Acta Universitatis Sapientiae

Informatica

Volume 16, Number 1, 2024

Sapientia Hungarian University of Transylvania
Scientia Publishing House

Journal Metrics (2022)

Impact Factor: **0.3**
Five Year Impact Factor: **0.8**
JCI: **0.09**
MEiN: **20**
Google Scholar h5-index: **21**
Google Scholar h5-median: **28**

Acta Universitatis Sapientiae Informatica
is covered by the following services:

ACM Digital Library
Baidu Scholar
Cabell's Journalytics
CNKI Scholar
CNPIEC – cnpLINKer Dimensions
DOAJ
EBSCO
Engineering Village
ExLibris
Google Scholar
Inspec
Japan Science and Technology Agency
J-Gate
JournalTOCs
KESLI-NDSL
MyScienceWork
Naver Academic
Naviga (Softweco)
QOAM
ReadCube
SCILIT
Semantic Scholar
Sherpa/RoMEO
TDNet
Ulrich's Periodicals Directory
WanFang Data
Web of Science – ESCI
WorldCat (OCLC)
zbMATH Open
X-MOL

Contents

Acta Univ. Sapientiae, Informatica, **16, 1** (2024) 1–19

DOI: 10.47745/ausi-2024-0001

TARNet for the Classification of Time Series with Missing Values: A Strong Baseline

Krisztian BUZA[a,b]
[a]Faculty of Finance and Accountancy
Budapest Business University
Buzogány utca 10-12
1049 Budapest, Hungary
chrisbuza@yahoo.com
0000-0002-7111-6452

Erik NOVAK[b]
[b]Department for Artificial Intelligence
Jožef Stefan Institute
Jamova cesta 39
1000 Ljubljana, Slovenia
erik.novak@ijs.si
0000-0002-7010-314X

Abstract. Due to sensor failures and other issues, real-world time series may contain missing values, often in consecutive segments. Classification of such time series is an important task with prominent applications in various domains such as medicine, manufacturing, social networks and environmental sciences. In this paper, we consider various approaches that have been designed for this task, in particular, fully-convolutional neural networks (FCNs) with sparsity-invariant convolution and dynamic time warping convolution. We compare their performance to that of a standard transformer, TARNet, which has *not* been tailored to the classification of time series with missing values. Our results indicate that even this simple transformer may outperform the aforementioned models that were designed to deal with missing values. As this observation is consistent for many datasets from various domains and various distributions of missing values, we conclude that transformers are an exceptionally strong baseline for the classification of time series with missing values. In order to support the reproduction of our results as well as follow-up works, we performed the aforementioned experiments on publicly available time series datasets using a publicly available implementation of TARNet.

Key words and phrases: time series classification, missing values, transformers, TARNet

1 Introduction

Time series classification is the common denominator of countless recognition tasks in numerous domains, including biology, medicine, healthcare, astronomy, geology, social networks, industry and finance. These tasks include biometric person identification, signature verification, speech recognition, earthquake prediction, the diagnosis of various diseases, intrusion detection and the detection of physical activity [1]–[6]. Due to the aforementioned applications, and many others, time series classification has been considered as one of the most prominent tasks in machine learning, for which various approaches have been introduced, including methods based on neural networks, Bayesian networks, hidden Markov models, genetic algorithms, support vector machines, decision trees, frequent patterns (also known as *motifs* or *shapelets*) and hubness-aware classifiers [7]–[15]. Related surveys provide more information on the topic [16], [17].

While the nearest neighbour with dynamic time warping (DTW) was considered "an exceptionally competitive classifier" [18] in the early 2000s, subsequent solutions were based on deep learning [14], [19]–[21]. Out of the many excellent works, we point out the study of Wang et al. [22], who found that a "fully convolutional network" (FCN) is a "strong baseline."

Most of the research, including the aforementioned studies, has been performed in the context of regularly sampled time series with constant time between subsequent observations. In contrast, as pointed out by Lim and Zohren [23], the analysis of time series containing missing values is rather understudied, despite its prominent applications in medicine [24], manufacturing [25], social networks [26], environmental sciences [27] and other domains. Nevertheless, as pointed out by Weerakody et al. [28], there is an increasing interest in methods to handle irregular time series due to "the growth of multi-sensor systems as well as the continued use of unstructured manual data recording mechanisms." Considering the classification of time series containing missing values, various approaches have been introduced, including methods based on convolutional networks [24], kernel methods [29] and end-to-end learning [30].

Inline with the advent of attention-based models [31], the most recent time series classifiers are based on the transformer architecture [32], a prominent representative of them is TARNet [33]. When training TARNet, time series reconstruction was considered as a surrogate task that allowed the model to learn an appropriate representation, therefore TARNet is especially promising for the classification of time series containing missing values.

In this paper, we focus on neural approaches for the classification of time series containing missing values. We point out that some neural networks were designed

to deal with missing values, such as FCN with sparsity-invariant convolution [34] or dynamic time warping convolution [35]. At the same time, attention-based models may potentially learn to ignore missing values on their own as they aim at identifying the most informative segments of time series. Therefore, transformers are very promising in the case of time series with missing values. To the best of our knowledge, our work is the first to explore the ability of transformers for the classification of time series with missing values. In particular, we compare the aforementioned time series transformer, TARNet, with various fully convolutional networks, including the variants designed to deal with missing values. We perform experiments on 30 publicly available time series datasets both in cases when values are missing uniformly at random and in cases when long continuous segments are missing. Our results show that vanilla TARNet outperforms the aforementioned networks designed to handle missing values which indicates that transformers are indeed very promising for the classification of time series with missing values.

2 Classification of Time Series with Missing Values

In this section, we describe the models used throughout our work. We begin with the formal definition of the problem, followed by neural networks designed for the classification of time series with missing values and TARNet.

2.1 Problem Formulation and Basic Definitions

Given a set C of class labels, and a set $\mathcal{D}$ of observed time series together with their class labels

$$\mathcal{D} = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}, \forall y^{(i)} \in C \tag{1}$$

where $x^{(i)}$ is an observed time series, $x^{(i)} = (x_1^{(i)}, \ldots, x_l^{(i)})$, we aim at finding a mathematical model $\mathcal{M}$ that is able to determine the class label $y' \in C$ of a new (test) time series $x' = (x'_1, \ldots, x'_l)$. The process of determining the values of the parameters of $\mathcal{M}$ is called *training*, while $\mathcal{D}$ is called *training data*, because $\mathcal{D}$ is used to find the appropriate values of the parameters.

Missing values are allowed in the aforementioned time series, i.e., each x_i within $x = (x_1, \ldots, x_l)$ is either a real number or a symbol indicating that the value is missing. In real-world applications, the ratio of missing values may be as high as 90% or even more [36].

2.2 SiConv: Sparsity-invariant Convolution

Considering convolutional neural networks working with data containing missing values, it is useful to encode the missing values by zeros because, in this case, the multiplication of a missing value (i.e., a zero) by the corresponding weight results in zero and, therefore the missing values are ignored in the weighted sum calculated by the convolutional layer, which is an intuitive behaviour. Furthermore, treating missing values as zeros is also inline with dropout, a widely-used regularisation for deep neural networks [37]. For these reasons, we follow the wide-spread convention of denoting missing values by zeros[1] [38], [39].

The intuition behind sparsity-invariant convolution [39] is to normalize the output of the convolutional layer according to the number of its non-missing inputs. As we focus on time series classification, we describe sparsity-invariant convolution in the context of time series. The input of SiConv is denoted as $x^{in} = (x_1^{in}, \ldots, x_l^{in})$, the size of the convolutional filter is s, the output is denoted as $x^{out} = (x_1^{out}, \ldots, x_{l-s+1}^{out})$. Let z_i be the number of non-zero (i.e., non-missing) values within the i-th convolutional window:

$$z_i = \sum_{j=0}^{s} \mathcal{I}(x_{i+j}^{in} \neq 0), \tag{2}$$

where $\mathcal{I}$ is an indicator function that returns 1 if its argument is true, otherwise it returns zero. Furthermore, let b and w_j denote the bias and weights of the convolutional filter. With these notations, the output of SiConv can be calculated as follows:

$$x_i^{out} = \frac{1}{z_i}\left(b + \sum_{j=0}^{s} w_j x_{i+j}^{in}\right), \tag{3}$$

if $z_i \neq 0$; otherwise: $x_i^{out} = 0$. SiConv is illustrated in Fig. 1.

We note that in case there are no missing values in the input of SiConv, SiConv is equivalent to "usual" convolution up to the scaling factor $z_i = s$. On the other hand, for those segments of the time series where all the inputs of SiConv are missing values (zeros) within the convolutional window, the output of SiConv is zero, which denotes a missing value according to our conventions. For these reasons, we may use SiConv not only in the first convolutional layer but in subsequent convolutional layers as well, especially in the case of highly sparse input. In such cases, each convolutional layer with SiConv decreases the ratio of missing values in the hidden representation.

[1] As the actual observations are made on a continuous scale, the probability of an observation being exactly zero is very low, therefore, denoting missing values by zeros does not cause confusion in case of the datasets we considered.

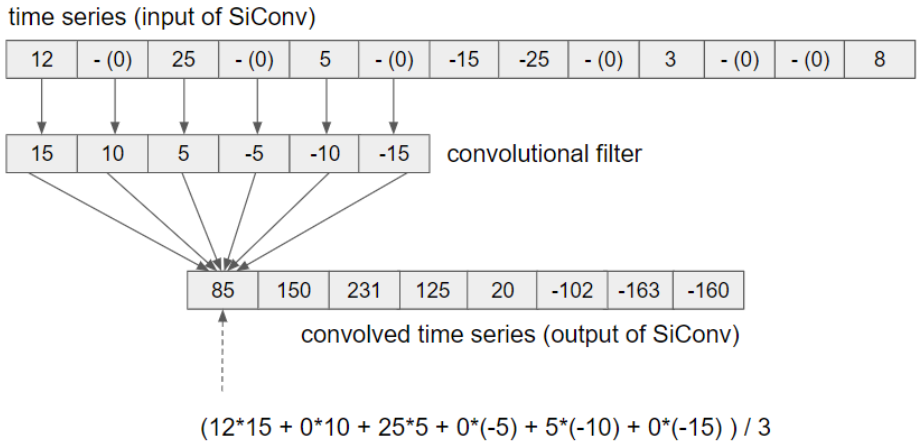


Figure 1: Illustration of sparsity-invariant convolution (SiConv). The input and output time series of SiConv are shown at the top and bottom, respectively. The convolutional filter in the center is expected to detect a decreasing trend. Compared to "usual" convolution, the difference is that the output values are divided by the number of non-missing input values.

2.3 DConv: Dynamic Time Warping Convolution

In a DConv block [35], dynamic time warping distances are calculated between the non-missing values of the time series segments and the convolutional kernel (i.e., the local pattern to be detected).

By design, DTW is able to compare time series of different lengths by matching the same value in the input time series to several values of the convolutional kernel. Thus, when comparing the convolutional kernel with a segment of the input containing missing values, we propose to simply keep the given (i.e., non-missing) values of the input segment and use that sequence in DTW calculations.

In the current study, we use dynamic time warping convolution together with "usual" convolution: our DConv block has two output channels, one for the values calculated by "usual" convolution and another one for the values calculated by dynamic time warping convolution. This is shown in Fig. 2.

Higher similarity between the time series segment and the convolutional kernel corresponds to higher values in the "usual" convolution. However, the opposite is true for DTW distances: in the case of DTW, the high similarity between the time series segment and the convolutional kernel is reflected by a distance close to zero. Therefore, to ensure that the activations on both channels are consistent, the

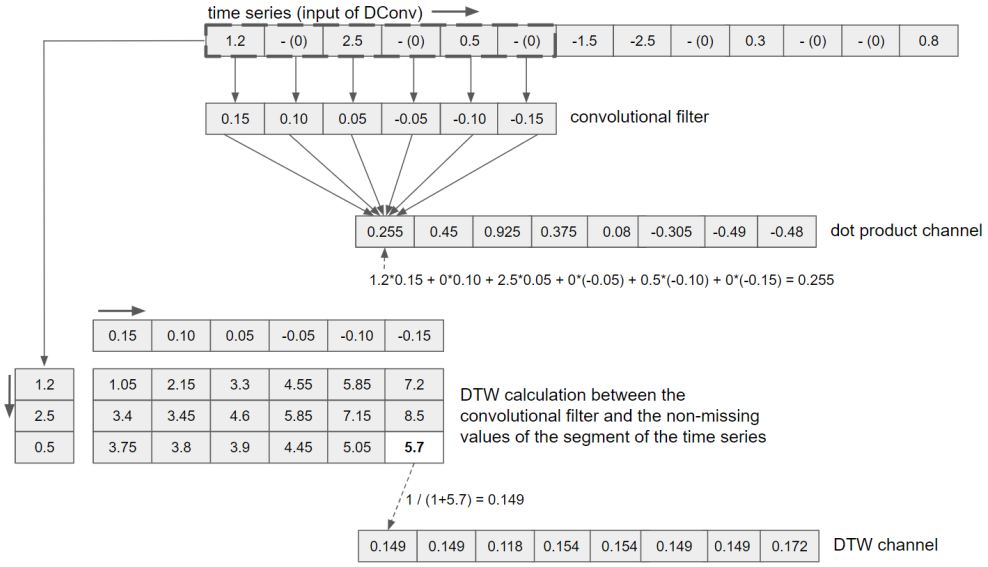


Figure 2: Our DConv block. We use dynamic time warping convolution together with "usual" convolution, thus our DConv block has two output channels, one for the values calculated by "usual" convolution and another one for the values calculated by dynamic time warping convolution. As dynamic time warping allows for comparing sequences of different length, missing values are simply omitted when we calculate the dynamic time warping distances.

activations of the DTW channel of our DConv block are calculated as follows:

$$out_{DTW}(t) = \frac{1}{1 + DTW(in[t : t + s], w)}, \tag{4}$$

where out_{DTW} denotes the activation of the DTW channel of the DConv block, $in[t : t + s]$ is the segment of the block's input between the t-th and $(t + s)$-th position[2], s is the size of the filter, w are the weights of the filter representing a local pattern and $DTW(.,.)$ is a function that calculates the DTW distance between two time series segments.

Training neural networks with DConv may be challenging because of the back-propagation of the gradients through the DTW calculations. Therefore, inline with [35], we propose to use DConv only in the first hidden layer of an FCN, and we train the FCN in a two-stage approach as follows: in the first stage, we train an analogous

[2]In Eq. (4) we use a Python-like syntax: the lower index, t is inclusive, the upper index, $t + s$ in exclusive in $in[t : t + s]$.

network with "usual" convolution instead of DConv. After this first stage, we freeze the weights of the DConv layer. In the subsequent stage, we calculate the activations of the DConv layer using those frozen weights, and we only train the rest of the network.

2.4 TARNet

Task-Aware Reconstruction for Time-Series Transformer, TARNet [33], is a transformer-based model tailored for solving a broad range of time-series-related tasks. Figure 3 depicts the model's architecture. To make sure that our manuscript is self-contained, we offer a brief overview of the model.

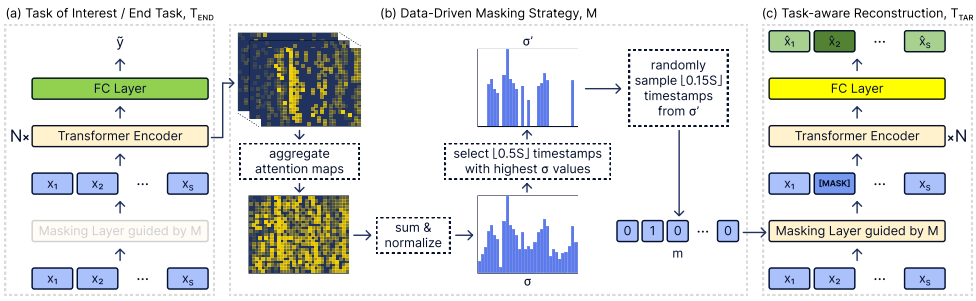


Figure 3: The TARNet architecture, as presented in the original paper [33].

2.4.1 Base model

The TARNet model consists of Transformer Encoders [31] and task-specific layers. Initially, the input time series are first mean-standardized per variable dimension. Next, it is linearly projected into a d-dimensional vector space, where d represents the dimension of the Transformer model's sequence element representations. Since the Transformer architecture is indifferent to the ordering of the input, positional encoding is added to the input vectors, enabling the model to recognize the sequential nature of the input time series. The data then traverses multiple layers of Transformer Encoder blocks. Finally, the output values are passed through the task-specific layers to produce the final output values. The following paragraphs describe how the base model is modified to solve specific tasks.

2.4.2 End task

The end task, denoted as T_{END}, is the main task the model aims to solve. As we focus on time series classification, in our case, the end task is time series classification. In order to solve this task, we used task-specific layers of the base model as follows: the values produced by the final transformer encoder block are fed through two fully connected layers and ReLU activation functions. The predictions are then passed through the softmax operation to give the probability distribution over the predefined set of classes. As we consider a classification task as the loss of the end task, $\mathcal{L}_{END}$, we utilized cross-entropy loss.

2.4.3 Task-aware reconstruction

The goal of the task-aware reconstruction (T_{TAR}) is to enhance the performance of the end task T_{END} by learning the data representations through reconstructing the input time series.

The reconstruction task is performed as follows: the original input time-series data undergoes a data-driven masking strategy, where specific timestamps being crucial for accurately solving the end task T_{END} are masked (hidden). Next, the masked time-series data traverses the sequence of Transformer Encoders shared with T_{END}, and finally, through two fully connected layers and the RELU activation functions. The label for this task is the raw input time-series data. To ensure an accurate reconstruction, the T_{TAR} loss function is the Mean Square Error (MSE) between the raw input data and the predicted values. The average MSE loss is computed separately for masked ($\mathcal{L}_{masked}$) and unmasked ($\mathcal{L}_{unmasked}$) time-series values. The final T_{TAR} loss is expressed as the linear combination of the masked and unmasked loss values:

$$\mathcal{L}_{TAR} = \lambda \mathcal{L}_{masked} + (1 - \lambda)\mathcal{L}_{unmasked},$$

where λ is a hyper-parameter controlling the relative weights between the losses. It is advisable to set $\lambda > 0.5$ since masked timestamps are more crucial for the end task than unmasked ones.

The total TARNet model loss function is given by

$$\mathcal{L}_{total} = \mu \mathcal{L}_{TAR} + (1 - \mu)\mathcal{L}_{END},$$

with μ as a hyper-parameter dictating the relative weights between the two task losses.

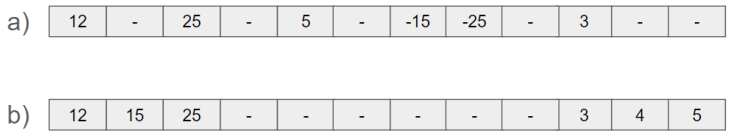


Figure 4: Missing value distributions considered in our work: a) missing uniformly at random, and b) missing in batches.

3 Distribution of Missing Values

In order to be able to study the performance of the considered models under various distributions of missing values, we considered time series datasets in which all the observations are given and we introduced missing values by setting 20%, 40%, 60% or 80% of the given values to zero which denotes missing value according to our conventions (see Section 2.2). In particular, we considered the following distributions of missing values.

- *Missing uniformly at random:* each value has the same probability of being a missing value; the given observations are distributed uniformly throughout the time series. In practice, this happens when different sensors make observations with different frequencies, but the data is represented at the highest sampling frequency.

- *Missing in batches:* in this setting, for each time series, we select a random position and beginning from that position, a continuous sequence of values is set to missing values. The length of this continuous sequence corresponds to 20%, 40%, 60% or 80% of the length of the original time series. Such a distribution of missing values is observed in practice when sensor failures occur.

These distributions are illustrated in Fig. 4. In the time series exemplified in the figure, 50% of all the values are missing, nevertheless, in our experiments, we set the ratio of missing values to 20%, 40%, 60% or 80%.

4 Experimental Evaluation

The goal of our experiments is to examine the ability of TARNet to classify time series with missing values.

4.1 Datasets

We performed experiments on 30 publicly available time series datasets from *The UEA & UCR Time Series Classification Repository*.[3] The datasets used in our experiments are listed[4] in the first column on Tab. 1 and Tab. 2.

As the time series of the aforementioned datasets do not contain missing values, we randomly selected 20%, 40%, 60% or 80% of the values of each time series and replaced them with missing values according to the distributions described in Section 3.

4.2 Compared Models

In both cases mentioned in Section 3 (that is, missing uniformly at random and missing in batches), we compared the following models:

1. FCN: fully convolutional neural network. When implementing FCN [22], we replaced the final "global average pooling" (GAP) layer with a fully connected layer with 128 units because we observed systematically better performance in the case of the fully connected layer.

2. FCN-Si: FCN with sparsity-invariant convolution.

3. FCN-DConv: FCN with dynamic time warping convolution (see Section 2.3) in the first convolutional layer, and "usual" convolution in the other convolutional layers.[5]

4. TARNet: the time-series transformer described in Section 2.4. Our experiments are based on the publicly available implementation of TARNet.[6]

4.3 Experimental Settings

We performed experiments according to the 10-fold cross-validation protocol. That is, we partition the data into 10 splits. Out of them, 9 splits are used as training data,

[3] https://www.timeseriesclassification.com

[4] Please note that some of the names are abbreviated, in particular: Chlorine = ChlorineConcentration, Diatom = DiatomSizeReduction, Italy = ItalyPowerDemand, Melbourne = MelbournePedestrian, SonyAIBO1 = SonyAIBORobotSurface1, SonyAIBO2 = SonyAIBORobotSurface2

[5] We implemented both FCN, FCN-Si and FCN-DConv in PyTorch. We trained all of them for 100 epochs. While doing so, we used cross-entropy loss and the Adam optimizer [40] with a learning rate of 10^{-5} and batch size of 16.

[6] We set the learning rate to 0.001, the batch size to 64 and the number of training epochs to 100. As for the other hyperparameters, we used their default values according to the TARNet repository: https://github.com/ranakroychowdhury/TARNet.

Table 1: Classification accuracy (averaged over 10 folds) and its standard deviation (after the ± symbol) in the case when 80% of the observations are missing uniformly at random. For each dataset, the best approach is underlined. We provide three symbols • or ◦ which denotes whether the differences between TARNet and (a) FCN, (b) FCN-Si and (c) FCN-DConv are statistically significant (•) or not (◦) according to paired t-test at a significance level of $p = 0.01$.

Dataset	FCN	FCN-Si	FCN-DConv	TARNet
Adiac	0.102 ± 0.022	0.113 ± 0.021	0.115 ± 0.030	0.307 ± 0.042 •/•/•
ArrowHead	0.635 ± 0.075	0.678 ± 0.095	0.659 ± 0.097	0.625 ± 0.122 ◦/◦/◦
Chinatown	0.670 ± 0.050	0.714 ± 0.011	0.761 ± 0.026	0.868 ± 0.034 •/•/•
Chlorine	0.438 ± 0.021	0.468 ± 0.018	0.440 ± 0.025	0.561 ± 0.022 •/•/•
Diatom	0.764 ± 0.066	0.839 ± 0.048	0.944 ± 0.039	0.978 ± 0.020 •/•/◦
Earthquakes	0.798 ± 0.009	0.759 ± 0.037	0.796 ± 0.010	0.774 ± 0.070 ◦/◦/◦
ECG200	0.755 ± 0.108	0.750 ± 0.077	0.730 ± 0.093	0.670 ± 0.068 ◦/◦/◦
ECG5000	0.921 ± 0.005	0.925 ± 0.004	0.932 ± 0.010	0.928 ± 0.010 ◦/◦/◦
ECGFiveDays	0.851 ± 0.027	0.837 ± 0.056	0.904 ± 0.024	0.884 ± 0.033 ◦/◦/◦
FaceAll	0.636 ± 0.047	0.521 ± 0.058	0.667 ± 0.042	0.697 ± 0.028 •/•/◦
FiftyWords	0.496 ± 0.033	0.398 ± 0.021	0.549 ± 0.034	0.595 ± 0.049 •/•/◦
FordA	0.498 ± 0.029	0.504 ± 0.016	0.490 ± 0.025	0.632 ± 0.033 •/•/•
FordB	0.508 ± 0.019	0.545 ± 0.026	0.503 ± 0.017	0.583 ± 0.053 •/◦/•
Haptics	0.367 ± 0.069	0.365 ± 0.054	0.393 ± 0.043	0.385 ± 0.063 ◦/◦/◦
Italy	0.701 ± 0.039	0.698 ± 0.024	0.758 ± 0.030	0.845 ± 0.021 •/•/•
Mallat	0.908 ± 0.046	0.921 ± 0.038	0.917 ± 0.040	0.971 ± 0.015 •/•/•
Melbourne	0.188 ± 0.018	0.150 ± 0.012	0.224 ± 0.018	0.629 ± 0.033 •/•/•
MoteStrain	0.856 ± 0.031	0.884 ± 0.014	0.893 ± 0.026	0.901 ± 0.032 ◦/◦/◦
OSULeaf	0.396 ± 0.086	0.392 ± 0.085	0.426 ± 0.070	0.414 ± 0.057 ◦/◦/◦
Phoneme	0.113 ± 0.022	0.130 ± 0.015	0.117 ± 0.023	0.205 ± 0.020 •/•/•
Plane	0.814 ± 0.045	0.881 ± 0.061	0.895 ± 0.067	0.819 ± 0.073 ◦/◦/◦
PowerCons	0.794 ± 0.052	0.811 ± 0.063	0.819 ± 0.060	0.911 ± 0.043 •/•/•
SonyAIBO1	0.759 ± 0.054	0.763 ± 0.046	0.800 ± 0.046	0.844 ± 0.054 •/•/◦
SonyAIBO2	0.813 ± 0.032	0.799 ± 0.038	0.797 ± 0.029	0.788 ± 0.040 ◦/◦/◦
Strawberry	0.688 ± 0.030	0.713 ± 0.031	0.806 ± 0.045	0.824 ± 0.038 •/•/◦
SwedishLeaf	0.466 ± 0.035	0.495 ± 0.041	0.584 ± 0.051	0.644 ± 0.046 •/•/◦
Symbols	0.879 ± 0.033	0.925 ± 0.022	0.939 ± 0.014	0.925 ± 0.055 ◦/◦/◦
TwoLeadECG	0.694 ± 0.043	0.721 ± 0.056	0.802 ± 0.028	0.860 ± 0.058 •/•/◦
Wafer	0.894 ± 0.001	0.906 ± 0.024	0.948 ± 0.028	0.990 ± 0.004 •/•/•
Yoga	0.656 ± 0.036	0.724 ± 0.034	0.766 ± 0.030	0.866 ± 0.025 •/•/•

Table 2: Classification accuracy (averaged over 10 folds) and its standard deviation (after the ± symbol) in the case when 80% of the observations are missing in batches. For each dataset, the best approach is underlined. We provide three symbols • or ◦ which denotes whether the differences between TARNet and (a) FCN, (b) FCN-Si and (c) FCN-DConv are statistically significant (•) or not (◦) according to paired t-test at a significance level of $p = 0.01$.

Dataset	FCN	FCN-Si	FCN-DConv	TARNet
Adiac	0.122 ± 0.034	0.091 ± 0.019	0.146 ± 0.031	0.337 ± 0.061 •/•/•
ArrowHead	0.407 ± 0.092	0.407 ± 0.091	0.378 ± 0.108	0.564 ± 0.094 •/•/•
Chinatown	0.653 ± 0.091	0.608 ± 0.077	0.812 ± 0.073	0.934 ± 0.022 •/•/•
Chlorine	0.560 ± 0.020	0.542 ± 0.017	0.548 ± 0.018	0.588 ± 0.019 ◦/•/•
Diatom	0.609 ± 0.077	0.416 ± 0.090	0.693 ± 0.070	0.758 ± 0.082 •/•/◦
Earthquakes	0.798 ± 0.009	0.798 ± 0.009	0.774 ± 0.030	0.807 ± 0.027 ◦/◦/◦
ECG200	0.695 ± 0.061	0.695 ± 0.088	0.685 ± 0.092	0.755 ± 0.057 •/◦/◦
ECG5000	0.928 ± 0.007	0.920 ± 0.004	0.923 ± 0.006	0.936 ± 0.012 ◦/•/◦
ECGFiveDays	0.894 ± 0.046	0.776 ± 0.049	0.882 ± 0.037	0.898 ± 0.024 ◦/•/◦
FaceAll	0.603 ± 0.024	0.392 ± 0.041	0.540 ± 0.031	0.720 ± 0.028 •/•/•
FiftyWords	0.232 ± 0.031	0.179 ± 0.032	0.223 ± 0.016	0.328 ± 0.043 •/•/•
FordA	0.590 ± 0.020	0.708 ± 0.024	0.583 ± 0.023	0.686 ± 0.026 •/◦/•
FordB	0.580 ± 0.020	0.702 ± 0.016	0.578 ± 0.017	0.679 ± 0.016 •/•/•
Haptics	0.337 ± 0.048	0.330 ± 0.058	0.294 ± 0.069	0.428 ± 0.087 ◦/•/•
Italy	0.735 ± 0.039	0.762 ± 0.042	0.723 ± 0.039	0.786 ± 0.036 •/◦/•
Mallat	0.513 ± 0.039	0.361 ± 0.031	0.542 ± 0.013	0.767 ± 0.058 •/•/•
Melbourne	0.154 ± 0.025	0.106 ± 0.013	0.157 ± 0.024	0.431 ± 0.040 •/•/•
MoteStrain	0.819 ± 0.028	0.787 ± 0.049	0.796 ± 0.036	0.865 ± 0.044 •/•/•
OSULeaf	0.342 ± 0.056	0.339 ± 0.066	0.312 ± 0.050	0.496 ± 0.058 •/•/•
Phoneme	0.124 ± 0.018	0.143 ± 0.017	0.109 ± 0.015	0.161 ± 0.035 ◦/◦/•
Plane	0.619 ± 0.077	0.367 ± 0.088	0.519 ± 0.075	0.681 ± 0.109 ◦/•/•
PowerCons	0.619 ± 0.103	0.686 ± 0.092	0.622 ± 0.074	0.647 ± 0.061 ◦/◦/◦
SonyAIBO1	0.828 ± 0.040	0.841 ± 0.042	0.865 ± 0.044	0.905 ± 0.028 •/•/◦
SonyAIBO2	0.843 ± 0.031	0.804 ± 0.036	0.837 ± 0.027	0.846 ± 0.025 ◦/•/◦
Strawberry	0.715 ± 0.025	0.657 ± 0.035	0.735 ± 0.046	0.825 ± 0.033 •/•/•
SwedishLeaf	0.400 ± 0.049	0.230 ± 0.035	0.380 ± 0.045	0.657 ± 0.051 •/•/•
Symbols	0.625 ± 0.054	0.707 ± 0.040	0.487 ± 0.033	0.808 ± 0.032 •/•/•
TwoLeadECG	0.676 ± 0.048	0.648 ± 0.024	0.639 ± 0.043	0.793 ± 0.050 •/•/•
Wafer	0.894 ± 0.001	0.894 ± 0.001	0.894 ± 0.001	0.987 ± 0.008 •/•/•
Yoga	0.696 ± 0.031	0.617 ± 0.022	0.675 ± 0.017	0.801 ± 0.024 •/•/•

while the remaining one is used as test data. The process is repeated 10-times, in each round of the cross-validation, a different split plays the role of the test set.

We evaluated the predicted time series in terms of classification accuracy, i.e., the ratio of correctly classified time series. We used paired t-test at a significance level (p-value) of 0.01 in order to assess whether the observed differences between our approach and its competitors are statistically significant or not.[7]

4.4 Results

Tab. 1 and Tab. 2 summarize our results in case when observations are missing uniformly at random and in case when observations are missing in a batch (entire subsequences are missing), respectively. Additionally, Fig. 5 and Fig. 6 show the classification accuracy as a function of the ratio of missing observations for selected datasets.

As one can see, in the vast majority of the datasets, TARNet outperforms all variants of FCN, including the ones that were designed to perform well in case of input with missing values. In most cases, the difference is statistically significant, see Tab. 1 and Tab. 2. In those few cases when TARNet performs worse than any of the examined models, the difference is not significant statistically. The only exception is the FordB dataset in the case if the observations are missing in batches, see Tab. 2. In this case, FCN with sparsity-invariant convolution performs significantly better than TARNet. Taking all the results into account, we conclude that TARNet is an extremely competitive baseline for the classification of time series with missing values. Most likely, this can be attributed to the powerful attention mechanism that allows the transformer to focus on the given part of the time series and to the fact that time series reconstruction is considered a surrogate task when TARNet is trained so that the model is encouraged to learn an appropriate representation even in the absence of a relatively large fraction of the observations.

[7] For each test time series t_j, let us consider an indicator variable $\mathcal{I}_j^{\mathcal{M}}$ which denotes whether the prediction of model $\mathcal{M}$ is correct ('1') or not ('0') for that instance. The difference between the accuracy of two models $\mathcal{M}_1$ and $\mathcal{M}_2$ may be written as $\sum_j \mathcal{I}_j^{\mathcal{M}_1} - \mathcal{I}_j^{\mathcal{M}_2}$. Considering $\mathcal{I}_j^{\mathcal{M}_1}$ and $\mathcal{I}_j^{\mathcal{M}_2}$ as random variables, their difference is a random variable as well, and the difference between the two models in terms of their accuracy can be seen as a sum of these random variables. According to the central limit theorem, the aforementioned sum will approximately follow a normal distribution in case of sufficiently large test data.

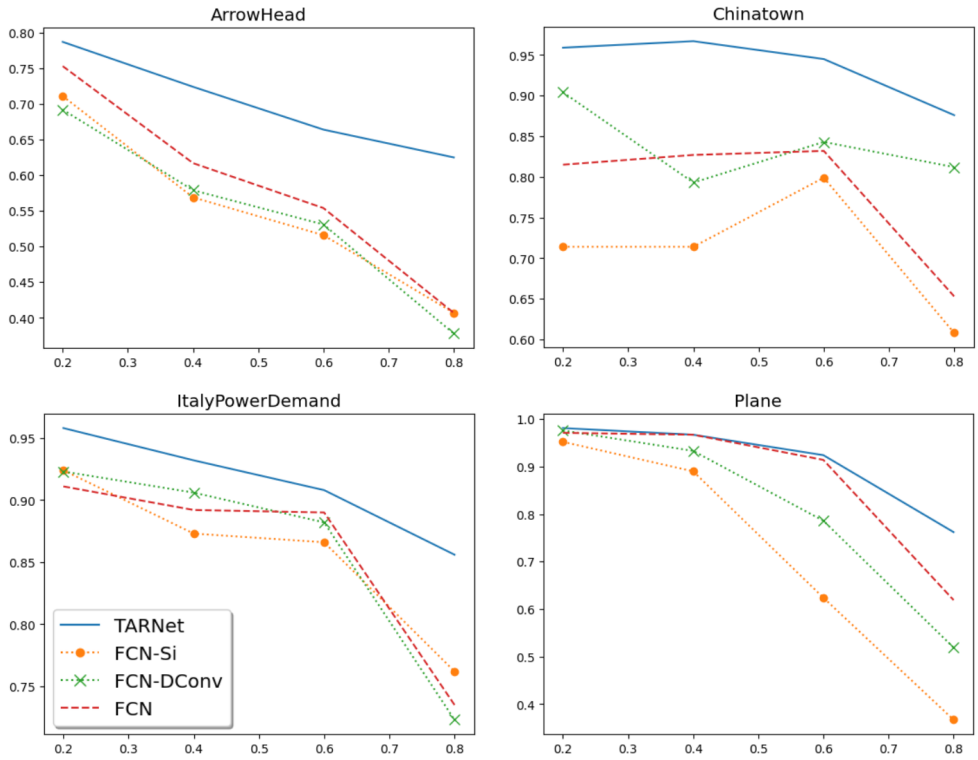


Figure 5: Classification accuracy as function of the ratio of missing observations (averaged over 10 folds) in case when the observations are missing uniformly at randoms.

4.4.1 Model parameters

According to our observations, FCN and its variants have approximately 4M parameters, whereas TARNet has only about 0.47M parameters. This indicates that the advantage of TARNet can indeed be attributed to the transformer architecture (including the attention mechanism) and not to the size of the model and that relatively simple transformers should be considered as alternatives to convolutional networks in case of time series classification with missing values.

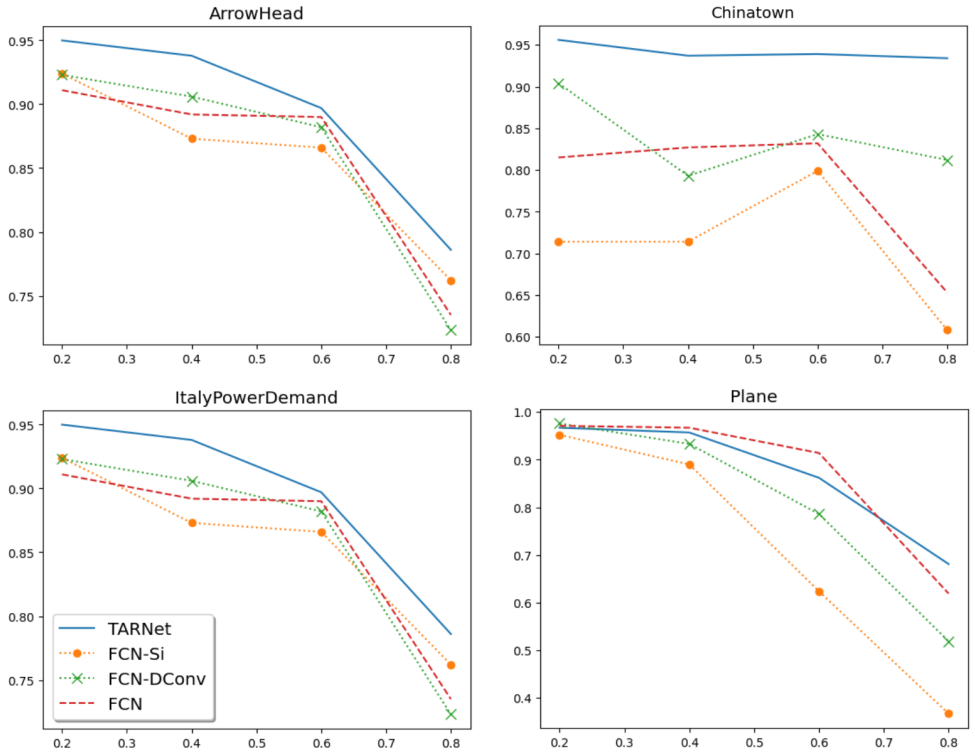


Figure 6: Classification accuracy as function of the ratio of missing observations (averaged over 10 folds) in case when the observations are missing in batches.

