Proceedings of the 9th Student Computing Research Symposium (SCORES'23)

Koper, Slovenia
October 5, 2023

Ina Bašić
Nina Chiarelli
Matjaž Krnc
Domen Šoberl
(Eds.)

https://www.scores.si

Kataložni zapis o publikaciji (CIP) pripravili v Narodni in univerzitetni knjižnici v Ljubljani
COBISS.SI-ID 163810307
ISBN 978-961-293-253-4 (Univerza na Primorskem, PDF)

Proceedings of the 9th Student Computing Research Symposium (SCORES'23)

Koper, Slovenia
October 5, 2023

Ina Bašić
Nina Chiarelli
Matjaž Krnc
Domen Šoberl
(Eds.)

Title **Proceedings of the 9th Student Computing Research Symposium (SCORES'23)**

Editors Ina Bašić
(Faculty of Mathematics, Natural Sciences and Information Technologies)

Nina Chiarelli
(Faculty of Mathematics, Natural Sciences and Information Technologies)

Matjaž Krnc
(Faculty of Mathematics, Natural Sciences and Information Technologies)

Domen Šoberl
(Faculty of Mathematics, Natural Sciences and Information Technologies)

Conference **9th Student Computing Research Symposium (SCORES'23)**

Venue University of Primorska
Faculty of Mathematics, Natural Sciences and Information Technologies
Glagoljaška 8, SI-6000 Koper, Slovenia

Date October 5, 2023

Published by **University of Primorska Press**
Titov trg 4, SI-6000 Koper, Slovenia
https://www.hippocampus.si/, zalozba@upr.si

Co-published by **University of Maribor**
Faculty of Electrical Engineering and Computer Science
Koroška cesta 46, SI-2000 Maribor, Slovenia
https://feri.um.si/en/, feri@um.si

Co-published by **University of Ljubljana**
Faculty of Computer and Information Science
Večna pot 113, SI-1000 Ljubljana, Slovenia
https://www.fri.uni-lj.si/en, dekanat@fri.uni-lj.si

Edition 1st

Publication type E-book

Published Koper, Slovenia, October 2023

ISBN 978-961-293-253-4

DOI https://doi.org/10.26493/scores23

Organisers and sponsors:

Editors' Foreword

In the ever-evolving landscape of computer science, where innovation knows no bounds, we welcome you to the 2023 Student Computing Research Symposium (SCORES). This annual gathering is a testament to the unwavering commitment of computer science faculties from all three Slovene public universities: the University of Ljubljana, the University of Maribor, and the University of Primorska Faculty of Mathematics, Natural Sciences, and Information Technologies (UP FAMNIT), which takes the helm as the primary organizer this year.

Recent years have witnessed remarkable strides in computer science, from breakthroughs in artificial intelligence and machine learning to advancements in cloud computing and quantum computing. In this rapidly evolving field, nurturing the next generation of computer scientists becomes more critical than ever. SCORES 2023 is our platform to empower undergraduate and master's students to contribute their unique insights and solutions to the ever-expanding frontiers of computer science. Our mission is clear: to provide a stage where these talented students can showcase their research, ideas, and innovations. The Student Computing Research Symposium is about bridging the gap between academic knowledge and real-world applications, and it's an opportunity for these young minds to not only present their work but also connect with their peers and mentors.

This year's conference program boasts a rich diversity of topics, all authored by students from various academic institutions. From user interface design for predicting football match results to real-time vehicle speed estimation from video data, the papers featured here offer fresh perspectives and practical solutions to challenges that impact our lives. We also delve into areas such as speech recognition, algorithm evaluation, community identification, procedural content generation, and much more.

As we embark on this exciting journey through the world of computer science, we invite you to join us in celebrating the dedication, creativity, and accomplishments of these emerging computer scientists. Their work represents the future of our field, and the Student Computing Research Symposium is the platform where it all comes to life.

SCORES 2023 organizing committee:
Ina Bašić, Nina Chiarelli, Matjaž Krnc, Domen Šoberl

Conference Program

Plenary Speakers

Marko Grobelnik

AiLab, Jožef Stefan Institute

Knowledge Graphs and Large Language Models

Large Language Models have many nice properties, but one which is not well spotted is extracting and manipulating structured knowledge. In the talk we'll overview and demonstrate a set of scenarios where LLMs can help transitioning between the textual content and structured knowledge based representations like knowledge graphs, first order logic and Prolog programming language. Scenarios will include also some of the more or less standard text manipulation tasks like entity extraction, emotion extraction, consequence generation and reasoning with the content.

Marko Grgurovič

Triternion & UP FAMNIT

Research and Development in the Video Game Industry

In this talk we will look at the changes and challenges in the video game industry over the last decade. With the advent of off-the-shelf engines and tech stacks, we have seen consolidation in terms of technology and a focus on content. But what is involved in the creation of a modern AA to AAA cross-platform title? Do video game companies engage in research? Do they benefit from published results? What kinds of problems do they face and how do they solve them?

Mouse cursor control using 3D head pose estimation

Blaž Kovačič
blaz.kovacic@student.um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

ABSTRACT

We developed a method of computer mouse cursor control based on 3D head pose estimation using a web camera image stream. This method can be especially applicable in the field of Assistive Technology interfaces for disabled persons. Our approach stands out by enabling direct conversion of head pose into an absolute position of the mouse cursor, the stability of which was achieved using a special approach of facial key-point filtering. In order to estimate head pose rotation angles we made use of the iterative version of the Perspective-n-Point algorithm together with a seven-point (symmetric) 3D head model, that adjusts itself to the head shape based on facial landmark positions in the starting head position. We demonstrate the effectiveness of our approach using practical comparative measurements.

KEYWORDS

Accessibility, 3D head pose estimation, Perspective-n-Point problem, dlib library, Human-computer interaction

1 INTRODUCTION

According to the World Health Organization, about 15 % of the world's population lives with some form of disability, out of which 2-4 % have significant difficulties in functioning [WHO 2011]. The field of Assistive Technology (AT) is concerned with the development of accessories that help persons with disabilities such as quadriplegia or cerebral palsy, in their everyday lives, work, communication, and have an impact on the general improvement of their quality of life.

Persons with special needs can make use of their devices through physical contact (for example using switches), with the use of facial movement (for example "Face Control" Android app [Obstino 2022]), with head movement (for example a gyroscopic mouse), with eye gaze (eye trackers), or by using their voice (for example Talon or Dragon Naturally Speaking [Nowogrodzki 2018]).

The choice of AT solution often depends on the type of disability and available mobility that a person has. For example, solutions that provide cursor control through head tracking, while requiring mobility of the head, may provide easier and more accurate cursor control for some disabled users compared to eye trackers [Zapała and Bałaj 2012], where the head is often required to be held still. Additional benefit of head trackers is that they can also be implemented in software, requiring only a web camera, presenting a cost effective AT solution.

Certain webcam head tracking solutions already exist on the application market, for example the CameraMouse [Betke et al. 2002] and eViaCam [Mauri 2019] PC software. The downside of the mentioned applications is that their underlying algorithm converts relative head motion into relative cursor movement, resulting in the effect that, with use, cursor will move to a position that is different from the current head gaze direction. For example, when we move the head from the starting position in a certain direction, and then back into the starting position, the mouse cursor is not at the same position anymore, but at a position that doesn't coincide with the head gaze direction, which might make it difficult for the end-user to interact with the computer.

To the best of our knowledge, an application solution that would remove the mentioned weakness doesn't yet exist on the application market, so we decided to research the feasibility of the approach of using the webcam image stream to convert 3D head pose into an *absolute* mouse cursor position, such that it always corresponds to the head gaze direction. Such an approach may potentially offer a favorable alternative to existing implementations.

2 RELATED WORK

[Nabati and Behrad 2010] developed an application solution for disabled persons, which captures an image stream using a monocular camera, and uses the head pose estimate (rotation, translation) using four marked points on the face to convert it into corresponding mouse movement. In their approach they divided the solution on a movement information retrieval module (3D head pose), and mouse movement module. When estimating the 3D head pose they assumed that 4 points on the face are on the same plane (a "N-point" planar face model where the front of the face is modeled as a flat surface) to obtain 3D rotation and translation between the web camera and head pose. Their approach of a 4-point head model is interesting and we expand upon the idea.