5 Conclusions

In this paper, we focused on the classification of time series with missing values, which is an understudied problem, despite its prominent applications in medicine, industry and science. We compared the performance of a recent time series transformer, TARNet, with models that were designed to work with sparse and irregularly sampled time series. To the best of our knowledge, ours is the first work that studied the ability of transformers to classify time series with missing observations. Our results show that TARNet is an exceptionally competitive baseline.

Due to the increasing use of multi-sensor systems, as well as the continued use of unstructured manual data recording mechanisms, we expect more and more applications with time series containing missing values. Our results show that TARNet is a reasonable default choice in such cases.

Data Availability: The datasets used in this study are available from *The UEA & UCR Time Series Classification Repository* (https://www.timeseriesclassification.com).

Conflicts of Interest: The authors declare no conflicts of interest.

References

[1] M. Okawa, "Time-series averaging and local stability-weighted dynamic time warping for online signature verification," *Pattern Recognition*, vol. 112, 2021 (⇒ 2).

[2] M. Antal, L. Z. Szabó, and T. Tordai, "Online signature verification on MOBISIG finger-drawn signature corpus," *Mobile Information Systems*, vol. 2018, pp. 1–15, 2018 (⇒ 2).

[3] K. Buza, A. Nanopoulos, L. Schmidt-Thieme, and J. Koller, "Fast classification of electrocardiograph signals via instance selection," in *First International Conference on Healthcare Informatics, Imaging and Systems Biology*, IEEE, 2011, pp. 9–16 (⇒ 2).

[4] S. M. Szilagyi, L. Szilagyi, D. Iclanzan, and Z. Benyo, "Unified neural network based adaptive ECG signal analysis and compression," *Scientific Bulletin of the Politechnica University of Timisoara, Transactions on Automatic Control and Computer Science*, vol. 51, pp. 27–36, 2006 (⇒ 2).

[5] M. Antal and E. Egyed-Zsigmond, "Intrusion detection using mouse dynamics," *IET Biometrics*, vol. 8, no. 5, pp. 285–294, 2019 (⇒ 2).

[6] L. Dénes-Fazakas, L. Szilágyi, J. Tasic, L. Kovács, and G. Eigner, "Detection of physical activity using machine learning methods," in *20th International Symposium on Computational Intelligence and Informatics*, IEEE, 2020, pp. 167–172 (⇒ 2).

[7] K. Buza, "ASTERICS: Projection-based classification of EEG with asymmetric loss linear regression and genetic algorithm," in *14th International Symposium on Applied Computational Intelligence and Informatics*, IEEE, 2020, pp. 35–40 (⇒ 2).

[8] B. Esmael, A. Arnaout, R. K. Fruhwirth, and G. Thonhauser, "Improving time series classification using Hidden Markov Models," in *12th International Conference on Hybrid Intelligent Systems*, IEEE, 2012, pp. 502–507 (⇒ 2).

[9] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: A review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019 (⇒ 2).

[10] M. Hüsken and P. Stagge, "Recurrent neural networks for time series classification," *Neurocomputing*, vol. 50, pp. 223–235, 2003 (⇒ 2).

[11] D. Jankowski, K. Jackowski, and B. Cyganek, "Learning decision trees from data streams with concept drift," *Procedia Computer Science*, vol. 80, pp. 1682–1691, 2016 (⇒ 2).

[12] J. Lines, L. M. Davis, J. Hills, and A. Bagnall, "A shapelet transform for time series classification," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2012, pp. 289–297 (⇒ 2).

[13] V. Pavlovic, B. J. Frey, and T. S. Huang, "Time-series classification using mixed-state Dynamic Bayesian Networks," in *Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE, vol. 2, 1999, pp. 609–615 (⇒ 2).

[14] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu, "Convolutional neural networks for time series classification," *Journal of Systems Engineering and Electronics*, vol. 28, no. 1, pp. 162–169, 2017 (⇒ 2).

[15] M. Radovanović, A. Nanopoulos, and M. Ivanović, "Time-series classification in many intrinsic dimensions," in *Proceedings of the International Conference on Data Mining*, SIAM, 2010, pp. 677–688 (⇒ 2).

[16] K. Buza, "Time series classification and its applications," in *Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics*, 2018, pp. 1–4 (⇒ 2).

[17] N. Tomašev, K. Buza, K. Marussy, and P. B. Kis, "Hubness-aware classification, instance selection and feature construction: Survey and extensions to time-series," *Feature Selection for Data and Pattern Recognition*, pp. 231–262, 2015 (⇒ 2).

[18] X. Xi, E. Keogh, C. Shelton, L. Wei, and C. A. Ratanamahatana, "Fast time series classification using numerosity reduction," in *Proceedings of the 23rd International Conference on Machine learning*, 2006, pp. 1033–1040 (⇒ 2).

[19] W. Chen and K. Shi, "A deep learning framework for time series classification using relative position matrix and convolutional neural network," *Neurocomputing*, vol. 359, pp. 384–394, 2019 (⇒ 2).

[20] F. Guzy and M. Woźniak, "Employing dropout regularization to classify recurring drifted data streams," in *International Joint Conference on Neural Networks*, IEEE, 2020, pp. 1–7 (⇒ 2).

[21] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, "Time series classification using multi-channels deep convolutional neural networks," in *International Conference on Web-age Information Management*, Springer, 2014, pp. 298–310 (⇒ 2).

[22] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *International joint conference on neural networks*, IEEE, 2017, pp. 1578–1585 (⇒ 2, 10).

[23] B. Lim and S. Zohren, "Time-series forecasting with deep learning: A survey," *Philosophical Transactions of the Royal Society A*, vol. 379, no. 2194, 2021 (⇒ 2).

[24] S. C.-X. Li and B. Marlin, "Learning from irregularly-sampled time series: A missing data perspective," in *International Conference on Machine Learning*, PMLR, 2020, pp. 5937–5946 (⇒ 2).

[25] C. A. Steed, W. Halsey, R. Dehoff, S. L. Yoder, V. Paquit, and S. Powers, "Falcon: Visual analysis of large, irregularly sampled, and multivariate time series data in additive manufacturing," *Computers & Graphics*, vol. 63, pp. 50–64, 2017 (⇒ 2).

[26] Z. Liu, Y. Yang, W. Huang, Z. Tang, N. Li, and F. Wu, "How do your neighbors disclose your information: Social-aware time series imputation," in *The World Wide Web Conference*, 2019, pp. 1164–1174 (⇒ 2).

[27] S. Dilmaghani, I. C. Henry, P. Soonthornnonda, E. R. Christensen, and R. C. Henry, "Harmonic analysis of environmental time series with missing data or irregular sample spacing," *Environmental Science & Technology*, vol. 41, no. 20, pp. 7030–7038, 2007 (⇒ 2).

[28] P. B. Weerakody, K. W. Wong, G. Wang, and W. Ela, "A review of irregular time series data handling with gated recurrent neural networks," *Neurocomputing*, vol. 441, pp. 161–178, 2021 (⇒ 2).

[29] S. C.-X. Li and B. M. Marlin, "Classification of sparse and irregularly sampled time series with mixtures of expected gaussian kernels and random features," in *Conference on Uncertainty in Artificial Intelligence*, 2015, pp. 484–493 (⇒ 2).

[30] S. C.-X. Li and B. M. Marlin, "A scalable end-to-end gaussian process adapter for irregularly sampled time series classification," *Advances in Neural Information Processing Systems*, vol. 29, 2016 (⇒ 2).

[31] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017 (⇒ 2, 7).

[32] M. Jin, Q. Wen, Y. Liang, *et al.*, "Large models for time series and spatio-temporal data: A survey and outlook," *arXiv 2310.10196*, 2023 (⇒ 2).

[33] R. R. Chowdhury, X. Zhang, J. Shang, R. K. Gupta, and D. Hong, "Tarnet: Task-aware reconstruction for time-series transformer," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 212–220 (⇒ 2, 7).

[34] K. Buza, "Sparsity-invariant convolution for forecasting irregularly sampled time series," in *International Conference on Computational Collective Intelligence*, Springer, 2023, pp. 151–162 (⇒ 3).

[35] K. Buza, "Classification of sparse and irregularly sampled time series with convolutional neural networks," in *9th SIGKDD International Workshop on Mining and Learning from Time Series*, 2023 (⇒ 3, 5, 6).

[36] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, "Recurrent neural networks for multivariate time series with missing values," *Scientific Reports*, vol. 8, no. 1, p. 6085, 2018 (⇒ 3).

[37] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014 (⇒ 4).

[38] A. N. Ramesh, F. Giovanneschi, and M. A. González-Huici, "Siunet: Sparsity invariant u-net for edge-aware depth completion," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023, pp. 5818–5827 (⇒ 4).

[39] J. Uhrig, N. Schneider, L. Schneider, U. Franke, T. Brox, and A. Geiger, "Sparsity invariant cnns," in *International Conference on 3D Vision*, IEEE, 2017, pp. 11–20 (⇒ 4).

[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv 1412.6980*, 2014 (⇒ 10).

Received: 24.01.2024; *Revised: 06.05.2024*; *Accepted: 20.05.2024*

Acta Univ. Sapientiae, Informatica, **16, 1** (2024) 20–38

DOI: 10.47745/ausi-2024-0002

A Memory-efficient Algorithm for Conservative Cuts in Disjoint Bilinear Programming

Xiaosong DING
International Business School
Beijing Foreign Studies University
Beijing, P.R.China, 100089
dingxiaosong@bfsu.edu.cn
0000-0003-3376-5774

Chao LIU
International Business School
Beijing Foreign Studies University
Beijing, P.R.China, 100089
202220316001@bfsu.edu.cn

Jun MA*
International Business School
Beijing Foreign Studies University
Beijing, P.R.China, 100089
*Corresponding author
202220316002@bfsu.edu.cn

Xi CHEN
International Business School
Beijing Foreign Studies University
Beijing, P.R.China, 100089
chenxi0109@bfsu.edu.cn
0000-0002-5803-8840

Qing SUN
International Business School
Beijing Foreign Studies University
Beijing, P.R.China, 100089
sy2009050823@163.com

Abstract. This paper presents a memory-efficient approach to the combinatorial challenges associated with the process in search of conservative cuts for degeneracy removal in disjoint bilinear programming. Drawing inspiration from Pascal's Triangle, the new randomized approach surpasses its predecessor from the perspective of memory, and thereby enables the possibility of its application to disjoint bilinear models with high degrees of degeneracy in imprecise decision analysis on a PC. Numerical reports are provided to validate its computational performance.

Key words and phrases: Conservative cuts, degeneracy, disjoint bilinear programming

1 Introduction

A typical disjoint bilinear programming (DBLP) intends to minimize a non-convex quadratic function over disjoint constraint sets defined by two bounded and non-empty polytopes. Mathematically, it can be formulated as below,

$$\begin{aligned} \min\ & f(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{c}^t\boldsymbol{x} + \boldsymbol{d}^t\boldsymbol{y} + \boldsymbol{x}^t\boldsymbol{Q}\boldsymbol{y}, \\ \text{s.t. } & \boldsymbol{x} \in \boldsymbol{X} = \{\boldsymbol{x} \in \mathbb{R}^{n_1} : \boldsymbol{A}\boldsymbol{x} \le \boldsymbol{a},\ \boldsymbol{x} \ge 0\}, \quad \boldsymbol{A} \in \mathbb{R}^{m_1 \times n_1},\ \boldsymbol{a} \in \mathbb{R}^{m_1}, \\ & \boldsymbol{y} \in \boldsymbol{Y} = \{\boldsymbol{y} \in \mathbb{R}^{n_2} : \boldsymbol{B}\boldsymbol{y} \le \boldsymbol{b},\ \boldsymbol{y} \ge 0\}, \quad \boldsymbol{B} \in \mathbb{R}^{m_2 \times n_2},\ \boldsymbol{b} \in \mathbb{R}^{m_2}. \end{aligned} \tag{1}$$

As a subset of bilinear programming, DBLP is a mathematical optimization framework that has gained significant attention due to its extensive applications in various fields including game theory, facility location, numerical linear algebra and stochastic processes; see [1]. More recent applications also cover DBLP related issues in supply chain management [2], [3], chemical engineering [4], two-dimensional packing [5], Markov decision process [6], [7], operations research [8], [9], etc. This paper focuses on the field of imprecise decision analysis, in which a typical decision model intends to solve many disjoint bilinear programs; see for example [10]–[15]. Each program in general possesses no more than 100 dimensions, but may encounter various degrees of degeneracy ranging from 1 to 5.

The unique challenge in solving DBLP lies in its bilinear objective function and disjoint constraints. Based on the structural properties, various solution techniques have been developed, among which, two major deterministic approaches are cutting plane methods and branch and bound methods.

In cutting plane methods, great effort has been devoted to the establishment of deep cuts like concavity cuts [16], polar cuts [17], decomposition cuts [18], etc. Nevertheless, the computational issue of degeneracy arising at a local solution can be frequently confronted in real-world applications. The development of an effective cut at a degenerate vertex has been long-standing with few computational results [17], [19]–[21]. Recently, the concept of conservative cuts was first introduced in [22]. From theoretical and computational viewpoints, a conservative cut neither sacrifices the local optimum by pivoting to a non-degenerate neighboring vertex [19], nor imposes too much computational load by generating a disjunctive cut [19], [23]. Accompanying the concept, a distance-following algorithm was proposed in search of a conservative cut, in which considerable computational effort is spent in the projection operation for each qualified adjacent vertex with respect to an established hyperplane. Additionally, the investigations of both a candidate's neighborhood and the distance between the degenerate vertex and an established hyperplane appear time-consuming. In [21], several heuristic algorithms aiming to further improve the computational performance for the location of a conservative hyperplane

were proposed, among which, **Algorithm 1** and **5** (will be referred to as **Random** throughout the rest of this paper; see **Appendix**), appeared quite promising in terms of computing time. By utilizing the property of non-uniqueness of a conservative hyperplane, **Random** is able to solve the bilinear models with low degrees of degeneracy arising in imprecise decision analysis, but may suffer from memory problems when the degree of degeneracy rises.

Since confronting heavy storage load of memory can severely restrict its application to a cutting plane method on PCs, we try to improve **Random** from the perspective of memory by utilizing Pascal's Triangle. With a static Pascal's Triangle table, we develop the inverse function of $nchoosek\_enum(n, k, i)$ to derive the serial number of a combination. It spares the need to save all combinations during the search in order to keep away from the memory problem. The improvement comes at a cost of computing time, and well illustrates the algorithmic space-time continuum in data structure.

In what follows, **Section 2** briefly describes the issue of degeneracy arising in the local optimization phase of a cutting plane method for DBLP. **Section 3** develops the memory-efficient algorithm for the location a conservative hyperplane. **Section 4** discusses the generation of required test instances, following which **Section 5** reports their computational performance based on the programs in imprecise decision analysis. **Section 6** concludes our paper.

2 Degeneracy

The essential solution property of DBLP exploited in the local optimization phase of almost all cutting plane methods is that even though $f(\boldsymbol{x}, \boldsymbol{y})$ is not quasi-concave, the global optimizer, $(\boldsymbol{x}^*, \boldsymbol{y}^*)$, is attained at a vertex of $\boldsymbol{X} \times \boldsymbol{Y}$, which means that $\boldsymbol{x}$ and $\boldsymbol{y}$ are vertices of $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively [24].

To facilitate our presentation, denote by $\boldsymbol{X}^i$ the original feasible region $\boldsymbol{X}$ when $i = 0$, or its subset obtained after i cuts have been introduced.

Definition 2.1. *A local minimizer of $g(\cdot)$ over $\boldsymbol{X}^i$ is a vertex, $\boldsymbol{x}_{\ell m}$, such that $g(\boldsymbol{x}_{\ell m}) \leq g(\boldsymbol{x})$ for each $\boldsymbol{x} \in \mathcal{B}_\delta(\boldsymbol{x}_{\ell m}) \cap \boldsymbol{X}^i$, where $\mathcal{B}_\delta(\boldsymbol{x}_{\ell m})$ is a δ-neighborhood around $\boldsymbol{x}_{\ell m}$ in $\boldsymbol{X}^i$, and $g(\boldsymbol{x}_{\ell m})$ is the corresponding local minimum.*

Definition 2.2. *A local star minimizer of $g(\cdot)$ over $\boldsymbol{X}^i$ is a vertex, $\boldsymbol{x}_{\ell sm}$, such that $g(\boldsymbol{x}_{\ell sm}) \leq g(\boldsymbol{x})$ for each $\boldsymbol{x} \in \mathcal{N}(\boldsymbol{x}_{\ell sm})$, where $\mathcal{N}(\boldsymbol{x}_{\ell sm})$ denotes the vertices adjacent to $\boldsymbol{x}_{\ell sm}$ in $\boldsymbol{X}^i$, and $g(\boldsymbol{x}_{\ell sm})$ is the corresponding local star minimum.*

Since $f(\boldsymbol{x}, \boldsymbol{y})$ is not quasi-concave, a local star minimum is not necessarily a local minimum, and thus the development of a cut from a local star minimizer cannot take

effect as usual for those with quasi-concave objective functions. Moreover, for DBLP (1), cuts involving variables associated with both $\boldsymbol{X}^i$ and $\boldsymbol{Y}$ may destroy their special structure, and thereby fail the existing efficient algorithms to solve sub-problems. As a result, to develop a cut that involves only the $\boldsymbol{x}$-variables and yet is convergent from a local minimizer, a concept more than **Definition 2.1, 2.2** is necessary [25].

Definition 2.3. *A vertex $(\overline{\boldsymbol{x}}^i, \overline{\boldsymbol{y}})$ in DBLP is a Pseudo-Global Minimizer (PGM) if $f(\overline{\boldsymbol{x}}^i, \overline{\boldsymbol{y}}) \leq f(\boldsymbol{x}, \boldsymbol{y})$ for each $\boldsymbol{x} \in \mathcal{B}_\delta(\overline{\boldsymbol{x}}^i) \cap \boldsymbol{X}^i$ and for each $\boldsymbol{y} \in \boldsymbol{Y}$.*

For DBLP (1), a vertex is adjacent to $(\overline{\boldsymbol{x}}^i, \overline{\boldsymbol{y}})$ if and only if it is either of the form $(\boldsymbol{x}^k, \overline{\boldsymbol{y}})$ or $(\overline{\boldsymbol{x}}^i, \boldsymbol{y}^k)$ where $\boldsymbol{x}^k \in \mathcal{N}_{\boldsymbol{X}^i}(\overline{\boldsymbol{x}}^i)$ and $\boldsymbol{y}^k \in \mathcal{N}_{\boldsymbol{Y}}(\overline{\boldsymbol{y}})$. For a PGM, further improvement may be achieved by an idea analogous to that suggested by **Definition 2.2**, i.e., we can examine those vertices adjacent to $\overline{\boldsymbol{x}}^i$ for a better solution. A so derived PGM can have the advantages from both a local minimum and a local star minimum. **Algorithm 1**, originated from [24] to identify a PGM, $(\overline{\boldsymbol{x}}^i, \overline{\boldsymbol{y}})$, is currently acting as a building block in the local optimization phase of a cutting plane method.

Algorithm 1: Augmented Mountain Climbing Method

Input: $\boldsymbol{Q}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{X}^i, \boldsymbol{Y}, \widetilde{\boldsymbol{y}} \in \boldsymbol{Y}$.
Output: $(\overline{\boldsymbol{x}}^i, \overline{\boldsymbol{y}})$.
1 **repeat**
2 $\quad \widetilde{\boldsymbol{x}} = \arg\min_{\boldsymbol{x} \in \boldsymbol{X}^i} f(\boldsymbol{x}, \widetilde{\boldsymbol{y}})$; $\quad \widetilde{\boldsymbol{y}} = \arg\min_{\boldsymbol{y} \in \boldsymbol{Y}} f(\widetilde{\boldsymbol{x}}, \boldsymbol{y})$;
3 **until** $\widetilde{\boldsymbol{x}}$ *converges*;
4 construct $\mathcal{N}_{\boldsymbol{X}^i}(\widetilde{\boldsymbol{x}})$;
5 **if** $\exists \check{\boldsymbol{x}} \in \mathcal{N}_{\boldsymbol{X}^i}(\widetilde{\boldsymbol{x}})$ *such that* $f(\check{\boldsymbol{x}}, \boldsymbol{y}^*) = \min_{\boldsymbol{y} \in \boldsymbol{Y}} f(\check{\boldsymbol{x}}, \boldsymbol{y}) < f(\widetilde{\boldsymbol{x}}, \widetilde{\boldsymbol{y}})$ **then**
6 $\quad$ go to **line 2** with $\widetilde{\boldsymbol{y}} = \boldsymbol{y}^*$;
7 **terminate** with $(\overline{\boldsymbol{x}}^i, \overline{\boldsymbol{y}}) = (\widetilde{\boldsymbol{x}}, \widetilde{\boldsymbol{y}})$ as a PGM.

In **Algorithm 1**, it turns out that $\overline{\boldsymbol{x}}^i$ in a PGM, $(\overline{\boldsymbol{x}}^i, \overline{\boldsymbol{y}})$, derived in **line 7**, can be degenerate. This will result in the inevitable computational difficulty in the establishment of a valid cut in the global optimization phase since we will have more than n_1 cutting points along the edges emanating from $\overline{\boldsymbol{x}}^i$, not to mention its effectiveness and efficiency.

Degeneracy can be further classified into weak degeneracy and strong degeneracy. In a two-dimensional (2D) program, as has always been done in the literature, it is only possible to introduce weak degeneracy by bringing in some redundant constraints; see **Example 1**.

Example 1. As shown in the left sub-figure of **Figure 1**, suppose that the reduced feasible region is defined by $\mathbb{P} \cap \mathbb{Q}$ as

$$\mathbb{P} = \left\{(x_1, x_2)^t : 0 \le x_1, x_2 \le 1\right\},$$
$$\mathbb{Q} = \left\{(x_1, x_2)^t : 4x_1 + x_2 \ge 1,\ \frac{4}{3}x_1 + x_2 \le 1\right\},$$

where $\mathbb{P}$ and $\mathbb{Q}$ can be regarded as the constraints to define the original feasible region and introduced cuts, respectively.

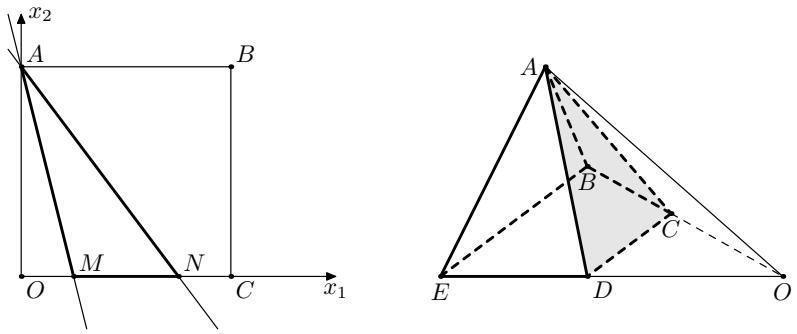


Figure 1: Weak Degeneracy versus Strong Degeneracy.

The feasible region in **Example 1** is bounded by three bold lines. Apparently, the degeneracy at vertex $\boldsymbol{A}$ can be resolved by removing two redundant linear constraints, $x_1 \ge 0$ and $x_2 \le 1$.

Intuitively, weak degeneracy occurs simply because of redundant constraints, and can be avoided by carrying out some pre-processing procedure [26], [27]. For any 2D program, barring the extreme case where the feasible region consists of a single point, only weak degeneracy exists, i.e., more than two linear constraints intersect at a single point.

In **Figure 1**, a three-dimensional (3D) instance in the right sub-figure illustrates strong degeneracy, in which $\boldsymbol{ACD}$ acts as an introduced cut to remove a portion of the feasible region $\boldsymbol{OACD}$. As a result, there are four edges emanating from $\boldsymbol{A}$ in the reduced feasible region. Having derived their respective maximal step-sizes, we can hardly expect four cutting points to be coplanar, thus leading to the computational difficulty in the generation of a valid cut from $\boldsymbol{A}$ in $\mathbb{R}^3$. Note that the removal of any constraint cannot take effect in the resolution of strong degeneracy because it changes the feasible region.

3 Conservative Cuts

3.1 Basic Knowledge

According to our experience, strong degeneracy is frequently confronted in DBLP (1) with multi-global minima. With one of the global minima derived as the current best objective value, a cut can at most reach another global minimum and may induce degeneracy therein. Consequently, it is essential to develop an efficient and effective approach for establishing a valid cut from a degenerate vertex. A promising technique specially designed for this purpose is conservative cuts.

In what follows, we will omit the superscript i in $\overline{\boldsymbol{x}}^i$ and $\boldsymbol{X}^i$ given a clear context. That is, in the i^{th} iteration, we simply take $\overline{\boldsymbol{x}}$ as $\overline{\boldsymbol{x}}^i$ in a PGM, $(\overline{\boldsymbol{x}}^i, \overline{\boldsymbol{y}})$, provided by **Algorithm 1**, and $\boldsymbol{X}$ as $\boldsymbol{X}^i$, the reduced feasible region. Besides, the procedure in search of a conservative hyperplane is formulated in $\mathbb{R}^n$ rather than $\mathbb{R}^{n_1}$ as in (1).

Geometrically, for a polytope defined by $\boldsymbol{X}$ in $\mathbb{R}^n$, a degenerate vertex $\overline{\boldsymbol{x}}$ has more than n incident edges. During the development of a cut, say, a polar cut, given the appropriate step-sizes along their positive or negative extensions, the probability that all cutting points are coplanar is fairly low. We thus have more than n points in $\mathbb{R}^n$ to establish a cut that should not exclude any potential optimal solution [22].

Denote by $\Delta = \mathcal{N}_{\boldsymbol{X}_0}(\overline{\boldsymbol{x}}) = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n, \boldsymbol{x}_{n+1}, \boldsymbol{x}_{n+2}, \ldots, \boldsymbol{x}_{n+\sigma}\}$ the set of all vertices adjacent to $\overline{\boldsymbol{x}}$, by σ ($\sigma \geq 1$) the degree of degeneracy, by Ω ($\Omega \subset \Delta$) the set containing n vertices selected from Δ to establish a hyperplane Π, and by $\overline{\Omega}$ the set of adjacent vertices that lie to the same side of Π as $\overline{\boldsymbol{x}}$ does.

Definition 3.1. *In $\mathbb{R}^n$, given a degenerate vertex, $\overline{\boldsymbol{x}}$, of a polytope, a conservative hyperplane, Π^*, is defined as the hyperplane generated by n vertices neighboring to $\overline{\boldsymbol{x}}$ such that $\overline{\Omega} = \emptyset$.*

Definition 3.2. *In $\mathbb{R}^n$, at a degenerate vertex, $\overline{\boldsymbol{x}}$, of a polytope, a conservative cut used to cut off $\overline{\boldsymbol{x}}$ is the inequality generated by a conservative hyperplane.*

3.2 Pascal's Triangle

Random, by utilizing implicit enumeration, acts as a promising procedure in search of a conservative cut. Although **Random** selects a candidate randomly in exchange for one of the vertices establishing the current hyperplane, it remains inevitable to keep a record on those already visited combinations. Otherwise, we may sacrifice the computing time for visiting the same combination and even step into infinite loops. In MATLAB, the function $nchoosek(\boldsymbol{V}, K)$ suits well for this purpose. It returns a matrix containing all possible combinations of the elements of vector $\boldsymbol{V}$ taken K at a time.

Suppose we confine an instance to the dimension of 125, which is sufficiently high for programs in imprecise decision analysis. To figure out a conservative cut, a PC equipped with 8G memory will throw an error such as "Requested $234531275x5$ (8.7GB) array exceeds maximum array size preference (8.0GB)" when **Random** carries out $nchoosek(\boldsymbol{V}, 5)$ where $\boldsymbol{V} = [1, 2, \ldots, 125]^t$. Though we can partially resolve this issue by configuring "MATLAB array size limit" in "Workspace", the generation process will run extremely slow and cause MATLAB to become unresponsive. This is reasonable because $nchoosek(\boldsymbol{V}, 5)$ intends to enumerate a total of 234531275 combinations. They will then serve as the indices to locate corresponding adjacent vertices in Δ to establish $\boldsymbol{\Omega}$. The number of combinations of $nchoosek(\boldsymbol{V}, K)$ increases particularly fast as K rises. We can hardly expect each PC is equipped with 8G memory so that the aforementioned memory problem can severely hinder **Random**'s application.

To circumvent this issue, we introduce a critical technique that is able to take effect throughout the implementation of our memory-efficient approach. To avoid building the full combination array in memory like what $nchoosek(\boldsymbol{V}, K)$ does, we take advantage of the function $nchoosek\_enum(n, k, i)$ with an enumerating selection of the i^{th} combination.

Consider an example for all six combinations of C_4^2, i.e., $(1, 1, 2)$, $(2, 1, 3)$, $(3, 1, 4)$, $(4, 2, 3)$, $(5, 2, 4)$, and $(6, 3, 4)$, with the first entry as the corresponding serial number. The 3^{rd} combination of C_4^2 derived by $nchoosek\_enum(4, 2, 3)$ is $(1, 4)$, the 5^{th} combination by $nchoosek\_enum(4, 2, 5)$ is $(2, 4)$, etc. By referring to the Pascal's Triangle table, we need to develop its inverse, $inverse\_nchoosek\_enum(n, k, \boldsymbol{c})$, such that, given an appropriate combination $\boldsymbol{c}$, we can derive its serial number. By "appropriate", we mean herein that all elements in $\boldsymbol{c}$ are organized in accordance with their natural order. Still with the previous example, the output for $inverse\_nchoosek\_enum(4, 2, [3, 4])$ is 6. The rationale can be found in **Example 1** with detailed explanations.

Example 1. Here comes an example with a detailed workflow for illustration. We list only the necessary rows of $Pascal(9)$ provided by MATLAB for our purpose; see **Table 1**.

We take the following procedures to figure out the serial number corresponding to the combination, $\boldsymbol{c} = [3, 5, 6, 8, 9]$, among all combinations of choosing 5 elements out of $\boldsymbol{V} = [1, 2, \ldots, 9]^t$. For the first element 3, two preceding elements are 1 and 2. First, for 1, there are eight elements, i.e., $2, 3, \ldots, 9$, left for the rest four positions, and therefore it possesses $C_8^4 = 70$ combinations in total. Once the four elements are selected, their order is fixed in consistence with the natural order. Next, for 2, there are seven elements, i.e., $3, 4, \ldots, 9$, left for the rest four positions, and therefore it

Table 1: Pascal(9)

1	1	1	1	1	1	1	1	1
1	2	③	4	5	6	7	8	9
1	3	6	⑩	15	21	28	36	45
1	4	10	20	㉟	56	84	120	165
1	5	15	35	⑦⓪	126	210	330	495

possesses $C_7^4 = 35$ combinations in total. When it comes to 5, we need to count the number of combinations starting with $[3, 4, \ldots]$. By an analogous logic, it is $C_5^3 = 10$ in total. Finally, we come to 6, which has C_3^2 combinations with $[8, 9]$ as the last one. As a result, the output of $inverse\_nchoosek\_enum(9, 5, [3, 5, 6, 8, 9])$ is $C_8^4 + C_7^4 + C_5^3 + C_3^2 = 70 + 35 + 10 + 3 = 118$, as indicated by the circled numbers.

Note that it is unnecessary to handle $[5, 6]$ and $[8, 9]$ in $[3, 5, 6, 8, 9]$ because they are two pairs of consecutive natural numbers.

Example 1 illustrates that by choosing appropriate numbers from a static Pascal's Triangle table, we can readily derive the serial number of an appropriate combination. Generally speaking, it is unnecessary to save a complete Pascal's Triangle table in memory because on the one hand, we could rarely expect such an ugly degenerate problem with $\sigma > 5$. On the other hand, since a Pascal's Triangle table is symmetric, the storage is so cheap that it can serve as a very efficient and effective means of deriving the serial number of some combination of choosing K out of N elements.

3.3 Implementation

With $inverse\_nchoosek\_enum()$, denote by $\#$ the serial number of an appropriate combination of choosing K out of N elements, by $\boldsymbol{v}$ the normal vector of Π^*, by $\boldsymbol{A}$ and $\boldsymbol{b}$ the parameters in the constraint set, $\boldsymbol{Ax} \leq \boldsymbol{b}$, by Q a structure to save the serial numbers of those Ωs whose candidates for exchange have already been exhausted; and by $\overline{Q}$ a structure array to save those Ωs with the same $|\overline{\Omega}|$, where $|\cdot|$ is the cardinality of a set. The so constructed structure array $\overline{\Omega}$ indexed by $|\overline{\Omega}|$ is used to accelerate the search among candidates. Moreover, there exists another structure, $abnormal$, which saves those Ωs constituting (close to) singular matrices. Such Ωs are considered inappropriate to establish hyperplanes.

Algorithm 2 realizes the memory-efficient randomized algorithm with details. Notice that it is the essential property of non-uniqueness of a conservative hyper-

plane that enables **Random** to surpass its other counterparts in [21], [22] and be competitive with **Algorithm 2** in terms of different σs.

Algorithm 2: Conservative Hyperplane (Improved Randomized)

Input: $\Delta, \overline{\boldsymbol{x}}$
Output: $\boldsymbol{v}, \Omega$

1 set $\boldsymbol{v} = \emptyset$;
2 **while** $\boldsymbol{v} = \emptyset$ **do**
3 randomly extract # from $\overline{Q}_\ell$ where $\ell = \arg\min_{\ell'}\{\ell' | \overline{Q}_{\ell'} \neq \emptyset\}$;
4 **if** $\ell = -1$ **then**
5 randomly collect n vertices from Δ into Ω to set up a Π;
6 **if** $|\overline{\Omega}| = 0$ **then return** $\boldsymbol{v}$ of Π^* and Ω;
7 **if** $\# \notin \overline{Q} \cup Q \cup abnormal$ **then**
8 insert # of Ω, and all $\boldsymbol{x}_k \in \overline{\Omega}$ into $\overline{Q}_{|\overline{\Omega}|}$;
9 **else**
10 recover Ω corresponding to the extracted #;
11 randomly select $\boldsymbol{x}_j \in \overline{\Omega}$, and set $\overline{\Omega} = \overline{\Omega} \backslash \{\boldsymbol{x}_j\}$;
12 **if** $|\overline{\Omega}| = 0$ **then** move # of Ω from $\overline{Q}_\ell$ to Q;
13 **foreach** $\boldsymbol{x}_k \in \Omega$ **do**
14 exchange $\boldsymbol{x}_j$ with $\boldsymbol{x}_k$ to set up Ω' and Π';
15 **if** $|\overline{\Omega'}| = 0$ **then return** $\boldsymbol{v}$ of Π^* and Ω';
16 **if** $\# \notin \overline{Q} \cup Q \cup abnormal$ **then**
17 insert # of Ω', and all $\boldsymbol{x}_k \in \overline{\Omega'}$ into $\overline{Q}_{|\overline{\Omega'}|}$;

In **Algorithm 2**, in order to validate the availability of one Ω, it is necessary to check Q, $\overline{Q}$, and $abnormal$ first; see **line 7, 16**. We can save in Q, $\overline{Q}$ and $abnormal$ only the serial number associated with each Ω rather than specific elements. This can greatly alleviate the usage of memory since it is unnecessary to visit all combinations in search of a conservative hyperplane. Otherwise, it would become quite unclear on the memory size we need to allocate in advance, or an arbitrary allocation may simply lead to "out of memory"; see **line 3**, **7**, **12** and **16**. Additionally, the utilization of actual combinations will compel the program to compare a sorted candidate with each entry, e.g., $ismember()$ in MATLAB. The comparison of vectors may also slow down the computations.

Several other aspects need to be further clarified. Firstly, a normal vector, $\boldsymbol{v}$, exists

provided that Ω is well-defined to establish a hyperplane, i.e., Ω is not in *abnormal*, otherwise $\boldsymbol{v}$ is kept empty. As a result, the only stopping criterion is either $|\overline{\Omega}| = 0$ or $|\overline{\Omega'}| = 0$ regardless of $\boldsymbol{v}$; see **line 6, 15**. One exception is that, in **line 12**, $|\overline{\Omega}| = 0$ indicates that all candidates for exchange regarding some Ω have been exhausted so that we should move it from $\overline{Q}$ to Q. Secondly, although it is suggested that Ω or $\overline{\Omega}$ in **Algorithm 2** contain qualified adjacent vertices, we actually take advantage of their row indices in Δ to indicate their positions and then extract them. By doing so, we intend to relieve the storage load of memory. Thirdly, in **line 4**, $\ell = -1$ means that no Ω exists for search, which may arise provided that all available Ωs have been exhausted or when initializing the entire algorithm. Finally, in **line 7, 16**, each operation inserts into $\overline{Q}$ one serial number and appends to it a set containing all qualified vertices. The algorithm extracts these candidates one by one for exchange until the set becomes empty (**line 11**, and **line 13** through **17**).

4 Test Instances

Algorithm 3 is used to generate a test instance with $n + \sigma$ vertices adjacent to a σ-degenerate vertex $\overline{\boldsymbol{x}}$ in $\mathbb{R}^n$.

Algorithm 3: Test Instances

Input: n, σ
Output: Δ

1 generate a polytope with $n+1$ points $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n, \boldsymbol{x}_{n+1}\}$ in $\mathbb{R}^n$;
2 fix one point, say, $\boldsymbol{x}_{n+1}$ as $\overline{\boldsymbol{x}}$, and collect the remaining into Δ;
3 **while** $|\Delta| \leq n + \sigma$ **do**
4 select $\boldsymbol{x}_i \in \Delta$;
5 **foreach** $\boldsymbol{x}_j$, $j \neq i$, *adjacent to* $\boldsymbol{x}_i$ **do**
6 generate a new point $\boldsymbol{x}_{|\Delta|+j}$ and set $\Delta = \Delta \cup \{\boldsymbol{x}_{|\Delta|+j}\}$

$$\boldsymbol{x}_{|\Delta|+j} = \sum_{k \neq j} \alpha_{jk} \boldsymbol{x}_k,\ \alpha_{jk} \in (0,1),\ k \neq j,\ \sum_{k \neq j} \alpha_{jk} = 1;$$

7 set $\Delta = \Delta \backslash \{\boldsymbol{x}_i\}$, and re-order so that $\Delta = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{|\Delta|}\}$;
8 randomly select $n + \sigma$ points from Δ so that $\Delta = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{n+\sigma}\}$;
9 **foreach** $\boldsymbol{x}_i \in \Delta$ **do** randomly extend or shorten $\overline{\boldsymbol{x}}\boldsymbol{x}_i$;

In **Algorithm 3**, between **line 3** and **7**, the while loop will not stop until the number of elements in Δ exceeds $n + \sigma$, the required number of adjacent vertices.

This provides us with great flexibility in the generation of a degenerate instance, despite the cost of some excessive computational effort. In **line 6**, the operation is in fact the convex combination of $\boldsymbol{x}_i$ with all the remaining $\boldsymbol{x}_k$s except $\boldsymbol{x}_j$ for each $j \neq i$. Therefore, we generate $n-1$ new points in total. Together with $\overline{\boldsymbol{x}}$, a hyperplane can be generated to remove $\boldsymbol{x}_i$, and the number of vertices adjacent to $\overline{\boldsymbol{x}}$ becomes $|\Delta| + n - 2$. In **line 9**, by randomly extending or shortening $\overline{\boldsymbol{x}}\boldsymbol{x}_i$, there may still exist more than n points on some hyperplane. However, a test instance is considered inappropriate only if all adjacent vertices lie on the same hyperplane. By carrying out the critical operation in **line 9**, this could seldom happen.

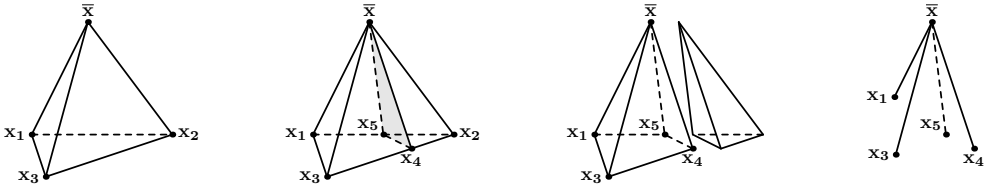


Figure 2: Illustration of **Algorithm 3**.

Algorithm 3 can be illustrated by a 3D example in **Figure 2**, where $n = 3$ and $\sigma = 1$. By **Figure 2**, each construction of a cut to remove some $\boldsymbol{x}_i$ will generate one more point adjacent to $\overline{\boldsymbol{x}}$. As a result, we are able to generate any number of points adjacent to $\overline{\boldsymbol{x}}$, regardless of how high the degree of degeneracy is.

5 Numerical Results

In order to evaluate the performance of two randomized algorithms, especially **Algorithm 2**, we take advantage of **Algorithm 3** to generate the required test instances and carry out the experiments on a PC equipped with Intel(R) Core(TM) i5-6267U CPU @ 2.90GHz and 4G memory. We deliberately impose a limit of 60 seconds over the total computing time for each instance, which appears reasonable for an interactive decision analysis software package. For each combination of dimension (n) and degree of degeneracy (σ), we generate 24 test instances and average their corresponding results with respect to total computing time, the number of performing $nchoosek\_enum(n, k, i)$, etc. In the following, we intend to evaluate the performance of two algorithms for test instances with low σ and n, high σ and low n, and low σ and high n, respectively. The exception is for test instances with high σ and n due to the memory issue raised by $nchoosek(n, k)$. With the current environment, we can only try to increase either n or σ, but not both, as will be demonstrated by the experiments.

5.1 Low Degree of Degeneracy and Dimension

Figure 3 illustrates the performance for test instances with low σ ($\sigma = 1, 2, 3$) and n ($n = 30, 35, \ldots, 100$). Three sub-figures on the first row of **Figure 3** illustrate the performance of two algorithms regarding $\sigma = 1, 2, 3$, respectively. It can be observed that **Random** runs faster than **Algorithm 2** most of the time. Nevertheless, the gaps in terms of computing time are rather small, say, within only around 1.5 seconds. Two sub-figures on the second row of **Figure 3** illustrate their individual performance with respect to $\sigma = 1, 2, 3$. For both algorithms, the computing time increases exponentially as n rises given $\sigma = 2, 3$, whereas the performance of **Random** keeps relatively steady when $\sigma = 1$, as apposed to that of **Algorithm 2**.

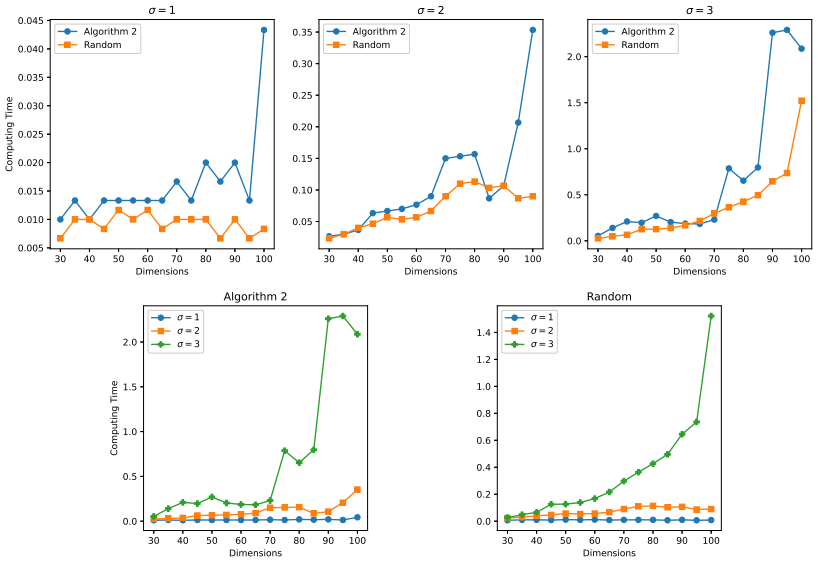


Figure 3: Performance of **Algorithm 2** and **Random** (Low σ and n).

The fact that **Random** runs faster than **Algorithm 2** in the current setting appears reasonable due to the sufficient memory. **Random** can take advantage of $nchoosek(\boldsymbol{V}, K)$, which generates all possible combinations and save them in memory. However, as n or σ increases, the memory problem will emerge.

5.2 High Degree of Degeneracy and Low Dimension

Figure 4 illustrates their results for instances with high σ ($\sigma = 4, 5$) when n goes from 30 to 100. Two sub-figures on the first row of **Figure 4** demonstrate their performance. Provided $\sigma = 4$, **Random** can run up to around $n = 100$ before its computing time exceeds the pre-specified time limit, or the test instance incurs "out of memory", whereas provided $\sigma = 5$, similar situations take place around $n = 75$. To some extent, this demonstrates that each increase in σ can dramatically impact the computational performance of **Random**. Meanwhile, the computing time of **Algorithm 2** is acceptable. By two sub-figures on the first row of **Figure 4**, the performance of **Algorithm 2** dominates that of **Random** for $\sigma = 4, 5$. Two sub-figures on the second row of **Figure 4** illustrate the number of performing nchoosek_enum(n,k,i) and the number of solving linear equations, respectively, regarding **Algorithm 2** when $\sigma = 4, 5$. In the bottom-left sub-figure, the numbers of performing nchoosek_enum(n,k,i) are competitive when $n \leq 70$. However, when $n \geq 75$, a clear increase in the number for $\sigma = 5$ can be observed, as compared with that for $\sigma = 4$. Besides, a higher σ, in general, corresponds to a larger number of solving linear equations; see the bottom-right sub-figure of **Figure 4**.

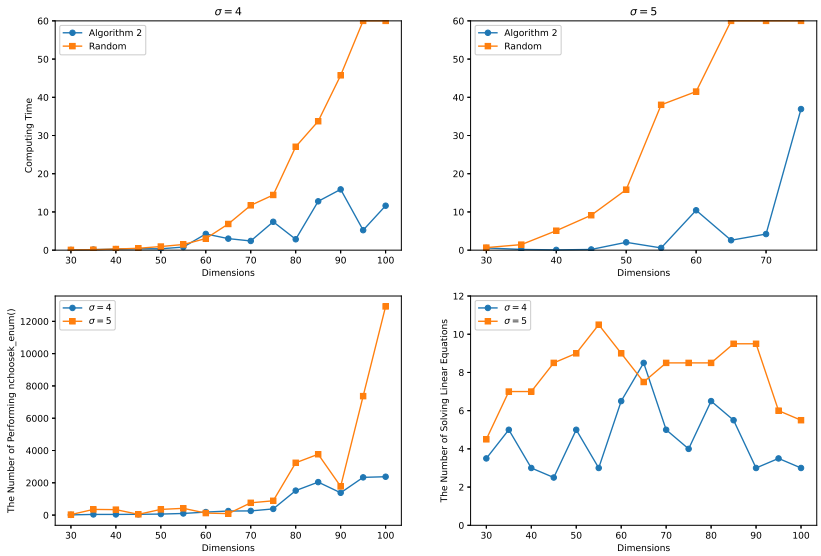


Figure 4: Performance of **Algorithm 2** and **Random** (High σ, Low n).

Most computing time of **Algorithm 2** is spent in $nchoosek\_enum(n,k,i)$ to generate serial numbers in exchange for memory. By contrast, solving linear equations only accounts for a very small portion of total computing time. As σ rises, the computing time rises sharply. Nonetheless, the impact from the rise in n on the total computing time seems much less when σ is not that high. Additionally, the higher the σ, the more number of $nchoosek\_enum(n,k,i)$ **Algorithm 2** performs. The trend becomes much clearer as n rises. This is reasonable since the number of candidates in search of a conservative hyperplane increases exponentially. By contrast, $inverse\_nchoosek\_enum(n,k,\boldsymbol{c})$ costs almost nothing since the Pascal's Triangle table is static.

5.3 Low Degree of Degeneracy and High Dimension

Figure 5 illustrates their performance for instances with low σ ($\sigma = 1,2,3$) and high n ($n = 100,105,\ldots,250$). For the purpose of comparison, it is impossible to perform **Random** with respect to these dimensions when $\sigma = 4,5$ due to the pre-specified time limit or "out of memory" issue; see also the previous illustration when $n \leq 100$. It can be observed that for $\sigma = 1$, their performance overlaps most of the time. Nevertheless, for $\sigma = 2,3$, **Algorithm 2** outperforms **Random** across all most dimensions, which demonstrates its ability and qualification in handling problems with higher dimensions. Note also that the gaps in computing time for $\sigma = 1,2$ are relatively small, showing two algorithms are very competitive.

6 Conclusions

This paper investigates an improved randomized algorithm, i.e., **Algorithm 2**, in addressing the memory challenge associated with the previously developed searching process, **Random**, for a conservative cut. By leveraging the inherent structure of Pascal's Triangle, the memory-efficient **Algorithm 2** well illustrates the algorithmic space-time continuum in data structure.

Computational experiments demonstrate that **Algorithm 2** is particularly beneficial to programs characterized by high n or σ, whereas **Random** is more suitable for programs with low σ. However, once the memory problem appears, **Random** will become less preferred so that **Algorithm 2** should come into play. By refining **Random**, we anticipate its application to a cutting plane method in imprecise decision analysis where DBLP plays a pivotal role.

What should be noted is that the Matlab environment utilized herein is an interpreter compiler, which slows down the test significantly unless a special-purpose command line was used in the compilation. To accelerate, an even better approach

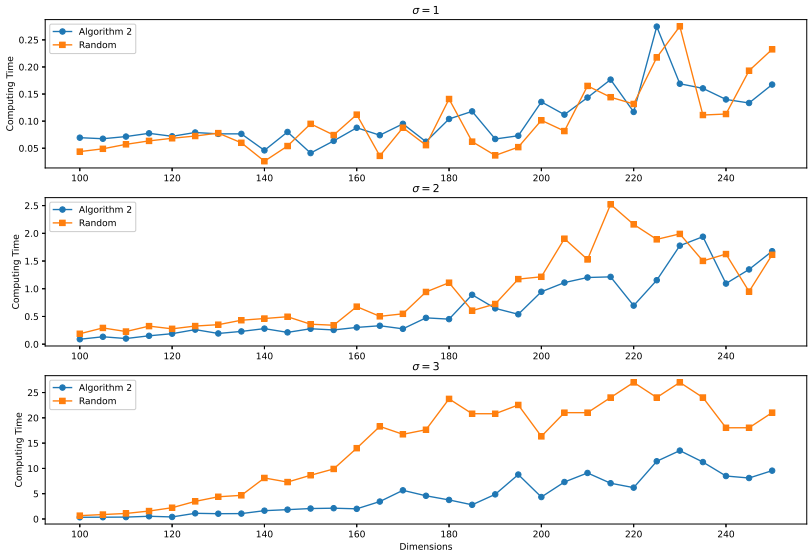


Figure 5: Performance of **Algorithm 2** and **Random** (Low σ and High n).

is to develop the entire cutting plane method using C/C++.

Author Contributions: Conceptualization, Xiaosong DING and Xi CHEN; Methodology, Xiaosong Ding and Chao LIU; Formal Analysis, Jun MA and Xi CHEN; Software, Xiaosong Ding; Writing – original draft, Xiaosong Ding; Writing – review & editing, Xi CHEN and Qing SUN. All authors have read and agreed to the manuscript's published version.

Funding: This research was partially funded by the Fundamental Research Funds for the Central Universities (2022TD001, 2023JJ002) and Young Faculty Research Fund of BFSU (2019-ZYQNJS-002).

Data Availability: This work did not generate new data.

Acknowledgments: We would like to thank two anonymous reviewers for their insightful remarks.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix

In $\mathbb{R}^n$, suppose that in a polytope, a degenerate vertex, $\overline{\boldsymbol{x}}$, has been located with $n+m$ adjacent vertices, $\Delta = \{B_1, B_2, \ldots, B_n, B_{n+1}, B_{n+2}, \ldots, B_{n+m}\}$. By randomly selecting n points, we can obtain $\mathcal{R}_\Theta$ to establish a hyperplane Ω and $\mathcal{S}_{\hat{\Theta}}$ with $|\mathcal{S}_{\hat{\Theta}}| = k$ where $|\cdot|$ is the number of elements in a set. For each possible k, we set up a queue, $\overline{Q}_k$, to save all $\mathcal{R}_\Theta$s with $|\mathcal{S}_{\hat{\Theta}}| = k$ that have been found but not yet utilized as the starting set of points for the branching process. Similarly, we establish another queue, Q_k, to save all $\mathcal{R}_\Theta$s with $|\mathcal{S}_{\hat{\Theta}}| = k$ that have already been utilized for the branching process.

Algorithm 4: General Algorithm

Input: $\Delta, \overline{\boldsymbol{x}}$
Output: $\mathcal{R}_\Theta$

randomly choose n points from Δ as $\mathcal{R}_\Theta$ and establish Ω with $\mathcal{R}_\Theta$;
update with $\ell = |\mathcal{S}_{\hat{\Theta}}| = k$, and save $\mathcal{R}_\Theta$ in $\overline{Q}_\ell$;
while $\ell \neq 0$ **do**
    extract one $\mathcal{R}_\Theta$ from $\overline{Q}_\ell$;
    update with $\overline{Q}_\ell = \overline{Q}_\ell / \{\mathcal{R}_\Theta\}$, $Q_\ell = Q_\ell \cup \{\mathcal{R}_\Theta\}$;
    ℓ = **Algorithm 5**($\mathcal{R}_\Theta$);
return $\mathcal{R}_\Theta$.

Algorithm 5: Branching Algorithm (Randomized)

Input: $\mathcal{R}_\Theta$
Output: ℓ

randomly choose one R_i in $\mathcal{R}_\Theta$ and one S_j in $\mathcal{S}_{\hat{\Theta}}$;
set $\mathcal{R}_{\Theta'} = \{\mathcal{R}_\Theta / \{R_i\}\} \cup \{S_j\}$;
if $\mathcal{R}_{\Theta'} \notin Q_{k'} \cup \overline{Q}_{k'}$, $\left(|\mathcal{S}_{\hat{\Theta}'}| = k'\right)$ **then** save $\mathcal{R}_{\Theta'}$ in $\overline{Q}_{k'}$, and set $\ell = k'$;
else $\ell = \arg\min_m \{m | \overline{Q}_m \neq \emptyset\}$;
return ℓ.

References

[1] H. Konno, "Bilinear programming: Part II. Application of bilinear programming," Tech. Rept. No. 71-10, Department of Operations Research, Stanford University, Stanford, Calif., 1971 (⇒ 21).

[2] S. Rebennack, A. Nahapetyan, and P. Pardalos, "Bilinear modeling solution approach for fixed charge network flow problems," *Optimization Letters*, vol. 3, no. 3, pp. 347–355, 2009 (⇒ 21).

[3] A. Nahapetyan, "Bilinear programming: Applications in the supply chain management," in *Encyclopedia of Optimization*, Springer, 2009, pp. 282–288 (⇒ 21).

[4] D. Wicaksono and I. Karimi, "Piecewise MILP under- and overestimators for global optimization of bilinear programs," *AIChE Journal*, vol. 54, no. 4, pp. 991–1008, 2008 (⇒ 21).

[5] A. Caprara and M. Monaci, "Bidimensional packing by bilinear programming," *Mathematical programming*, vol. 118, no. 1, pp. 75–108, 2009 (⇒ 21).

[6] M. Petrik and S. Zilberstein, "Robust approximate bilinear programming for value function approximation," *The Journal of Machine Learning Research*, vol. 12, pp. 3027–3063, 2011 (⇒ 21).

[7] M. Valdebenito, C. Pérez, H. Jensen, and M. Beer, "Approximate fuzzy analysis of linear structural systems applying intervening variables," *Computers & Structures*, vol. 162, no. 1, pp. 116–129, 2016 (⇒ 21).

[8] A. Nahapetyan and P. Pardalos, "A bilinear relaxation based algorithm for concave piecewise linear network flow problems," *Journal of Industrial and Management Optimization*, vol. 3, no. 1, p. 71, 2007 (⇒ 21).

[9] A. Nahapetyan and P. Pardalos, "A bilinear reduction based algorithm for solving capacitated multi-item dynamic pricing problems," *Computers & Operations Research*, vol. 35, no. 5, pp. 1601–1612, 2008 (⇒ 21).

[10] M. Danielson, "Generalized evaluation in decision analysis," *European Journal of Operational Research*, vol. 162, no. 2, pp. 442–449, 2005 (⇒ 21).

[11] M. Danielson and L. Ekenberg, "A framework for analysing decisions under risk," *European Journal of Operational Research*, vol. 104, no. 3, pp. 474–484, 1998 (⇒ 21).

[12] M. Danielson and L. Ekenberg, "Computing upper and lower bounds in interval decision trees," *European Journal of Operational Research*, vol. 181, no. 2, pp. 808–816, 2007 (⇒ 21).

[13] M. Danielson and L. Ekenberg, *Real-Life Decision-Making*, 1st ed. CRC Press, 2023, ISBN: 9781003406709 (⇒ 21).

[14] A. Sage and C. White, "Ariadne: A knowledge-based interactive system for planning and decision support," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 14, no. 1, pp. 35–47, 1984 (⇒ 21).

[15] A. Salo and R. Hämäläinen, "Preference programming through approximate ratio comparisons," *European Journal of Operational Research*, vol. 82, no. 3, pp. 458–475, 1995 (⇒ 21).

[16] H. Tuy, "Concave programming under linear constraints," *Soviet Maththematics*, vol. 5, pp. 1437–1440, 1964 (⇒ 21).

[17] H. Sherali and C. Shetty, "A finitely convergent algorithm for bilinear programming problems using polar cuts and disjunctive face cuts," *Mathematical Programming*, vol. 19, no. 1, pp. 14–31, 1980 (⇒ 21).

[18] M. Porembski, "How to extend the concept of convexity cuts to derive deeper cutting planes," *Journal of Global Optimization*, vol. 15, no. 4, pp. 371–404, 1999 (⇒ 21).

[19] S. Alarie, C. Audet, B. Jaumard, and G. Savard, "Concavity cuts for disjoint bilinear programming," *Mathematical Programming*, vol. 90, no. 2, pp. 373–398, 2001 (⇒ 21).

[20] M. Porembski, "On the hierarchy of γ-valid cuts in global optimization," *Naval Research Logistics*, vol. 55, no. 1, pp. 1–15, 2008 (⇒ 21).

[21] X. Chen, J. Zhang, X. Ding, T. Yang, and J. Qian, "Location of a conservative hyperplane for cutting plane methods in disjoint bilinear programming," *Optimization Letters*, vol. 13, no. 7, pp. 1677–1692, 2019 (⇒ 21, 28).

[22] J. Zhang, X. Chen, and X. Ding, "Degeneracy removal in cutting plane methods for disjoint bilinear programming," *Optimization Letters*, vol. 11, no. 3, pp. 483–495, 2017 (⇒ 21, 25, 28).

[23] C. Audet, P. Hansen, B. Jaumard, and G. Savard, "A symmetrical linear maxmin approach to disjoint bilinear programming," *Mathematical Programming*, vol. 85, no. 3, pp. 573–592, 1999 (⇒ 21).

[24] H. Konno, "A cutting plane algorithm for solving bilinear programs," *Mathematical Programming*, vol. 11, no. 1, pp. 14–27, 1976 (⇒ 22, 23).

[25] H. Vaish and C. Shetty, "A cutting plane algorithm for the bilinear programming problem," *Naval Research Logistics*, vol. 24, no. 1, pp. 83–94, 1977 (⇒ 23).

[26] T. Gal, "A method for determining redundant constraints," in *Redundancy in Mathematical Programming*, Berlin, Heidelberg: Springer Berlin Heidelberg, 1983, pp. 36–52, ISBN: 978-3-642-45535-3 (⇒ 24).

[27] T. Gal, "Weakly redundant constraints and their impact on postoptimal analyses in LP," *European Journal of Operational Research*, vol. 60, no. 3, pp. 315–326, 1992 (⇒ 24).

Received: 26.01.2024; *Revised: 13.05.2024*; *Accepted: 10.06.2024*

Acta Univ. Sapientiae, Informatica, **16, 1** (2024) 39–43

DOI: 10.47745/ausi-2024-0003

Density of α-powers in repetitions of a primitive word

József BUKOR
J. Selye University
Department of Informatics
Komárno, Slovakia
bukorj@ujs.sk
0000-0003-4134-2181

Abstract. We consider words over fixed alphabet. A word u is said to be primitive if it cannot be expressed as a nontrivial power of another word. For a positive real $\alpha > 1$ the α-power of a word v is defined by $v^\alpha = v^{\lfloor \alpha \rfloor} p$ where p is a prefix of v of length $\lfloor (\alpha - \lfloor \alpha \rfloor)|v| \rfloor$. We prove that for any primitive word u and for any $\alpha > 1$ the number of distinct α-powers in u^k is asymptotically equal to $\frac{1}{\alpha}|u^k|$ for $k \to \infty$.

Key words and phrases: repetitions, combinatorics on words

1 Introduction

We recall the basic definitions and notations; see, e.g., [1] Let Σ denote a finite alphabet. The elements of Σ are called letters. We denote by Σ^* the set of finite words over Σ and by ε the empty word. The set Σ^* is a monoid with concatenation. A factor (subword) of a word is its fragment consisting of a number of consecutive letters. More precisely, a word w is a factor of a word u if $u = xwy$ for some words x, y. We say that w is a prefix (resp. suffix) of u if $x = \varepsilon$ (resp. $y = \varepsilon$). The length of a word w, that is, the number of letters of w, is denoted by $|w|$.

For a word w and a non-negative integer n the nth power of w is defined inductively as $w^n = ww^{n-1}$, where $w^0 = \varepsilon$. Words of the form $w^2 = ww$ are called squares. The maximum number of different square factors were studied by many authors, see [2]–[4].

A word u is said to be primitive if it cannot be expressed as a nontrivial power of another word. Patawar and Kapoor [5] studied the density of distinct squares in a

repetition of a primitive word. They proved that for a primitive word u the density of distinct squares in a repetition u^k converges to $\frac{1}{2}$ for $k \to \infty$. The aim of this note is to investigate the asymptotic behaviour of the number of distinct α-powers in a repetition for any $\alpha > 1$.

For a positive real $\alpha > 1$ the α-power of a word v is defined by $v^\alpha = v^{\lfloor \alpha \rfloor} p$ where p is a prefix of v of length $\lfloor (\alpha - \lfloor \alpha \rfloor)|v| \rfloor$. We note that $|v^\alpha| = \lfloor \alpha |v| \rfloor$. For example, the e-power of the word abc is $abcabcab$ (e is the Euler's number).

We prove that for any primitive word u and for any $\alpha > 1$ the number of distinct α-powers in u^k is asymptotically equal to $\frac{1}{\alpha}|u^k|$ for $k \to \infty$.

2 Results

The main result of this note is the following.

Theorem 1. *Let u be a primitive word and k be a positive integer. Let $D_\alpha(u^k)$ denote the number of distinct α-powers in the word u^k. Let $\rho_\alpha(u^k)$ be the density of distinct α-powers in the word u^k. Then we have*

$$\lim_{k\to\infty} \rho_\alpha(u^k) = \lim_{k\to\infty} \frac{D_\alpha(u^k)}{|u^k|} = \frac{1}{\alpha}. \tag{1}$$

The following lemma shows that arbitrary sufficiently long subword of u^k has a unique representation.

Lemma 2. *Let u be a primitive word and let $w = a_1 \cdots a_n$ be a subword of u^k. Furthermore, let u be a subword of w. Then, there are unique words x, y, and a positive integer $i \geq 1$ satisfying $w = xu^i y$ where $|x|, |y| < |u|$.*

Proof. To contrary, let us suppose that the word

$$w = xu^i y = x'u^j y' \quad (i \neq j \text{ or } x \neq x' \text{ or } y \neq y')$$

is a counterexample for the statement of the lemma of minimal length. Then we have $x = \varepsilon$ or $x' = \varepsilon$, otherwise the word $a_2 \cdots a_n$ is shorter counterexample. So, let us suppose that $x' = \varepsilon$ and let us consider the counterexample

$$w = xu^i y = u^j y' \quad (x \neq \varepsilon).$$

In this case u appears in the "middle" of uu which yields to a contradiction, see, e.g. [4] or Lemma 2.1 in [5].
In the remaining case, when the counterexample is $w = u^i y = u^j y'$, we clearly have $i = j$ and $y = y'$, which is a contradiction, too. □

As a consequence of the previous lemma, we have the following observation. Let u be a nonprimitive word of length t and let $u^k = b_1 \cdots b_t \cdots b_{tk}$. Then all the subwords w of u^k whose first term is b_i, $1 \le i \le t$, and whose length satisfies $|w| \ge 2|u| - 1$, are distinct. The proof follows from the fact that the conditions above imply that u is a subword of w.

Definition 1. *We call a subword w of u^k "long" if it is an α-power of a word z for which we have $\lfloor \alpha |z| \rfloor - |z| \ge |u|$ and $|z| \ge |u|$, otherwise we call it "short".*

We note that for a given nonprimitive word u and a real number $\alpha > 1$ there exists a constant n_0 such that for any $n \ge n_0$ we have $\lfloor \alpha n \rfloor - n \ge |u|$ and $n \ge |u|$. Let us denote by $f(u, \alpha)$ the minimal value of n_0 satisfying the conditions above and which is concurrently divisible by $|u|$.

We estimate the number of short and long subwords of u^k. For a nonprimitive word $u = b_1 \cdots b_t$ let $P_{\alpha,long}(u^k, i)$ (resp. $P_{\alpha,short}(u^k, i)$) denote the set of long (resp. short) prefixes of the word $b_i b_{i+1} \cdots b_t u^{k-1}$.

If we consider all short prefixes of $b_i b_{i+1} \cdots b_t u^{k-1}$ we get the upper bound

$$\left| P_{\alpha,short}(u^k, i) \right| \le f(u, \alpha). \tag{2}$$

In what follows we show that long subwords of u^k are α-powers of some word, which length is a multiple of $|u|$.

Lemma 3. *Let $u = b_1 \cdots b_t$ be a primitive word and $x = b_1 \cdots b_{i-1}$, $y = b_i \cdots b_t$. Let w be an α-power of a word $z = yxp$ ($p \in \Sigma^*$). If w is a long subword of u^k then $|z|$ is divisible by $|u|$.*

Proof. From the conditions of the lemma, we find that $yxpyx$ is a subword of u^k. As $u = xy$, using the Lemma 2, we get that $p = (yx)^m$ for some positive integer m. Then $z = (yx)^{m+1}$ and the assertion follows. □

Proof of Theorem 1. Consider the longest word w of the set $P_{\alpha,long}(u^k, i)$. It means that w is an α-power of a word z for that

$$|z| \ge f(u, \alpha), \qquad |z| \text{ is divisible by } |u| \qquad \text{and} \qquad (i-1) + \lfloor |z^\alpha| \rfloor \le k|u|.$$

By Lemma 3, we have that $|z| = l|u|$ for a positive integer l. Therefore,

$$(i-1) + \lfloor \alpha l |u| \rfloor \le k|u|. \tag{3}$$

As l is the largest positive integer satisfying (3), there hold the following inequalities

$$\frac{k}{\alpha} - 2 < l < \frac{k}{\alpha} + 1. \tag{4}$$

For the cardinality of the set $P_{\alpha,long}(u^k, i)$ we have the upper bound l and the lower bound $l - f(u,\alpha)/|u|$ which together with the inequalities (4) produce the estimations

$$\frac{k}{\alpha} - 2 - \frac{f(u,\alpha)}{|u|} < \left|P_{\alpha,long}(u^k, i)\right| < \frac{k}{\alpha} + 1. \tag{5}$$

Taking into account that

$$\sum_{i=1}^{|u|} |P_{\alpha,long}(u^k, i)| < D_\alpha(u^k) \leq \sum_{i=1}^{|u|} \left(|P_{\alpha,long}(u^k, i)| + |P_{\alpha,short}(u^k, i)|\right)$$

by (2) and (5) we get that

$$|u|\left(\frac{k}{\alpha} - 2 - \frac{f(u,\alpha)}{|u|}\right) < D_\alpha(u^k) < |u|\left(\frac{k}{\alpha} + 1 + f(u,\alpha)\right)$$

and the limit (1) follows by squeeze theorem. □

We remark that Theorem 1 is also valid for nonprimitive u, as such u is v^s for an integer s and a primitive v, and $u^k = v^{sk}$.

3 Concluding remark

Throughout of this section, we assume that u is a primitive word. We have proved that the density of distinct α-powers in the word u^k is $1/\alpha$. It is worth to mention that if we extend our study to the case $\alpha = 1$, we get that the density of distinct α-powers in the word u^k (which is a function of variable α) is discontinuous in $\alpha = 1$ under the assumption that $|u| \geq 2$. The proof of the fact that the density of distinct subwords in the word u^k is $|u|$ is left to the reader as an exercise.

Declaration of competing interest

The author declares that he has no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] M. Lothaire, *Algebraic Combinatorics on Words* (Encyclopedia of Mathematics and its Applications). Cambridge University Press, 2002 (⇒ 39).

[2] S. Brlek and S. Li, "On the number of distinct squares in finite sequences: Some old and new results," in *Combinatorics on Words*, A. Frid and R. Mercaş, Eds., Cham: Springer Nature Switzerland, 2023, pp. 35–44 (⇒ 39).

[3] A. S. Fraenkel and J. Simpson, "How many squares can a string contain?" *Journal of Combinatorial Theory, Series A*, vol. 82, no. 1, pp. 112–120, 1998, ISSN: 0097-3165. DOI: https://doi.org/10.1006/jcta.1997.2843. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0097316597928430 (⇒ 39).

[4] L. Ilie, "A simple proof that a word of length n has at most 2n distinct squares," *Journal of Combinatorial Theory, Series A*, vol. 112, no. 1, pp. 163–164, 2005, ISSN: 0097-3165. DOI: https://doi.org/10.1016/j.jcta.2005.01.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0097316505000130 (⇒ 39, 40).

[5] M. Patawar and K. Kapoor, "Density of distinct squares in non-primitive words," *Information Processing Letters*, vol. 182, p. 106 367, 2023, ISSN: 0020-0190. DOI: https://doi.org/10.1016/j.ipl.2023.106367. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020019023000108 (⇒ 39, 40).

Received: 30.01.2024; *Revised: 04.03.2024*; *Accepted: 06.03.2024*

Acta Univ. Sapientiae, Informatica, **16, 1** (2024) 44–61

DOI: 10.47745/ausi-2024-0004

Supporting the debugging of Erlang programs by symbolic execution

Zsófia ERDEI
ELTE, Eötvös Loránd University
Budapest, Hungary
zsanart@inf.elte.hu
0000-0002-5089-4984

Melinda TÓTH
ELTE, Eötvös Loránd University
Budapest, Hungary
toth_m@inf.elte.hu
0000-0001-6300-7945

István BOZÓ
ELTE, Eötvös Loránd University
Budapest, Hungary
bozo_i@inf.elte.hu
0000-0001-5145-9688

Abstract. Programmers can benefit from static source code analysis techniques in various ways: they can understand their code better, test it more effectively, debug it more efficiently, and so on. However, they often face the challenge of discovering how to reproduce faulty executions that cause runtime errors. The term symbolic execution refers to a static source code analysis method that can help with this challenge. In this paper, we are showing a symbolic execution-based analysis method to find the source of a runtime error. The method uses a control-flow graph to select the execution paths reaching the targeted runtime error. The algorithm is implemented as part of the RefactorErl static program analysis and transformation framework.

Key words and phrases: static analysis, fault localization, symbolic execution, Erlang

1 Introduction

Localising the reason for faulty behaviours is a crucial activity in software development and maintenance that aims to identify the exact locations of program faults that cause failures. Faults are defects or errors in the source code that may produce

incorrect or unexpected behaviour when executed. Fault localization is often very tedious, costly and time-consuming, as it requires developers to examine a large amount of code and test cases to find the root cause of failures. Fault localization also depends on developers' skills, knowledge, experience and intuition. To reduce manual effort and improve the efficiency of fault localization, various automated techniques have been proposed that leverage different sources of information.

Symbolic execution [1], [2] is a technique that can be used for fault detection by exploring different execution paths of a program with symbolic values instead of concrete values. Symbolic values represent a range of possible values that can satisfy certain constraints. Symbolic execution can find errors that are hard to detect with conventional testing methods, such as buffer overflows, division by zero errors, etc.

Symbolic execution works by maintaining a symbolic state and a path condition for each execution path. The symbolic state contains the symbolic values of variables. The path condition contains the constraints on the symbolic values that are derived from branch conditions along the path. Symbolic execution uses a constraint solver to check the feasibility of each path and to generate concrete inputs that can trigger faults.

The aim of our work is to help Erlang developers in debugging processes to reproduce a runtime error. We would like to use and extend the static analyser framework of RefactorErl with new algorithms to support this fault localisation process. We build our algorithm on the top of the intermediate source code representation of the tool, the so-called Semantic Program Graph [3] and heavily use its control flow analysis [4] backend. Besides RefactorErl we use the Z3 [5] SMT solver and build a direct symbolic execution engine for Erlang programs.

The paper is structured as follows: In Section 2, we define the problem we want to solve through examples. In Sections 3 and 4 we present our algorithm and a few examples to demonstrate how it works. In Section 5 we briefly introduce the tool RefactorErl and the built prototype implementation. Finally, Sections 6 and 7 introduce some related work and concludes the paper.

2 Problem definition

In general, the line-reachability problem in static analysis is a variant of the more general problem of control flow analysis, which aims to determine the flow of control in a program. Control flow analysis is essential for many program analysis tasks, including optimization, program verification, and testing.

To solve the line-reachability problem with static analysis, we need to construct a control flow graph (CFG) of the program, which represents the program's control

flow structure. Once we have the CFG, we can use standard algorithms, such as depth-first search or data-flow analysis, to determine whether a given line of code is reachable or not.

This problem can be generalized to finding a specific path in a program that causes it to enter a particular state. This can be useful for debugging or regression testing. For example, if we know that a line in the code can cause an error, finding the path - and the conditions and input values that lead to it - can make it easier to fix.

Let us use a division by zero as a simple example of the bug we want to find in the next examples.

The analysis begins from the user-specified target line that may cause an error. On the function level, we use forward symbolic execution to find a possible path from the start of the function to the target line. Based on the function's control-flow graph, we use depth-first traversal to find a path to the target expression. When we see a conditional branch, we update a set of conditions for the current path. We can use an SMT solver to check if the path we explored is feasible.

```
1  -module(example1).
2  -export([eg1/2]).
3  eg1(X, Y) ->
4    if
5      X * Y > 0 ->
6        X / Y;
7      true ->
8        Z = X * Y,
9        X / Z
10   end.
```

Figure 1: An example that may lead to runtime error

Let us consider the Erlang code in Figure 1. Suppose that a division by zero error has occurred during the execution in line 9. Our goal is to find out what set of inputs can trigger the line with the error. First, we build the control-flow graph of the function. Exploring it from the start node we can observe that the graph splits at the branch of the if expression. In order to find an execution path to the target line, the first guard of the if expression has to be evaluated as false and the second guard has to be evaluated as true. In our example, the second guard of the if expression is the default "true", which means otherwise. Therefore our constraint

set contains in this case the negation of the first condition ($\neg(X * Y > 0)$) and the "true", which can be omitted. From our first condition, we can see that the error can occur only if the product of the variables X and Y is non-positive. However, the conditions generated from the branches of the if expression do not provide enough information to calculate proper input values to reproduce the runtime error. The error occurs only if the value of the variable Z is equal to zero. Therefore, we need to add the condition $Z = 0$ to our set of constraints.

$$\neg(X * Y > 0)$$
$$Z = 0$$

Although Z is not an input of the function, thus considering only this constraint does not help us to find the proper set of inputs. We need to add further conditions about the possible values of Z that connect the inputs and Z. The first assignment of the selected branch provides some context for the value of the variable Z: $Z = X * Y$. Examined with the previous conditions, we can see that the value of Z can only be zero if either X or Y is zero. This establishes the following set of conditions:

$$\neg(X * Y > 0)$$
$$Z = 0$$
$$Z = X * Y$$

We can see that these conditions provide constraints for both input parameters X and Y. Using an SMT solver we can generate input values that can lead us to the discovered runtime error.