[Fu and Huang 2007] describe an approach of using the relative position of the head's bounding box in webcam image to set coarse cursor position on the screen, and they fine tune cursor position by means of estimating the head rotation. Although not specifically aimed at disabled population, the approach is interesting as it uses both head translation and rotation for fine cursor control.

[Tu et al. 2005] describe a 3D model based camera mouse control, implementing what they termed "direct mode" as a one-to-one mapping from obtained head rotations to cursor screen coordinates. Though exhibiting a degree of cursor localization error, with ease of implementation not being it's strong point, the approach does shows promise and prompts us to further research this method.

DOI: https://doi.org/10.26493/scores23.02

3 PROBLEM DESCRIPTION

Our goal was to develop a solution that uses the web camera image stream to estimate the 3D head pose and convert the obtained head rotation angles to absolute mouse cursor position such that it always corresponds to the direction of the head gaze.

The primary challenge that we address in this work is obtaining very stable estimates of head rotation angles through facial key-point filtering, and implementing a mapping function to translate head rotations to cursor screen coordinates.

The problem can be divided into the following parts: 1) image acquisition and face region detection, 2) detection of facial landmarks within the face region, 3) head modeling (for example an N-point 3D model), 4) 3D head pose estimation, 5) stabilization of the obtained head rotation angles, and 5) conversion of the head rotation angles into stable mouse movements.

3D head pose estimation problem

3D head pose estimation is a subset of the more general problem called pose estimation, where the 3D pose (rotation and translation) of an arbitrary object is being estimated. A 3D pose can be estimated by minimizing the so-called reprojection error. A reprojection error is the smallest at that rotation and translation of object points, such that the difference between the positions of those points projected from the 3D object coordinates to the 2D image plane, and between corresponding (observed landmark) points in the captured image is as small as possible (see Fig. 1).

The problem of reprojection error minimization is solved by the so-called Perspective-n-Point (PnP) algorithm [OpenCV.org 2023b], which, additionally, takes into account the camera parameters described by a so-called camera intrinsic matrix (containing information about, for example, focal length and optical center) and distortion coefficients of the matrix (describes how much straight lines are distorted when the image is captured).

The 3D pose estimation is, therefore, obtained when we find the appropriate rotation and translation matrix that minimizes the error when projecting object points into the image plane (point (u, v)). This projection can be written mathematically as [OpenCV.org 2023b] the matrix product given in Eq. (1). The equation describes the projection of world coordinate points $(X_w, Y_w, Z_w, 1)$ onto the image plane points $(u, v, 1)$, where **r** and **t** are parameters of the rotation and translation matrix, and **f** and **c** are focal lengths and optical centers respectively.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{1}$$

4 SOLUTION IMPLEMENTATION

The solution was implemented in the C++ programming language, where the popular OpenCV library was used for computer vision algorithms. To obtain 68 key-points of the facial region, also known as facial landmarks, we used the dlib C++ library, a toolkit containing machine learning and computer vision algorithms and tools [Dlib 2017].

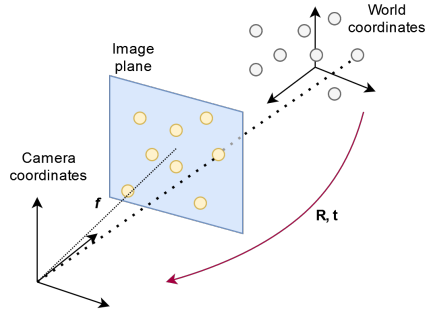


Figure 1: Visualization of the Perspective-n-Point problem (according to [OpenCV.org 2023b]).

Table 1: Computation of 7-point 3D head model given object points in starting head position.

Point	x	y	z
Between the eyebrows	0	$27.y - 33.y$	$0.56 \cdot (45.x - 36.x)$
Left eye corner	$-(27.x - 36.x)$	$36.y$ - $33.y$	$0.56 \cdot (45.x - 36.x)$
Right eye corner	$(27.x - 36.x)$	$36.y - 33.y$	$0.56 \cdot (45.x - 36.x)$
Tip of the nose	0	0	$1.36 \cdot (45.x - 36.x)$
Left mouth corner	$-(54.x - 48.x)/2$	$48.y - 33.y$	$0.70 \cdot (45.x - 36.x)$
Right mouth corner	$(54.x - 48.x)/2$	$48.y - 33.y$	$0.70 \cdot (45.x - 36.x)$
Middle of the chin	0	$8.y - 33.y$	$0.56 \cdot (45.x - 36.x)$

We captured the image stream with 3-channel RGB images of resolution 640x480. First, we detected the facial region using the widely used Viola-Jones algorithm [Viola and Jones 2001] upon which we decided because of its fast OpenCV implementation, using the default FrontalFace pre-trained model from the OpenCV library. Then, we tracked the obtained facial region using the MOSSE (Minimum Output Sum of Squared Error) tracker [Bolme et al. 2010] from the OpenCV contrib repository, which helps reduce computation time, and therefore increases the resulting frame rate, which would otherwise have been affected negatively if we were to detect the facial region repeatedly using the Viola-Jones algorithm. What follows is the detection of facial landmarks using the dlib library, which, when given an input image and location of the facial region, returned 68 landmark points (see image 2) in the starting head position.

When modeling the head, we used seven of those landmark points, namely, the points corresponding to the eye corners (points 36 and 45), the point between the eyebrows (27), nosetip point (33), corners of the mouth (48 and 54), and the point in the middle of the chin (8). We centered the model in a way shown in Table 1. The Table shows the x, y, and z coordinates of the model points, where, for example, $27.y$ means the y position of the 27th point (the point between the eyebrows) that we obtained using the facial landmark detector at the starting head position.

4.1 Obtaining head rotation angles using OpenCV

In our implementation we measured the camera intrinsic matrix and distortion coefficients using the example calibration tool that comes with the OpenCV library package. The tool measures these

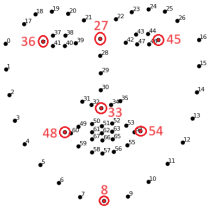


Figure 2: Distribution of 68 facial landmark points [1].

parameters by being shown an image of the chessboard pattern from different angles through the webcam image stream.

Inside the OpenCV environment we made use of the *solvePnP* function [OpenCV.org 2023a], in order to obtain the rotation and translation vector, followed by the use of the *Rodrigues* function [OpenCV.org 2023a], in order to transform the rotation vector into a rotation matrix, and, finally, we used the function *RQDecomp3x3* [OpenCV.org 2023a], in order to transform the rotation matrix into Euler rotation angles. Out of those angles, we made use of the yaw (horizontal head rotation) and pitch (vertical head rotation) angles, and converted them into the absolute position of the mouse cursor.

4.2 Filtering algorithm

With the intent of stabilizing the obtained head rotation angles, we made use of the approach of filtering the x and y positions of each of seven points (a total of 14 filters) that the dlib facial landmark detector returns. For this purpose we implemented an exponential smoothing (IIR) filter with difference equation given by Eq. (2).

$$y[n \quad 1] = (1 - \alpha)x[n] \quad \alpha y[n] \tag{2}$$

We defined $\alpha = 1 - \frac{2\pi f_0}{f_s}$ as the memory factor, basing the definition on properties of the RC low-pass filter from whose Laplace transfer function the Eq. (2) can be derived by taking the inverse Laplace transform and substituting derivatives with finite differences. Here f_0 is the filter cutoff frequency, and $f_s = 30\,\text{Hz}$ is our frame rate.

The final solution implementation used Eq. (2) to filter the seven landmark coordinate points $(u_i[n], v_i[n])$ for $1 \leq i \leq 7$, by applying the filter equation to u and v coordinate components. After we used these points in the PnP algorithm, we obtained estimations of the head rotation angles. We then transformed these rotations into the final cursor position point, and, finally applied the same filter on this point as well, which resulted in an even more stable cursor position.

We determined the appropriate cutoff frequency for filters applied at each point as $f_0 = 0.19\,\text{Hz}$ empirically, which corresponds to $\alpha = 0.96$.

The undesired effect of filtering is the time delay introduced into the movement of cursor during application use. From the filter's transfer function, the filter step response can be shown to be equal to $y(t) = 1 - e^{-2\pi f_0 t}$. The time at which this function reaches 80 % of it's maximum value can be shown to be $t_{delay} = \frac{\ln(0.2)}{2\pi f_0}$. Using this, we could estimate the time delay of our filter at a specified cutoff frequency as $t_{delay}(f_0 = 0.19\,\text{Hz}) \approx 1.35\,\text{s}$, which in a practical setting, manifests as a time delay, that is needed in order for the cursor to move from one side of the screen to the other, when the head is moved quickly between the two positions.

4.3 Conversion of rotation angles into an absolute cursor position

After having obtained the Euler rotation angles of the head, we transformed them into an absolute cursor position according to the equations (3) and (4) given by:

$$x_{pix} = \frac{W}{2} \times \left(1 - \frac{\tan(\theta_x)}{\tan(\theta_{xmax})}\right) \tag{3}$$

$$y_{pix} = \frac{H}{2} \times \left(1 - \frac{\tan(\theta_y)}{\tan(\theta_{ymax})}\right) \tag{4}$$

Here, W and H are the screen width and height in pixels, θ_x and θ_y Euler angles for yaw (horizontal head rotation) and pitch (vertical head rotation) respectively (obtained through the PnP algorithm), and θ_{xmax} and θ_{ymax} are the estimated head yaw and pitch angles at the extreme head positions (when looking at leftmost/rightmost and topmost/bottom side of the screen). In our case we set $\theta_{xmax} = 6.5°$ and $\theta_{ymax} = 8.0°$, which enabled us to use smaller head movements to obtain extreme positions of the cursor.

Eq. (3) was derived by observing $\tan(\theta_x) = \frac{\hat{x}}{D}$, where $\hat{x}$ is the desired cursor x position relative to the center of the screen, θ_x is the yaw angle of the head rotation, and D is the distance of the user's head to the computer monitor. For example, $\hat{x} = 0$ would, in this case, correspond to the center of the computer screen. Then, we observed that, in practice, there exists a maximum yaw angle θ_{xmax}, where the user is looking at the extreme side of the computer screen (for example rightmost), and that there also a similar relationship holds, namely, $\tan(\theta_{xmax}) = \frac{\hat{x}_{max}}{D} = \frac{W/2}{D}$. Dividing the two resulting equations eliminates D, giving $\frac{\tan(\theta)}{\tan(\theta_{xmax})} = \frac{\hat{x}}{W/2}$, which we can now solve easily for $\hat{x}$. Additionally, accounting for the pixel offset of $\frac{W}{2}$ pixels at $\hat{x} = 0$ resulted in $x_{pix} = \frac{W}{2} - \hat{x}$, which, after insertion of $\hat{x}$, yielded Eq. (3). Without loss of generality, the same approach can be used to derive Eq. (4).

5 MEASUREMENTS AND RESULTS

We evaluate our solution by measuring cursor stability during use with three cursor movement tasks, and compare the results to those when using noisy baseline head rotation estimates.

The measurements were performed by an able bodied test subject at a distance of about 60cm from the screen, using a 144 PPI (Pixels Per Inch) screen with a screen resolution of 1920x1080 pixels, using the default parameter values described in Section 4.3 and the filter cutoff frequency $f_0 = 0.19\,\text{Hz}$.

In the first part of the measurements we compared graphs of cursor x and y displacement from a reference center point at the task where the user sequentially moved the head in two directions: 1) left-right, 2) up-down. We measured the stability of the cursor position relative to horizontal axis of the computer screen (x displacement) and vertical axis of the computer screen (y displacement). The results can be seen in Fig. 3, where we can see that the non-filtered cursor positions exhibited a large degree of noise.

[1] Based on: https://www.researchgate.net/figure/The-68-landmarks-detected-by-dlib-library-This-image-was-created-by-Brandon-Amos-of-CMU_fig2_329392737

Table 2: Results of a t-test for cursor position when observing a fixed point.

	No filter	Filter
Mean [pixels]	46.25	8.39
Variance	787.73	16.10
Number of samples	132	201
t Stat	15.39	
P(T<=t)	< 0.00001	
t critical two-sided	1.97	

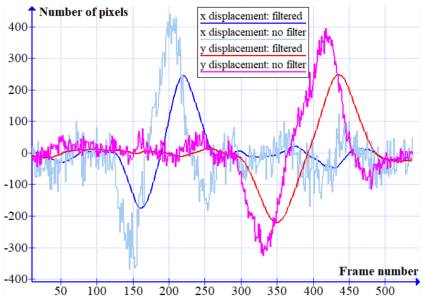


Figure 3: Visualization of filtered and non-filtered cursor displacement along the x and y axes when looking left-right and up-down sequentially.

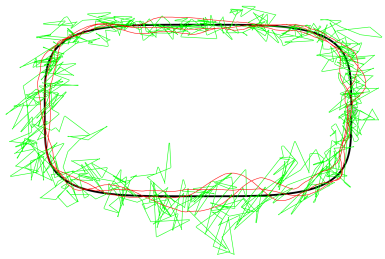


Figure 4: Display of filtered (red) and non-filtered (green) cursor path during a detour around the reference curve (black).

In the second part of the measurements we recorded the cursor location when observing a fixed point - first without the filtering algorithm, and then with the filter. Measurement samples are absolute values of cursor deviation from the point. Then, we performed the t-test, where we obtained the results shown in Table 2. We can see that filtering introduces a statistically significant difference in the results.

In the final part of the measurements we measured cursor deviation magnitude from an ellipse-like curve (with half the width and height the screen dimensions, i.e. 25% side-margins). The subject had the task of moving the cursor using the head mouse three times around the curve, starting and ending in mid-right side of the curve.

For this sample of measurements we obtained the following average values with their Standard Deviations (units are the number of pixels of cursor deviation magnitude from the reference curve): $\mu_{\text{no filter}} = 83.5$ ($\sigma_{\text{no filter}} = 101.1$); $\mu_{\text{filter}} = 17.9$ ($\sigma_{\text{filter}} = 13.8$).

6 CONCLUSION

In this paper we presented a promising method for computer mouse cursor control using webcam 3D head pose estimation, based on filtering of facial key-points. Such an approach is relevant and applicable in the field of Assistive Technology Human-Computer Interfaces.

Our approach stands out by it's ability to map the head rotation vector, $(yaw, pitch)$, directly into corresponding (x, y) positions of the cursor. This differs from available application solutions on the market, which instead translate relative head motion into relative mouse movement, where the head gaze direction need not coincide with cursor position.

In our experimental work we demonstrated the effectiveness of our key-point filtering approach on cursor stability during use. During the task of observing a fixed point, the mean absolute difference between the observed point and actual cursor position was 8.39 pixels with $\sigma^2 = 16.10$, as opposed to the mean of 46.25 pixels with $\sigma^2 = 787.73$ without the filter. During the task of user moving the cursor around the reference curve, we obtained a mean absolute cursor position difference (relative to the curve) of 17.9 pixels ($\sigma = 13.8$ pixels), as opposed to the mean of 83.5 pixels ($\sigma = 101.1$ pixels). Additionally, we demonstrated the cursor denoising ability visually during the task of sequential head movements.

In the future work, we would like to improve the cursor stability by increasing the filter memory factor when the head is held still, which would allow fine-tuned cursor movements and potentially improve the ease of use. Additionally, we would like to include the implementation of facial gesture detection for mouse click simulation and automatic fixation of cursor on UI elements (e.g. buttons).

REFERENCES

Margrit Betke, James Gips, and Peter Fleming. 2002. The camera mouse: visual tracking of body features to provide computer access for people with severe disabilities. *IEEE Transactions on neural systems and Rehabilitation Engineering* 10, 1 (2002), 1–10.

David S Bolme, J Ross Beveridge, Bruce A Draper, and Yui Man Lui. 2010. Visual object tracking using adaptive correlation filters. In *2010 IEEE computer society conference on computer vision and pattern recognition*. IEEE, 2544–2550.

C++ Dlib. 2017. Dlib C++ Library. *Dlib. net* (2017).