3 Overview of the algorithm

Our symbolic execution algorithm is designed to find a path from the programs start to the target expression that has been selected. The algorithm is presented in detail in the following section. The function takes four arguments. The first one is the control flow graph (G). The second one is a node that represents where the path search starts (S). The third one is a node that contains the target expression we want to know the path to (T). The last one is a list of nodes that we have visited so far (P). This algorithm can help us to find out what input values can make us reach the target line and what conditions they have to meet. The algorithm uses a kind of symbolic backward execution called call-chain backward symbolic execution [6]. This is a

type of symbolic execution that mixes forward and backward symbolic execution. Inside each function, it explores the execution paths forward, but it follows the call chain backwards from the target point to the program's entry point.

We start at the target expression. First, we find the path from the entry point of the function containing the target expression itself. This intraprocedural part of the algorithm (see Algorithm 1) uses the control-flow graph of the function to look for a possible path to the target node.

A depth-first traversal of the function's control-flow graph returns the first path found ($find\_path$), which yields a set of expression nodes in the graph representing the analysed code. This set can be used to determine the initial set of conditions. Upon completion of the path, we check the set of conditions for satisfiability to ensure the path's validity. If the conditions are not satisfiable, we discard the path and search for another one. We step back ($step\_back$) on the path until we reach the last non dead end node that generated a condition and resume the traversal of the graph. Dead ends on the graph are marked accordingly.

Algorithm 1: Selecting the path

```
Data: Control flow graph G, starting node S, target node T, current path P
Result: Path to target node
Cand ← find_path(G, S, T);
if no candidate found then
    NewPath ← step_back(P);
    backtrack(G, end_of(NewPath), T, NewPath);
else
    FullPath ← concat(Cand, P);
    if sat(FullPath) then
        return Cand;
    else
        NewPath ← step_back(FullPath);
        if no path found then
            return T is unreachable;
        end
        backtrack(G, end_of(NewPath), T, NewPath);
    end
end
```

We collect the initial set of conditions from the guards of the targeted expression. The condition set is extended with further conditions we found on the execution

path to the target point. These include not just the current guard of a branching expression, but also the negated conditions of the previously evaluated guards as well. This is needed because in Erlang, the branches of an if expression are scanned sequentially until a guard sequence that evaluates to true is found. As variable assignments on the execution path can also contribute to the path constraint set (as it was demonstrated in our previous example in Section 2) we add this as a condition as well.

In order to construct the set of conditions, it is necessary to keep track of the variables in the conditions: an execution path may contain variables with the same names from different scopes, thus we need to handle these cases. We map the variables we found during the exploration of the control-flow graph to a scope-sensitive semantic variable node. This node is saved in a map data structure with its original name. If a different semantic variable node with the same name is later encountered, the new variable is renamed in the constraint set.

Each candidate is part of a possible execution path of the program and will consist of a path from the function entry point to the current target node. We find the initial candidate by traversing the tree in a depth-first order. If no such path is found, the target node is unreachable. While traversing the graph we keep track of the visited nodes. When a node is reached from where the target is unreachable, it is marked as a dead end. Examples of such nodes are the leaves or nodes having all children already marked as dead ends. If the conditions of the selected path cannot be satisfied, a new path has to be searched for. In order to find a new path we use backtracking, which is a basic strategy to solve constraint satisfaction problems. After a candidate path has been selected, we produce the set of conditions defined by the route. We then determine the function's callers and recursively repeat this process along the backward call chains. The path search stops when we reach the predefined entry point of the program.

4 Working examples

In this section, we will showcase the workings of our algorithm through some example code snippets. The first example will dissect a straightforward scenario where we detect a path of a runtime error. The second instance demonstrated in Subsection 4.2 shows a case where the first found path is unjustifiable and we have to use backtracking to uncover the correct path. Lastly, the third example provide an overview of how our intraprocedural algorithm operates. These working examples will showcase our algorithm's functionality in distinct scenarios.

4.1 Simple intraprocedural example

Consider this simple example in Figure 2. Similarly to the previous example, this code snippet contains divisions, and if the denominator C is zero, an error occurs (Figure 3). Suppose that the error occurred in line 10. We can use the algorithm to find a realizable path to the target expression from the entry point of the program, and also determine a set of input values that may trigger the error.

```
1  -module(example1).
2  -export([foo/2]).
3  foo(A, B) ->
4    C = A + B,
5    if
6      C < A -> 0;
7      C == A ->1;
8      C > A ->
9        if
10         B > C -> A / C;
11         true -> B / C
12       end
13   end.
```

Figure 2: Another example that may lead to runtime error

```
5> example1:foo(A,B).
** exception error: an error occurred when evaluating an arithmetic expression
     in function  example1:foo/2 (example1.erl, line 10)
```

Figure 3: Runtime error

The algorithm starts by determining a path in the CFG from the function containing the target expression to the entry point of the function. In this case, the path found will be the one demonstrated in Figure 4.

First, we need to collect the conditions along the path to building our initial set of constraints. Since the error occurred due to a division by zero, our first condition will be determined from the expression containing the division A/C. This error can only occur if the denominator C is zero, so the first condition for the execution path will be $C = 0$. The next step is to examine the selected execution path. As we can see, the path contains two if expressions. At the first one, the first two conditions will not be met, but the third will evaluate to true. This means we need to add the

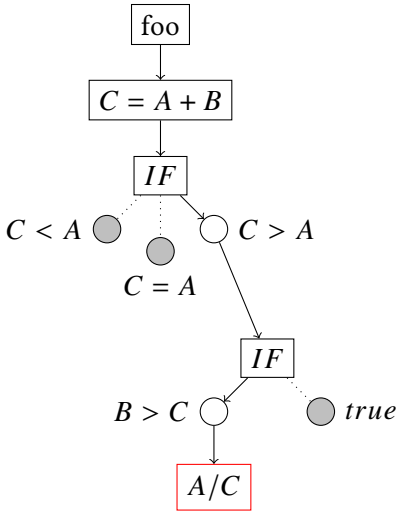


Figure 4: Possible execution path of example presented on Figure 2

conditions derived from the guard expressions $\neg(C < A), \neg(C = A), C > A$ to the set of constraints.

After we evaluated all necessary conditions of the first if expression, we can continue to walk the selected path. When we get to the second if expression, we can see, that this time the first guard will evaluate to true. This time, we only need to add the first condition to our set.

$$
\begin{aligned}
&C = 0\\
&\neg(C < A)\\
&\neg(C = A)\\
&C > A\\
&B > C
\end{aligned}
$$

This will be our initial set of constraints determined from the if expressions on the selected path. However as we could see in Section 2, these constraints are neither enough to determine with which inputs we reach the expression where the error occurred nor to provide valuable information on the cause of the error.

The next step is to determine the constraints for the variables used in the set of conditions: A, B and C. For this, we need to find any assignment expression containing any of these variables on the left side of the path. Using these we can build extra constraints for our set of conditions. In this example, the only such

expression is $C = A + B$ on line 4. Since in the example the second, third and fourth conditions can be simplified to $C > A$, our set of path conditions will be as follows:

$$C = 0$$
$$C > A$$
$$B > C$$
$$C = A + B$$

From this, we get a set of path conditions that when satisfied the program will run into the division by zero error at line 10. Using a constraint solver on this set of conditions will provide us with a set of input parameters that will guarantee that the error will occur. If the set of conditions would be unsatisfiable, a new path would be determined by backtracking on the CFG of the function.

4.2 Execution path selection

```
1  -module(example1).
2  -export([foo/2]).
3  foo(A, B) ->
4    if
5      A == B -> B;
6      A == -B -> A;
7      true -> 1
8    end,
9    if
10     A == 0 -> A;
11     A rem 2 == 1 -> C = A + B, A / C;
12     true -> C = A - B, A / C
13   end.
```

Figure 5: An example of a path selection

The example in Figure 5 shows why it is important to check the set of conditions for satisfiability during path selection. In this example code, a division by zero error may occur on lines 11 and 12, if the value of C equals zero. Let us assume, that during execution we found the error on line 11. We can see, that since the value of C is calculated as the sum of A and B, this error may only occur if $A = -B$.

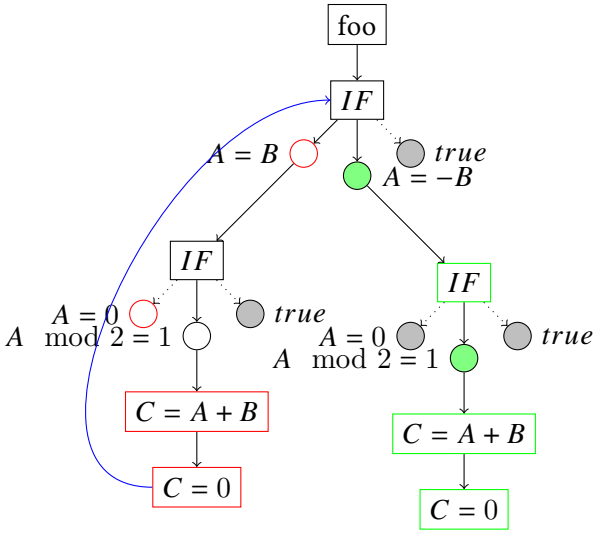


Figure 6: Possible execution path of example presented on Figure 5

During the analysis if we determine a path in the CFG from the functions entry point to the target expression, we get the following set of conditions:

$$A = B$$
$$\neg(A = 0)$$
$$A \mod 2 = 1$$
$$C = A + B$$
$$C = 0$$

If we examine these conditions, we can see that we have a contradiction between our original assumption (that the error may occur on that line only in the case when $A = -B$) and the condition $A = B$, since $A = B = 0$ is not possible, because then the first guard $A == 0$ would evaluate to true. This means that this execution path could not occur during actual execution, so we need to find another path in the CFG. Using backtracking we continue to traverse the graph to find a different path to the target. When the first if expression is reached again, this time the algorithm will choose the next branch. Similarly to the previously examined path, we need to determine the constraints. On this newly selected path we will have the following conditions:

$$\neg(A = B)$$
$$A = -B$$
$$\neg(A = 0)$$
$$A \mod 2 = 1$$
$$C = A + B$$
$$C = 0$$

Using a constraint solver on this set of conditions we can ascertain that the new path is feasible, and we can find a set of values for parameters that will trigger the error found at the target expression.

Figure 6 illustrates how the algorithm works on the provided example traversing the CFG. The first found path is illustrated with red nodes where a contradiction is found between the conditions. At that point we step back on the tree to the last condition generating node (the upper if expression) from where a possible path is found to the target expression. When the new path is checked, this time the set of conditions is satisfiable. This correct path marked with green is returned by our analysis.

4.3 Interprocedural example

The previous examples showed how the algorithm determines the execution path in a function. Our next example, presented in Figure 7, will demonstrate how we can use the call chain to find the full execution path. For the sake of simplicity, instead of an actual error, the target will be the atom "fail" on line 17. Similarly to the previous example, a path from the function entry point to the target will be determined. The constraints will only consist of the constraints built from the guards of the only if expression of the function $f2$: $\neg(A < 2)$ and $true$. Examining our set of conditions, it only contains the variable A, we save it in our variable map. Since in this example function, there is only one if expression and no other expressions, we do not have to add further constraints on the variables to the constraint set. By checking the set of conditions with a constraint solver we can verify that the path is so far feasible.

The next step is to determine the callers of the function $f2$. Using RefactorErl we can collect all expressions that contain such a function call. In this example, we only get one result, the expression in line 7. This expression will be our next target, and the new starting point will be the function containing the expression: f. Similarly to the previous function, we find a path to the target node on the CFG and collect the path conditions. For the sake of simplicity of the example, we only

```
1  -module(multi_fun_example).
2  -export([f/1]).
3
4  f(A) ->
5      if
6          1 < A ->
7              f2(3);
8          true ->
9              ok
10     end.
11
12 f2(A) ->
13     if
14         A < 2 ->
15             ok;
16         true ->
17             fail
18     end.
```

Figure 7: Example of following the execution path through multiple functions

get one such condition: $1 < A$. Again, the variables used in the conditions have to be saved in the variable map. However since the map already contains a variable with the name A which refers to a different variable, it will be renamed in the map to $A2$. After checking the path for feasibility the following set of conditions were established: $\neg(A < 2)$, $true$, $1 < A2$. As the analysis reached the main entry point of the function, that means we found a full execution path. The constraint solver with this set of conditions will provide us with a set of input parameters so the original target expression can be reached.

5 Realisation in RefactorErl

RefactorErl [7] is a source code analysis and transformation tool for Erlang. It provides a wide variety of functionalities to support code comprehension, dependency analysis, source code checks, refactorings, etc. The tool uses a rich intermediate source code representation, the so-called Semantic Program Graph (SPG). The SPG stores lexical, syntactic and semantic information about the source code, calculated by various static semantic analysers.

RefactorErl also has control-flow and control dependence analyses [4]. Thus we can build our direct symbolic execution engine on top of the available control-flow information and the syntactic and semantic information available in the SPG.

The prototype implementation of the algorithm uses multiple program representations provided by the RefactorErl tool. The Control-Flow Graph (CFG) represents all the possible execution paths of the program that can be chosen for every possible input. We traverse the CFG in order to search for possible execution paths to the target node. Each candidate path consists of a series of expressions leading to the target expression itself. While traversing the graph, we keep track of the variables on the path and also which vertices we visited and which vertices lead only to a dead end. This is necessary because if the constraints for the candidate path turn out to be unsatisfiable, we will have to look for a different path.

If a candidate path is selected, we have to determine the set of constraints on the path. To do this, we use the Semantic Program Graph (SPG). The SPG is a rooted, directed, labelled and indexed graph that stores lexical, syntactic and semantic information about the source code, calculated by various static semantic analysers. Since the SPG uses the same identifiers for the vertices in the CFG, we can use this to extract more information from the expressions on the candidate path. Using this we can determine the set of conditions as previously described in Section 3.

Once we have found a possible route in the tree from the starting point to the target expression node, we use Z3 to check the constraints found on the path for satisfiability. The Google protocol buffer (protobuf) [8] is used to serialize the set of constraints. The possible forms of the conditions were predefined in a proto file and we use the gpb tool [9] to compile the definitions. Z3 takes as input simple-sorted formulas that are built from atomic variables and logical connectives. By adding the conditions defined by the candidate path to a Z3 model and using the solver to try to find a solution, it either produces the verdict *unsat* or *sat* and in the case of the latter we also get a solution to the possible values of the variables. The complete solution is serialized again and the results are processed on the Erlang side. If the case that the constraints on the candidate path are unsatisfiable, it is necessary to find an alternative path.

6 Related Work

Symbolic execution is a technique used by many program analysis and transformation techniques, such as partial evaluation, test-case generation or model checking. While symbolic execution is not a new topic in the Erlang ecosystem, previously published papers mostly focus on formal [10], [11] and informal [12] definitions

with the aim of program verification. In this paper, we present a symbolic execution technique for Erlang that can support debugging processes of Erlang developers through the RefactorErl framework. Our goal is not to verify Erlang programs, but to support their debugging processes through the RefactorErl framework. Previous work on symbolic execution in Erlang focused on verification [10], [11] or used informal definitions [12].

Symbolic execution can be used for testing purposes as well. CutEr [13], [14] is a concolic testing [15] tool for Erlang. It can automatically generate test cases that cover different execution paths of an Erlang program by combining concrete and symbolic execution. CutEr can also detect runtime errors such as arithmetic exceptions, bad arguments, or pattern-matching failures. We can find similarities and differences in our and CutEr's approaches. Both tools are using the Z3 SMT solver to evaluate the collected symbolic constraints, but the evaluated execution path selection is different. We use a static control-flow path and traverse it backwards. CutEr takes a real/dynamic execution path and traverses it forward.

In [16] a symbolic execution-based runtime error detection algorithm is presented. The presented method transforms Erlang programs into Prolog facts, evaluates those on symbolic input data and reports input patterns that lead to runtime errors within a given bound. In contrast, the method presented in this paper is looking at the problem from the other direction. It takes the known runtime error as input and searches for execution paths that may lead to faulty behaviour.

In [6], a generic algorithm for call-chain backward execution was presented, which in combination with any forward search strategy resulted in an efficient way to solve the line-reachability problem. In this algorithm, the main difference to the traditional symbolic backward execution is that, while it follows the call chain backwards from the target point, inside each function the exploration is based on a traditional forward symbolic execution. The algorithm starts by determining a valid path in the function containing the target line, then it moves to one of the callers of the function. This process is recursively repeated until the full path from the program's main entry point to the target is determined. The authors implemented these strategies in Otter [17], a C source code symbolic executor, and studied their performance and compared the results with a range of state-of-the-art forward search strategies like KLEE [18], SAGE [19].

KLEE uses two main search strategies: Random Path Selection and State-Based Search. Random Path Selection maintains a binary tree recording the program path followed for all active states, where the internal nodes are the ones where the execution has forked and the leaves represent the current states. The states are selected by traversing this tree from the root and randomly selecting the path to follow at branch points. During the symbolic execution when an internal node is reached, all

child nodes of the given node have an equal probability to be selected by the algorithm regardless of the size of the subtrees. The biggest advantage of this strategy is that it avoids starvation occurring in loops containing symbolic conditions and resulting quick new state creation.

SAGE uses the generational search, which is an algorithm designed for dynamic test generation that tolerates divergences. This strategy maximises the number of generated new input tests during the symbolic execution, it can generate thousands of new tests by a single symbolic execution. For a given path constraint, SAGE negates all the constraints in that path, conjuncts them with the initial constraint and use a constraint solver to solve it.

Both call-chain backward execution combined with KLEE and the shortest distance symbolic execution (the other technique presented in the paper) outperformed other strategies. One mentioned disadvantage was the difficulty of the construction of the inter-procedural control-flow graph for call-chain backward execution. However, RefactorErl is already able to build this, thus this option suited us the most.

7 Conclusion and Future Work

The identification of the sources of a runtime error is a common task for Erlang developers. Dynamic and static tools can assist in this task. Dynamic tools are used to analyze the behaviour of a program during execution, while static tools are used to analyze the program's source code without executing it.

In this paper, we propose a method based on static analysis of Erlang programs to identify execution paths that may lead to a given runtime error. The proposed method uses the control-flow graph of RefactorErl, which is a static code analysis tool that can be used to analyze and refactor existing Erlang code bases.

The presented algorithm uses the control-flow graph of RefactorErl and applies dynamic backward symbolic execution to gather the constraints of the execution. Dynamic backward symbolic execution is a technique that starts from the error location and works backwards to the input parameters of the function. The technique generates a set of constraints that must be satisfied by the input parameters to reach the error location.

We use the Z3 SMT solver to decide the reachability of a path and calculate possible input values for real execution. The Z3 SMT solver is a tool for solving logical formulas. The proposed method can be used to identify execution paths that may lead to a runtime error.

We have implemented the presented algorithm for a reasonable subset of the Erlang language. Our main goal in the future is to improve language coverage. We

also would like to improve the execution path selection algorithm.

Funding: The research was (partially) supported by the ÚNKP-22-3 New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund.

Data Availability: The study did not generate new data.

References

[1] J. C. King, "Symbolic execution and program testing," vol. 19, no. 7, pp. 385–394, Jul. 1976, ISSN: 0001-0782. DOI: 10.1145/360248.360252. [Online]. Available: https://doi.org/10.1145/360248.360252 (⇒ 45).

[2] R. Baldoni, E. Coppa, D. C. D'elia, C. Demetrescu, and I. Finocchi, "A survey of symbolic execution techniques," *ACM Comput. Surv.*, vol. 51, no. 3, May 2018, ISSN: 0360-0300. DOI: 10.1145/3182657. [Online]. Available: https://doi.org/10.1145/3182657 (⇒ 45).

[3] Z. Horváth, L. Lövei, T. Kozsik, *et al.*, "Modeling semantic knowledge in Erlang for refactoring," in *Knowledge Engineering: Principles and Techniques, Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques, KEPT 2009*, ser. Studia Universitatis Babeş-Bolyai, Series Informatica, vol. 54(2009) Sp. Issue, Cluj-Napoca, Romania, Jul. 2009, pp. 7–16 (⇒ 45).

[4] M. Tóth and I. Bozó, *Static analysis of complex software systems implemented in erlang*, Central European Functional Programming Summer School – Fourth Summer School, CEFP 2011, Revisited Selected Lectures, Lecture Notes in Computer Science (LNCS), Vol. 7241, pp. 451-514, Springer-Verlag, ISSN: 0302-9743, 2012 (⇒ 45, 56).

[5] L. de Moura and N. Bjørner, "Z3: An efficient smt solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and J. Rehof, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340, ISBN: 978-3-540-78800-3 (⇒ 45).

[6] K.-K. Ma, K. Yit Phang, J. S. Foster, and M. Hicks, "Directed symbolic execution," in *Static Analysis*, E. Yahav, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 95–111, ISBN: 978-3-642-23702-7 (⇒ 47, 57).

[7] I. Bozó, D. Horpácsi, Z. Horváth, *et al.*, "Refactorerl - source code analysis and refactoring in erlang," in *Proceedings of the 12th Symposium on Programming Languages and Software Tools, ISBN 978-9949-23-178-2*, Tallin, Estonia, Oct. 2011, pp. 138–148 (⇒ 55).

[8] protocolbuffers, *Protocol buffers documentation*, [Accessed: Feb, 2023]. [Online]. Available: https://protobuf.dev/ (⇒ 56).

[9] Tomas Abrahamsson, *Gpb*, [Accessed: Feb, 2023]. [Online]. Available: https://protobuf.dev/ (⇒ 56).

[10] G. Vidal, "Towards symbolic execution in erlang," in *Perspectives of System Informatics*, A. Voronkov and I. Virbitskaite, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 351–360, ISBN: 978-3-662-46823-4 (⇒ 56, 57).

[11] G. Vidal, "Towards erlang verification by term rewriting," in *Logic-Based Program Synthesis and Transformation*, G. Gupta and R. Peña, Eds., Cham: Springer International Publishing, 2014, pp. 109–126, ISBN: 978-3-319-14125-1 (⇒ 56, 57).

[12] C. B. Earle, "Symbolic program execution using the erlang verification tool," in *9th International Workshop on Functional and Logic Programming, WFLP'2000, Benicassim, Spain, September 28-30, 2000*, M. Alpuente, Ed., 2000, pp. 42–55 (⇒ 56, 57).

[13] K. Sagonas, *A cuter tool. talk at erlang factory 2016*, [Accessed: Feb, 2023], 2016. [Online]. Available: http://www.erlang-factory.com/static/upload/media/1457739488660923kostissagonasacutertool.pdf (⇒ 57).

[14] CutEr, *Cuter github page*, [Accessed: Feb, 2023]. [Online]. Available: https://github.com/cuter-testing/cuter (⇒ 57).

[15] K. Sen, D. Marinov, and G. Agha, "Cute: A concolic unit testing engine for c," *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 5, pp. 263–272, Sep. 2005, ISSN: 0163-5948. DOI: 10.1145/1095430.1081750. [Online]. Available: https://doi.org/10.1145/1095430.1081750 (⇒ 57).

[16] E. De Angelis, F. Fioravanti, A. Palacios, A. Pettorossi, and M. Proietti, "Bounded symbolic execution for runtime error detection of erlang programs," in *Proceedings 5th Workshop on Horn Clauses for Verification and Synthesis, HCVS 2018, Oxford, UK, 13th July 2018*, T. Kahsai and G. Vidal, Eds., ser. EPTCS, vol. 278, 2018, pp. 19–26. DOI: 10.4204/EPTCS.278.4. [Online]. Available: https://doi.org/10.4204/EPTCS.278.4 (⇒ 57).

[17] R. Majumdar and K. Sen, "Hybrid concolic testing," in *Proceedings of the 29th International Conference on Software Engineering*, ser. ICSE '07, USA: IEEE Computer Society, 2007, pp. 416–426, ISBN: 0769528287. DOI: 10.1109/ICSE.2007.41. [Online]. Available: https://doi.org/10.1109/ICSE.2007.41 (⇒ 57).

[18] C. Cadar, D. Dunbar, and D. Engler, "Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs," in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'08, San Diego, California: USENIX Association, 2008, pp. 209–224 (⇒ 57).

[19] P. Godefroid, M. Y. Levin, and D. Molnar, "Automated whitebox fuzz testing," in *Proceedings of the Network and Distributed System Security Symposium (NDSS'08)*., vol. 8, Nov. 2008, pp. 151–166. [Online]. Available: https://www.microsoft.com/en-us/research/publication/automated-whitebox-fuzz-testing/ (⇒ 57).

Received: 30.01.2024; *Accepted: 04.03.2024*

Acta Univ. Sapientiae, Informatica, **16, 1** (2024) 62–77

DOI: 10.47745/ausi-2024-0005

Integrating Optical Flow into Deep Learning based Distortion Correction

Szabolcs-Botond LŐRINCZ-MOLNÁR
Independent Researcher
lorincz.szabolcs.botond@gmail.com
0000-0002-2202-9491

Szabolcs PÁVEL
Babeş-Bolyai University
szabolcs.pavel@ubbcluj.ro
0000-0002-8825-2768

Abstract. One crucial task in 3D computer vision is the correction of geometric distortions, since most algorithms rely on the assumption that the image formation process can be described by a specific camera model, e.g. the pinhole camera model. In an autonomous driving scenario, however, the front-facing camera is most commonly placed behind the windshield, causing complex, nonlinear distortions.

Previous attempts have been made to undistort such images using deep learning based methods. The input of these deep networks usually consists of one or more images, and they optionally include additional tasks such as semantic segmentation to improve the results.

We hypothesize that the well-constrained nature of optical flow in rigid, static scenes provides useful cues for the process of image undistortion. By using optical flow as an additional input, we present a multi-view distortion correction method achieving superior results on both synthetic and real-world images compared to previous works, demonstrating the usability of optical flow for correcting highly complex distortions.

Key words and phrases: camera calibration, distortion correction, deep learning

1 Introduction

Front-facing cameras in autonomous driving systems are most commonly placed behind the vehicles' windshields, therefore the captured image sequences suffer from

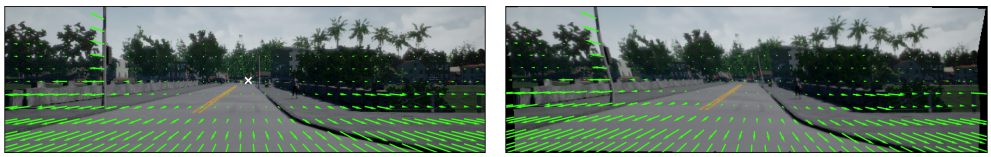


Figure 1: In the case of a calibrated camera and pure forward motion (left) all optical flow vectors emerge from the focus of expansion (white cross), situated at the principal point. In the case of geometric distortions (right), aberrations are induced in both the magnitude and direction of the vectors. The irregularity of distorted optical flow vectors provides useful cues for distortion correction.

complex geometric distortions caused by light refraction. This anomaly impacts the performance of various computer vision pipelines, inducing errors for instance in scene reconstruction, depth estimation and in camera based driver assistance systems in general. Such kind of errors are not admissible in critical systems, thus, these distortions must be corrected.

Several attempts have been made to correct distortions caused by different refractive surfaces, *e.g.* radial distortions caused by the camera lens [1] or tangential distortions, the root of which lies at the nonparallelism of the lens and the image plane. Distortions caused by wide angle (fisheye) lenses typically used in autonomous driving systems have also been succesfully corrected [2].

Recent experiments show that the correction of more complex distortions, such as the ones caused by windshields is also feasible, by employing deep neural networks and using additional tasks to guide the process of undistortion, such as semantic segmentation [3].

In this work, we extend previously proposed methods for correcting complex distortions caused by car windshields and achieve superior performance on both synthetic and real-world data sets. We also show experimentally that using optical flow as an additional input to a deep learning based distortion correction method improves the performance of the system by exploiting the predictability and regularity of optical flow vector directions and magnitudes. The proposed method can be trained without the need of conducting rigorous measurements to obtain ground truth distortion fields, by leveraging differentiable image sampling, enabling us to produce the undistorted image jointly with estimating the distortion parameters.

2 Related Work

Calibration The task of predicting distortion parameters and correcting distortions has been studied for a few years by now. Early methods tried to correct simpler distortions such as radial or tangential distortions, the former being caused by the camera lens, the later by the lens being non parallel relative to the image plane. These methods can be categorized into two major groups: self-calibration methods exploiting geometric constraints based on multiple view image sequences [4]–[6], and static calibration approaches using a calibration object or pattern [7]–[11] where relative positions of specific keypoints are known, but their perceived positions are distorted during projection.

Deep Learning based Distortion Correction Recent distortion correction methods started to employ deep learning, specifically convolutional neural networks (CNNs) to predict radial distortion parameters based on arbitrary single view input images [1], without the need of having a calibration pattern. Later, CNNs have been used to estimate the parameters of more complex distortions, such as fisheye distortions caused by wide angle cameras [2] or distortions caused by windshields [3], using semantic segmentation as an additional task to guide the correction process.

Distortion Model Each distortion estimation or correction algorithm employs a specific distortion model, ranging from simple models *e.g.* Brown's polynomial model [12] for radial and tangential distortions to the more complex methods introduced in [13]–[15], explicitly modeling the refractive surface. In this work, we use thin plate spline (TPS) interpolation [16] to model the two dimensional geometric distortions caused by windshields.

Optical Flow based Image Reconstruction Studies of using optical flow in the distortion correction process have also been conducted. Non-rigid geometric distortions caused by atmospheric turbulences were corrected in [17] using an optical flow scheme and a non local total variation (TV) regularization, while in [18] distortions of the same kind are corrected using a non-rigid image registration algorithm based on B-splines, embedded in a Bayesian framework with bilateral TV regularization. Both methods focus on restoring images using video sequences having no camera motion, assuming a constant scene. In an autonomous driving scenario this assumption does not hold, therefore a different approach must be taken.

Supervision through View Synthesis Several methods rely on direct supervision from ground truth distortion parameters [1], [19]. In the case of simple forms of distortions it is possible to obtain these parameters for specific settings separately. However, the diversity of windshields and the complexity of the distortion caused by them makes the collection of large data sets a tedious task. The introduction

of differentiable inverse warping by [20] led to the emergence of several methods employing novel view synthesis [21] as supervision to solve the problem of layered 3D scene representation [22], unsupervised learning of depth, ego-motion [23] and optical flow [24].

In this work we employ a reconstruction loss as supervision based on Multi Scale Structural Similarity [25]. In contrast with novel view synthesis based methods, instead of synthesizing a new image from a different view given a single image, we generate corrected images given three distorted images and two corresponding optical flow maps. This enables the method to be trained even in the case of real-world, complex and highly variable distortions, when the estimation of ground truth parameters is not feasible, but distorted and correct image pairs can be obtained.

3 Motion Field Constraints

Our main hypothesis is that using optical flow as an additional input to a distortion correction network provides important information about the distortion field, and as a result can improve the performance of the deep learning system. In an ideal scenario the optical flow corresponds to the motion field – the projection of 3D velocity vectors to the image plane [26], [27]. Assuming a static, rigid scene the 3D velocity vectors are only influenced by the ego-motion of the camera. In a front-facing camera used in autonomous driving scenarios further assumptions can be made, such that the dominant motion component is the forward translation, and in some cases the yaw rotation. These assumptions cause the optical flow to become well predictable, and deviations from the predicted optical flow field are in part caused by the geometric distortions of the imaging system.

Let $\boldsymbol{P} = (X, Y, Z)$ be a 3D point in the scene, with a corresponding velocity vector (time derivative of $\boldsymbol{P}$) $\boldsymbol{V} = (V_x, V_y, V_z)$. The perspective projection of $\boldsymbol{P}$ to the image plane is denoted by $\boldsymbol{p} = (x, y)$, and is given by the first two components of the vector $\frac{f\boldsymbol{P}}{Z}$, where f denotes the focal distance of the camera. Then the 2D velocity vectors $\boldsymbol{v} = (v_x, v_y)$ of the motion field can be computed as a function of the 3D position $\boldsymbol{P}$ and velocity $\boldsymbol{V}$ by differentiating the 2D pose $\boldsymbol{p}$ w.r.t. the time, resulting in:

$$v_x = \frac{fV_x - xV_z}{Z} \qquad v_y = \frac{fV_y - yV_z}{Z}. \tag{1}$$

The 3D velocity vector can be written as $\boldsymbol{V} = \boldsymbol{T} + \boldsymbol{\Omega} \times \boldsymbol{P}$, where $\boldsymbol{T} = (T_x, T_y, T_z)$ is a linear velocity (translation), and $\boldsymbol{\Omega}$ is the angular velocity. Assuming zero angular velocity, the velocity vector is equal to the translation $\boldsymbol{T}$ of the 3D points, and it is

independent of the 3D position of the point. As a result, Eq. (1) is reduced to:

$$v_x = \frac{fT_x - xT_z}{Z} \qquad v_y = \frac{fT_y - yT_z}{Z}. \tag{2}$$

In this case we can observe, that the motion field is composed of radial vectors emerging from a common point on the image called the focus of expansion (in the case of forward motion). The focus of expansion is influenced by the T_x and T_y components of the translation. If both x and y components are zero ($\boldsymbol{T} = (0, 0, T_z)$), i.e. we have pure forward motion, the focus of expansion corresponds to the principal point of the image. One more desirable property is that the scene structure (the depth Z of the 3D points) only influences the magnitude, but not the direction of the velocity vectors. As a consequence the direction of these vectors by themselves can provide useful constraints, while the large changes in magnitude can signal object boundaries or occlusions. An example of the motion field with pure forward motion can be seen in Fig. 1, both in the case of a calibrated camera and in the presence of complex distortions caused by a windshield.

One important limitation of optical flow based distortion estimation in forward motion scenarios is that optical flow provides no information about the distortions in the radial direction [19]. An ambiguity exists where scene depth and radial distortions both influence only the magnitude of the motion field vectors, and an infinite number of depth - distortion pairs can result in the same motion field. In fact, this is a major factor in making optical flow based distortion estimation using classic computer vision challenging. A learning-based system however, despite this ambiguity, is still able to filter out the information relevant for the given task, therefore optical flow remains a useful input for distortion estimation.

4 Methods

4.1 Distortion Model

The distortion model in this work is identical to the one proposed in [3] to provide a fair comparison between the two methods and to allow us to properly quantify the effects of integrating optical flow into the distortion correction process.

The model relies on a pair of *thin plate splines (TPS)* forming a two dimensional linear map. The TPS transformation $\boldsymbol{f}_{tps}$ consists of two parts, the first being an affine transformation, while the second corresponding to the superposition of geometrically independent affine-free deformations [16], and is given by

$$\boldsymbol{f}_{tps}(G_i) = A \begin{bmatrix} G_i \\ 1 \end{bmatrix} + \sum_{k=1}^{n} \varphi(\left\| \boldsymbol{p}'_k - G_i \right\|_2) \cdot \boldsymbol{w}_k, \tag{3}$$

Table 1: Mean and Standard Deviation of Original Distortion Vector Norms

Data Set	Mean (px)	SD (px)
DC Test	8.46	3.92
DK 00	8.59	3.32

where $G_i = [x_i, y_i]^\top$ represents the image coordinates on the undistorted target image and n corresponds to the number of control points, in our case $n = 16$. The TPS kernel is denoted by $\varphi(r) = r^2 log(r)$, where r represents the L^2 distance between two points, with $P' = [\boldsymbol{p}'_1, \boldsymbol{p}'_2, \ldots, \boldsymbol{p}'_n] \in \mathbb{R}^{2\times n}$ being the coordinates of target control points. In our case points P' are evenly distributed and fixed on a 4×4 grid, whereas the coordinates of source control points $P = [\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n] \in \mathbb{R}^{2\times n}$ have to be estimated based on the distorted images and optical flows. The sampling grid is obtained by interpolating the displacements between point correspondences in P' and P. The transformation can be efficiently implemented by matrix operations as detailed in [3].

The properties of the map enables it to model various types of complex two dimensional deformations such as skeletal shape abnormalities caused by Apert syndrome [16], or even geometric distortions caused by refractive surfaces [3].

In this work, we applied the same parametric distortions sampled from a distribution derived from real-world measurements in the presence of windshields as in [3]. The mean and standard deviation of distortion norms reported in Table 1 is expressed in pixels in the distorted images.

4.2 Proposed Solution

In order to solve the problem of geometric distortion correction, we propose an end-to-end architecture similar to the architecture presented in [3] with some modifications.

First, the inputs of the network are three consequent RGB images, making our approach a member of multi-view distortion correction methods. As a direct consequence, the outputs of the network are also three consequent, corrected images, in addition to the estimated distortion parameters.