Yun Fu and Thomas S. Huang. 2007. hMouse: Head Tracking Driven Virtual Computer Mouse. In *2007 IEEE Workshop on Applications of Computer Vision (WACV '07)*. 30–30. https://doi.org/10.1109/WACV.2007.29

Cesar Mauri. 2019. *Enable Viacam Website.* http://eviacam.crea-si.com/index.php Accessed on 2023-06-30.

Masoomeh Nabati and Alireza Behrad. 2010. Camera mouse implementation using 3D head pose estimation by monocular video camera and 2D to 3D point and line correspondences. (2010), 825–830. https://doi.org/10.1109/ISTEL.2010.5734136

ANNA Nowogrodzki. 2018. Writing code out loud. *Nature* 559, 7712 (2018), 141–142.

Obstino Association - Društvo Obstino. 2022. *Face Control Android application.* https://play.google.com/store/apps/details?id=com.obstino.facecontrol

OpenCV.org. 2023a. *OpenCV documentation Camera Calibration and 3D Reconstruction.* https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html Accessed on 2023-3-3.

OpenCV.org. 2023b. *OpenCV documentation Perspective-n-Point (PnP) pose computation.* https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html Accessed on 2023-3-3.

Jilin Tu, T. Huang, and Hai Tao. 2005. Face as mouse through visual face tracking. In *The 2nd Canadian Conference on Computer and Robot Vision (CRV'05)*. 339–346. https://doi.org/10.1109/CRV.2005.39

P. Viola and M. Jones. 2001. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Vol. 1. I–I. https://doi.org/10.1109/CVPR.2001.990517

WHO. 2011. *World Report on Disability 2011.* https://www.who.int/teams/noncommunicable-diseases/sensory-functions-disability-and-rehabilitation/world-report-on-disability Accessed on 2023-05-17.

Dariusz Zapała and Bibianna Bałaj. 2012. Eye Tracking and Head Tracking–The two approaches in assistive technologies. (2012).

App to Help Children with Special Needs to Improve Their Eye Movements and Focus

Christeena Varghese*
christeena.varghese@study.thws.de
Technical University of Applied Sciences Würzburg-Schweinfurt
Würzburg, Germany

Vincent Wahyudi*
vincent.wahyudi@study.thws.de
Technical University of Applied Sciences Würzburg-Schweinfurt
Würzburg, Germany

Viony Tengguna*
viony.tengguna@study.thws.de
Technical University of Applied Sciences Würzburg-Schweinfurt
Würzburg, Germany

ABSTRACT

This paper presents a low-cost eye-tracking system and training game application aimed to improve eye gaze control of children aged three to six, especially for those with conditions such as Dyspraxia and Autism Spectrum Disorders. The traditional method of manually evaluating eye movement can often result in inaccurate results. Our application combines eye-tracking technology with a game-like approach designed to encourage children to focus their gaze in different directions. The application tracks their gaze using a laptop's built-in camera and provides comprehensive analytics. This work has the potential for further development and application in the field of eye gaze treatment.

KEYWORDS

Eye-tracking application, Autism Spectrum Disorder, children, Computer Vision, OpenCV

1 INTRODUCTION

Eye contact plays a crucial role in human social interaction. Individuals with conditions such as Dyspraxia and Autism Spectrum Disorders face difficulties in shifting their gaze, which leads to difficulty in synchronizing their central and peripheral vision [9]. Impaired eye movement can increase the risk of accidents, restlessness in movement, and increase stress. If detected in its early stages, the dexterity of the eye gaze can be improved through the implementation of therapeutic exercises, commonly referred to as "eye gaze treatments".

Traditionally, the therapist uses an item (a stick or a pen) for the eye gaze treatment sessions. The therapist moves the stick in various directions, encouraging the child to focus on the object. The therapist visually evaluates the movement of the child's eye and manually tracks the improvements. However, this method is extremely imprecise and subjective.

The inaccuracy of the results can be reduced by the use of eye-tracking systems, which are readily available for purchase and can accurately determine the position and movement of the gaze [10]. However, these systems can be costly and there are a limited number of affordable eye-tracking systems on the market.

Our eye-tracking software aims to improve traditional methods in a cost-effective way by providing more accurate results in eye gaze treatment. The source code of this project is available on GitHub at the following link: https://github.com/vincentw1997/Eye_Tracking_Project_Module.git. The eye-tracking application is covered up as an interactive game that can be appealing to children of ages three to six [11]. The game that is connected to the eye-tracking system mimics the traditional treatment in an attractive way that helps the children to improve their eye gaze in a joyful way. The therapist can evaluate each child's progress accurately from the results generated by the application.

The software tracks the child's gaze through the built-in laptop's camera and provides analytics about the estimated gaze for the therapist. The system records the coordinate data of the child's eye movement and compares it to the recorded coordinates of the moving characters in the game. These data are useful for therapists to assess the speed and accuracy of the eye gaze movement during the therapy session.

This paper is structured into four sections. The first section provides a concise overview of various eye-tracking methods and offers explanations of similar works related to cost-effective eye-tracking pipelines. Next, this paper describes the method used to develop the eye-tracking application, including elements such as head-tracking, game components, and data analytics. The third section presents the experimental setup, the results of the controlled experiments, and a discussion of the obtained results. Lastly, the fourth section provides a short conclusion and potential improvements for future works.

2 RELATED WORK

Generally, eye-tracking methods can be categorized into two main groups [4]: appearance-based and model-based approaches. The appearance-based approach tackles the problem by learning a mapping function from eye images to gaze estimation. This approach usually requires large training data and a properly trained model [13]. On the other hand, the model-based approach aims to calculate a 3D gaze direction vector, based on the known anatomy of the human eye, such as the iris, sclera, and pupil. This method usually relies on precise metric information such as camera calibration, positioning, monitor positioning, and orientation. In this paper, we implemented the model-based approach by simplifying assumptions and skipping some traditional steps in order to fit into our specific use case. Cazzato et al. [2] approaches eye-tracking problems for soft biometrics using the model-based approach that uses cheaper hardware such as Microsoft Kinect and ASUS Xtion Pro Live. These are commercial depth sensors, which are considerably more accessible to the general public compared to Tobii infrared eye trackers or similar laboratory hardware that can cost multiple times more. Our paper aims to leverage the concept of making eye-tracking technology more accessible to the general public, without

*Equal contribution

DOI: https://doi.org/10.26493/scores23.05

the need for additional hardware such as commercial depth sensors. However, this approach comes at the cost of lowered tracking accuracy.

3 METHOD

In this section, we divide the description of our application into four parts. The first description begins with an overview of the eye-tracking architecture. Secondly, we elaborate on the head-tracking architecture and its integration with the eye-tracking architecture. Thirdly, we explain the game component designed to captivate the child's attention. Lastly, we provide data analytics based on the data collected from the first three architectures. We visualize the data for performance analysis evaluated by the therapist. An overview of how the application functions can be seen in Figure 1. A flowchart of all the development processes can be seen in the Github repository.

3.1 Eye-Tracking

The eye-tracking architecture's main objective is to isolate the eye with respect to the whole screen captured by the webcam of the laptop. It starts with grayscaling [14] the frames captured to reduce color complexity. Facial landmarks are further detected from the grayscaled frames using dlib library [5] that assigns specific coordinates to the facial landmarks. Masking is executed to isolate the precise locations of the right and left eye coordinates derived from dlib landmarks [12], thereby producing frames that exclusively isolate the eye shape, which we call eye-only frames. The estimation of the pupils' position is achieved by detecting circular shapes in the eye-only frames.

General pipelines for shape detection are implemented to estimate the pupil's position in eye-only frames. These include grayscaling, filtering of the frames to emphasize contours of the iris and the sclera (white part of the eye), and thresholding of the frames where pixels are replaced into either black or white depending on the pixel value being higher or lower than the predetermined threshold value. The contours of the eye are clearly shown after the thresholding step and the circular shape of the iris will be clearly outlined.

The determined iris shape can be used to detect the region of interest (ROI). In the ROI frame, pupils are estimated by detecting the circular shape using Circular Hough Transform [8] inside the iris. The coordinates of the estimated iris and pupil are stored. Plotting the stored coordinates with respect to time will provide comparable data with the character movement in the game part. The ROI is implemented by magnifying the area on only one of the eyes as the movement of the left and right eye are assumed to be similar. The ROI frame can be seen in Figure 1. I. Eye Tracking (magnified version of the frame) with a green circle highlighting the iris of the child's eye.