Furthermore, we feed two optical flows corresponding to the three images for guiding the process of distortion correction instead of employing an additional task

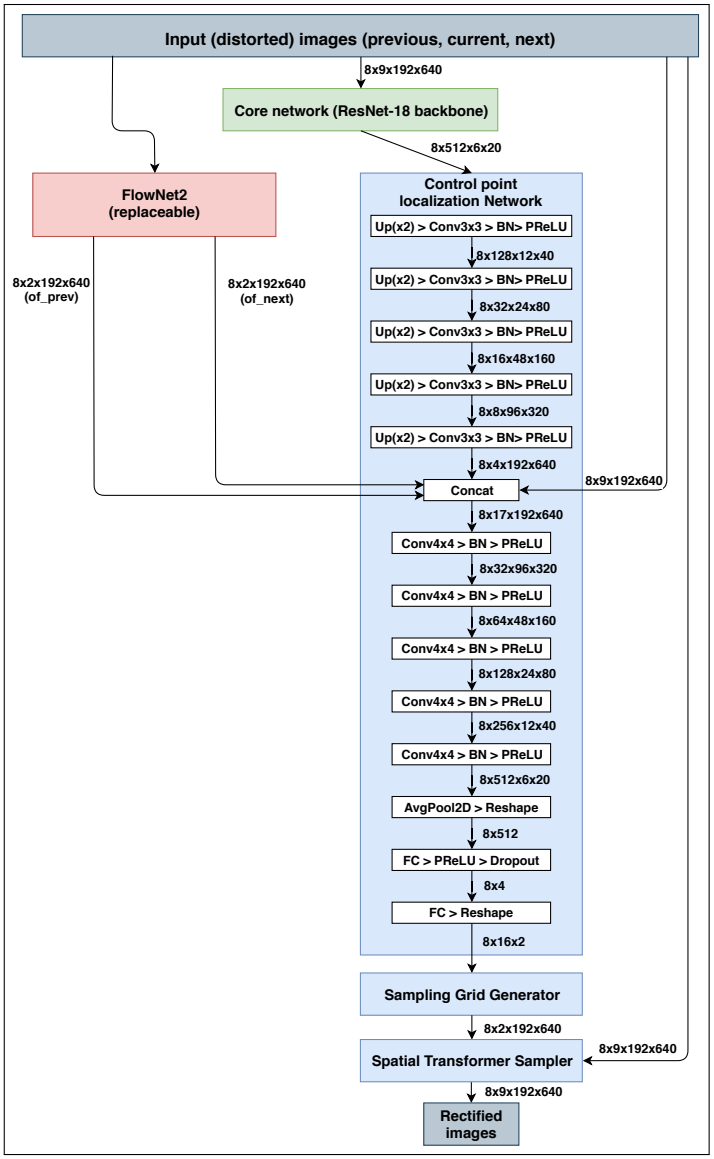


Figure 2: The proposed architecture consists of three key modules: the core network (green), the optical flow network (red) and a Spatial Transformer module (blue). The input of the network is formed of three consequent distorted images, based on which it produces two optical flows. The optical flow maps are concatenated to the input images and the upsampled features generated by the core network and they are jointly used for estimating the parameters of the TPS transformation to correct the images.

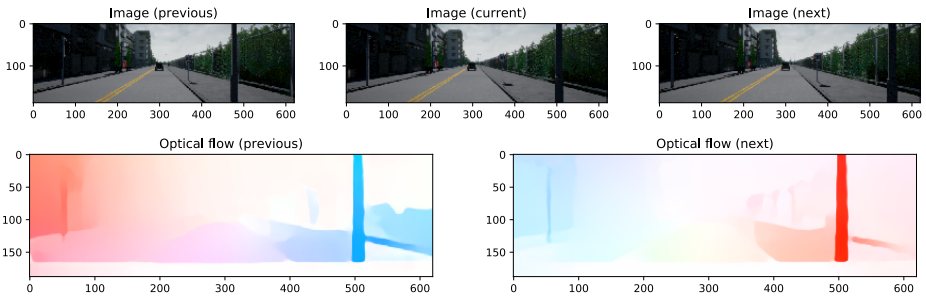


Figure 3: Using three consequent image frames (top), we calculate the optical flow between the second- and the first, and between the second- and the third image (bottom) using FlowNet 2.0, which we later use for distortion correction.

such as semantic segmentation as in [3].

The model corrects images in two steps: a feature extraction step, and a distortion correction step. For feature extraction we use ResNet-18 [28] pre-trained on ImageNet [29] as the core network. To obtain optical flow based on the three consequent input images, we use the pre-trained PyTorch [30] implementation [31] of FlowNet 2.0 [32].

For distortion correction we utilize the Spatial Transformer module [20], a differentiable image warping mechanism, also integrating the TPS interpolation based distortion model into the framework.

The Spatial Transformer module first estimates the parameters of the TPS transformation, being the coordinates of the source control points. Then, a sampling grid is generated based on the source and target control point coordinates, in our case this corresponds to the inverse of the distortion field. Lastly, the sampling grid is used to sample the distorted images, then the pixel values in the corrected images are calculated by bilinear interpolation. Since all three steps are differentiable, end-to-end learning is achievable. The detailed architecture is presented in Fig. 2.

In our experiments we explore the possibility of minimizing multiple loss functions separately and in a joint fashion. In order to enforce the reconstruction of the corrected images in terms of luminance, contrast and structure, the reconstruction loss $\mathcal{L}_r$ proposed in [3] given by Eq. (4) is used which is based on Multiscale Structural Similarity (MS-SSIM) [25] between a ground truth correct image (I) and the predicted corrected image ($\hat{I}$).

$$\mathcal{L}_r = -\frac{\text{MS-SSIM}(I, \hat{I}) + 1}{2} \tag{4}$$

By minimizing the reconstruction loss, the network is trainable even on data sets containing real-world distortions, when the ground truth sampling grid is hard or even impossible to obtain accurately, but distorted and correct image pairs are available.

In our experiments synthetic distortions are applied, thus, the ground truth sampling grid can be determined. For this reason, we experiment with direct minimization of the mean squared error (MSE) between the ground truth sampling grid and the predicted sampling grid, namely the grid loss $\mathcal{L}_g$ proposed in [3]. We also quantify its effect on model performance.

The joint loss is a linear combination of the two losses given by the following equation:

$$\mathcal{L} = \mathcal{L}_r + \lambda_1 \mathcal{L}_g, \tag{5}$$

where λ_1 is a weighting coefficient balancing the two loss functions set to $\lambda_1 = 100$ experimentally.

4.3 Experimental Setup

With the purpose of being able to correct not only synthetic but distorted real-world images also, we followed the experimental setups of Lőrincz et al. [3] and constructed two data sets, which we name Distorted Carla (DC) and Distorted KITTI (DK).

To construct the DC data set, 10000 images are generated using Carla driving simulator [33] with the same settings as in [3]. DK data set contains images from KITTI odometry data set [34] consisting of real-world sequences captured in Karlsruhe, Germany. In our experiments the first seven sequences of KITTI odometry data set are utilized (ranging from 00 to 06), specifically images captured by the left camera, which means a total of 15223 images.

Since we use optical flow merely as an additional input rather than as an additional task for guiding distortion correction, for both data sets we generate in advance two optical flows with FlowNet 2.0 based on every three consequent images. The first optical flow corresponds to the optical flow between the second and first frames, while the second optical flow is generated based on the second and the third frame of a sequence, as demonstrated in Fig. 3, resulting in 9998 optical flow pairs in the case of DC data set and 15221 pairs in the case of DK data set.

We follow the same data set splitting and model training procedure as in [3]. The training of the networks is conducted on DC Train (8,000 images) for a total of 10 epochs, testing is conducted on both DC Test (2,000 images) and on sequence 00 of DK data set (4,539 images). The trained networks are further fine-tuned on

Table 2: Mean and Standard Deviation of Residual Distortion Vector Norms

Method	Fine-Tune	Test	$\mathcal{L}_r$	$\mathcal{L}_g$	Sem. Seg.	Opt. Flow	Mean (px)	SD (px)
Lőrincz et al. [3]	✗	DC Test	✓				2.26	1.49
				✓			2.25	1.59
			✓	✓			2.15	1.47
			✓		✓		1.98	1.40
				✓	✓		2.15	1.53
			✓	✓	✓		2.06	1.45
Ours			✓			✓	1.72	1.07
				✓		✓	1.52	0.62
			✓	✓		✓	1.43	0.71
Lőrincz et al. [3]	✗	DK 00	✓				1.75	1.10
				✓			2.28	1.52
			✓	✓			1.99	1.35
			✓		✓		1.65	1.09
				✓	✓		1.37	0.88
			✓	✓	✓		2.53	1.31
Ours			✓			✓	4.99	1.75
				✓		✓	2.65	1.09
			✓	✓		✓	3.47	1.18
Lőrincz et al. [3]	DK 01-06	DK 00	✓				1.24	0.72
				✓			1.33	0.67
			✓	✓			1.30	0.70
			✓		✓		1.30	0.69
				✓	✓		1.22	0.72
			✓	✓	✓		1.24	0.70
Ours			✓			✓	1.16	0.75
				✓		✓	1.06	0.71
			✓	✓		✓	1.06	0.68

sequences ranging from 01 to 06 of DK data set (10,684 images) for another 10 epochs and are also evaluated on sequence 00.

In our experiments we compared the performance of the proposed distortion correction network with the method presented in [3]. We also performed an ablation study to quantify the individual effects of integrating optical flow into the distortion correction process. Additionally, we examined the shortcomings of the proposed method and discussed further directions worth discovering to improve the current results.

5 Results

In Table 1 we present the mean and standard deviation of distortion vector norms in the two data sets providing a basis of comparison for estimating the performance of the variants of the proposed distortion correction method. The performance comparison of our solution and the method proposed by [3] is presented in Table 2 in terms of mean and standard deviation of residual distortion vector norms measured in pixels.

The effect of integrating optical flow into the distortion correction process is also quantified in Table 2 in all different settings: with- and without fine-tuning on DK data set and by minimizing separately or jointly the reconstruction and grid loss functions. Overall, one can see that our proposed distortion correction method using optical flow as an additional input performs the best in almost all cases. The best performing optical flow based model on DC Test reduces distortion vector norms to 1.43 ± 0.71 pixels, while on DK 00 this value is equal to 2.65 ± 1.09 pixels without fine-tuning and 1.06 ± 0.68 pixels with fine-tuning.

Similarly to the segmentation based system proposed by [3], we investigated the performance of the model in the case in which only the distorted and correct images are known, and we do not make use of the ground truth sampling grid during training, which is considered to be unknown (similar to the case of real-world distortions). In this case, the optical flow based model achieves 1.72 ± 1.07 pixels on DC Test, 4.99 ± 1.75 pixels on DK 00 without fine-tuning, and 1.16 ± 0.75 pixels with fine-tuning. One can see, that the model achieves comparable results without direct supervision from ground truth sampling grids, demonstrating the applicability of the network in the case of real-world distortions.

6 Discussion

The only exceptions where the optical flow based model does not outperform the method proposed in [3] are the tests conducted on real world images (DK 00) without fine-tuning the network on real-world images (sequences 01–06 of DK).

This phenomena is explained by the optical flow network used in our experiments producing substantially different optical flows based on synthetic images compared to the real-world images. Thus, our method needs to refine its distortion parameter estimations by fine-tuning it on the real-world images and corresponding optical flows.

Our experiments do not address certain possibilities, therefore further potential extensions need to be mentioned. First, the optical flow based distortion correction method processes image sequences, which have to be captured from multiple views, consequently, the camera has to be in motion in order to have optical flow vectors suitable for distortion correction.

In this case, it is possible to exploit the constant nature of the distortion caused by refractive surfaces by estimating the distortion parameters using images captured in adequate scenes, when optical flow vector norms are above a certain threshold, then calculating the mean sampling grid. Based on the calculated sampling grid, each new image can be sampled to correct the geometric distortions, even if the camera is not in motion.

Further experiments are also needed to achieve robustness regarding the method used for generating optical flow. Our results show that currently the proposed method is sensitive to the quality of the produced optical flow to a certain degree, without fine-tuning on real-world images its performance is inferior to the method proposed in [3]. This could be avoided by training with various optical flow methods, injecting variety in the data set. This was out of scope of our current experiments, however.

Another potential improvement would be achieved by extending the scope of this paper to real-world distortions by collecting real-world distorted and undistorted image pairs, enabling us to train and test the system in the presence of real-world distortions, in contrast to the data sets used in this work, which only contain images on which synthetic distortions were applied.

7 Conclusion

In this work we presented a deep learning based distortion correction method which is capable of correcting a wider range of geometric distortions compared to existing methods, based on three consequent RGB images using two corresponding optical

flows as additional inputs for guiding the process of distortion correction.

We also showed that the predictability and regularity of optical flow vector directions typical to autonomous driving scenarios can be exploited to assist the distortion correction method.

Our experimental results proved the hypothesis, that using optical flow as an additional input enhances the distortion correction method compared to employing an additional task such as semantic segmentation introduced in [3].

We detailed the disadvantages and constraints of the proposed system as well, and proposed solutions for each potential failure case. Addressing the mentioned problems would be the most important direction of development.

Data Availability: This work uses data sets derived from the public KITTI odometry dataset [34] and a data set generated using the open-source CARLA simulator [33]. To generate the distorted images, proprietary windshield distortion measurement data was used, and as a consequence the final derived data set can not be published.

References

[1] J. Rong, S. Huang, Z. Shang, and X. Ying, "Radial lens distortion correction using convolutional neural networks trained with synthesized images," in *Computer Vision – ACCV 2016*, S.-H. Lai, V. Lepetit, K. Nishino, and Y. Sato, Eds., Cham: Springer International Publishing, 2017, pp. 35–49, ISBN: 978-3-319-54187-7 (⇒ 63, 64).

[2] X. Yin, X. Wang, J. Yu, M. Zhang, P. Fua, and D. Tao, "Fisheyerecnet: A multi-context collaborative deep network for fisheye image rectification," *arXiv preprint arXiv:1804.04784*, 2018 (⇒ 63, 64).

[3] S.-B. Lőrincz, S. Pável, and L. Csató, "Single view distortion correction using semantic guidance," in *2019 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2019, pp. 1–6 (⇒ 63, 64, 66, 67, 69–74).

[4] A. W. Fitzgibbon, "Simultaneous linear estimation of multiple view geometry and lens distortion," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, IEEE, vol. 1, 2001, pp. I–I (⇒ 64).

[5] R. Hartley and S. B. Kang, "Parameter-free radial distortion correction with center of distortion estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 8, pp. 1309–1321, 2007 (⇒ 64).

[6] G. P. Stein, "Lens distortion calibration using point correspondences," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, IEEE, 1997, pp. 602–608 (⇒ 64).

[7] C. Bräuer-Burchardt and K. Voss, "Automatic lens distortion calibration using single views," in *Mustererkennung 2000*, Springer, 2000, pp. 187–194 (⇒ 64).

[8] B. Prescott and G. McLean, "Line-based correction of radial lens distortion," *Graphical Models and Image Processing*, vol. 59, no. 1, pp. 39–47, 1997 (⇒ 64).

[9] R. Tsai, "A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses," *IEEE Journal on Robotics and Automation*, vol. 3, no. 4, pp. 323–344, 1987 (⇒ 64).

[10] A. Wang, T. Qiu, and L. Shao, "A simple method of radial distortion correction with centre of distortion estimation," *Journal of Mathematical Imaging and Vision*, vol. 35, no. 3, pp. 165–172, 2009 (⇒ 64).

[11] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, 2000 (⇒ 64).

[12] D. C. Brown, "Decentering distortion of lenses," *Photogrammetric Engineering and Remote Sensing*, 1966 (⇒ 64).

[13] A. Agrawal, S. Ramalingam, Y. Taguchi, and V. Chari, "A theory of multi-layer flat refractive geometry," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2012, pp. 3346–3353 (⇒ 64).

[14] S. Morinaka, F. Sakaue, J. Sato, K. Ishimaru, and N. Kawasaki, "3d reconstruction under light ray distortion from parametric focal cameras," *Pattern Recognition Letters*, vol. 124, pp. 91–99, 2019 (⇒ 64).

[15] S. Pável, C. Sándor, and L. Csató, "Distortion estimation through explicit modeling of the refractive surface," in *International Conference on Artificial Neural Networks*, Springer, 2019, pp. 17–28 (⇒ 64).

[16] F. L. Bookstein, "Principal warps: Thin-plate splines and the decomposition of deformations," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 11, no. 6, pp. 567–585, 1989 (⇒ 64, 66, 67).

[17] Y. Mao and J. Gilles, "Non rigid geometric distortions correction-application to atmospheric turbulence stabilization," *Inverse Problems & Imaging*, vol. 6, no. 3, p. 531, 2012 (⇒ 64).

[18] X. Zhu and P. Milanfar, "Image reconstruction from videos distorted by atmospheric turbulence," in *Visual Information Processing and Communication*, International Society for Optics and Photonics, vol. 7543, 2010, 75430S (⇒ 64).

[19] B. Zhuang, Q.-H. Tran, G. H. Lee, L. F. Cheong, and M. Chandraker, "Degeneracy in self-calibration revisited and a deep learning solution for uncalibrated slam," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 3766–3773 (⇒ 64, 66).

[20] M. Jaderberg, K. Simonyan, A. Zisserman, *et al.*, "Spatial transformer networks," in *NIPS*, 2015, pp. 2017–2025 (⇒ 65, 69).

[21] R. Szeliski, "Prediction error as a quality metric for motion and stereo," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, IEEE, vol. 2, 1999, pp. 781–788 (⇒ 65).

[22] S. Tulsiani, R. Tucker, and N. Snavely, "Layer-structured 3d scene inference via view synthesis," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 302–317 (⇒ 65).

[23] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1851–1858 (⇒ 65).

[24] Z. Yin and J. Shi, "Geonet: Unsupervised learning of dense depth, optical flow and camera pose," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1983–1992 (⇒ 65).

[25] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *Asilomar Conference on Signals, Systems & Computers*, vol. 2, 2003, pp. 1398–1402 (⇒ 65, 69).

[26] H. C. Longuet-Higgins and K. Prazdny, "The interpretation of a moving retinal image," *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 208, no. 1173, pp. 385–397, 1980 (⇒ 65).

[27] A. Distante and C. Distante, *Handbook of Image Processing and Computer Vision: Volume 3: From Pattern to Object*. Springer Nature, 2020 (⇒ 65).

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016 (⇒ 69).

[29] O. Russakovsky, J. Deng, H. Su, *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, Dec. 2015, ISSN: 1573-1405. DOI: 10.1007/s11263-015-0816-y. [Online]. Available: https://doi.org/10.1007/s11263-015-0816-y (⇒ 69).

[30] A. Paszke, S. Gross, S. Chintala, *et al.*, "Automatic differentiation in pytorch," 2017 (⇒ 69).

[31] F. Reda, R. Pottorff, J. Barker, and B. Catanzaro, *Flownet2-pytorch: Pytorch implementation of flownet 2.0: Evolution of optical flow estimation with deep networks*, https://github.com/NVIDIA/flownet2-pytorch, 2017 (⇒ 69).

[32] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017. [Online]. Available: http://lmb.informatik.uni-freiburg.de//Publications/2017/IMKDB17 (⇒ 69).

[33] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16 (⇒ 70, 74).

[34] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012 (⇒ 70, 74).

Received: 19.06.2024; *Revised: 30.06.2024*; *Accepted: 07.07.2024*

Acta Univ. Sapientiae, Informatica, **16, 1** (2024) 78–104

DOI: 10.47745/ausi-2024-0006

Vertex eccentric connectivity index of chemical graphs obtained from (α)bis(pyridine)-cobalt(III)chloride

Abhinaya B M
Department of Mathematics, CHRIST (Deemed to be University), Bangalore-560029, Karnataka, India
abhinaya.bm@res.christuniversity.in
0009-0009-1297-6119

Adhithya R
Department of Mathematics, CHRIST (Deemed to be University), Bangalore-560029, Karnataka, India
adhithya.ravi@science.christuniversity.in

Tabitha A M
Department of Mathematics, CHRIST (Deemed to be University), Bangalore-560029, Karnataka, India
tabitha.rajashekar@christuniversity.in
0000-0002-2960-9392

Abstract. Let $G = (V, E)$ be a graph. Topological indices are numerical descriptors that provide information about the molecular structure based on the structural properties of the corresponding molecular graph. Among the various topological indices available for graphs, eccentricity-based indices such as vertex eccentric and modified vertex eccentric connectivity indices are particularly significant for QSAR/QSPR studies. In this paper, these indices are computed for self-centered graphs, regular graphs, and graphs obtained by graph operations such as join and Cartesian product. Further, we examine these indices for a molecular graph of (α)bis(pyridine)-cobalt(III)chloride.

Key words and phrases: Vertex eccentric connectivity index, Modified vertex eccentric connectivity index, Topological indices, Chemical graphs, Graph operations.

1 Introduction

Graph theory is a branch of mathematics that focuses on the analysis of networks. It is a highly significant discipline with several applications in computer science,

chemistry, and social sciences. Chemical graph theory, in particular, is essential in analysing molecules, topological indices, isomerism, and their practical consequences in quantum chemistry and stereochemistry [1]. The effect of stereocenters on topological values has been studied in [2].

A chemical structure can be visualised as a graph, with the atoms in a molecule represented as vertices and the bonds connecting them as edges [3]. In QSAR (Quantitative Structure-Activity Relationships) and QSPR (Quantitative Structure-Property Relationship), topological indices of chemical structures are used to create connections between chemical structures and compare their characteristics or reactivity. Topological indices serve as molecular descriptors to predict various physicochemical parameters such as boiling point, enthalpy of vaporisation, and stability [4]. A QSPR model is developed using linear regression to predict properties such as molecular weight, molar volume, flash point and boiling point [5]. Additionally, QSPR analysis of nonsteroidal anti-inflammatory drugs using topological indices demonstrates a strong correlation between these indices and the physical properties of the chemical compound [6]. In [7], the atom bond connectivity index and geometric arithmetic index of oxide and chain silicate are computed. Ashrafi et al. have investigated the eccentric connectivity index for nanotubes and nanotorus in [8]. The super-augmented eccentric connectivity index was computed for 6-arylbenzonitriles and models were developed for anti-HIV-1 activity prediction in [9]. Topological indices are also used to investigate the chemical structure of anti-heart attack drugs [10].

2 Literature Review

After the Wiener index was introduced by Harry Wiener in 1947 [11], several other topological indices were developed. While indices are based on vertex or edge degrees, others focus on distances between vertices or edges. Here, we investigate topological indices called the vertex eccentric connectivity indices. The vertex eccentric connectivity index was introduced by Sharma et al. in 1997 [12], and several authors have conducted further research to explore and investigate this index. In [13], the relationship of pyrazole carboxylic acid hydrazide analogues with the Wiener index and eccentric connectivity index is investigated. Anti-HIV activity of 2, 3-diaryl-1, 3-thiazolidin-4-one derivative was investigated using this index in [14]. Later, M.J. Morgan et al. gave bounds for this index [15]. Vertex eccentric connectivity index was also computed for the diamond graph, and Dutch windmill graph in [16] and [17], respectively.

The eccentric connectivity index was studied for composite graphs by Tomislav

et al. [18]. In recent studies, vertex eccentric connectivity index was calculated for k-uniform hyper-cacti [19], benzenoid hourglass network [20], and benzenoid structure [21]. In this article, we present the vertex and modified vertex eccentric connectivity index of (α)bis(pyridine)-cobalt(III)chloride.

3 Terminologies

Let $G = (V, E)$ be a graph of order n and size m. For any vertex $v \in V(G)$, the open neighbourhood of v is defined as $N(v) = \{u \in V(G) | uv \in E(G)\}$. The degree of v is the cardinality of $N(v)$ and is denoted by $deg(v)$ or specifically $deg_G(v)$ is the degree of v in G. The distance $d(u, v)$ is the shortest distance between vertices u and v. The eccentricity of a vertex v is the largest distance between v and any other vertex of G, and it is denoted by $\epsilon(v)$ or specifically $\epsilon_G(v)$ is the eccentricity of v in G. The join of two graphs, G_1 and G_2, is a graph obtained by adding edges from each vertex of G_1 to every vertex of G_2, and it is denoted by $G_1 \vee G_2$. Cartesian product of two graphs G_1 and G_2, denoted by $G_1 \square G_2$, with vertex set $V(G_1) \times V(G_2)$ and two vertices, (u_1, v_1) and (u_2, v_2), of $G_1 \square G_2$ are adjacent if and only if $u_1 = u_2$ and $v_1v_2 \in E(G_2)$ or $v_1 = v_2$ and $u_1u_2 \in E(G_1)$, where $u_1, u_2 \in V(G_1)$ and $v_1, v_2 \in V(G_2)$.

The vertex eccentric connectivity index of graph G, denoted by $\xi^c(G)$, is defined as

$$\xi^c(G) = \sum_{v \in V(G)} \epsilon(v) deg(v).$$

The modified vertex eccentric connectivity index, denoted by $\Lambda^c(G)$, was defined by Ashrafi et al. in 2011 [22] and is given as

$$\Lambda^c(G) = \sum_{v \in V(G)} \epsilon(v)\delta(v), \text{ where } \delta(v) = \sum_{u \in N(v)} deg(u).$$

4 Results

4.1 Vertex eccentric connectivity index of certain graphs

Theorem 4.1. *Let G be a k-regular self-centered graph. Then, $\xi^c(G) = nk\mu$ and $\Lambda^c(G) = nk^2\mu$, where μ is the diameter of G.*

Proof. Consider a k-regular self-centered graph G of order n and eccentricity μ $\forall v \in V(G)$. Then, the vertex eccentric connectivity index of G is given as

$$\xi^c(G) = \sum_{v \in V(G)} \epsilon(v) deg(v) = \sum_{v \in V(G)} \mu k = nk\mu.$$

We have $\delta(v) = \sum_{u \in N(v)} deg(u) = \sum_{u \in N(v)} k = k^2$. The modified vertex eccentric connectivity index of a graph G is given as

$$\Lambda^c(G) = \sum_{v \in V(G)} \epsilon(v)\delta(v) = \sum_{v \in V(G)} \mu k^2 = nk^2\mu.$$

□

Theorem 4.2. *The vertex eccentric connectivity index and modified vertex eccentric connectivity index of a k-regular graph G of order n satisfy $\xi^c(G) \geq nk$ and $\Lambda^c(G) \geq nk^2$, respectively.*

Proof. Consider a k-regular graph G. Then $deg(v) = k \forall v \in V(G)$ and $\epsilon(v) \geq 1$. Then, the vertex eccentric connectivity index of G satisfies

$$\xi^c(G) = \sum_{v \in V(G)} \epsilon(v) deg(v) \geq \sum_{v \in V(G)} k \geq nk.$$

We have $\delta(v) = \sum_{u \in N(v)} deg(u) = \sum_{u \in N(v)} k = k^2$. Further, the modified vertex eccentric connectivity index of a graph G satisfies

$$\Lambda^c(G) = \sum_{v \in V(G)} \epsilon(v)\delta(v) \geq \sum_{v \in V(G)} k^2 \geq nk^2.$$

□

4.2 Vertex eccentric connectivity index for graphs obtained by certain graph operations

Theorem 4.3. *Let G_1 and G_2 be two graphs of orders n_1 and n_2, respectively, and sizes m_1 and m_2, respectively. Let k_1 and k_2 be the number of universal vertices in G_1 and G_2, respectively. Then,*

$$\xi^c(G_1 \vee G_2) = (k_1 + k_2)(1 - n_1 - n_2) + 4n_1n_2 + 4(m_1 + m_2).$$

Proof. Consider G_1 and G_2 to be two connected graphs with $|V(G_1)| = n_1$, $|V(G_2)| = n_2$, $|E(G_1)| = m_1$ and $|E(G_2)| = m_2$. Let k_1 and k_2 be the number of universal vertices in G_1 and G_2, respectively. For every vertex $v \in V(G_1)$, $deg_{G_1 \vee G_2}(v) = deg_{G_1}(v) + n_2$ in $G_1 \vee G_2$ and for every vertex $v \in V(G_2)$, $deg_{G_1 \vee G_2}(v) = deg_{G_2}(w) + n_1$ in $G_1 \vee G_2$. Further, we note that the eccentricity of every vertex in $G_1 \vee G_2$ is at most 2. If a vertex is a universal vertex in G_1 or G_2, then its eccentricity is 1, and all other vertices will have eccentricity 2 in $G_1 \vee G_2$.

By the handshaking lemma, we have

$$\begin{aligned} 2m_1 &= \sum_{v\in V(G_1)} deg_{G_1}(v) \\ &= \sum_{\substack{v\in V(G_1)\\ deg_{G_1}(v)=n_1-1}} deg_{G_1}(v) + \sum_{\substack{v\in V(G_1)\\ deg_{G_1}(v)<n_1-1}} deg_{G_1}(v) \\ &= k_1(n_1-1) + \sum_{\substack{v\in V(G_1)\\ deg_{G_1}(v)<n_1-1}} deg_{G_1}(v). \end{aligned}$$

Hence we have

$$\sum_{\substack{v\in V(G_1)\\ deg_{G_1}(v)<n_1-1}} deg_{G_1}(v) = 2m_1 - k_1n_1 + k_1.$$

Similarly,

$$\sum_{\substack{v\in V(G_2)\\ deg_{G_2}(v)<n_2-1}} deg_{G_2}(v) = 2m_2 - k_2n_2 + k_2.$$

Therefore, the vertex eccentric connectivity index is given as

$$\begin{aligned} \xi^c(G_1\vee G_2) &= \sum_{v\in V(G_1\vee G_2)} \epsilon_{G_1\vee G_2}(v) deg_{G_1\vee G_2}(v) \\ &= \sum_{v\in V(G_1)} \epsilon_{G_1\vee G_2}(v) deg_{G_1\vee G_2}(v) + \sum_{v\in V(G_2)} \epsilon_{G_1\vee G_2}(v) deg_{G_1\vee G_2}(v) \\ &= \sum_{\substack{v\in V(G_1)\\ deg_{G_1}(v)=n_1-1}} \epsilon_{G_1\vee G_2}(v) deg_{G_1\vee G_2}(v) \\ &\quad + \sum_{\substack{v\in V(G_1)\\ deg_{G_1}(v)<n_1-1}} \epsilon_{G_1\vee G_2}(v) deg_{G_1\vee G_2}(v) \\ &\quad + \sum_{\substack{v\in V(G_2)\\ deg_{G_2}(v)=n_2-1}} \epsilon_{G_1\vee G_2}(v) deg_{G_1\vee G_2}(v) \\ &\quad + \sum_{\substack{v\in V(G_2)\\ deg_{G_2}(v)<n_2-1}} \epsilon_{G_1\vee G_2}(v) deg_{G_1\vee G_2}(v) \\ &= \sum_{\substack{v\in V(G_1)\\ deg_{G_1}(v)=n_1-1}} (deg_{G_1}(v)+n_2) \end{aligned}$$

$$
\begin{aligned}
&+ \sum_{\substack{v\in V(G_1)\\ deg_{G_1}(v)<n_1-1}} 2(deg_{G_1}(v)+n_2) + \sum_{\substack{v\in V(G_2)\\ deg_{G_2}(v)=n_2-1}} (deg_{G_2}(v)+n_1)\\
&+ \sum_{\substack{v\in V(G_2)\\ deg_{G_2}(v)<n_2-1}} (deg_{G_2}(v)+n_1)\\
&= \sum_{\substack{v\in V(G_1)\\ deg_{G_1}(v)=n_1-1}} (n_1-1+n_2) + \sum_{\substack{v\in V(G_1)\\ deg_{G_1}(v)<n_1-1}} 2deg_{G_1}(v)\\
&+ \sum_{\substack{v\in V(G_1)\\ deg_{G_1}(v)<n_1-1}} n_2 + \sum_{\substack{v\in V(G_2)\\ deg_{G_2}(v)=n_2-1}} (n_2-1+n_1)\\
&+ \sum_{\substack{v\in V(G_2)\\ deg_{G_2}(v)<n_2-1}} deg_{G_2}(v) + \sum_{\substack{v\in V(G_2)\\ deg_{G_2}(v)<n_2-1}} n_1\\
&= k_1(n_1-1+n_2) + 2(2m_1 - k_1n_1 + k_1) + 2(n_1-k_1)n_2\\
&\quad + k_2(n_2-1+n_1) + 2(2m_2 - k_2n_2 + k_2) + 2(n_2-k_2)n_1\\
&= (k_1+k_2)(1-n_1-n_2) + 4n_1n_2 + 4(m_1+m_2).
\end{aligned}
$$

Hence $\xi^c(G_1 \vee G_2) = (k_1+k_2)(1-n_1-n_2) + 4n_1n_2 + 4(m_1+m_2)$. □

We partition the vertex set of a graph into sets of vertices. All the vertices with the same eccentricity and degree belong to the same set.

Theorem 4.4. *The modified vertex eccentric connectivity index of $P_n \square C_m$, where $n, m \geq 3$ is given by*

$$
\Lambda^c(P_n \square C_m) = \begin{cases} m(-30n - 7m + 12n^2 + 8mn + 23), & \text{if } n \text{ is even and } m \text{ is odd,}\\ m(-22n - 7m + 12n^2 + 8mn + 12), & \text{if } n \text{ is odd and } m \text{ is even,}\\ m(-22n - 7m + 12n^2 + 8mn + 16), & \text{if } n \text{ and } m \text{ are even,}\\ m(-30n - 7m + 12n^2 + 8mn + 19), & \text{if } n \text{ and } m \text{ are odd.} \end{cases}
$$

Proof. Let $G = P_n \square C_m$. These are the four cases depending on n and m.
Case 1: Consider n to be even and m to be odd.
As seen in Table 1, we have $\frac{n}{2}$ sets of vertices in G based on the degrees and eccentricities of vertices. The modified vertex eccentric connectivity index of a graph G is given as

$$
\Lambda^c(G) = \sum_{v\in V(G)} \epsilon(v)\delta(v)
$$

$$
\begin{aligned}
&= 20m\left(\frac{2n+m-3}{2}\right) + 30m\left(\frac{2n+m-3}{2}-1\right) \\
&\quad + 32m\left[\left(\frac{2n+m-3}{2}-2\right) + \left(\frac{2n+m-3}{2}-3\right)\right. \\
&\qquad \left. + \ldots + \left(\frac{2n+m-3}{2} - \frac{n-2}{2}\right)\right] \\
&= 10m(2n+m-3) + 15m(2n+m-5) + 32m\sum_{k=1}^{\frac{n-4}{2}} \frac{2n+m-3}{2} - k - 1 \\
&= -30mn - 7m^2 + 23m + 12mn^2 + 8m^2.
\end{aligned}
$$

Table 1: Sets of vertices of $P_n \square C_m$ when n is even and m is odd.

S. No of sets of vertices	Frequency	$\epsilon(v)$	$deg(v)$	$\delta(v)$
1	2m	$\frac{2n+m-3}{2}$	3	10
2	2m	$\frac{2n+m-3}{2} - 1$	4	15
3	2m	$\frac{2n+m-3}{2} - 2$	4	16
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\frac{n}{2}$	2m	$\frac{2n+m-3}{2} - \frac{n}{2} + 1$	4	16

Case 2: Consider n to be odd and m to be even.
As seen in Table 2, we have $\frac{n+1}{2}$ sets of vertices in G based on the degrees and eccentricities of vertices. The modified vertex eccentric connectivity index of a graph G is given as

$$
\begin{aligned}
\Lambda^c(G) &= \sum_{v \in V(G)} \epsilon(v)\delta(v), \\
&= 20m\left(\frac{2n+m-2}{2}\right) + 30m\left(\frac{2n+m-2}{2}-1\right) \\
&\quad + 32m\left[\left(\frac{2n+m-2}{2}-2\right) + \left(\frac{2n+m-2}{2}-3\right)\right.
\end{aligned}
$$

$$+\ldots+\left(\frac{2n+m-2}{2}-\frac{n-3}{2}\right)\Bigg]+16m\left(\frac{2n+m-2}{2}-\frac{n-1}{2}\right)$$
$$=10m(2n+m-2)+15m(2n+m-4)+8m(n+m-1)$$
$$+32m\sum_{k=1}^{\frac{n-5}{2}}\frac{2n+m-3}{2}-k-1$$
$$=-22mn-7m^2+12m+12mn^2+8m^2n.$$

Table 2: Sets of vertices of $P_n\square C_m$ when n is odd and m is even.

S. No of sets of vertices	Frequency	$\epsilon(v)$	$deg(v)$	$\delta(v)$
1	$2m$	$\frac{2n+m-2}{2}$	3	10
2	$2m$	$\frac{2n+m-2}{2}-1$	4	15
3	$2m$	$\frac{2n+m-2}{2}-2$	4	16
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\frac{n-1}{2}$	$2m$	$\frac{2n+m-2}{2}-\frac{n-3}{2}$	4	16
$\frac{n+1}{2}$	m	$\frac{2n+m-2}{2}-\frac{n-1}{2}$	4	16

Case 3: Consider n and m both to be even.
As seen in Table 3, we have $\frac{n}{2}$ sets of vertices in G based on the degrees and eccentricities of vertices. The modified vertex eccentric connectivity index of a graph G is given as

$$\Lambda^c(G)=\sum_{v\in V(G)}\epsilon(v)\delta(v),$$
$$=20m\left(\frac{2n+m-2}{2}\right)+30m\left(\frac{2n+m-2}{2}-1\right)$$
$$+32m\left[\left(\frac{2n+m-2}{2}-2\right)+\left(\frac{2n+m-2}{2}-3\right)\right.$$
$$\left.+\ldots+\left(\frac{2n+m-2}{2}-\frac{n-2}{2}\right)\right]$$

$$= 10m(2n+m-2) + 15m(2n+m-4) + 32m \sum_{k=1}^{\frac{n-4}{2}} \frac{2n+m-2}{2} - k - 1$$
$$= -22mn - 7m^2 + 16m + 12n^2m + 8m^2n.$$

Table 3: Sets of vertices of $P_n \square C_m$ when n and m are even.

S. No of sets of vertices	Frequency	$\epsilon(v)$	$deg(v)$	$\delta(v)$
1	$2m$	$\frac{2n+m-2}{2}$	3	10
2	$2m$	$\frac{2n+m-2}{2} - 1$	4	15
3	$2m$	$\frac{2n+m-2}{2} - 2$	4	16
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\frac{n}{2}$	$2m$	$\frac{2n+m-2}{2} - \frac{n-2}{2}$	4	16

Case 4: Consider both n and m to be odd.
As seen in Table 4, we have $\frac{n+1}{2}$ sets of vertices in G based on the degrees and eccentricity of vertices. The modified vertex eccentric connectivity index of a graph G is given as

$$\begin{aligned}
\Lambda^c(G) &= \sum_{v \in V(G)} \epsilon(v)\delta(v) \\
&= 20m\left(\frac{2n+m-3}{2}\right) + 30m\left(\frac{2n+m-3}{2} - 1\right) \\
&\quad + 32m\left[\left(\frac{2n+m-3}{2} - 2\right) + \left(\frac{2n+m-3}{2} - 3\right)\right. \\
&\qquad \left. + \ldots + \left(\frac{2n+m-3}{2} - \frac{n-3}{2}\right)\right] \\
&\quad + 16m\left(\frac{2n+m-3}{2} - \frac{n-1}{2}\right) \\
&= 10m(2n+m-3) + 15m(2n+m-5) + 8m(n+m-2)
\end{aligned}$$

$$+ 32m \sum_{k=1}^{\frac{n-5}{2}} \frac{2n+m-3}{2} - k - 1,$$

$$\Lambda^c(G) = -30nm - 7m^2 + 19m + 12n^2m + 8m^2n.$$

Table 4: Sets of vertices of $P_n \square C_m$ when n and m both are odd.