3.2 Head-Tracking

The head-tracking architecture detects and follows the movement of a person's head, which includes the position and orientation of the head. The integration of head-tracking with eye-tracking ensures a controlled alignment between the camera and the child. Good alignment provides accurate readings of the estimated eye movements. Face detection, facial landmarks, and pose estimation are the main components of the head-tracking part. In this model, we use OpenCV [1] for camera live input, Dlib for the facial detector, and solvePnP [7] for pose estimation. A warning will show in the application if the patient is not aligned perfectly with the camera as seen in Figure 1. II. Head Tracking.

3.3 Game

The game architecture is created to serve as a facade for eye-tracking therapy. The game offers several settings such as cartoon character selections, the speed of the character's movement, and the direction. This promotes different movement variations for eye gaze training. The available direction options are horizontal, vertical, and free movement. The movement speed of the character can be set to slow, medium, or high.

The game was built using Python and libraries such as Pygame and Tkinter. SQLite was used as the backend database to store information such as game progress, patient details, and login credentials. Therapists can register patients, view patient records, start therapy sessions, and create an account using the "Create Account" option before logging in. These features provide secured information storage that the therapy can use.

3.4 Data Analytics

Data analytics can be done once eye-tracking, head-tracking and the game architecture are combined together into one integrated application. Data analytics analyzes the eye gaze performance of the child. Eye gaze data are only collected as long as the child maintains good head orientation during the session. Alerts in the application will remind the user to fix the user's head position and orientation as explained in the previous section.

The gathered data are represented in the form of two graphs [15] for each therapy session. The first graph contains coordinates that represent the pupil gaze positions in the ROI frame. It is plotted with respect to time and shows all the positions of the pupil from the start of the session until the session ends. This graph can be seen in Figure 2. (Pupil position in the ROI). The second graph contains the comparison of the cartoon character game coordinates and the pupil gaze coordinates in the ROI frame. Matching these coordinate systems with time will show whether the child is following the movement of the cartoon character or not. If the child is looking at something else besides the cartoon character, there will be a big difference between the two coordinates. An example of the graph can be seen in Figure 2. (Comparison between Game and Eye Coordinates).

4 EXPERIMENTAL SETUP, RESULTS AND DISCUSSION

In this section, we will explain the experimental setup, results, and discussion. The experimental setup describes the implemented settings. The result shows the data collected when implementing the settings in the experimental setup and discussion on the results obtained.

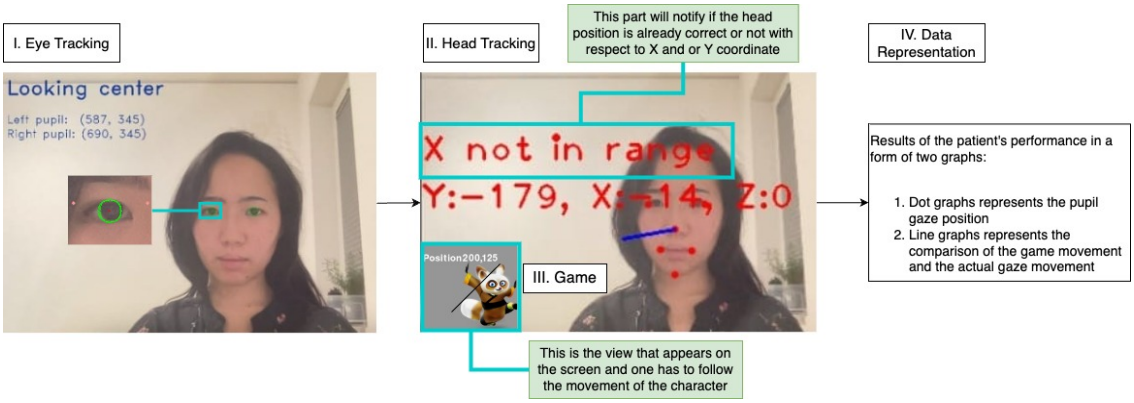


Figure 1: An Overview of the eye-tracking application. I. Eye Tracking shows the Region of Interest (ROI) frame that isolates the eye and highlights the iris as a green circle. II. Head Tracking shows the head position and orientation of the user in red. "Good" means the user is in the perfect position. "X/Y not in range means that the user needs to adjust his/her positioning. The small rectangle on the bottom left (III. Game) is the view that the user will see when the application runs. The user must follow the cartoon character's movement while remaining still. IV. Data representation represents the recorded data into graphs for analytics purposes."

4.1 Experimental Setup

The application is ideally designed for home usage and we simulate similar conditions during our experiment. The application is running on a regular laptop with a built-in webcam with a resolution of 720p. The room was well-lit with one warm white light in the ceiling and a window overlooking the side of the laptop. Experiments were done with no direct sunlight shining into the room. The tester is required to remove glasses during the experiment as it adds noise to the data.

The application was run on 5 different settings. The first and second setups run at medium speeds with different movement, which is horizontal and vertical respectively. The third and fourth setups have the same horizontal movement with different speed settings, low speed, and high speed respectively. The fifth setup will have the cartoon character moving at medium speed with a free movement function. These setups aim to show the effect of different parameters on a common use case.

The tester must be perpendicular to the camera, remain still, and maintain his left eye in between the pink diamonds in the ROI frame during the testing session. If the tester is maintaining an ideal position, the application display will show "Good!" in red color. Otherwise, the application will show warnings, which the tester must adjust their position similar to Figure 2. II. Head Tracking. X not in range means the tester is not in the right orientation horizontally.

4.2 Result

Figure 2. shows the recorded result for the first experimental setup (Horizontal movement and medium speed). The first graph in Figure 2. represents the estimated pupil position in the ROI frame. Both axes are in pixel units. The time recorded for the estimated pupil position is determined by the color. Darker colors represent earlier time stamps in the session and color lightens as time increases. The second graph in Figure 2. shows the movement data of the cartoon character (Game X and Y coordinate) and the eye movement in the ROI frame (Eye X and Y coordinate).

Figure 3. shows the recorded result for the second experimental setup (Vertical movement and medium speed). The two graphs in Figure 3. follows the same format as explained in the previous paragraph. The second setup maintains the same speed parameter while changing the cartoon character's movement from horizontal to vertical to test the tracking capabilities of the application. Results from the third to the fifth experimental setup are not shown here and can be seen in the GitHub repository.

4.3 Discussion

Figure 2. and Figure 3. demonstrates the effect of different movement directions. The pupil position in the ROI graph in Figure 2. shows the initial dots of the estimated pupil positions start moving from the bottom left corner of the screen which suggests the tester is trying to find the cartoon character on the screen. In the following time stamps, the positions moved toward the center of the ROI frames and oscillated horizontally between 250 to 300 pixels X position. These movements indicate that the tester is trying to follow the movement of the cartoon character that is moving horizontally. The horizontal movement of the pupil is further supported by the result of the Eye X coordinate in the second graph of Figure 2. (green line). The Eye X coordinate mimics the oscillating movement of the Game X coordinate (blue line) and has similar peaks. As this setup only evaluates the horizontal movement, the Game Y coordinate (orange line) remains in place for the whole session. The Eye Y coordinate (red line) will never be as constant as the Game Y coordinate (orange line) as humans have micro-movement in their gaze when tracking any moving object. The first graph in Figure 3. shows similar behavior in the first few dots. It started to move from the bottom left towards the center of the ROI frame which suggests the tester trying to track the cartoon character on the screen. Subsequent movements suggest that the estimated pupil