S. No of sets of vertices	Frequency	$\epsilon(v)$	$deg(v)$	$\delta(v)$
1	$2m$	$\frac{2n+m-3}{2}$	3	10
2	$2m$	$\frac{2n+m-3}{2} - 1$	4	15
3	$2m$	$\frac{2n+m-3}{2} - 2$	4	16
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\frac{n-1}{2}$	$2m$	$\frac{2n+m-3}{2} - \frac{n-3}{2}$	4	16
$\frac{n+1}{2}$	m	$\frac{2n+m-3}{2} - \frac{n-1}{2}$	4	16

□

Corollary 4.4.1. *The modified vertex eccentric connectivity index of $P_n \square C_m$ is given by*

$$\Lambda^c(P_n \square C_n) = \begin{cases} 20n^3 - 29n^2 + 16n, & \text{if } n \text{ is even,} \\ 20n^3 - 37n^2 + 19n, & \text{if } n \text{ is odd.} \end{cases}$$

Theorem 4.5. *The modified vertex eccentric connectivity index of $C_n \square C_m$ is given by*

$$\Lambda^c(C_n \square C_m) = \begin{cases} 8mn(m+n), & \text{if } n \text{ and } m \text{ are even,} \\ 8mn(m+n-2), & \text{if } n \text{ and } m \text{ are odd,} \\ 8mn(m+n-1), & \text{either one of } n \text{ and } m \text{ are even.} \end{cases}$$

Proof. These are the three cases depending on the values of m and n.
Case 1: Consider both n and m to be even.
All vertices of $C_n \square C_m$ has the same eccentricity $\frac{m+n}{2}$, and are of same degree 4.

Since a vertex of the graph has four neighbours of degree 4, $\delta(v) = 16\ \forall v \in V(G)$. The modified vertex eccentric connectivity index of a graph G is given as

$$\Lambda^c(G) = \sum_{v \in V(G)} \epsilon(v)\delta(v) = \sum_{v \in V(G)} 16\left(\frac{m+n}{2}\right).$$

Since the order of G is mn, we have

$$\Lambda^c(G) = 16mn\left(\frac{m+n}{2}\right) = 8mn(m+n).$$

Hence,

$$\Lambda^c(G) = 8mn(m+n).$$

Case 2: Consider both n and m to be odd.
All vertices of $C_n \square\, C_m$ have the same eccentricity $\frac{n+m-2}{2}$ and are of same degree 4. Since a vertex of the graph has four neighbours of degree 4, $\delta(v) = 16\ \forall v \in V(G)$. The modified vertex eccentric connectivity index of a graph G is given as

$$\Lambda^c(G) = \sum_{v \in V(G)} \epsilon(v)\delta(v) = \sum_{v \in V(G)} 16\left(\frac{n+m-2}{2}\right).$$

Since the order of G is mn, we have

$$\Lambda^c(G) = 16mn\left(\frac{n+m-2}{2}\right) = 8mn(n+m-2).$$

Hence,

$$\Lambda^c(G) = 8mn(n+m-2).$$

Case 3: Consider either n or m to be even. Without loss of generality, let n be even and m be odd.
All vertices of $C_n \square\, C_m$ have the same eccentricity $\frac{n+m-1}{2}$ and are of same degree 4. Since a vertex of the graph has four neighbours of degree 4, $\delta(v) = 16\ \forall v \in V(G)$. The modified vertex eccentric connectivity index of a graph G is given as

$$\Lambda^c(G) = \sum_{v \in V(G)} \epsilon(v)\delta(v) = \sum_{v \in V(G)} 16\left(\frac{n+m-1}{2}\right).$$

Since the order of G is mn, we have

$$\Lambda^c(G) = 16mn\left(\frac{n+m-1}{2}\right) = 8mn(n+m-2).$$

□

Corollary 4.5.1. *The modified vertex eccentric connectivity index of $C_n \square C_n$ is given by*

$$\Lambda^c(C_n \square C_n) = \begin{cases} 16n^3, & \text{if } n \text{ and } m \text{ are even,} \\ 16n^2(n-1), & \text{if } n \text{ and } m \text{ are odd.} \end{cases}$$

4.3 Vertex eccentric connectivity index of chemical graph of (α)bis(pyridine)-cobalt(III)chloride

We consider the compound bis(pyridine)-cobalt(III)chloride with the chemical formula $Co(py)_2Cl_2$, which exists in two forms: α form, which is pink in color and a blue-colored β form[23]. In this paper, we are focusing on the α form. As shown in Figure 1, it has a polymeric chain-like structure. It is stable because chlorine atoms and pyridine rings are packed within a single chain[24]. Due to the repetitive nature of the α form, we can construct a graph that represents this compound. The graph obtained is a multigraph. We denote this family of graphs as CC_t, where t is the number of units. The simplest recorded structure of α $Co(py)_2Cl_2$ has $t = 3$ as seen in Figure 2.

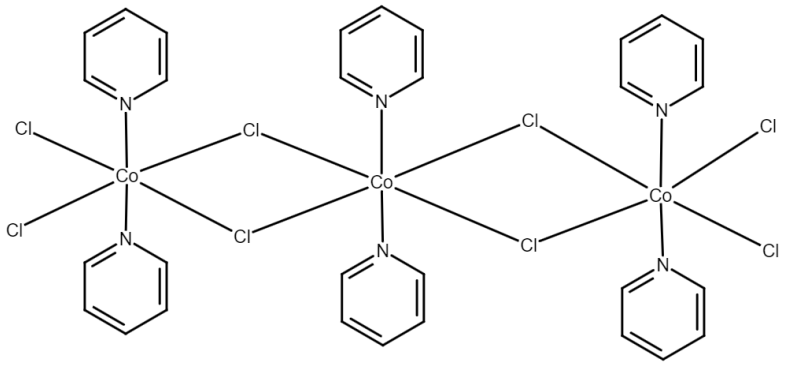


Figure 1: (α)bis(pyridine)-cobalt(III)chloride [$Co(py)_2Cl_2$].

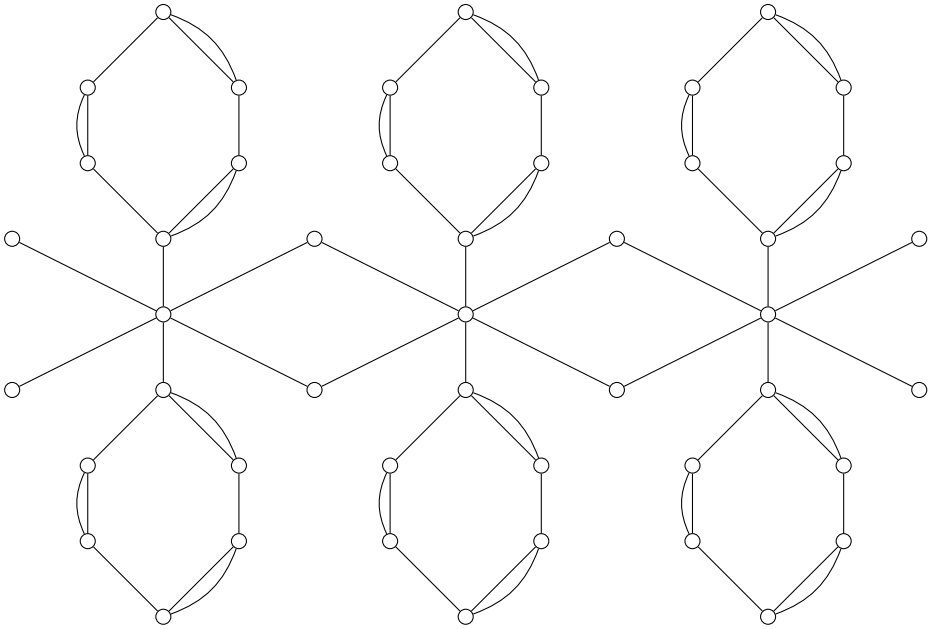


Figure 2: Graph representing $Co(py)_2Cl_2$.

Theorem 4.6. *The vertex eccentric connectivity index of CC_t is given by*

$$\xi^c(CC_t) = \begin{cases} 72t^2 + 236t, & \text{if } t \text{ is even,} \\ 72t^2 + 236t - 20, & \text{otherwise.} \end{cases}$$

Proof. We categorise the vertices based on their degrees. Then, we partition them into sets of vertices depending on their eccentricities. That is, the vertices in one particular set have the same eccentricities. We have two cases depending on t.
Case 1: Consider t to be even.
As seen in Tables 5, 7, 9, 11, and 13, there are $\frac{t}{2}$ sets of vertices of degree 6, $\frac{t}{2}$ sets of vertices of degree 4, $t+1$ sets of vertices of degree 3, $\frac{t}{2}$ sets of vertices of degree 2, and one set of vertices of degree 1. The vertex eccentric connectivity index of a graph CC_t is given by

$$\begin{aligned} \xi^c(CC_t) &= \sum_{v \in V(CC_t)} \epsilon(v) deg(v) \\ &= \sum_{\substack{v \in V(CC_t) \\ deg(v)=1}} \epsilon(v) + \sum_{\substack{v \in V(CC_t) \\ deg(v)=2}} 2\epsilon(v) + \sum_{\substack{v \in V(CC_t) \\ deg(v)=3}} 3\epsilon(v) \end{aligned}$$

$$+ \sum_{\substack{v \in V(CC_t) \\ deg(v)=4}} 4\epsilon(v) + \sum_{\substack{v \in V(CC_t) \\ deg(v)=6}} 6\epsilon(v).$$

From Table 7, we have

$$\sum_{\substack{v \in V(CC_t) \\ deg(v)=1}} \epsilon(v) = \sum_{\substack{v \in V(CC_t) \\ deg(v)=1}} (2t+3) = 4(2t+3) = 8t+12.$$

From Table 11, we have

$$\begin{aligned}\sum_{\substack{v \in V(CC_t) \\ deg(v)=2}} 2\epsilon(v) &= 8\left[(2t+6-5)+(2t+6-7)+\ldots+(2t+6-t-1)\right] \\ &\quad + 4(2t+6-t-3) \\ &= 4t+12+\sum_{k=1}^{\frac{t-2}{2}} 2t+6-2k-3 = 6t^2+4t-12.\end{aligned}$$

From Table 5, we have

$$\begin{aligned}\sum_{\substack{v \in V(CC_t) \\ deg(v)=3}} 3\epsilon &= 12(2t+6)+24\left[(2t+6-1)+(2t+6-3)+\ldots+2t+6-(t-1)\right] \\ &\quad + 36\left[(2t+6-2)+(2t+6-4)+\ldots+(2t+6-(t-2))\right] \\ &\quad + 24(2t+6-t) \\ &= 24t+72+24(2t+6-1)+24(t+6)+24\sum_{k=1}^{\frac{t-2}{2}} 2t+6-(2k+1) \\ &\quad + 36\sum_{k=1}^{\frac{t-2}{2}} 2t+6-2k \\ &= 45t^2+174t.\end{aligned}$$

From Table 9, we have

$$\sum_{\substack{v \in V(CC_t) \\ deg(v)=4}} 4\epsilon(v) = 16\left[(2t+6-3)+(2t+6-5)+\ldots+(2t+6-(t+1))\right]$$

$$= 16 \sum_{k=1}^{\frac{t}{2}} 2t + 6 - 2k - 1 = 12t^2 + 32t.$$

From Table 13, we have

$$\sum_{\substack{v \in V(CC_t) \\ deg(v)=6}} 6\epsilon(v) = 12\left[(2t + 6 - 4) + (2t + 6 - 6) + \ldots + (2t + 6 - (t + 2))\right]$$

$$= 12 \sum_{k=1}^{\frac{t}{2}} 2t + 6 - 2k - 2 = 9t^2 + 18t.$$

Substituting the above equations in equation (1), we get

$$\xi^c(CC_t) = 72t^2 + 236t, \text{ when } t \text{ is even.}$$

Case 2: Consider t to be odd.
As seen in Tables 6, 8, 10, 12, and 14, there are $\frac{t-1}{2}$ sets of vertices of degree 6, $\frac{t+1}{2}$ sets of vertices of degree 4, $t + 2$ sets of vertices of degree 3, $\frac{t-1}{2}$ sets of vertices of degree 2, and one set of vertices of degree 1. The vertex eccentric connectivity index of a graph CC_t is given by

$$\begin{aligned}\xi^c(CC_t) &= \sum_{v \in V(CC_t)} \epsilon(v) deg(v) \\ &= \sum_{\substack{v \in V(CC_t) \\ deg(v)=1}} \epsilon(v) + \sum_{\substack{v \in V(CC_t) \\ deg(v)=2}} 2\epsilon(v) + \sum_{\substack{v \in V(CC_t) \\ deg(v)=3}} 3\epsilon(v) + \sum_{\substack{v \in V(CC_t) \\ deg(v)=4}} 4\epsilon(v) \\ &\quad + \sum_{\substack{v \in V(CC_t) \\ deg(v)=6}} 6\epsilon(v).\end{aligned}$$

From Table 8, we have

$$\sum_{\substack{v \in V(CC_t) \\ deg(v)=1}} \epsilon(v) = \sum_{\substack{v \in V(CC_t) \\ deg(v)=1}} (2t + 3) = 4(2t + 3) = 8t + 12.$$

From Table 12, we have

$$\sum_{\substack{v \in V(CC_t) \\ deg(v)=2}} 2\epsilon(v) = 8\left[(2t + 6 - 5) + (2t + 6 - 7) + \ldots + (2t + 6 - t - 2)\right]$$

$$= \sum_{k=1}^{\frac{t-1}{2}} 2t + 6 - 2k - 3 = 6t^2 + 4t - 10.$$

From Table 6, we have

$$\begin{aligned}\sum_{\substack{v \in V(CC_t) \\ deg(v)=3}} 3\epsilon(v) &= 12(2t+6) + 24\left[(2t+6-1) + (2t+6-3) + \ldots + 2t + 6 - (t-2)\right] \\ &\quad + 36\left[(2t+6-2) + (2t+6-4) + \ldots + (2t+6-(t-3))\right] \\ &\quad + 30(t+7) + 12(2t+11) \\ &= 24t + 72 + 24(2t+6-1) + 30(t+7) + 12(2t+11) \\ &\quad + 24\sum_{k=1}^{\frac{t-3}{2}} 2t + 6 - (2k+1) + 36\sum_{k=1}^{\frac{t-3}{2}} 2t + 6 - 2k \\ &= 45t^2 + 174t - 15.\end{aligned}$$

From Table 10, we have

$$\begin{aligned}\sum_{\substack{v \in V(CC_t) \\ deg(v)=4}} 4\epsilon(v) &= 16\left[(2t+6-3) + (2t+6-5) + \ldots + (2t+6-t))\right] \\ &\quad + 8(2t+6-t-2) \\ &= \sum_{k=1}^{\frac{t-1}{2}} 2t + 6 - 2k - 1 + 8(2t+6-t-2) = 12t^2 + 32t - 4.\end{aligned}$$

From Table 14, we have

$$\begin{aligned}\sum_{\substack{v \in V(CC_t) \\ deg(v)=6}} 6\epsilon(v) &= 12\left[(2t+6-4) + (2t+6-6) + \ldots + (2t+6-(t+1))\right] \\ &\quad + 6(2t+6-t-3) \\ &= 6(t+3) + 12\sum_{k=1}^{\frac{t-1}{2}} 2t + 6 - 2k - 2 = 9t^2 + 18t - 3.\end{aligned}$$

Substituting the above equations in equation (2), we get

$$\xi^c(CC_t) = 72t^2 + 236t - 20, \text{ when } t \text{ is odd.}$$

□

Theorem 4.7. *The modified vertex eccentric connectivity of CC_t is given by*

$$\xi^c(CC_t) = \begin{cases} 192t^2 + 576t - 8, & \text{if } t \text{ is even,} \\ 192t^2 + 576t - 48, & \text{otherwise.} \end{cases}$$

Proof. We categorise the vertices based on their degrees. Then, we partition them into sets of vertices depending on their eccentricities. That is, the vertices in one particular set have the same eccentricities. We have two cases depending on t.
Case 1: Consider t to be even.
As seen in Tables 5, 7, 9, 11, and 13, there are $\frac{t}{2}$ sets of vertices of degree 6, $\frac{t}{2}$ sets of vertices of degree 4, $t+1$ sets of vertices of degree 3, $\frac{t}{2}$ sets of vertices of degree 2, and one set of vertices of degree 1. The vertex eccentric connectivity index of a graph CC_t is given by

$$\Lambda^c(CC_t) = \sum_{v \in V(CC_t)} \epsilon(v)\delta(v).$$

From Table 7, we have

$$\sum_{\substack{v \in V(CC_t) \\ deg(v)=1}} \epsilon(v)\delta(v) = \sum_{\substack{v \in V(CC_t) \\ deg(v)=1}} (2t+3) = 24(2t+3) = 48t + 72.$$

From Table 11, we have

$$\begin{aligned} \sum_{\substack{v \in V(CC_t) \\ deg(v)=2}} \epsilon(v)\delta(v) &= 48\left[(2t+6-5) + (2t+6-7) + \ldots + (2t+6-t-1)\right] \\ &\quad + 24(2t+6-t-3) \\ &= 24t + 72 + \sum_{k=1}^{\frac{t-2}{2}} 2t + 6 - 2k - 3 = 36t^2 + 24t - 72. \end{aligned}$$

From Table 5, we have

$$\begin{aligned} \sum_{\substack{v \in V(CC_t) \\ deg(v)=3}} \epsilon(v)\delta(v) &= 48\left[(2t+6-1) + (2t+6-3) + \ldots + 2t + 6 - (t-1)\right] \\ &\quad + 24\left[(2t+6-2) + (2t+6-4) + \ldots + (2t+6-(t-2))\right] \\ &\quad + 56\left[(2t+6-2) + (2t+6-4) + \ldots + (2t+6-(t-2))\right] \\ &\quad + 56(2t+6-t) + 24(2t+6) \end{aligned}$$

$$\begin{aligned}
&= 48t + 144 + 48(2t + 6 - 1) + 56(t + 6) \\
&\quad + 48 \sum_{k=1}^{\frac{t-2}{2}} 2t + 6 - (2k + 1) + 80 \sum_{k=1}^{\frac{t-2}{2}} 2t + 6 - 2k \\
&= 96t^2 + 368t.
\end{aligned}$$

From Table 9, we have

$$\sum_{\substack{v \in V(CC_t) \\ deg(v)=4}} \epsilon(v)\delta(v) = 48\left[(2t + 6 - 3) + (2t + 6 - 5) + \ldots + (2t + 6 - (t + 1))\right]$$

$$= 48 \sum_{k=1}^{\frac{t}{2}} 2t + 6 - 2k - 3 = 36t^2 + 96t.$$

From Table 13, we have

$$\begin{aligned}
\sum_{\substack{v \in V(CC_t) \\ deg(v)=6}} \epsilon(v)\delta(v) &= 28(2t + 6 - 4) \\
&\quad + 32\left[(2t + 664) + (2t + 6 - 8) + \ldots + (2t + 6 - (t + 2))\right] \\
&= 28(2t + 2) + 32 \sum_{k=1}^{\frac{t-2}{2}} 2t + 6 - 2k - 4 = 24t^2 + 40t - 8.
\end{aligned}$$

Substituting the above equations in equation (3), we get

$$\Lambda^c(CC_t) = 192t^2 + 576t - 8, \text{ when } t \text{ is even.}$$

Case 2: Consider t to be odd.
As seen in Tables 6, 8, 10, 12, and 14, there are $\frac{t-1}{2}$ sets of vertices of degree 6, $\frac{t+1}{2}$ sets of vertices of degree 4, $t + 2$ sets of vertices of degree 3, $\frac{t-1}{2}$ sets of vertices of degree 2, and one set of vertices of degree 1. The vertex eccentric connectivity index of a graph CC_t is given by

$$\Lambda^c(CC_t) = \sum_{v \in V(CC_t)} \epsilon(v)\delta(v).$$

From Table 8, we have

$$\sum_{\substack{v \in V(CC_t) \\ deg(v)=1}} \epsilon(v)\delta(v) = \sum_{\substack{v \in V(CC_t) \\ deg(v)=1}} (2t + 3) = 24(2t + 3) = 48t + 72.$$

From Table 12, we have

$$\sum_{\substack{v\in V(CC_t)\\ deg(v)=2}} \epsilon(v)\delta(v) = 48\left[(2t+6-5)+(2t+6-7)+\ldots+(2t+6-t-2)\right]$$
$$= 48\sum_{k=1}^{\frac{t-1}{2}} 2t+6-2k-3 = 36t^2+24t-60.$$

From Table 6, we have

$$\sum_{\substack{v\in V(CC_t)\\ deg(v)=3}} \epsilon(v)\delta(v) = 24(2t+6)$$
$$+48\left[(2t+6-1)+(2t+6-3)+\ldots+2t+6-(t-2)\right]$$
$$+24\left[(2t+6-2)+(2t+6-4)+\ldots+(2t+6-(t-3))\right]$$
$$+56\left[(2t+6-2)+(2t+6-4)+\ldots+(2t+6-(t-3))\right]$$
$$+12(t+7)+56(t+7)+24(t+6)+28(t+5)$$
$$= 24(2t+6)+48(2t+6-1)+12(t+7)+56(t+7)$$
$$+24(t+6)+28(t+5)+48\sum_{k=1}^{\frac{t-3}{2}} 2t+6-(2k+1)$$
$$+24\sum_{k=1}^{\frac{t-3}{2}} 2t+6-2k+56\sum_{k=1}^{\frac{t-3}{2}} 2t+6-2k$$
$$= 96t^2+368t-32.$$

From Table 10, we have

$$\sum_{\substack{v\in V(CC_t)\\ deg(v)=4}} \epsilon(v)\delta(v) = 48\left[(2t+6-3)+(2t+6-5)+\ldots+(2t+6-t)\right]$$
$$+24(2t+6-t-2)$$
$$= \sum_{k=1}^{\frac{t-1}{2}} 2t+6-2k-1+24(2t+6-t-2) = 36t^2+96t-12.$$

From Table 14, we have

$$\sum_{\substack{v\in V(CC_t)\\ deg(v)=6}} \epsilon(v)\delta(v) = 28(2t+6-4)+16(2t+6-t-3)$$

$$
\begin{aligned}
&+ 32\left[(2t+6-6)+(2t+6-8)+\ldots+(2t+6-(t+1))\right] \\
&= 28(2t+2)+16(t+3)+32\sum_{k=1}^{\frac{t-3}{2}} 2t+6-2k-4 \\
&= 24t^2+40t-16.
\end{aligned}
$$

$$\Lambda^c(CC_t) = 192t^2+576t-48, \text{ when } t \text{ is odd.}$$

□

Table 5: Sets of vertices with degree 3 in CC_t for even t.

S. No of sets of vertices	Frequency	$\epsilon(v)$	$\delta(v)$
1	4	$2t+6$	6
2	8	$2t+6-1$	6
3	4	$2t+6-2$	6
	8	$2t+6-2$	7
4	8	$2t+6-3$	6
5	4	$2t+6-4$	6
	8	$2t+6-4$	7
⋮	⋮	⋮	⋮
$t-1$	4	$2t+6-(t-2)$	6
	8	$2t+6-(t-2)$	7
t	8	$2t+6-(t-1)$	6
$t+1$	8	$2t+6-t$	7

Table 6: Sets of vertices with degree 3 in CC_t for odd t.

S. No of sets of vertices	Frequency	$\epsilon(v)$	$\delta(v)$
1	4	$2t+6$	6
2	8	$2t+6-1$	6
3	4	$2t+6-2$	6
	8	$2t+6-2$	7
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$t-2$	4	$2t+6-(t-3)$	6
	8	$2t+6-(t-3)$	7
$t-1$	8	$2t+6-(t-2)$	6
t	2	$2t+6-(t-1)$	6
	8	$2t+6-(t-1)$	7
$t+1$	4	$2t+6-t$	6
$t+2$	4	$2t+6-t-1$	7

Table 7: Sets of vertices with degree 1 in CC_t for even t.

S. No of sets of vertices	Frequency	$\epsilon(v)$	$\delta(v)$
1	4	$2t+6-3$	6

Table 8: Sets of vertices with degree 1 in CC_t for odd t.

S. No of sets of vertices	Frequency	$\epsilon(v)$	$\delta(v)$
1	4	$2t+6-3$	6

Table 9: Sets of vertices with degree 4 in CC_t for even t.

S. No of sets of vertices	Frequency	$\epsilon(v)$	$\delta(v)$
1	4	$2t+6-3$	12
2	4	$2t+6-5$	12
3	4	$2t+6-7$	12
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\frac{t}{2}$	4	$2t+6-(t+1)$	12

Table 10: Sets of vertices with degree 4 in CC_t for odd t.

S. No of sets of vertices	Frequency	$\epsilon(v)$	$\delta(v)$
1	4	$2t+6-3$	12
2	4	$2t+6-5$	12
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\frac{t-1}{2}$	4	$2t+6-t$	12
$\frac{t+1}{2}$	2	$2t+6-(t+2)$	12

Table 11: Sets of vertices with degree 2 in CC_t for even t.

S. No of sets of vertices	Frequency	$\epsilon(v)$	$\delta(v)$
1	4	$2t+6-5$	12
2	4	$2t+6-7$	12
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\frac{t}{2}-1$	4	$2t+6-(t+1)$	12
$\frac{t}{2}$	2	$2t+6-(t+3)$	12

Table 12: Sets of vertices with degree 2 in CC_t for odd t.

S. No of sets of vertices	Frequency	$\epsilon(v)$	$\delta(v)$
1	4	$2t+6-5$	12
2	4	$2t+6-7$	12
3	4	$2t+6-9$	12
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\frac{t-1}{2}$	4	$2t+6-(t+2)$	12

Table 13: Sets of vertices with degree 6 in CC_t for even t.

S. No of sets of vertices	Frequency	$\epsilon(v)$	$\delta(v)$
1	2	$2t+6-4$	14
2	2	$2t+6-6$	16
3	2	$2t+6-8$	16
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\frac{t}{2}$	2	$2t+6-(t+2)$	16

Table 14: Sets of vertices with degree 6 in CC_t for odd t.

S. No of sets of vertices	Frequency	$\epsilon(v)$	$\delta(v)$
1	2	$2t+6-4$	14
2	2	$2t+6-6$	16
3	2	$2t+6-8$	16
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\frac{t-1}{2}$	2	$2t+6-(t+1)$	16
$\frac{t+1}{2}$	1	$2t+6-(t+3)$	16

5 Conclusion

In this article, vertex eccentricity connectivity indices are computed for graphs obtained by operations of graphs and for the chemical graph of (α)bis(pyridine)-cobalt(III)chloride with the chemical formula $Co(py)_2Cl_2$. These indices are connected to a substance's core physical and chemical characteristics, and the results can be very useful in QSAR (Quantitative Structure-Activity Relationships) and QSPR (Quantitative Structure-Property Relationship) research. The work can be extended to find other topological indices for the chemical graph of (α)bis(pyridine)-cobalt(III)chloride.

References

[1] A. Balaban, "Topological and stereochemical molecular descriptors for databases useful in qsar, similarity/dissimilarity and drug design," *SAR and QSAR in Environmental Research*, vol. 8, no. 1-2, 1998. DOI: https://doi.org/10.1080/10629369808033259 (⇒ 79).

[2] H. P. Schultz, E. B. Schultz, and T. P. Schultz, "Topological organic chemistry. 9. graph theory and molecular topological indices of stereoisomeric organic compounds," *Journal of chemical information and computer sciences*, vol. 35, no. 5, pp. 864–870, 1995. DOI: https://doi.org/10.1021/ci00027a011 (⇒ 79).

[3] N. Trinajstic, *Chemical graph theory*. Routledge, 2018 (⇒ 79).

[4] G. Rücker and C. Rücker, "On topological indices, boiling points, and cycloalkanes," *Journal of chemical information and computer sciences*, vol. 39, no. 5, pp. 788–802, 1999. DOI: https://doi.org/10.1021/ci9900175 (⇒ 79).

[5] S. Zaman, H. S. A. Yaqoob, A. Ullah, and M. Sheikh, "Qspr analysis of some novel drugs used in blood cancer treatment via degree based topological indices and regression models," *Polycyclic Aromatic Compounds*, pp. 1–17, 2023. DOI: https://doi.org/10.1080/10406638.2023.2217990 (⇒ 79).

[6] L. R. M. Gnanaraj, D. Ganesan, and M. K. Siddiqui, "Topological indices and qspr analysis of nsaid drugs," *Polycyclic Aromatic Compounds*, pp. 1–17, 2023. DOI: https://doi.org/10.1080/10406638.2022.2164315 (⇒ 79).

[7] S. Hayat and M. Imran, "Computation of topological indices of certain networks," *Applied Mathematics and Computation*, vol. 240, pp. 213–228, 2014. DOI: https://doi.org/10.1016/j.amc.2014.04.091 (⇒ 79).

[8] A. R. Ashrafi, M. Saheli, and M. Ghorbani, "The eccentric connectivity index of nanotubes and nanotori," *Journal of Computational and Applied Mathematics*, vol. 235, no. 16, pp. 4561–4566, 2011. DOI: https://doi.org/10.1016/J.CAM.2010.03.001 (⇒ 79).

[9] H. Dureja, S. Gupta, and A. Madan, "Predicting anti-hiv-1 activity of 6-arylbenzonitriles: Computational approach using superaugmented eccentric connectivity topochemical indices," *Journal of Molecular Graphics and Modelling*, vol. 26, no. 6, pp. 1020–1029, 2008. DOI: https://doi.org/10.1016/J.JMGM.2007.08.008 (⇒ 79).

[10] M. W. Rasheed, A. Mahboob, and I. Hanif, "An estimation of physicochemical properties of heart attack treatment medicines by using molecular descriptor's," *South African Journal of Chemical Engineering*, vol. 45, pp. 20–29, 2023. DOI: https://doi.org/10.1016/j.sajce.2023.04.003 (⇒ 79).

[11] H. Wiener, "Structural determination of paraffin boiling points," *Journal of the American chemical society*, vol. 69, no. 1, pp. 17–20, 1947. DOI: https://doi.org/10.1021/ja01193a005 (⇒ 79).

[12] V. Sharma, R. Goswami, and A. Madan, "Eccentric connectivity index: A novel highly discriminating topological descriptor for structure- property and structure- activity studies," *Journal of chemical information and computer sciences*, vol. 37, no. 2, pp. 273–282, 1997. DOI: https://doi.org/10.1021/ci960049h (⇒ 79).

[13] S. Gupta, M. Singh, and A. Madan, "Application of graph theory: Relationship of eccentric connectivity index and wiener's index with anti-inflammatory activity," *Journal of Mathematical Analysis and Applications*, vol. 266, no. 2, pp. 259–268, 2002. DOI: https://doi.org/10.1006/JMAA.2000.7243 (⇒ 79).

[14] V. Kumar, S. Sardana, and A. K. Madan, "Predicting anti-hiv activity of 2, 3-diaryl-1, 3-thiazolidin-4-ones: Computational approach using reformed eccentric connectivity index," *Journal of molecular modeling*, vol. 10, pp. 399–407, 2004. DOI: https://doi.org/10.1007/S00894-004-0215-8 (⇒ 79).

[15] M. Morgan, S. Mukwembi, and H. C. Swart, "On the eccentric connectivity index of a graph," *Discrete Mathematics*, vol. 311, no. 13, pp. 1229–1234, 2011. DOI: https://doi.org/10.1016/j.disc.2009.12.013 (⇒ 79).

[16] M. Ö. Turaci, "The values of eccentricity-based topological indices of diamond graphs," *Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü Dergisi*, vol. 22, pp. 285–289, 2018. DOI: https://doi.org/10.19113/SDUFBED.61724 (⇒ 79).

[17] M. Ö. Turaci, "On vertex and edge eccentricity-based topological indices of a certain chemical graph that represents bidentate ligands," *Journal of Molecular Structure*, vol. 1207, p. 127 766, 2020. DOI: https://doi.org/10.1016/j.molstruc.2020.127766 (⇒ 79).

[18] T. Došlic and M. Saheli, "Eccentric connectivity index of composite graphs," *Util. Math*, vol. 95, pp. 3–22, 2014 (⇒ 80).

[19] Y. Wang and Z. Zhu, "On the eccentric connectivity index of k-uniform hypercacti," *Discrete Applied Mathematics*, vol. 332, pp. 101–118, 2023. DOI: https://doi.org/10.1016/j.dam.2023.02.006 (⇒ 80).

[20] H. Iqbal, M. H. Aftab, A. Akgul, *et al.*, "Further study of eccentricity based indices for benzenoid hourglass network," *Heliyon*, 2023. DOI: https://doi.org/10.1016/j.heliyon.2023.e16956 (⇒ 80).

[21] K. Jebreen, H. Iqbal, M. H. Aftab, I. Yaqoob, M. I. Sowaity, and A. Barham, "Study of eccentricity based topological indices for benzenoid structure," *South African Journal of Chemical Engineering*, 2023. DOI: https://doi.org/10.1016/j.sajce.2023.05.010 (⇒ 80).

[22] A. Ashrafi, M. Ghorbani, and M. Hossein Zadeh, "The eccentric connectivity polynomial of some graph operations," eng, *Serdica Journal of Computing*, vol. 5, no. 2, pp. 101–116, 2011. DOI: https://doi.org/10.55630/sjc.2011.5.101-116. [Online]. Available: http://eudml.org/doc/196270 (⇒ 80).

[23] S. Cotton, "Cobalt chloride," *COBALT CHLORIDE - Molecule of the Month June 2016 - JSMol version*, Jun. 2016. [Online]. Available: https://www.chm.bris.ac.uk/motm/cobalt-chloride/cobalt-chloridejs.htm (⇒ 89).

[24] J. D. Dunitz, "The crystal structures of copper dipyridine dichloride and the violet form of cobalt dipyridine dichloride," *Acta Crystallographica*, vol. 10, no. 4, pp. 307–313, 1957. DOI: https://doi.org/10.1107/S0365110X57000894 (⇒ 89).

Received: 15.02.2024; *Revised: 20.06.2024*; *Accepted: 10.07.2024*

Acta Univ. Sapientiae, Informatica, **16, 1** (2024) 105–123

DOI: 10.47745/ausi-2024-0007

Exploring the Impact of Backbone Architecture on Explainable CNNs' Interpretability

Ábel PORTIK
Faculty of Mathematics and Computer Science, Babeş–Bolyai University
str. M. Kogălniceanu, nr. 1, RO-400084, Cluj-Napoca
abel.portik@stud.ubbcluj.ro
0009-0005-3496-9703

Adél BAJCSI
Faculty of Mathematics and Computer Science, Babeş–Bolyai University
str. M. Kogălniceanu, nr. 1, RO-400084, Cluj-Napoca
adel.bajcsi@ubbcluj.ro
0009-0007-9620-8584

Annamária SZENKOVITS
Faculty of Mathematics and Computer Science, Babeş–Bolyai University
str. M. Kogălniceanu, nr. 1, RO-400084, Cluj-Napoca
annamaria.szenkovits@ubbcluj.ro
0009-0001-8579-6962

Zalán BODÓ
Faculty of Mathematics and Computer Science, Babeş–Bolyai University
str. M. Kogălniceanu, nr. 1, RO-400084, Cluj-Napoca
zalan.bodo@ubbcluj.ro
0000-0002-4857-878X

Abstract. The growing demand for interpretable models in machine learning underscores the importance of transparency in decision-making processes for building trust and ensuring accountability in AI systems. Unlike complex black-box models, interpretable models shed light on the reasoning behind predictions or classifications. In image processing, convolutional networks often serve as backbone models that – obviously, but not entirely transparently – highly influence overall performance. This research focuses on assessing and comparing the performance of explainable neural network-based image classification models using various backbone architectures. The evaluation includes various performance metrics, such as prediction accuracy and specialized measurements tailored to assess interpretability, providing insights into the effectiveness of interpretable models in image classification tasks.

Key words and phrases: Explainable AI, Computer Vision, Convolutional Neural Networks, Image Classification, Interpretability Metrics

1 Introduction

Interpretability of the outputs of machine learning systems has always been a question of central interest, nevertheless, providing explanations for the predictions of *low-complexity*, e.g. linear models can be done in a rather simplistic manner. Linear or generalized linear models, however, are not the only ones considered highly interpretable, but here we can also mention decision trees, decision rules, or the k-nearest neighbor classifier [1]. However, when a more complex predictor is applied, giving explanations ceases to be simple anymore. Black-box predictions cannot facilitate root cause analysis, an important component of process improvement in high-risk systems, for example, driver-assistance systems. Besides accuracy and robustness, transparency of decisions is also prescribed by the Artificial Intelligence Act when working with high-risk scenarios.[1] Explainability can also alleviate social acceptance: if we know how something works, we trust it more. And last but not least, explainability might also facilitate performance improvement. However, interpretability can also lead to easier manipulation of the system; therefore, it must be handled with care [4].

In computer vision, explainable models focus mostly on image classification, with only a few tackling other tasks like semantic segmentation. A significant part of these methods are prototype-based models: the explanation is formulated by specifying the most important image parts that are similar to some prototypes extracted from training images [5]–[8]. From an architectural perspective, these neural networks first extract the features from the input image usually by employing a neural network (*backbone*), mostly a convolutional neural network (CNN), based on which the prototypes are determined and subsequently compared to the encoded image regions (see Figure 2). The results of the comparisons between the encoded image parts and prototypes are then fed into an interpretable, i.e. linear classifier layer. Although any convolutional backbone can be substituted in these models, the backbone has a major impact on the accuracy and explainability of the prediction. In this paper, we study the influence of the backbone model on the prediction accuracy and interpretability in interpretable image classification models, by using various convolutional nets to extract the features, comparing the results obtained and attempting to give explanations for the outcomes.

The present paper is structured as follows. Section 2 provides an introduction to explainable and self-explainable models in computer vision, used in image classification and semantic segmentation. In Section 3 evaluation metrics applied in the literature are presented to measure interpretability, some of which will also appear

[1] The AI Act proposed by the European Commission in 2021 enters into force in June–July 2024, but will become fully applicable 36 months after [2], [3].

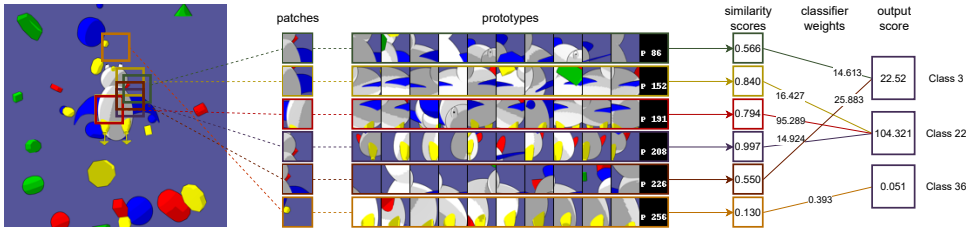


Figure 1: Example explanation given by the PIP-Net self-explaining model [7]: highlighted image patches are compared to the learned prototypes, from which similarity scores are obtained, finally, the similarity scores are weighted by *explainable* classification weights.

in our experiments. Section 4 presents the circumstances under which the main idea for the present paper was formulated and describes the backbone architectures used in our experiments. The experiments carried out and the results obtained are presented in Section 5, while Section 6 concludes the paper by discussing the results and possible future directions.

2 Explainable models in image classification

Although explainable models can be discussed independently of the application domain, in this section, we briefly describe the models used in computer vision; [9] gives a detailed introduction as well as a categorization of interpretable models in machine learning. Here we focus only on deep neural – mostly convolutional – network architectures for image classification and semantic segmentation, biased towards prototype-based approaches, mentioning only some of the other types of existing interpretable models. Although other categorizations exist as well, when considering explainable models, we differentiate here between (i) *post-hoc* approaches, i.e. methods where explanations are given using an ex-post procedure, and (ii) *self-interpretable* models, where explainability is an intrinsic property. The work [10] is a recent comprehensive survey on explainable approaches in artificial intelligence that covers the last few years of research in this field.

2.1 Post-hoc methods

The most notable explainable post-hoc methods are probably Grad-CAM (Gradient-weighted Class Activation Mapping) [11] and LIME (Local Interpretable Model-agnostic Explanations) [12]. Grad-CAM for CNNs calculates class-specific gradi-

ents and propagates them back to a given convolutional layer to assign importance to feature maps, while LIME is a generic method providing explanations by building an interpretable, e.g., linear model locally around the prediction. While LIME is model-agnostic, Grad-CAM is model-specific, since it can only be applied to (convolutional) neural networks. Grad-CAM is a generalization of CAM (Class Activation Maps) [13], capable of producing an activation heatmap of an arbitrary convolutional layer by the following formula

$$L^c_{\text{Grad-CAM}} = \text{ReLU}\left(\sum_k \text{GlobalAvgPool}\left(\frac{\partial y^c}{\partial A^k}\right) A^k\right),$$

where A^k denotes the k-th feature map of the selected layer and y^c is the class score before softmax. Unlike CAM, this method is applicable regardless of the neural network structure. The variants of Grad-CAM proposed over the years (Grad-CAM++ [14], Eigen-CAM [15], XGrad-CAM [16], etc.) try to correct the shortcomings of the base algorithm and thus create a more accurate heatmap for saliency visualization. Grad-CAM and its variants can also be used to provide high-resolution localization maps [17].

2.2 Self-explainable models

Attention-based models [18], [19], dominating natural language processing applications in the present, are becoming more and more popular and successful in solving computer vision tasks as well – in these cases, visualization can be performed, for example, using the attention maps obtained directly [20], [21].

Prototype-based models explain their decisions using an aggregated comparison of image regions to prototypes obtained from the training images. In the case of image classification, these models can explain the predictions by pinpointing the prototypes activated for a given image and their corresponding similarities and classification weights. Figure 2 shows the general architecture of prototype-based self-explainable image classification models. One of the first prototype-based such models, PrototypeDL [5] uses an autoencoder to obtain the features, the prototypes here having the same size as the input image. Its successor model, ProtoPNet [6], replaces the auto-encoder with a CNN and uses prototypes corresponding to smaller regions in the input space of the images. PIP-Net [7] no longer stores prototypes explicitly, but only the similarities of image patches to prototypes. It tries to correct the shortcomings of the earlier self-explainable models, in which a semantic gap between the input and latent space has been observed, originating from the possibly erroneous assumption that images from the same class have the same prototypes, thus resulting in a sometimes meaningless mixture of different concepts.

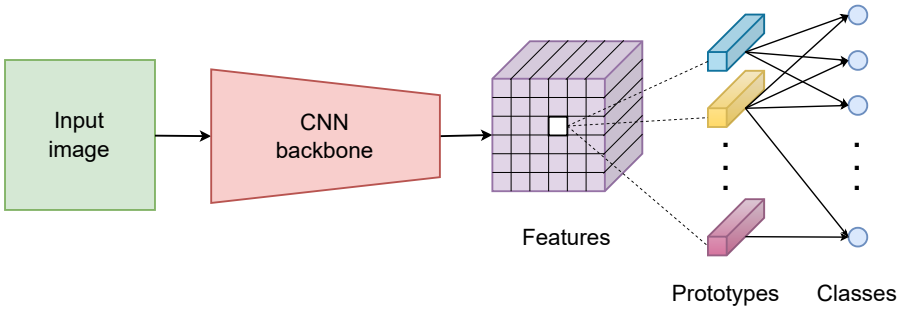


Figure 2: The general architecture of prototype-based self-explainable image classification models using a CNN backbone to extract the features: the feature vectors provided by the backbone neural network are compared to the prototypes, the predictions being based on the similarity scores obtained from this.

Its loss function has three parts, $\mathcal{L} = \lambda_C \mathcal{L}_C + \lambda_A \mathcal{L}_A + \lambda_T \mathcal{L}_T$,, where $\mathcal{L}_C$ denotes the negative log-likelihood classification loss, $\mathcal{L}_A$ is the aligment loss, while $\mathcal{L}_T$ is responsible for maintaining diversity. The alignment loss introduces robustness into the model by restraining that two views of the same image patch to belong to the same prototype,

$$\mathcal{L}_A = -\frac{1}{HW} \sum_{(h,w)} \log \left(\mathrm{z}'_{h,w} \cdot \mathrm{z}''_{h,w} \right),$$

where z denote the latent patches of the $H \times W$ feature map. When training, the network is fed with pairs instead of one image, generated using geometric augmentation. The tanh loss regularizer $\mathcal{L}_T$ was introduced to prevent the naive solution for the $\mathcal{L}_A$ term, forcing every prototype to be at least once present in a mini-batch,

$$\mathcal{L}_T = -\frac{1}{D} \sum_{d} \log \left[\tanh \left(\sum_{b \in B} \mathrm{p}_b \right) + \epsilon \right],$$

where p_b is the prototype similarity vector, D denotes the number of prototypes and B represents a mini-batch. It uses *scoring sheet reasoning*, applying a ReLU activation function on the classification weights, thus producing a sparse classification layer with positive and hence interpretable weights. The model can also abstain itself from prediction if low similarity scores are obtained for every prototype. By allowing only positive weights in the classification layer, the model will output only positive predictions for every class; therefore, the reasoning process will rely only on concepts that are present in the input image and not on missing concepts. This way the model can predict out-of-distribution data when all the classes get near-zero scores, i.e. none of the learned concepts were found in the input image.

3 Metrics of interpretability

By evaluating supervised learning methods, we usually understand measuring the accuracy of the prediction with respect to some ground-truth data, different cases requiring different such metrics (e.g. accuracy versus F_1-score [22]). However, in our situation, explainability should also be measured to be able to compare different approaches from this perspective as well. In what follows, we enumerate some of the interpretability metrics used in the computer vision literature.

3.1 Intersection over union

The intersection of union metrics (IoU) are used to evaluate the object detection and semantic segmentation methods in computer vision [23] – in both problems, a specific part of the image is sought, and the better the overlap between ground truth and output, the better the prediction should be. IoU can also be used to measure interpretability by calculating the overlap between the ground truth object or the object part and the thresholded activation map [24], [25].

3.2 Location instability

Location instability was introduced in [26] and was further applied to evaluate explainability in [25]. The underlying idea is to measure the variance of the inference positions relative to some landmark points. As stated in [25], "if f represented the shoulder, then the distance between the shoulder and the head should remain stable through different objects", hence the metric computes the mean of the variances of these distances.[2]

3.3 Prototype purity

The purity of the prototype was introduced in [7] to measure interpretability in classification scenarios. This metric measures the extent to which prototypes represent the same object part by selecting the most similar images to a given prototype and returning the maximum such ratio. A prototype is said to represent a specific part if the center of the object part lies in the image patch corresponding to the prototype. The most similar images based on which this measure is calculated are determined

[2]Because one cannot guarantee the same ratio between the size of the object and the of the image, normalization should be done for example by the average distance between landmark points, assuming the presence of at least two landmarks, instead of the width and height of the image, as it appears in the proposed formula.

in two ways in [7]: (i) selecting the top-k most similar images versus (ii) selecting all images where similarity exceeds an ε threshold.

3.4 The "FunnyBirds" metrics

In [27] the synthetic dataset *FunnyBirds* is introduced together with a set of metrics that measure the interpretability from three perspectives: *completeness*, *correctness*, and *contrastivity*. In our experiments, we used these metrics (and the dataset as well) in order to be able to put the results side by side. We also mention that while these measures offer a versatile evaluation of interpretability, their disadvantage is that a significant subset of these can only be used for synthetically generated datasets.

1. Completeness:
    - Controlled synthetic data check (CSDC) – overlap between the parts estimated to be important and the sufficient parts to correctly categorize the image, normalized by the size of the sufficient parts set.
    - Preservation check (PC) – how often the prediction remains unchanged if parts estimated to be unimportant are deleted from the image.
    - Deletion check (DC) – how often the prediction changes if the parts estimated to be important are removed from the image.[3]
    - Distractibility (D) – how many actually unimportant parts are correctly estimated to be unimportant by the explanation; the opposite of CSDC (overcompleteness).
2. Correctness:
    - Single deletion (SD) – Spearman's rank correlation coefficient between the explanation's importance score of each part and the change in the logit of the target class when removing that part.
3. Contrastivity:
    - Target sensitivity (TS) – benchmarks how sensitive an explanation is to the target class; to measure this, the relative frequency of events is considered when the relation between the returned importance scores between two classes with nonoverlapping parts is correct.

[3] Deletion check requires that the prediction to be different than the ground-truth when important parts are removed from the image, however, all that can really be said is that the prediction might change in this case. Although we do not agree with this metric, we left it in the set of evaluation measures used throughout the experiments.

A metric that falls outside of the above mentioned three categories is background independence (BI), measuring the level of independence between the prediction and the image background by calculating the ratio of background objects whose removal causes the target logit to drop by less than 5%. However, following the definition of *correctness* for interpretable models from [27], in our opinion, BI could also be placed alongside SD.

4 Changing the backbone architecture

As described in Section 2.2, ProtoPNet [6] and PIP-Net [7] use CNNs to extract the features from the input images. In this section, we briefly describe three successful and popular CNN architectures that can be used as the backbone providing the features in self-explaining models. ProtoPNet was chosen because of its model's clarity, while PIP-Net is considered to be an extension of ProtoPNet, claiming its superiority in terms of explainability – see Section 3.3 for the prototype purity metric. The authors of [7] obtained exceptional results using ConvNeXt as the model backbone; therefore, in addition to VGG and ResNet, ConvNeXt will also be part of the architectures used as backbones in our experiments.

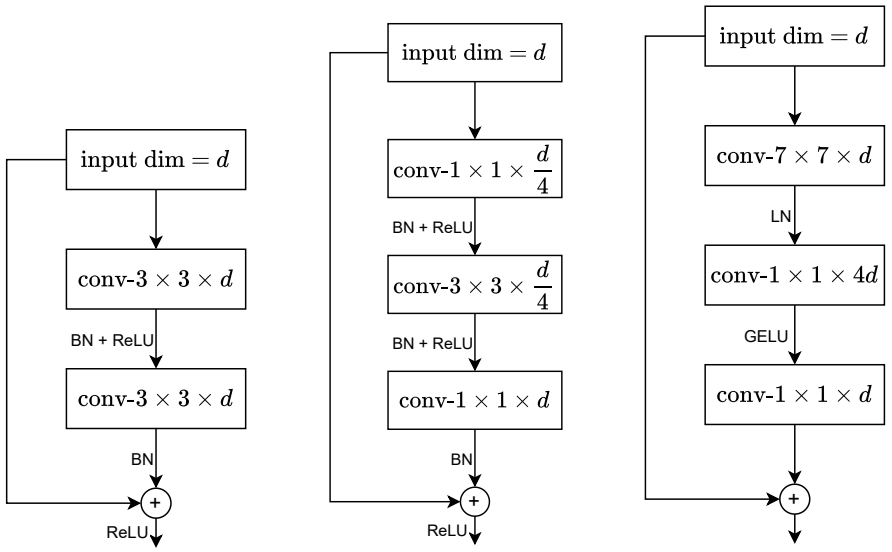
4.1 VGG

VGG [28] is architecturally the simplest of the three CNNs included in the current study. The network is built of (3×3) convolutional filters and max-pooling layers to reduce feature dimensions. Simonyan and Zisserman [28] introduced the architecture in multiple configurations, four of which were subsequently named VGG11, VGG13, VGG16, and VGG19, denoting the number of weighted layers used in the configuration.

The architecture of VGG networks, which are designed for image classification tasks, consists of five convolutional layers, followed by three fully connected layers and a softmax layer. In a self-explaining model, only the convolutional layers serve as the backbone, the fully connected and softmax layers are excluded. The total number of convolutional filters in the five layers ranges from 8 to 16 in the different configurations.

4.2 ResNet

ResNet [29] marked a paradigm shift by addressing the problem of vanishing gradients that plagued deeper neural networks. The ResNet architecture is primarly built on the VGG architecture, both featuring five convolutional layers in the feature ex-



(a) ResNet Basic Block (b) ResNet Bottleneck (c) ConvNeXt Block

Figure 3: Building blocks of ResNet and ConvNeXt architectures.

traction segment. Its innovative use of skip connections, or residual connections, allows gradients to flow through the network more effectively, enabling the training of much deeper networks without a degradation in performance. This design allows the network to use identity functions between layers as needed, which means that layers can be skipped if they do not contribute to the overall performance, leading to more efficient learning processes.

Compared to the architecture of VGG, ResNet also has five convolutional layers that can serve as the backbone of self-explaining models. Unlike VGG, the first layer features a single 7×7 convolution with a stride of 2 followed by a max-pool for downsampling. Moreover, max-pooling layers are omitted after the next layers, and downsampling is performed by the first block of each layer, having a stride of 2 on the 3×3 convolution.

The residual connections in ResNet address the issue of vanishing gradients. Connections are introduced through every group of two consecutive convolutional layers in the VGG architecture, allowing the network to bypass these layers, forming a basic block, as illustrated in Figure 3a.

By overcoming the problem of vanishing gradients, it became possible to con-

struct networks with more layers. To maintain computational efficiency, bottleneck blocks are utilized in place of basic blocks Figure 3b. In a bottleneck block, the feature map is reduced in depth dimension before the 3×3 convolution and restored after, using 1×1 convolutions.

Having the ability to build neural networks with greater depth, [28] introduces two smaller ResNet configurations built of basic blocks (see Figure 3a) and three larger ones built of bottleneck blocks (see Figure 3b), the total number of convolutional filters in the five layers ranging from 17 to 151.

4.3 ConvNeXt

ConvNeXt [30] represents a contemporary architecture that modernizes the ResNet design. The authors begin by applying advanced training techniques to enhance the performance of the original ResNet architecture, which serves as a baseline. Subsequently, they methodically develop a new convolutional network architecture, integrating insights from transformer models and ResNeXt [31].

As a first step, the block number ratio between convolutional layers is aligned with the ratios used in vision transformer models. Then the initial 7×7 convolution with stride 2 and max-pool is replaced by a single 4×4 convolution to migrate the *patchify* strategy of the vision transformers.

Based on the main principle of ResNeXt, ConvNeXt employs grouped convolutions and broadens the network width. Depthwise convolutions serve to diminish the FLOPs. Another benefit of depth-wise convolution is their ability to separate spatial and channel mixing, similarly to vision transformers, where each operation mixes information across spatial or channel dimension, but not both.

In ConvNeXt, inspired by vision transformers, bottleneck blocks have been replaced with inverted bottleneck blocks. Moving beyond the foundation of the VGG architecture, the developers of ConvNeXt investigated the use of spatial convolutions larger than 3×3. This required swapping the first 1×1 convolution and depthwise convolution within the bottleneck blocks. Their experiments indicated that optimal performance was achieved with 7×7 convolutions.

Finally, some minor vision transformer-inspired adjustments have been applied to the network architecture, including (i) the interchanging rectified linear unit (ReLU) [32] activations to Gaussian error linear unit (GELU) [33]; (ii) the reduction of activation functions and normalization layers; (iii) the replacement of batch normalization with layer normalization and (iv) the implementation of a separate downsampling layer instead of using 3×3 convolution with stride 2 at the start of each layer. The final structure of the ConvNeXt architecture building blocks is depicted in Figure 3c.

5 Experimental results

In the current study, we compared different self-explainable and interpretable post-hoc networks by measuring their explainability. The goal is to obtain empirical evidence that supports the importance of the backbone network in the architectures studied. In the following sections, we describe the dataset used, the setup of the experiments, and present the obtained results.

5.1 Dataset

In the current experiments, images from the FunnyBirds dataset [27] were used[4], a synthetic dataset generated to quantitatively analyze XAI models. In [27], the following five human-interpretable "concepts" were considered in the development of the dataset: (1) beak, (2) wings, (3) feet, (4) eyes and (5) tail. Each class in the dataset consists of a unique combination of these concepts, the dataset imitating real-word data/scenarios by adding random background objects and changing the illumination and view point when generating the images.

The FunnyBirds dataset consists of 50 500 images (50 000 train and 500 test samples) of 50 species of synthetic birds. In addition to the images, the dataset also contains a pixel-wise annotation of each object and concept to facilitate realistic image modifications (e.g. removing parts).

5.2 Setup

In the case of the ProtoPNet experiments, we used Adam optimizer, a learning rate of 0.003 for the warm-up and joint training phases, and 0.0001 for the fine-tuning step. A batch size of 128 was used for all three backbone families.

For the PIP-Net model, the Adam optimizer with a learning rate of 0.0005 was used. During the pre-training phase, a batch size of 64 and 32 was used during joint training, respectively, except when ConvNeXt-Tiny was used as the backbone, where the pre-training and joint batch size were set to 32 and 16, respectively.

Grad-CAM and Grad-CAM++ experiments were performed using the `pytorch-grad-cam`[5] library. For every architecture, the feature map considered was the last layer preceding the classification layer. For ResNet, Adam optimizer was used with a learning rate of 0.001, while for the rest of the models, the SGD optimizer was used with the same learning rate and 0 momentum. The batch sizes selected were 128 for ResNet and 32 for the larger VGG and ConvNeXt models. Each model was

[4] https://github.com/visinf/funnybirds-framework
[5] https://github.com/jacobgil/pytorch-grad-cam

Metric \ Backbone	ResNet-18	ResNet-34	VGG-11	VGG-16	ConvNeXt-T
ACC	0.76	0.86	0.94	0.95	0.96
BI	0.99	0.99	1.00	1.00	0.97
CSDC	0.94	0.94	0.96	0.97	1.00
PC	0.95	0.93	0.98	0.99	0.97
DC	0.97	0.97	0.98	0.98	0.21
D	0.33	0.29	0.46	0.43	0.99
SD	0.28	0.53	0.23	0.36	0.80
TS	0.57	0.56	0.64	0.72	0.50

Table 1: ProtoPNet results using various backbone architectures, measured by the eight evaluation metrics.

Metric \ Backbone	ResNet-18	ResNet-34	VGG-11	VGG-16	ConvNeXt-T
ACC	0.98	0.97	0.97	0.81	0.98
BI	0.98	0.99	0.98	0.98	0.99
CSDC	0.95	0.95	0.87	0.74	0.97
PC	0.96	0.97	0.80	0.61	1.00
DC	0.95	0.94	0.81	0.78	0.97
D	0.38	0.46	0.78	0.75	0.20
SD	0.59	0.57	0.55	0.61	0.63
TS	0.23	0.48	0.44	0.06	0.51

Table 2: PIP-Net results using various backbone architectures, measured by the eight evaluation metrics.

trained for 60 epochs.

The experiments were performed using NVIDIA GeForce RTX 3050, 3060 and 3070 GPUs having 8 to 12GB memory, depending on availability.

5.3 Results

We conducted experiments on the FunnyBirds synthetic dataset using self-explainable models (ProtoPNet and PIP-Net) and post-hoc approaches (Grad-CAM and Grad-CAM++). We used ResNet [28], VGG [29], and ConvNeXt [30] serving as backbones of the interpretable models. From the available ResNet and VGG configurations, ResNet-18, ResNet-34, VGG-11, and VGG-16 were used, respectively, supplemented by the ConvNeXt-Tiny model. All these backbones were initialized with weights obtained by pre-training on ImageNet.

The results obtained are shown in Tables 1 to 4: the columns represent the differ-

Metric \ Backbone	ResNet-18	ResNet-34	VGG-11	VGG-16	ConvNeXt-T
ACC	0.95	0.97	0.96	0.96	0.97
BI	0.89	0.92	0.99	0.98	1.00
CSDC	0.61	0.57	0.73	0.73	0.32
PC	0.45	0.43	0.60	0.65	0.16
DC	0.54	0.45	0.70	0.70	0.21
D	0.91	0.90	0.94	0.93	0.89
SD	0.75	0.69	0.84	0.76	0.78
TS	0.29	0.29	0.68	0.64	0.77

Table 3: Grad-CAM results using various backbone architectures, measured by the eight evaluation metrics.

Metric \ Backbone	ResNet-18	ResNet-34	VGG-11	VGG-16	ConvNeXt-T
ACC	0.95	0.97	0.96	0.96	0.97
BI	0.89	0.92	0.99	0.98	1.00
CSDC	0.55	0.54	0.78	0.74	0.20
PC	0.38	0.36	0.67	0.63	0.07
DC	0.48	0.40	0.77	0.70	0.12
D	0.90	0.89	0.93	0.91	0.90
SD	0.73	0.68	0.83	0.75	0.75
TS	0.68	0.64	0.62	0.63	0.74

Table 4: Grad-CAM++ results using various backbone architectures, measured by the eight evaluation metrics.

ent backbone networks in increasing order of their complexity, i.e. number of trainable parameters, while the rows are the metrics computed in the experiments (see Section 3.4 and [27]). There is an observable general tendency that more complex models provide better scores; however, there are a few exceptions. These exceptions are probably due to the small number of learning epochs relative to the complexity of some of the models employed – compare, for example, the Grad-CAM results with the VGG-11 and VGG-16 models in Table 3, or the PIP-Net results obtained with the same two models in Table 2. Some of the outstandingly low scores obtained, e.g. DC of ProtoPNet + ConvNeXt, D of PIP-Net + ConvNeXt, TS of PIP-Net + VGG-16, or PC of both Grad-CAM and Grad-CAM++ with ConvNeXt; however, are not straightforward to explain, therefore these cases require a comprehensive investigation.

Background independence was the most consistent metric across all methods and

Backbone	XAI method	Com.	Cor.	Con.	mX	$\overline{\text{mX}}$
ResNet-18	ProtoPNet	0.64	0.28	0.57	0.50	0.57
ResNet-18	PIP-Net	0.67	0.59	0.23	0.50	
ResNet-18	Grad-CAM	0.72	0.75	0.29	0.59	
ResNet-18	Grad-CAM++	0.69	0.73	0.68	0.70	
ResNet-34	ProtoPNet	0.62	0.53	0.56	0.57	0.60
ResNet-34	PIP-Net	0.71	0.57	0.48	0.59	
ResNet-34	Grad-CAM	0.69	0.69	0.29	0.56	
ResNet-34	Grad-CAM++	0.66	0.68	0.64	0.66	
VGG-11	ProtoPNet	0.72	0.23	0.64	0.53	0.67
VGG-11	PIP-Net	0.80	0.55	0.44	0.60	
VGG-11	Grad-CAM	0.81	0.84	0.68	0.78	
VGG-11	Grad-CAM++	0.84	0.83	0.62	0.76	
VGG-16	ProtoPNet	0.71	0.36	0.72	0.60	0.64
VGG-16	PIP-Net	0.73	0.61	0.06	0.47	
VGG-16	Grad-CAM	0.81	0.76	0.64	0.74	
VGG-16	Grad-CAM++	0.80	0.75	0.63	0.73	
ConvNeXt-T	ProtoPNet	0.86	0.80	0.50	0.72	0.67
ConvNeXt-T	PIP-Net	0.59	0.63	0.51	0.58	
ConvNeXt-T	Grad-CAM	0.56	0.78	0.77	0.70	
ConvNeXt-T	Grad-CAM++	0.52	0.75	0.74	0.67	

Table 5: Results obtained averaged into the three explainability dimensions of *completeness*, *correctness* and *contrastivity* from [27]. mX denotes the mean explainability score obtained as the average of these, and the last column $\overline{\text{mX}}$ averages these scores over different backbones.

backbones, reaching almost its maximum value in every case. The self-explainable models achieved considerably higher completeness scores (CSDC, PC, DC) than the post-hoc approaches tested; however, self-explainable models tend to provide over-complete explanations (D) in contrast to post-hoc methods. Overall, correctness and contrastivity metrics did not perform well in the case of self-explainable models, with a slight improvement observed for the post-hoc methods, meaning that the importances are not always estimated correctly, and the explanations given are not always constrastive.

Table 5 summarizes the results obtained into the three explainability dimensions, simplifying the comparison between the results of [27] and our scores.

6 Conclusion

The experiments performed show that the choice of backbone architecture is in fact important when building a prototype-based self-explaining model. As models built on more complex backbones produced slightly higher accuracies, the models did not achieve higher scores on every interpretability metric. In general, based on the experiments, we can say that by increasing the complexity of the backbone CNN an increasing tendency was observed for most of the protocols, but there were exceptions, probably due to insufficient learning in most cases – Table 5 supports this claim too. An early stopping condition with the right parameters would probably be a unifying solution here, but due to lack of resources, this has not been achieved so far. Background independence was accomplished for every method tested, while the self-interpretable models produced better completeness scores but lower over-completeness results, compared to Grad-CAM and Grad-CAM++.

Using PIP-Net, slightly better scores have been observed, again with some exceptions, namely the contrastivity dimension, for which almost consistently worse results were obtained; however, PIP-Net with ConvNeXt produced unexpectedly low explainability scores overall.

Although the authors of [27] claim the 500 test images to be sufficient to produce stable results, we somewhat disagree with their statement considering the ease of achieving a test accuracy of 95% after only a few epochs, training accuracy being able to catch up only after *quite* a few iterations – the rapid convergence of accuracy and loss on the test set might suggest a non-representative data sample. However, evaluation is very costly for even a small dataset of this size. We plan to generate a larger dataset with better properties or using other datasets to evaluate interpretability. We recommend using semantic segmentation datasets such as PASCAL VOC[6] or PASCAL-Part[7] for classification, where the segmentation masks could be used to compute some of the metrics proposed in [27] (e.g., background independence or preservation check).

Regarding the hyperparameters used during the training, we observed that the self-explaining models were really sensitive even to slight modifications of these. Parameters such as learning rate, batch size, and weights of different loss function parts should be set carefully for both ProtoPNet and PIP-Net. Incorrectly set hyperparameters could cause the learned prototypes to be useless for classification.

Normalizing the FunnyBirds images is also expected to provide better results, however, normalization is not discussed in [27] or its supplement[8], nor could it be

[6] http://host.robots.ox.ac.uk/pascal/VOC/

[7] https://roozbehm.info/pascal-parts/pascal-parts.html

[8] https://openaccess.thecvf.com/content/ICCV2023/html/Hesse_FunnyBirds_A_

found in the code provided by the authors (see Footnote 4); therefore, we decided to perform the experiments without it as well.

Since no clear correlation between accuracy or loss and the scores provided by the interpretable metrics could be obtained, we propose that interpretability should be considered during the training phase as well (e.g. by incorporating some of these metrics into the loss function), allowing the adjustment of the training process to achieve the desired level of interpretability alongside prediction accuracy.

Funding: This research was funded by the Babeş–Bolyai University through the Special Scholarships for Scientific Activity (SSSA) for the 2023–2024 academic year no. 36260/24.11.2023, and the Hungarian Academy of Sciences through the Domus Scholarship no. 50/9/2024/HTMT.

Data Availability: The code of the experiments performed can be accessed at https://github.com/PortikAbel/XAI-Interpretability. In order to successfully set up we recommend consulting the https://github.com/visinf/funnybirds-framework repository prior to running our code.

References

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*. Berlin, Heidelberg: Springer-Verlag, 2006, ISBN: 0387310738 (⇒ 106).

[2] *Proposal for a Regulation of the European Parliament and of the Council Laying Down Harmonised Rules on Artificial Intelligence (Artificial Intelligence Act) and Amending Certain Union Legislative Acts*, European Commission, 2021. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A52021PC0206 (⇒ 106).

[3] *EU Artificial Intelligence Act. Timeline of Developments*, 2024. [Online]. Available: https://artificialintelligenceact.eu/developments/ (⇒ 106).

[4] C. Molnar, *Interpretable Machine Learning: A Guide For Making Black Box Models Explainable*. 2024, ISBN: 9798411463330. [Online]. Available: https://christophm.github.io/interpretable-ml-book/ (⇒ 106).

Synthetic_Vision_Dataset_for_a_Part-Based_Analysis_of_ICCV_2023_paper.html

[5] O. Li, H. Liu, C. Chen, and C. Rudin, "Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018, pp. 3530–3537. DOI: 10.1609/aaai.v32i1.11771 (⇒ 106, 108).

[6] C. Chen, O. Li, C. Tao, A. J. Barnett, J. Su, and C. Rudin, "This Looks like That: Deep Learning for Interpretable Image Recognition," in *NeurIPS*, 2019, pp. 8930–8941. DOI: 10.5555/3454287.3455088 (⇒ 106, 108, 112).

[7] M. Nauta, J. Schlötterer, M. van Keulen, and C. Seifert, "PIP-Net: Patch-Based Intuitive Prototypes for Interpretable Image Classification," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 2744–2753. DOI: 10.1109/cvpr52729.2023.00269 (⇒ 106–108, 110–112).

[8] M. Sacha, D. Rymarczyk, Ł. Struski, J. Tabor, and B. Zieliński, "ProtoSeg: Interpretable Semantic Segmentation With Prototypical Parts," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023, pp. 1481–1492. DOI: 10.1109/wacv56688.2023.00153 (⇒ 106).

[9] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, "Explaining explanations: An overview of interpretability of machine learning," in *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, 2018, pp. 80–89. DOI: 10.1109/DSAA.2018.00018 (⇒ 107).

[10] S. Ali, T. Abuhmed, S. El-Sappagh, *et al.*, "Explainable Artificial Intelligence (XAI): What we know and what is left to attain Trustworthy Artificial Intelligence," *Information Fusion*, vol. 99, p. 101 805, 2023. DOI: 10.1016/j.inffus.2023.101805 (⇒ 107).

[11] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 618–626. DOI: 10.1109/iccv.2017.74 (⇒ 107).

[12] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why should i trust you?" Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1135–1144. DOI: 10.1145/2939672.2939778 (⇒ 107).

[13] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2921–2929. DOI: 10.1109/cvpr.2016.319 (⇒ 108).

[14] A. Chattopadhay, A. Sarkar, P. Howlader, and V. N. Balasubramanian, "Grad-CAM++: Generalized gradient-based visual explanations for deep convolutional networks," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, IEEE, 2018, pp. 839–847. DOI: 10.1109/WACV.2018.00097 (⇒ 108).

[15] M. B. Muhammad and M. Yeasin, "Eigen-CAM: Class activation map using principal components," in *2020 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2020, pp. 1–7. DOI: 10.1109/ijcnn48605.2020.9206626 (⇒ 108).

[16] R. Fu, Q. Hu, X. Dong, Y. Guo, Y. Gao, and B. Li, *Axiom-based Grad-CAM: Towards accurate visualization and explanation of CNNs*, arXiv preprint arXiv:2008.02312, 2020. DOI: 10.48550/arXiv.2008.02312 (⇒ 108).

[17] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, *Striving for simplicity: The all convolutional net*, arXiv preprint arXiv:1412.6806, 2014. DOI: 10.48550/arXiv.1412.6806 (⇒ 108).

[18] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017. DOI: 10.5555/3295222.3295349 (⇒ 108).

[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of deep bidirectional transformers for language understanding*, arXiv preprint arXiv:1810.04805, 2018. DOI: 10.48550/arXiv.1810.04805 (⇒ 108).

[20] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, *An image is worth 16 words: Transformers for image recognition at scale*, arXiv preprint arXiv:2010.11929, 2020. DOI: 10.48550/arXiv.2010.11929 (⇒ 108).

[21] H. Chefer, S. Gur, and L. Wolf, "Transformer interpretability beyond attention visualization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 782–791. DOI: 10.1109/CVPR46437.2021.00084 (⇒ 108).

[22] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge University Press, 2008, ISBN: 0521865719 (⇒ 110).

[23] S. Hao, Y. Zhou, and Y. Guo, "A brief survey on semantic segmentation with deep learning," *Neurocomputing*, vol. 406, pp. 302–321, 2020. DOI: 10.1016/j.neucom.2019.11.118 (⇒ 110).

[24] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba, "Network dissection: Quantifying interpretability of deep visual representations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6541–6549. DOI: 10.1109/CVPR.2017.354 (⇒ 110).

[25] Q. Zhang, Y. N. Wu, and S.-C. Zhu, "Interpretable convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8827–8836. DOI: 10.1109/cvpr.2018.00920 (⇒ 110).

[26] Q. Zhang, R. Cao, F. Shi, Y. N. Wu, and S.-C. Zhu, "Interpreting CNN knowledge via an explanatory graph," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018. DOI: 10.1609/aaai.v32i1.11819 (⇒ 110).

[27] R. Hesse, S. Schaub-Meyer, and S. Roth, "Funnybirds: A synthetic vision dataset for a part-based analysis of explainable ai methods," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 3981–3991. DOI: 10.1109/ICCV51070.2023.00368 (⇒ 111, 112, 115, 117–119).

[28] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, arXiv preprint arXiv:1409.1556, 2015. DOI: 10.48550/arXiv.1409.1556 (⇒ 112, 114, 116).

[29] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, arXiv preprint arXiv:1512.03385, 2015. DOI: 10.48550/arXiv.1512.03385 (⇒ 112, 116).

[30] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 11 976–11 986. DOI: 10.1109/CVPR52688.2022.01167 (⇒ 114, 116).

[31] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, *Aggregated residual transformations for deep neural networks*, arXiv preprint arXiv:1611.05431, 2017. DOI: 10.48550/arXiv.1611.05431 (⇒ 114).

[32] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, Omnipress, 2010, pp. 807–814. DOI: 10.5555/3104322.3104425 (⇒ 114).

[33] D. Hendrycks and K. Gimpel, *Gaussian Error Linear Units (GELUs)*, arXiv preprint arXiv:1606.08415, 2023. DOI: 10.48550/arXiv.1606.08415 (⇒ 114).

Received: 19.06.2024; *Revised: 09.07.2024*; *Accepted: 01.08.2024*

Acta Univ. Sapientiae, Informatica, **16, 1** (2024) 124–138

DOI: 10.47745/ausi-2024-0008

On maximum spectral radius of $\{H(3,3),\ H(4,3)\}$-free graphs

Amir Rehman
University of Kashmir
Srinagar, India
aamirnajar786@gmail.com

S. PIRZADA
University of Kashmir
Srinagar, India
pirzadasd@kashmiruniversity.ac.in

Abstract. Let G be a simple connected graph of size m. Let A be the adjacency matrix of G and let $\rho(G)$ be the spectral radius of G. A graph is said to be H-free if it does not contain a subgraph isomorphic to H. Let $H(\ell,3)$ be the graph formed by taking a cycle of length ℓ and a triangle on a common vertex. Recently, Li, Lu and Peng [Y. Li, L. Lu, Y. Peng, Spectral extremal graphs for the bowtie, Discrete Math. 346(12) (2023) 113680.] showed that the unique m-edge $H(3,3)$-free spectral extremal graph is the join of K_2 with an independent set of $\frac{m-1}{2}$ vertices if $m \geq 8$ and the condition $m \geq 8$ is tight. In particular, if G does not contain $H(3,3)$ as induced subgraph, they proved that $\rho(G) \leq \frac{1+\sqrt{4m-3}}{2}$ and equality holds when G is isomorphic to $S_{\frac{m+3}{2},2}$. Note that Li et al. denoted $H(3,3)$ by F_2. In this paper, we find the maximum spectral radius and identify the graph with the largest spectral radius among all $\{H(3,3), H(4,3)\}$-free graphs of size odd m, where $m \geq 259$. Coincidently, we show that $\rho(G) \leq \frac{1+\sqrt{4m-3}}{2}$ when G forbids both $H(3,3)$ and $H(4,3)$. In our case, the equality holds when G is isomorphic to the same graph.

Key words and phrases: H-free graph, adjacency matrix, spectral radius, induced subgraph, forbidden subgraph

1 Introduction

Let G be a simple graph with order n and size m and let $V(G)$ be the vertex set of G. The adjacency matrix of G is defined as $A(G) = (a_{ij})$, where

$$a_{ij} = \begin{cases} 1 & \text{if there is an edge between vertices } i \text{ and } j, \\ 0 & \text{otherwise.} \end{cases}$$

The largest eigenvalue of $A(G)$, denoted by ρ, is called the spectral radius of G. In case of a connected graph G, the Perron-Frobenius theorem asserts the existence of a unique positive eigenvector associated with $\rho(G)$, termed as the Perron vector of G. Additional definitions and notations can be found in [1], [2].

For a subset $S \subseteq V(G)$, $G[S]$ represents the subgraph of G induced by S. Further $e(S,T)$ denotes the number of edges with one end in S and the other in T, where S and T are subsets of $V(G)$. Also, we use $e(S)$ to denote $e(S,S)$. We write $N^k(v)$ for the set of vertices at a distance of k from vertex v, with $N^1(v)$ being denoted by $N(v)$. We define $N[v]$ as $N(v) \cup \{v\}$. For $S \subseteq V(G)$, let $N_S(v)$ represent the set of neighbors of v in S and $d_S(v)$ be the cardinality of $N_S(v)$.