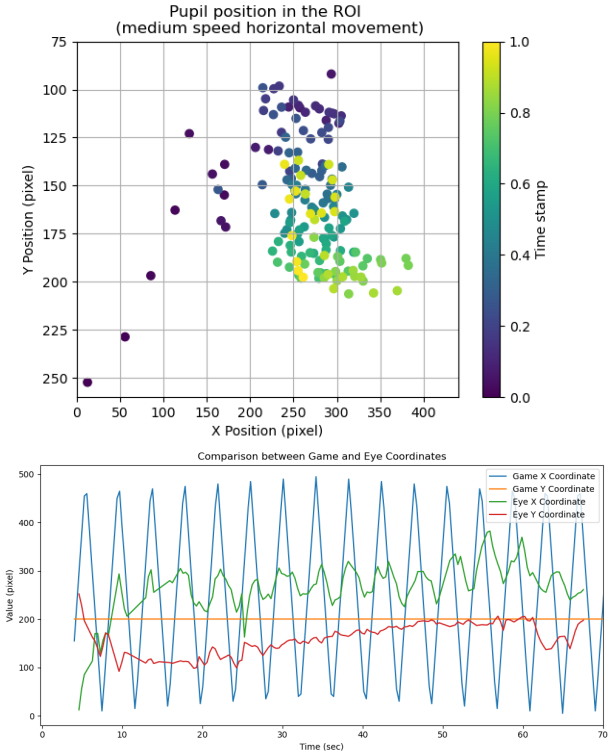


Figure 2: Results for the first experimental setup. Parameter settings for this setup are horizontal movement and medium speed. The first graph represents the tracked estimated pupil position in the ROI frame. The axes are in pixels. Darker points mean an earlier time stamp. The tracked pupil positions are more horizontally spread, and oscillations from left to right can be observed between 250 to 300 pixels X Position. The second graph illustrates the movement of the cartoon characters in the game as Game X/Y coordinates and the tracked eye movement as Eye X/Y coordinates. The Eye X coordinate (green line) mimics the pattern of the Game X coordinate (blue line) when the user tracks the horizontally moving character.

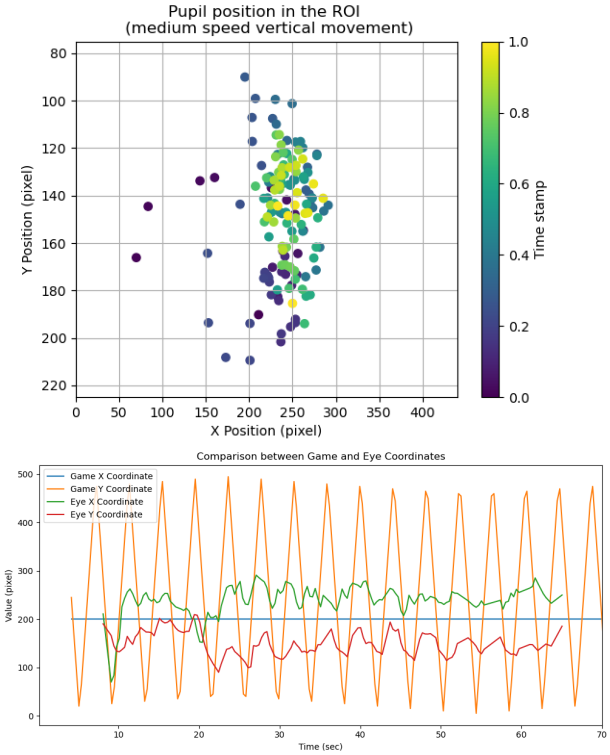


Figure 3: Results of the second experimental setup. Parameter settings for this setup are vertical movement and medium speed. The representation is the same as the previous Figure 2. The estimated pupil positions in the ROI are more vertically spread between 100 to 200 Y pixels Position. The estimated pupil positions generally remain centered on the X-axis. The second graph shows the Eye Y coordinate (red line) mimics the Game Y coordinate (orange line). The movements on the vertical plane are not as defined as the movements on the horizontal plane as in Figure 2.

position also centers around 200 to 250 pixels X position. Due to the vertical movement of the cartoon character, the estimated pupil positions oscillate vertically between 100 and 200 Y pixels position. This is further reinforced by the result in the second graph in Figure 3. Eye Y coordinate (red line). It mimics the movement of the Game Y coordinates (orange line) and has similar peaks. However, these peaks and nearly constant values of the Eye coordinates are not as obvious as in the previous horizontal setup. This shows the weakness in the application in capturing the estimated vertical movement of the eye. Another point to address is the difference between the Eye X coordinate and the Game X coordinate very far apart. This is due to the difference in the coordinate values during the development and the limitation of the application in capturing the extreme values on the edge of the ROI frames. This means that the Eye coordinates will have less range compared to the Game coordinates.

5 CONCLUSION AND FUTURE WORKS

This paper develops an affordable game-like eye-tracking application that does not need additional hardware. The application provides analytics for the therapist to track the performance of the child. Based on the experimental results, the application performs better in tracking horizontal movement compared to vertical movement. Other results testing the effect of speed suggest that the application performs best in a medium-speed environment. The free movement setup shows the limitation of the application in capturing extreme coordinates in the ROI frames. Based on the discussion section, the application still has a lot of sectors to improve. Children with glasses cannot use this application due to the effect of glass on the recorded video. Gwon et al. [3] use additional hardware to solve this limitation. Differences in the Game and Eye coordinate values can be tackled by building both coordinates of

the same magnitude making it easier to compare both values. The application has limited controls (no pause or run time settings) during the data recording session which made it hard to operate the application. The implemented approach in this paper is a naive approach that has its limitations in the accuracy and usability sector. Rebuilding the whole architecture using the appearance-based approach [6] requires a larger pre-trained model with a larger dataset might be a better solution that can improve the tracking accuracy of the application.

6 ACKNOWLEDGEMENTS

The authors express great appreciation to Prof. Dr. Magda Gregorová for her constructive suggestions during the experiments and writing of this paper.

REFERENCES

[1] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[2] Dario Cazzato, Andrea Evangelista, Marco Leo, Pierluigi Carcagnì, and Cosimo Distante. 2015. A low-cost and calibration-free gaze estimator for soft biometrics: An explorative study. *Pattern Recognition Letters* 82 (11 2015). https://doi.org/10.1016/j.patrec.2015.10.015

[3] Su Gwon, Chul Cho, Eui Chul Lee, Won Lee, and Kang Park. 2014. Gaze Tracking System for User Wearing Glasses. *Sensors (Basel, Switzerland)* 14 (02 2014), 2110–34. https://doi.org/10.3390/s140202110

[4] Dan Witzner Hansen and Qiang Ji. 2010. In the Eye of the Beholder: A Survey of Models for Eyes and Gaze. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32, 3 (2010), 478–500. https://doi.org/10.1109/TPAMI.2009.30

[5] Davis E. King. 2009. Dlib-ml: A Machine Learning Toolkit. *Journal of Machine Learning Research* 10 (2009), 1755–1758.

[6] Kyle Krafka, Aditya Khosla, Petr Kellnhofer, Harini Kannan, Suchendra Bhandarkar, Wojciech Matusik, and Antonio Torralba. 2016. Eye Tracking for Everyone. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[7] Eric Marchand, Hideaki Uchiyama, and Fabien Spindler. 2016. Pose Estimation for Augmented Reality: A Hands-On Survey. *IEEE Transactions on Visualization and Computer Graphics* 22, 12 (Dec. 2016), 2633 – 2651. https://doi.org/10.1109/TVCG.2015.2513408

[8] Jiri Matas, C. Galambos, and J. Kittler. 2000. Robust Detection of Lines Using the Progressive Probabilistic Hough Transform. *Computer Vision and Image Understanding* 78 (04 2000), 119–137. https://doi.org/10.1006/cviu.1999.0831

[9] Michael Miller, Leanne Chukoskie, Marla Zinni, Jeanne Townsend, and Doris Trauner. 2014. Dyspraxia, motor function and visual–motor integration in autism. *Behavioural brain research* 269 (2014), 95–102.

[10] Pramodini A. Punde, Mukti E. Jadhav, and Ramesh R. Manza. 2017. A study of eye tracking technology and its applications. In *2017 1st International Conference on Intelligent Systems and Information Management (ICISIM)*. 86–90. https://doi.org/10.1109/ICISIM.2017.8122153

[11] Tony Renshaw, Richard Stevens, and Paul Denton. 2009. Towards understanding engagement in games: An eye-tracking study. *On the Horizon* 17 (09 2009), 408–420. https://doi.org/10.1108/10748120910998425

[12] Christos Sagonas, Georgios Tzimiropoulos, Stefanos Zafeiriou, and Maja Pantic. 2013. 300 Faces in-the-Wild Challenge: The First Facial Landmark Localization Challenge. 397–403. https://doi.org/10.1109/ICCVW.2013.59

[13] Yusuke Sugano, Yasuyuki Matsushita, and Yoichi Sato. 2014. Learning-by-Synthesis for Appearance-Based 3D Gaze Estimation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 1821–1828. https://doi.org/10.1109/CVPR.2014.235

[14] C. Tomasi and R. Manduchi. 1998. Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. 839–846. https://doi.org/10.1109/ICCV.1998.710815

[15] Antony Unwin. 2020. Why Is Data Visualization Important? What Is Important in Data Visualization? *Harvard Data Science Review* 2, 1 (jan 31 2020). https://hdsr.mitpress.mit.edu/pub/zok97i7p.