For $1 \leq k \leq n$, the graph $S_{n,k}$ of order n is obtained by joining each vertex of the complete graph K_k to $n-k$ isolated vertices. Let C_n represent the cycle on n vertices. Let $H(\ell,3)$ be the graph formed by a cycle of length ℓ and a triangle on a common vertex. For example, the graphs $H(3,3)$ and $H(4,3)$ are shown in Figure 1. Define $G(m,t)$ to be the graph of size m obtained by joining a vertex of maximum degree in $S_{\frac{m-t+3}{2},2}$ to t isolated vertices (see Fig. 2(a)). For $t=0$, the graph $S_{\frac{m+3}{2},2}$ is called the book graph. Let K_4^m be the graph of size m obtained by joining a vertex from K_4 to $m-6$ isolated vertices (see Fig. 2(b)).

For a family of graphs $\mathcal{H}$, a graph G is said to be $\mathcal{H}$-free if it does not contain a subgraph isomorphic to any graph in $\mathcal{H}$. In particular, if $\mathcal{H} = \{H\}$, we simply say that G is H-free. A classical problem in the extremal graph theory is the Turán problem which asks for the maximum size of an H-free graph of order n, where the maximum size is known as the Turán number of H. Nikiforov [3], proposed a spectral analogue of the Turán problem which asks for the maximum spectral radius of an H-free graph of size m or order n. In [4], Nosal proved that $\rho(G) \leq \sqrt{m}$ for every graph of size m, when H is a triangle. Nikiforov [5] showed that $\rho(G) \leq \sqrt{m}$ for all C_4-free graphs of size m. Zhai, Lin, Shu [6] showed that $\rho(G) \leq \frac{1+\sqrt{4m-3}}{2}$ for any C_5-free graph of size $m \geq 8$ or C_6-free graph of size $m \geq 22$, with equality if and only if $G \cong S_{\frac{m+3}{2},2}$. In [7], Li and Peng determined the maximum spectral radius of graphs with no intersecting odd cycles. Let F_k be the graph obtained from k triangles sharing a common vertex. Then F_2 is the graph $H(3,3)$. In 2023, Li, Lu and Peng [8] characterized F_2-free graphs with given number of edges. They proved that the unique m-edge $H(3,3)$-free spectral extremal graph is the join of K_2 with an independent set of $\frac{m-1}{2}$ vertices (that is, $S_{\frac{m+3}{2},2}$) if $m \geq 8$, and the condition $m \geq 8$ is tight. The problem has been investigated for various graphs H, as can be seen in [5], [9]–[16].

The objective of this paper is to determine the maximum spectral radius of the graphs when $\mathcal{H} = \{H(3,3), H(3,4)\}$. This is stated in the following theorem.

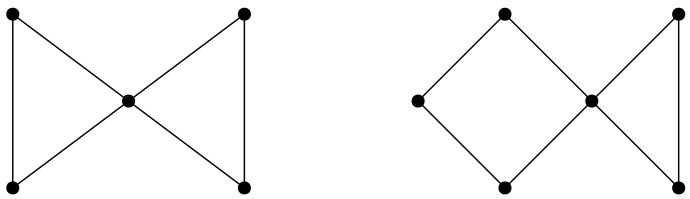


Figure 1: Graphs (a) H(3,3) and (b) H(4,3)

Theorem 1.1. *If G is an $\{H(3,3), H(3,4)\}$-free connected graph with odd size $m \geq 259$, and G contains no isolated vertices, then $\rho(G) \leq \frac{1+\sqrt{4m-3}}{2}$, unless $G \cong S_{\frac{m+3}{2},2}$.*

Coincidentally, the above bound is same as obtained by Li et al. [8] for $H(3,3)$-free graphs.

In this paper, we use G^* to represent the connected graph with maximum spectral radius among all graphs of size m that are free of induced subgraphs $H(3,3)$ and $H(4,3)$. Let $\rho^* = \rho(G^*)$, and consider X^* as the Perron vector of G^* with coordinates x_u corresponding to the vertex $u \in V(G^*)$. Let $x_{u^*} = \max\{x_u : u \in V(G^*)\}$, representing the coordinate associated with the vertex u^* in G^*. We term G^* as the extremal graph and u^* as the extremal vertex of G^*.
Define $A(G^*) = A$, and let $N_0(u^*) = \{v : v \in N(u^*), d_{N(u^*)}(v) = 0\}$, and $N_1(u^*) = N(u^*) \setminus N_0(u^*)$. Additionally, define $N_j^2(u^*) = \{w \in N^2(u^*) : d_{N_j(u^*)}(w) \geq 1\}$ for $j = 0, 1$ and let $W = V(G) \setminus N[u^*]$. By utilizing the eigenequations of A on u^*, the following equation is obtained

$$\rho^* x_{u^*} = (AX^*)u^* = \sum_{u \in N_0(u^*)} x_u + \sum_{v \in N(u^*) \setminus N_0(u^*)} x_v. \tag{1}$$

As ρ^{*2} represents the spectral radius of A^2, the eigenequations of A^2 on u^* lead to

$$\rho^{*2} x_{u^*} = d(u^*)x_{u^*} + \sum_{u \in N(u^*) \setminus N_0(u^*)} d_{N(u^*)}(u)x_u + \sum_{w \in N^2(u^*)} d_{N(u^*)}(w)x_w. \tag{2}$$

The rest of the paper is organised as follows. In Section 2, we present lemmas which will be required to prove Theorem 1.1 in Section 3.

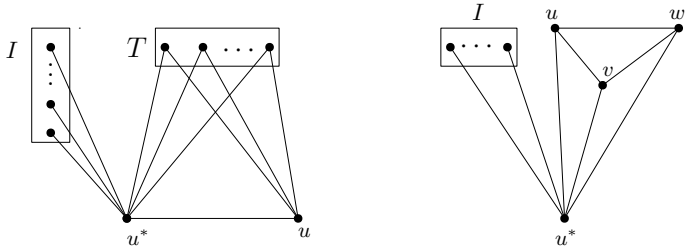


Figure 2: Graphs (a) $G(m,t)$ and (b) K_4^m

2 Lemmas

In this section, we present several lemmas that will be required to prove the main theorem.

Lemma 2.1. [5] *Let A and A' be the adjacency matrices of two graphs G and G' on the same vertex set. Suppose that $N_G(u) \subsetneq N_{G'}(u)$ for some vertex u. If some positive eigenvector to $\rho(G)$ satisfies $X'A'X \geq X'AX$, then $\rho(G') > \rho(G)$.*

Lemma 2.2. [1] *Let G be a connected graph and let H be a proper subgraph of G. Then $\rho(H) < \rho(G)$.*

Definition 2.1. [1] *Given a graph G, the vertex partition $P : V(G) = V_1 \cup V_2 \cup \cdots \cup V_k$ is said to be an equitable partition if, for each $v \in V_i$, $|V_j \cap N(v)| = c_{ij}$ is a constant depending only on $i,\ j$ $(1 \leq i,\ j \leq k)$. The matrix $A_P = (c_{ij})$ is called the quotient matrix of G with respect P.*

Lemma 2.3. [1] *Let $P : V(G) = V_1 \cup V_2 \cup \cdots \cup V_k$ be an equitable partition of G with quotient matrix A_P. Then $\det(xI - A_P) \mid \det(xI - A(G))$. Furthermore, the largest eigenvalue of A_P is just the spectral radius of G.*

Lemma 2.4. [4] *Let G be a graph of size m without isolated vertices. If G is triangle free, then $\rho(G) \leq \sqrt{m}$, with equality if and only if G is a complete bipartite graph.*

The following lemma shows that the spectral radius of the book graph $S_{\frac{m+3}{2},2}$ is greater than or equal to the spectral radius of $G(m,t)$.

Lemma 2.5. *Let $G(m,t)$ be the graph as shown in Figure 2(a) with $m > t+2$. If $t \geq 0$ is even, then $\rho(G(m,t)) \leq \rho(S_{\frac{m+3}{2},2})$, and equality holds when $t = 0$.*

Proof. The quotient matrix of $G(m,t)$ corresponding to the partition $P_3 : V(G(m,t)) = \{u^*\} \cup \{u\} \cup T \cup I$, where $|I| = t$ and $|T| = \frac{m-t-1}{2}$, is

$$A_{P_3} = \begin{array}{c} \\ \{u^*\} \\ \{u\} \\ T \\ I \end{array} \begin{array}{c} \begin{array}{cccc} \{u^*\} & \{u\} & T & I \end{array} \\ \begin{pmatrix} 0 & 1 & \frac{m-t-1}{2} & t \\ 1 & 0 & \frac{m-t-1}{2} & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

Let $f(x) = \det(xI_4 - A_{P_3}) = x^4 - mx^2 - (m-t-1)x + \frac{t}{2}(m-t-1)$. The largest root of $S_{\frac{m+3}{2},2}$ satisfies $g(x) = x^3 - mx - (m-1)$. Then $h(x) = f(x) - xg(x) = tx + \frac{t}{2}(m-t-1) \geq 0$ for $x > 0$ and $t \geq 0$. This demonstrates that the largest root of $g(x)$ is greater than or equal to the largest root of $f(x)$. Hence the result follows by Lemma 2.3. □

Now, we show that the spectral radius of the book graph is strictly greater than the spectral radius of K_4^m.

Lemma 2.6. *Let K_4^m be the graph of size m, where $m \geq 8$, shown in Figure 2(b). Then $\rho(K_4^m) < \rho(S_{\frac{m+3}{2},2})$.*

Proof. The vertex set of K_4^m has equitable partition $P_4 : V(K_4^m) = \{u^*\} \cup R \cup I$, where $R = \{u, v, w\}$ and the quotient matrix with respect to P_4 is

$$A_{P_3} = \begin{array}{c} \\ \{u^*\} \\ R \\ I \end{array} \begin{array}{c} \begin{array}{ccc} \{u^*\} & R & I \end{array} \\ \begin{pmatrix} 0 & 3 & m-6 \\ 1 & 2 & 0 \\ 1 & 0 & 0 \end{pmatrix} \end{array}.$$

Then $f(x) = \det(xI_3 - A_{P_3}) = x^3 - 2x^2 - (m-3)x + 2(m-6)$. If $\rho_1 = \rho(S_{\frac{m+3}{2},2})$, then $\rho_1^2 = \rho_1 + (m-1)$. Therefore, we have

$$\begin{aligned} f(\rho_1) &= \rho_1^2 + (m-1)\rho_1 - 2(\rho_1 + m - 1) - (m-3)\rho_1 + 2(m-6) \\ &= \rho_1 + m - 11. \end{aligned}$$

Given that $m \geq 8$, it follows that $f(\rho_1) > 0$. Additionally, the derivative $f'(x) = 3x^2 - 4x - (m-3) > 0$ holds for $x \geq \rho_1$. Consequently, by Lemma 2.3, it can be inferred that $\rho(K_4^m) < \rho(S_{\frac{m+3}{2},2})$. □

In the following lemma, we prove that any vertex of degree one in G^* is joined to the extremal vertex u^*. In particular this shows that $d(v) \geq 2$ for any $v \in W$.

Lemma 2.7. *Let* G^* *be a graph forbidding the subgraphs* $H(3,3)$ *and* $H(4,3)$*, and* u^* *be an extremal vertex in* G^**. Then every pendent vertex in* G^* *is joined to* u^**.*

Proof. Suppose there exists a vertex $w \in V(G^*) \setminus N(u^*)$ with $d(w) = 1$. Let $N(w) = \{v\}$. Consider the graph G' obtained by removing the edge wv from G^* and adding the edge wu^*. The resulting graph G' has m edges and is free of induced subgraphs $H(3,3)$ and $H(4,3)$. Furthermore, $N_{G^*}(u^*) \subsetneq N_{G'}(u^*)$, and the following inequality holds.

$$\begin{aligned}\sum_{uz\in E(G')} x_u x_z &= \sum_{uz\in E(G^*)} x_u x_z + x_w(x_u^* - x_v)\\ &\geq \sum_{uz\in E(G^*)} x_u x_z.\end{aligned}$$

According to Lemma 2.1, this implies that $\rho(G') > \rho(G^*)$. However, this contradicts the definition of G^*. □

In the following lemma we give an upper bound of $e(W)$.

Lemma 2.8. *Let* G *be a graph of size* m *and* $W = V(G) \setminus N[u^*]$*. If* $\rho \geq \frac{1+\sqrt{4m-3}}{2}$*, then*

$$e(W) \leq e(N(u^*)) - |N(u^*) \setminus N_0(u^*)| + 1.$$

Proof. From Eqs. 1 and 2, we can deduce the following expression

$$\begin{aligned}(\rho^2 - \rho)x_u^* = d(u^*)x_{u^*} &+ \sum_{u\in N(u^*)\setminus N_0(u^*)} (d_{N(u^*)}(u) - 1)x_u + \sum_{w\in W} d_{N(u^*)}(w)x_w\\ &- \sum_{u\in N_0(u^*)} x_u.\end{aligned}$$

Given $\rho^2 - \rho \geq m - 1$, it implies that

$$\begin{aligned}(m-1)x_u^* &\leq d(u^*)x_{u^*} + \sum_{u \in N(u^*)\setminus N_0(u^*)} (d_{N(u^*)}(u) - 1)x_u + \sum_{w \in W} d_{N(u^*)}(w)x_w \\ &\quad - \sum_{u \in N_0(u^*)} x_u \\ &\leq \left\{ d(u^*) + e(N(u^*), W) + \sum_{u \in N(u^*)\setminus N_0(u^*)} (d_{N(u^*)}(u) - 1) \right. \\ &\quad \left. - \sum_{u \in N_0(u^*)} \frac{x_u}{x_u^*} \right\} x_u^* \\ &= \left\{ m - e(W) + e(N(u^*)) - |N(u^*) \setminus N_0(u^*)| - \sum_{u \in N_0(u^*)} \frac{x_u}{x_u^*} \right\} x_u^*.\end{aligned}$$

This yields

$$\begin{aligned}e(W) &\leq e(N(u^*)) - |N(u^*) \setminus N_0(u^*)| + 1 - \sum_{u \in N_0(u^*)} \frac{x_u}{x_u^*} \\ &\leq e(N(u^*)) - |N(u^*) \setminus N_0(u^*)| + 1. \qquad (3)\end{aligned}$$

This concludes the proof. □

3 Proof of the main theorem

Since G^* does not contain a subgraph isomorphic to $H(3,3)$, it implies that $G^*[N(u^*)]$ cannot have two or more independent edges. As a result, $G^*[N(u^*)]$ can be categorized into one of the following (1) it either consists solely of isolated vertices, or (2) it includes isolated vertices along with a copy of a star S_k, where $k \leq \frac{m+1}{2}$, or (3) it comprises isolated vertices and a triangle.

The following lemma shows that the only non trivial component of $G^*[N(u^*)]$ is a star.

Lemma 3.1. *Let G^* be the graph with maximum spectral radius among all $\{H(3,3), H(4,3)\}$-free graphs with odd size $m \geq 259$ and let $X^* = (x_1, x_2, \cdots, x_n)^T$ be the Perron vector of G^*, and $x_{u^*} = max\{x_i : i \in V(G^*)\}$. If $\rho(G^*) \geq \frac{1+\sqrt{4m-3}}{2}$, then $G^*[N(u^*)]$ has exactly one non trivial component S_k, where $k \leq \frac{m+1}{2}$.*

Proof. First, we suppose that $G^*[N(u^*)]$ consists of isolated vertices only. Since the star S_{m+1} with m edges does not include $H(3,3)$ and $H(4,3)$ as subgraphs, consequently, due to the extremality of G^*, it follows that $\rho(G^*) \geq \rho(S_{m+1}) = \sqrt{m}$. Then,

from Eq. 2, we derive the following inequality

$$\begin{aligned}(m-d(u^*))x_{u^*} &= \sum_{u\in N(u^*)\setminus N_0(u^*)} d_{N(u^*)}(u)x_u + \sum_{w\in N^2(u^*)} d_{N(u^*)}(w)x_w \\ &\leq (2e(N(u^*)) + e((N(u^*), N^2(u^*)))x_{u^*}.\end{aligned}$$

This leads to the conclusion that $e(W) \leq e(N(u^*))$. In other words, $e(W) = 0$. According to Lemma 2.7, there are no pendent vertices in W. Consequently, G^* is a triangle-free graph. Therefore, by Lemma 2.4, G^* is a complete bipartite graph. Thus, $\rho(G^*) = \sqrt{m} < \frac{1+\sqrt{4m-3}}{2}$ for $m \geq 2$, which is a contradiction.

Next, suppose that $G^*[N(u^*)]$ contains a copy of K_3. In this case, we observe that $d_{N(u^*)}(w) \leq 1$. It is clear that K_4^m does not include $H(3,3)$ and $H(4,3)$ as subgraphs. As S_{m-2} is a proper subgraph of K_4^m, according to Lemma 2.2, we must have $\rho(K_4^m) > \rho(S_{m-2}) = \sqrt{m-3}$. Moreover, the definition of G^* implies that $\rho(G^*) > \rho(K_4^m) > \sqrt{m-3}$. Then, based on Eq. 2 and using $x_u^* \geq x_u$ for any vertex u, it can be deduced that

$$\begin{aligned}(m-d(u^*)-3)x_{u^*} &< (\rho^{*2} - d(u^*)x_{u^*} \\ &\leq (2e(N(u^*)) + e((N(u^*), N^2(u^*)))x_{u^*}.\end{aligned}$$

This implies that $e(W) \leq 5$. We claim that $e(W) = 0$. To prove the claim, we consider the following possibilities.

Case 1. $2 \leq \mathrm{e(W)} \leq 5$.

Given that $e(W) \leq 5$, it follows that $d(w) \leq 6$ for any $w \in W$ and $d(u) \leq 8$ for any vertex $u \in N_1(u^*)$. Additionally, as $x_u^* \geq x_u$ for any vertex u, we obtain the following inequalities for any vertex $w \in W$ and any vertex $u \in N_1(u^*)$.

$$x_w \leq \frac{6}{\rho^*}x_u^* \text{ and } x_u \leq \frac{8}{\rho^*}x_u^*. \tag{4}$$

From Eq. 2 and inequalities 4, we have

$$\begin{aligned}\rho^{*2}x_{u^*} &= d(u^*)x_{u^*} + \sum_{u\in N(u^*)\setminus N_0(u^*)} d_{N(u^*)}(u)x_u + \sum_{w\in N^2(u^*)} d_{N(u^*)}(w)x_w \\ &\leq \left\{d(u^*) + \frac{8}{\rho^*}\sum_{u\in N(u^*)\setminus N_0(u^*)} d_{N(u^*)}(u) + \frac{6}{\rho^*}\sum_{w\in N^2(u^*)} d_{N(u^*)}(w)\right\}x_{u^*} \\ &= \left\{d(u^*) + \frac{16}{\rho^*}e(N_1(u^*) + \frac{6}{\rho^*}(e(W, N(u^*)))\right\}x_{u^*}.\end{aligned}$$

Since $\rho^* > \sqrt{m-3} \geq 16$, from above, we have

$$\begin{aligned} \rho^{*2} x_{u^*} &< \left\{ d(u^*) + e(N_1(u^*) + \frac{6}{16}(e(W, N(u^*))) \right\} x_{u^*} \\ &= \left\{ m - e(W) - \frac{10}{16}(e(W, N(u^*))) \right\} x_{u^*}. \end{aligned} \tag{5}$$

Recall that $d(w) \geq 2$ for any $w \in W$. Therefore

$$e(W, N(u^*)) \geq \begin{cases} 2 & e(W) = 2, \\ 1 & e(W) \geq 3. \end{cases}$$

Using this in 5, we conclude that $\rho^* < \sqrt{m-3}$, leading to a contradiction.

Case 2. $\mathrm{e(W) = 1}$.

In this situation, we observe that $x_w \leq \frac{2}{\rho^*} x_u^*$ for any $w \in W$ and $x_u \leq \frac{4}{\rho^*} x_u^*$ for any $u \in N_1(u^*)$. Following the steps outlined in case 1, we conclude that $\rho^* < \sqrt{m-3}$, contradicting the hypothesis.

Considering both cases 1 and 2, we can deduce that $e(W) = 0$. Consequently, the observation that all pendent vertices are joined to u^* and $d_{N(u^*)}(w) = 1$ for any $w \in N^2(u^*)$ implies that $W = \emptyset$. Hence, G^* is isomorphic to K_4^m when $G^*[N(u^*)]$ contains a copy of K_3. By Lemma 2.6, we have $\rho(K_4^m) < \rho(S_{\frac{m+3}{2},2}) = \frac{1+\sqrt{4m-3}}{2}$, which is again a contradiction. This completes the proof of the lemma. □

Proof. **(Theorem 1.1.)** As $G^*[N(u^*)]$ is a tree, by Lemma 2.8, we have $e(N(u^*)) - |N(u^*) \backslash N_0(u^*)| = -1$. Considering that $S_{\frac{m+3}{2},2}$ does not contain $H(3,3)$ and $H(4,3)$ as subgraphs, and by the definition of G^*, it follows that $\rho(G^*) \geq \rho(S_{\frac{m+3}{2},2}) = \frac{1+\sqrt{4m-3}}{2}$. According to Lemma 2.8, this implies that

$$e(W) = 0 \tag{6}$$

and $W = N^2(u^*)$. In view of Lemma 3.1, $G^*[N(u^*)]$ includes isolated vertices along with a copy of a star S_k, where $k \leq \frac{m+1}{2}$. Now, we consider the following cases.

Case 1. $\mathrm{G^*[N(u^*)]}$ **contains a copy of** $\mathrm{K_2}$.

Consider the unique edge v_1v_2 in $G^*[N_1(u^*)]$. In view of Lemma 2.7, there are no pendent vertices in W. Also G^* is restricted from containing the graphs $H(3,3)$ and $H(4,3)$, no vertex in W can have two distinct neighbors in $N_0(u^*)$. Therefore,

$d_{N(u^*)}(w) \leq 3$ for any $w \in W$.

Assume that $w \in W$ and $d_{N(u^*)}(w) = 3$. This implies that $W = \{w\}$ and w is adjacent to v_1, v_2 and some vertex u_1 in $N_0(u^*)$. Let G_1 be the graph obtained from G^* by deleting the edge wu_1 and adding the edge wu^*. The graph G_1 is $\{H(3,3), H(4,3)\}$-free, and $N_{G^*}(u^*) \subsetneq N_{G_1}(u^*)$. Also, $x_{u^*} \geq x_{u_1}$. According to Lemma 2.1, this implies that $\rho(G_1) > \rho(G^*)$. However, this contradicts the definition of G^*. Therefore, $d_{N(u^*)}(w) \leq 2$.

Suppose that $d_{N(u^*)}(w) = 2$ for some $w \in W$. If w is adjacent to both v_1 and v_2, then $N_0^2(u^*) \cap N_1^2(u^*) = \emptyset$. Consequently, $N^2(u^*) = N_1^2(u^*)$. Let $w_1, w_2, ..., w_s$ be the vertices in $N_1^2(u^*)$ that are adjacent to both v_1 and v_2. Consider $G_2 = G^* - \{w_i v_1 : w_i \in W\} + \{w_i u^* : w_i \in W\}$. As previously established by Lemma 2.1, we derive that $\rho(G_2) > \rho(G^*)$, which is not feasible.

Now, let each $w \in W$ be adjacent to one vertex in $N_0(u^*)$ and one vertex in $N_1(u^*)$. Let $w_1, w_2 \in W$ such that $N_{N(u^*)}(w_1) = \{v_1, u_1 : u_1 \in N_0(u^*)\}$ and $N_{N(u^*)}(w_2) = \{v_2, u_2 : u_2 \in N_0(u^*)\}$. Without loss of generality, assume that $x_{v_1} \geq x_{v_2}$. Consider $G_3 = G^* - w_2v_2 + w_2v_1$. The graph G_3 is $\{H(3,3), H(4,3)\}$-free, and $N_{G^*}(v_1) \subsetneq N_{G_3}(v_1)$. Therefore, by Lemma 2.1, we have $\rho(G_3) > \rho(G^*)$, contradicting the definition of G^*. Thus, every $w \in W$ is adjacent to either v_1 or v_2 and some vertex $u \in N_0(u^*)$. Assume that every $w \in W$ is adjacent to v_1. Since $d(w) = 2$, so w is adjacent to v_1 and some vertex u_1 in $N_0(u^*)$. Applying the same procedure as before, we obtain the graph G_4 from G^* by deleting the edge wu_1 and adding the edge wu^*. Clearly, the hypothesis of Lemma 2.1 holds, and it follows that $\rho(G_4) > \rho(G^*)$. Thus, we conclude that $W = \emptyset$, and hence G^* is isomorphic to $G(m,t)$, where $t = m - 3$. By Lemma 2.5, we have $\rho(G(m, m-3)) < \rho(S_{\frac{m+3}{2},2})$ and thus contradicting the definition of G^*.

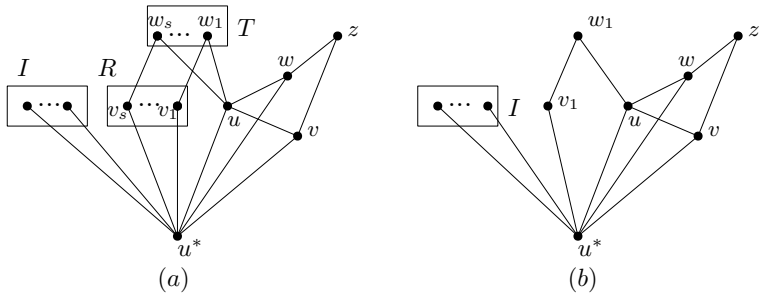


Figure 3: Graphs (a) H and (b) H_1

Case 2. $\mathrm{G}^*[\mathrm{N}(\mathrm{u}^*)]$ **contains a copy of star** S_3.

Let u, v and w be the vertices of the star component, with u as its center. The following observations apply to any vertex $w \in W$

i. w cannot be adjacent to any two vertices in $N_0(u^*)$.

ii. w cannot be adjacent to both u and v, or both u and w.

iii. w cannot be adjacent to v (or w) and a vertex in $N_0(u^*)$.

Consider $z \in W$ such that $N_{N(u^*)}(z) = \{v, w\}$, and let $T = \{w_1, \cdots, w_s\}$ represent the set of vertices in W, and $R = \{v_1, \cdots, v_s\}$ represent the set of vertices in $N(u^*)$, satisfying $N_{N(u^*)}(w_i) = \{u, v_i\}$ for $i = 1, \cdots, s$. Denote this graph by H, as depicted in Figure 3(a). We aim to demonstrate that $\rho(H) < \rho(H_1)$, where H_1 is a graph with m edges and $|T| = 1$ as illustrated in Figure 3(b). To facilitate the analysis, we partition the vertex set of H as $P : \{u^*\} \cup \{u\} \cup \{v, w\} \cup R \cup I \cup T \cup \{z\}$, where $|T| = |R| = t$, $|I| = m - 3t - 7$. The quotient matrix of H concerning the partition P is given by

$$A_P = \begin{array}{c} \\ \{u^*\} \\ \{u\} \\ \{v,w\} \\ R \\ I \\ T \\ \{z\} \end{array} \begin{array}{c} \begin{array}{ccccccc} \{u^*\} & \{u\} & \{v,w\} & R & I & T & \{z\} \end{array} \\ \begin{pmatrix} 0 & 1 & 2 & t & m-3t-7 & 0 & 0 \\ 1 & 0 & 2 & 0 & 0 & t & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

Consider the polynomial $\phi_t(x) = \det(xI_7 - A_P) = x^7 + (-m+t-1)x^5 - 4x^4 + (mt - 2t^2 + 5m - 16t - 26)x^3 + (-4t+4)x^2 + (-2mt + 4t^2 - 4m + 30t + 26)x$. Consequently, $\phi_1(x) = x^7 - mx^5 - 4x^4 + (6m - 44)x^3 + (-6m + 60)x$, and the difference $f(x) = \phi_t(x) - \phi_1(x) = x(t-1)g(x)$, where $g(x) = x^4 + (m - 2t - 18)x^2 - 4x - 2m + 4t + 34$. To establish $f(x) > 0$, it suffices to show that $g(x) > 0$. Given that $2 \leq t \leq \frac{m-7}{3}$, we have

$$\begin{aligned} g(x) &\geq x^4 + (m - 2(\frac{m-7}{3}) - 18)x^2 - 4x - 2m + 42 \\ &= \frac{1}{3}(3x^4 + (m - 40)x^2 - 12x - 6m + 126). \end{aligned}$$

Furthermore, considering that $m \geq 259$, it follows that

$$g\left(\frac{\sqrt{m}}{2}\right) = \frac{5}{16}m^2 - 16m - 6\sqrt{m} + 126 > 0.$$

Now, for $x > \frac{\sqrt{m}}{2}$, the derivative of g is $g'(x) = 4x^3 + \frac{2}{3}(m-40)x - 12 > 0$, and $\frac{\sqrt{m}}{2} < \frac{1+\sqrt{4m-3}}{2}$. Consequently, it follows that $f(x) > 0$ for $x > \frac{1+\sqrt{4m-3}}{2}$, leading to the conclusion that $\rho(H_1) > \rho(H)$. We have the following claim.

Claim. $\rho(H_1) < \rho(G(m,t)$, where $t = m-5$.
Proof of the claim. The quotient matrix of the graph $G(m, m-5)$, as depicted in Figure 4(b), concerning the partition $P_1 : \{u^*\} \cup \{u\} \cup R \cup I$ is given by

$$A_{P_1} = \begin{array}{c} \\ \{u^*\} \\ \{u\} \\ R \\ I \end{array} \begin{array}{c} \begin{array}{cccc} \{u^*\} & \{u\} & R & I \end{array} \\ \begin{pmatrix} 0 & 1 & 2 & m-5 \\ 1 & 0 & 2 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

Consider $\phi_2(x) = \det(xI_4 - A_{P_1}) = x^4 - mx^2 - 4x + 2m - 10$. Define

$$\begin{aligned} \psi_1(x) &= \frac{1}{x}\phi_1(x) - x^2\phi_2(x) \\ &= x^6 - mx^4 - 4x^3 + (6m-44)x^2 + (-6m+60) \\ &\quad - (x^6 - mx^4 - 4x^3 + (2m-10)x^2) \\ &= (4m-34)x^2 - 6m + 60. \end{aligned}$$

Observe that $\psi_1'(x) = 2(4m-34)x > 0$ and $\psi_1(x) > 0$, for $x > \frac{1+\sqrt{4m+3}}{2}$. So it is evident that $\rho(G(m, m-5)) > \rho(H_1)$. This completes the proof of the claim.

Consider the case where $|T| = 1$, and let $w_1 \in T$ be adjacent to both u and v_1. In the event that w and v lack a common neighbor in W, form the graph $G_1 = G^* - v_1w_1 + u^*w_1$. Applying Lemma 2.1 leads to a contradiction.

Alternatively, if $T = \emptyset$ and v and w share a common neighbor in W, as illustrated by graph H_2 in Figure 4(a). The quotient matrix of the graph H_2 concerning the partition $P_2 : \{u^*\} \cup \{u\} \cup \{v,w\} \cup \{z\} \cup I$ is given as

$$A_{P_2} = \begin{array}{c} \\ \{u^*\} \\ \{u\} \\ \{v,w\} \\ \{z\} \\ I \end{array} \begin{array}{c} \begin{array}{ccccc} \{u^*\} & \{u\} & \{v,w\} & \{z\} & I \end{array} \\ \begin{pmatrix} 0 & 1 & 2 & 0 & m-7 \\ 1 & 0 & 2 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

The characteristic polynomial of A_{P_2} is $\phi_3(x) = x^5 - mx^3 - 4x^2 + (4m-26)x$. Define $\psi_2(x) = \phi_3(x) - x\phi_2(x) = (2m-16)x > 0$ for $x > 0$. This implies that

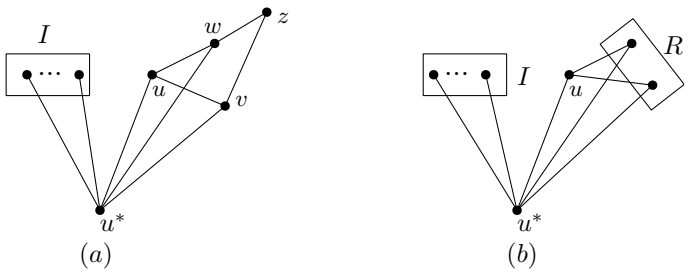


Figure 4: Graphs (a) H_2 and (b) $G(m, m-5)$

$\rho(G(m, m-5)) > \rho(H_2)$.

In light of the preceeding discussion, it follows that G^* is isomorphic to $G(m, t)$, where $t = m-5$. By Lemma 2.5, we have $\rho(G(m, m-5)) < \rho(S_{\frac{m+3}{2},2})$, which contradicts the definition of G^*.

Case 3. $\mathrm{G^*[N(u^*)]}$ **contains a copy of star** $\mathrm{S_k},\ \mathrm{k} \geq 4$.

Consider u as the central vertex of the star S_k. With $\rho(G^*) \geq \frac{1+\sqrt{4m-3}}{2}$, it follows that $e(W) = 0$. Furthermore, as G^* is free of induced subgraphs $H(3,3)$ and $H(4,3)$, it ensures that no two vertices in $N_0(u^*)$ or $N_1(u^*)$ share a common neighbor in W. If $w \in N_0^2(u^*) \cap N_1^2(u^*)$, then $|N_{N(u^*)}(w)| = 2$ and w must be adjacent to u. Let $N_{N(u^*)}(w) = \{u, v : v \in N_0(u^*)\}$. Define $G_1 = G^* - vw + u^*w$. Applying Lemma 2.1 in this context leads to a contradiction. Consequently, $W = \emptyset$ and by Lemma 2.5, it follows that G^* is isomorphic to $S_{\frac{m+3}{2},2}$. This completes the proof of the theorem. □

4 Conclusion

In this paper, we have established that the graph denoted by $S_{\frac{m+3}{2},2}$ has the maximum spectral radius within the class of $\{H(3,3), H(4,3)\}$-free graphs with odd size greater than or equal to 259. For cases where m is less than 259, we identify Lemma 3.1 as a crucial technical obstacle. To investigate and to overcome this obstacle, extending our proof to the case where $m < 259$ would be an interesting problem for further research. Moreover, it's important to highlight that Theorem 1.1 establishes the proof for the case of odd m. This naturally raises the question: What is the maximum spectral radius among $\{H(3,3), H(4,3)\}$-free graphs when the size is even? This question invites further investigation and we leave it as an open problem.

Data Availability: The paper does not use or produce any dataset.

Conflicts of Interest: The authors declare no conflicts of interest.

Acknowledgements: The research of first author is supported by NBHM-DAE research project No. NBHM/02011/20/2022. The research of Amir Rehman is supported by SRF financial assistance by Council of Scientific and Industrial Research (CSIR) New Delhi, India.

References

[1] D. Cvetković, P. Rowlinson, and S. Simić, *An introduction to the theory of graph spectra* (London Mathematical Society Student Texts (75)). Cambridge University Press, 2009 (⇒ 125, 127).

[2] S. Pirzada, *An Introduction to graph theory*. Universities Press, Orient BlackSwan, Hyderabad, 2012 (⇒ 125).

[3] V. Nikiforov, "The spectral radius of graphs without paths and cycles of specified length," *Linear Algebra and its Applications*, vol. 432, no. 9, pp. 2243–2256, 2010 (⇒ 125).

[4] E. Nosal, "Eigenvalues of graphs (Master's thesis)," 1970 (⇒ 125, 127).

[5] V. Nikiforov, "The maximum spectral radius of C_4-free graphs of given order and size," *Linear Algebra and its Applications*, vol. 430, no. 11-12, pp. 2898–2905, 2009 (⇒ 125, 127).

[6] M. Zhai, H. Lin, and J. Shu, "Spectral extrema of graphs with fixed size: Cycles and complete bipartite graphs," *European Journal of Combinatorics*, vol. 95, p. 103 322, 2021 (⇒ 125).

[7] Y. Li and Y. Peng, "The spectral radius of graphs with no intersecting odd cycles," *Discrete Mathematics*, vol. 345, no. 8, p. 112 907, 2022 (⇒ 125).

[8] Y. Li, L. Lu, and Y. Peng, "Spectral extremal graphs for the bowtie," *Discrete Mathematics*, vol. 346, no. 12, p. 113 680, 2023 (⇒ 125, 126).

[9] M.-Z. Chen, A.-M. Liu, and X.-D. Zhang, "Spectral extremal results with forbidding linear forests," *Graphs and Combinatorics*, vol. 35, pp. 335–351, 2019 (⇒ 125).

[10] M.-Z. Chen, A.-M. Liu, and X.-D. Zhang, "On the spectral radius of graphs without a star forest," *Discrete Mathematics*, vol. 344, no. 4, p. 112 269, 2021 (⇒ 125).

[11] S. Cioabă, D. N. Desai, and M. Tait, "The spectral radius of graphs with no odd wheels," *European Journal of Combinatorics*, vol. 99, p. 103 420, 2022 (⇒ 125).

[12] L. Feng, G. Yu, and X.-D. Zhang, "Spectral radius of graphs with given matching number," *Linear Algebra and its Applications*, vol. 422, no. 1, pp. 133–138, 2007 (⇒ 125).

[13] V. Nikiforov, *Some new results in extremal graph theory*. Surveys in Combinatorics, Cambridge University Press, 2012 (⇒ 125).

[14] V. Nikiforov, "A spectral condition for odd cycles in graphs," *Linear Algebra and its Applications*, vol. 428, no. 7, pp. 1492–1498, 2008 (⇒ 125).

[15] M. Zhai, B. Wang, and L. Fang, "The spectral Turán problem about graphs with no 6-cycle," *Linear Algebra and its Applications*, vol. 590, pp. 22–31, 2020 (⇒ 125).

[16] L.-P. Zhang and L. Wang, "The maximum spectral radius of graphs without spanning linear forests," *Graphs and Combinatorics*, vol. 39, no. 1, p. 9, 2023 (⇒ 125).

Received: 30.04.2024; *Revised: 15.05.2024*; *Accepted: 16.05.2024*