Real-Time Vehicle Speed Estimation from Video

Mitko Nikov
mitko.nikov@student.um.si
Faculty of Electrical Engineering and Computer Science, University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

Mitja Žalik
mitja.zalik@um.si
Faculty of Electrical Engineering and Computer Science, University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

Domen Mongus
domen.mongus@um.si
Faculty of Electrical Engineering and Computer Science, University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

ABSTRACT

Vehicle tracking and speed estimation are essential in many intelligent traffic management systems. In this paper, we propose a new real-time architecture for estimating the speed of the vehicles from monocular video streams. While following traditional steps, including vehicle detection, its tracking, and mapping of coordinates from pixel to real-world space, notable progress beyond state-of-the-art is achieved by introducing a new filtering schema with a camera calibration that allows for speed estimations from arbitrary viewing angles. This enables monitoring to be performed from moving platforms, such as Unmanned Aerial Vehicle (UAV), while also observing multiple vehicles at once. As confirmed by the results, even in these circumstances, the proposed approach achieves comparable results to state-of-the-art techniques applied on stationary video streams.

KEYWORDS

vehicle speed estimation, video stream processing, unmanned aerial vehicle, traffic management systems

1 INTRODUCTION

With the increase in traffic density, real-time vehicle speed estimation is actively used in traffic-aware route planning and forecasting [11], speed limit enforcement [13] and other intelligent traffic management systems' services. Traditionally, methods used for vehicle speed estimation from video streams, adopt a so-called tracking-by-detection framework [20], that consists of vehicle detection, vehicle tracking and speed estimation steps [14, 15, 19, 25, 26].

In general, vehicles are detected and annotated with bounding boxes on a frame-by-frame basis. In pipelines of older work [8, 27, 29] this was achieved using histograms, thresholds, and other means of separating moving objects from static backgrounds. Accordingly, these methods are susceptible to noise and need extensive manual calibrations for specific environments and setups. Nevertheless, deep learning approaches proved to be efficient in dealing with these issues, and state-of-the-art today rely almost exclusively on variations of Convolutional Neural Networks (CNNs) such as YOLO[1] [23], Faster-RCNN[2] [7] and Mask-RCNN[2] [10] for detection as well as classification of vehicles [9, 15, 18, 25, 26, 30].

Speed estimation, however, requires vehicle tracking over a set of successive frames even in those cases when occlusions occur. Predicting their movements is, therefore, inevitably required for successfully associating their detected positions. The Simple Online Realtime Tracking (SORT) algorithm [2] and different variations of the Kalman filter are commonly used for this purpose [20]. In order to enhance vehicle matching robustness across frames, the recently introduced DeepSORT further extends the SORT algorithm by extracting vehicle features using a pre-trained CNN [18, 28], while increased accuracy of vehicle tracking and the extraction of movement parameters using optical flow was also examined [4, 14, 19, 22]. In all of these cases, however, movement vectors are estimated in pixel-space. Their mapping into real-world coordinates is, therefore, addressed next.

Mapping of movement vectors inevitably requires camera calibration that establishes a correlation between the pixel and the real-world displacement of the vehicle. In most cases, calibration approaches are based either on manually defined coefficients [14, 19, 22, 24, 27, 29] and intrusion lines [16] or linear image transformations [15, 18]. However, these approaches require extensive environment-specific parameter calibrations and measurements, due to factors such as lighting conditions, camera angles, occlusions or car-specific parameters [19, 29]. Therefore, they often remain limited to the video-streams from stationary cameras. On the other hand, some recently introduced approaches address these issues by using CNNs and optical flow to estimate the average speed of traffic rather than individual vehicles. [5, 31]

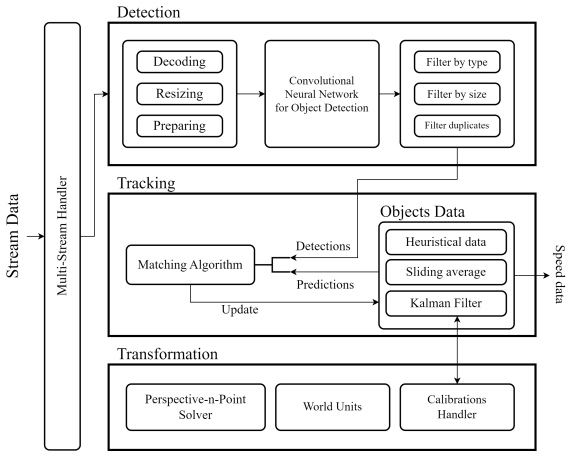


Figure 1: High-level architecture of the proposed vehicle speed estimation method.

Accordingly, while vehicle detection, tracking and even average speed estimation of traffic are already well-researched, estimating

[1]YOLO: You Only Look Once
[2]RCNN: Region-based Convolutional Neural Network

DOI: https://doi.org/10.26493/scores23.14

speed of individual vehicles poses a greater challenge. The non-linear effects stemming from the perspective projection of real-world objects onto the 2D image space, coupled with the intricate terrain configurations, further aggravates the problem, preventing applications of the discussed methods for processing video-streams from moving platforms, such as Unmanned Aerial Vehicle (UAV).

In order to address these issues, we propose a new camera calibration technique that is based on an iterative solver for the Perspective-n-Point (PnP) problem [6], together with several improvements made upon the known movement vector estimation with ray tracing and Kalman filtering [17]. Details on the proposed method are explained in the next section. Section 3 provides the achieved results, while conclusions are presented in section 4.

2 METHOD

Following the pattern of tracking-by-detection frameworks, the processing pipeline of the proposed method consists of detection, tracking and space transformation steps, as shown in Figure 1.

2.1 Vehicle Detection

In the detection step, the stream data is first decoded, resized and preprocessed before being fed into the CNN for object detection. Processing the image in the CNN is the most computationally intensive task of the proposed pipeline. For the real-time object detection, we obtained a YOLOv4 model [3], pretrained on the ImageNet and MS COCO datasets. Since the mentioned datasets contain mostly images of different objects from the ground level, the pretrained model performed badly in our high elevation and high meter-to-pixel domain, where objects are captured from above and appear relatively small. Therefore, we improved the model via transfer learning using custom created dataset. The training dataset consists of approximately 61500 images and 550000 annotated vehicles. Vehicle annotation was performed manually over 6 month period and aided by general object tracking [1, 12, 21] in order to obtain multiple samples of vehicles from different angles. The model reached convergence after being trained on 35000 batches of 64 images in order to fine tune its weights and, thus, transfer the knowledge to the required domain.

Further improvements on object detection are achieved by applying several filters for removal of duplicate objects and objects that are irrelevant for speed estimation because of their class or size. Still, this does not solve the commonly present shift of bounding-boxes over consecutive frames, as indicated by Hua et al. [14]. Nevertheless, the integration of the Kalman filter substantially mitigates this error during vehicle tracking, as discussed.

2.2 Vehicle Tracking

The proposed implementation of tracking is based on the SORT algorithm [2] for Multi-Object Tracking (MOT). Given the information about prior movement of objects $[\dot{x}, \dot{y}]^T$, the SORT algorithm uses a Kalman filter with the following state vector:

$$\mathbf{x} = [x, y, s, r, \dot{x}, \dot{y}, \dot{s}]^T, \tag{1}$$

where the point (x, y) is the center of the bounding box, s is the area and r - the aspect ratio of the bounding box. Moreover, the displacement vector $[\dot{x}, \dot{y}]^T$ in 2D space can be directly extracted from the state. However, the perspective projection of the 3D world makes the problem of object tracking in the 2D image space non-linear. Therefore, with the use of the mapping function R that projects image coordinates onto a real-world plane, formally defined in section 2.3, we are able to model the problem directly in 3D space with the following Kalman filter state:

$$\mathbf{x}_{\text{ours}} = \left[R_0(c)_x, R_0(c)_y, s, r, \dot{R}_0(c)_x, \dot{R}_0(c)_y, \dot{s}\right]^T, \tag{2}$$

where c is a corner of the bounding box whose projection on the ground plane is the closest to the camera. The vehicle velocity is directly described by the vector $v = \left[\dot{R}_0(c)_x, \dot{R}_0(c)_y\right]^T$. Moreover, the inclusion of the z coordinate from the 3D domain in the Kalman filter state is unnecessary and will always be 0 (ground plane).

Following the prediction of the Kalman filter and re-projection of the object's position in the current frame, we proceed to match the predicted bounding box with a bounding box obtained by object detection, prioritizing the highest Intersection over Union (IOU) metric. We also impose a IOU_{min} threshold that restricts the matching algorithm from linking bounding boxes that have a really low IOU correspondence. Furthermore, we implement an object age heuristic, which assigns priority to older objects in cases where a single bounding box exhibits similar Intersection over Union (IOU) correspondence with two or more tracked objects.

2.3 Space Transformations

As already mentioned in the previous section, space transformation from the 2D image space to the 3D world captured by the camera, is necessary for tracking as well as the actual vehicle speed estimation. Additionally, to project the ground plane and bounding box predictions from 3D to 2D space, a reverse transformation is also required.

We propose an effective methodology that requires only 6 pairs of image-to-world correlation coordinates. With the use of PnP iterative solver, we manually provide 4 image-to-world correlation coordinate pairs on the ground plane ($z = 0$) and 2 more on the $z = 1$ plane. This results in translation and rotation vectors describing the relative 3D space. We use landmarks with approximately equal length, width and height to ensure uniformity across all units in the 3D space (x, y and z). Although subpixel precision is required for extremely precise calibration, we found the Kalman filter to be sufficiently flexible, delivering comparable results even in its absence.

Using the camera matrix, translation and rotation vectors provided by our camera calibration, it is possible to define a function $R_z : \mathbb{R}^2 \to \mathbb{R}^2$ that maps the 2D image space to some predefined z plane in the 3D space. It is implemented using a ray-plane intersection. By defining the R function in regards to a fixed z plane, we can show that R_z is a bijectional mapping from the 2D image space to the predefined plane in the 3D world. Let us assume that all vehicles are driving on the ground plane where $z = 0$ and thus only be interested in R_0.

Nevertheless, establishing a correlation between the 2D and 3D space remains incomplete until the equivalence of one unit in our 3D space to a specific distance in meters is determined. Using our forementioned functions, we can introduce a routine to calibrate the "world units coefficient" as follows: pick two points p_1 and p_2

in the 2D image space for which, you know the real-world distance between their projections in the $z = 0$ plane. The 3D world unit coefficient is the ratio of the known distance between the two points in the real world and the Euclidean distance of the projected points $R_0(p_1)$ and $R_0(p_2)$.

At last, using time data and the world unit coefficient, we can convert the mentioned velocity vector $v = \left[\dot{R}_0(c)_x, \dot{R}_0(c)_y\right]^T$ to a speed measurement.

3 RESULTS

This research is a part of the development of the GeMMa Fusion Machine Learning (GFML) application written in C++, where all of our tests were also conducted. For the given results, we evaluated the pipeline on Intel Core i7-9700K @ 3.6GHz with 32GB RAM and NVIDIA GeForce RTX2070 SUPER GPU.

3.1 Test Data

(a) (b) (c)

Figure 3: Test (a) Stream 1, (b) Stream 2, and (c) Stream 3 with estimated vehicle speed.

A DJI Mini 3 Pro and a DJI Phantom 4 Advanced drones were used to record most of our test footage. The footage was initially recorded in 4K resolution, with 25 frames per second (FPS) D-Cinelike 8-bit color profile, then resized for comparison to 1920 × 1080 px and 720×576 px resolution without applying a color lookup table (LUT). The ground truth data is calculated using a Global Positioning System (GPS) measurement device with 30 connected GPS satellites and synced to the footage using time codes.

Amongst these, three different scenes were selected for testing. Stream 1 (shown in Fig. 3a) monitors vehicle during acceleration, Stream 2 (shown in Fig. 3b) monitors vehicle during driving with constant speed, while Stream 3 (shown in Fig. 3c) monitors individual vehicle in heavy traffic.

3.2 Time complexity

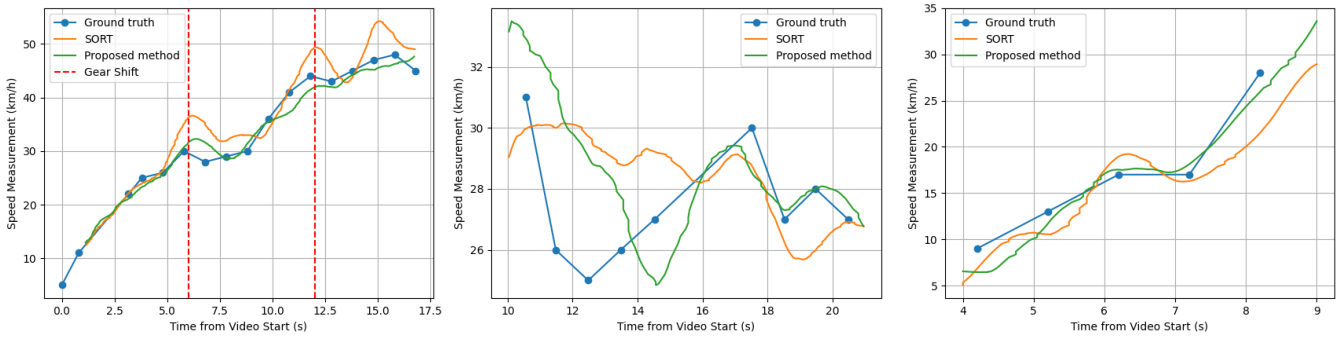
With our custom dataset, a pipeline integrating the YOLOv4 model, as well as its reduced version YOLOv4-tiny, was examined. As presented in Table 1, we achieved speed estimation of 11-18 frames per second (FPS). Most of the computing time is spent in the YOLO CNN, requiring 20-50 milliseconds, as seen in the OD column in Table 1, depending on the model and the input image size. There were no significant time differences with and without the use of our modified Kalman state.

Table 1: FPS rates and object detection times (OD) achieved on tested video streams with 1920 × 1080 px and 720 × 576 px resolutions and two different CNNs, namelly YOLOv4 and YOLOv4-tiny.

Stream size [px]	CNN	Stream 1		Stream 2		Stream 3	
		FPS	OD [ms]	FPS	OD [ms]	FPS	OD [ms]
1920× 1080	YOLO v4	9	28	9	29	9	33
	YOLO v4-tiny	11	26	11	26	10.5	27
720× 576	YOLO v4	11	22	11	22	14	47
	YOLO v4-tiny	18	36	17	33	17	33

3.3 Accuracy

In Figure 2a, we can observe a sample measurement of a vehicle acceleration from aerial footage in Stream 1, in order to compare the initial SORT state [2] of the Kalman filter with our modified 3D-based state. The accuracy of the ground truth and measurement data, is further corroborated by the fact that one can notice the two initial gear shifts. A second sample of measurement of a very



(a) (b) (c)

Figure 2: Comparison of the results achieved using SORT and the proposed method with ground truth data on (a) Stream 1, (b) Stream 2, and (c) Stream 3.

small speed variation across a 10 second period can be seen on Figure 2b. When the SORT's 2D-based Kalman state is used, because of its linear nature, we found that it will quite often overshoot, especially when the camera's elevation angle is relatively small. Nevertheless, we can see that even with imprecise camera and world unit calibration, one can achieve comparable results to the ground truth. Furthermore, as shown by Fig. 2c, comparable results are achieved also when observing an individual vehicle within a heavy traffic. It should, however, be noted that inaccuracies in the GPS and time data, camera and world unit calibration and bounding box detections are unavoidable.

4 CONCLUSION

In this paper, we presented our pipeline for vision-based vehicle speed estimation with many improvements of the algorithms used, both in object tracking and space transformations. We can conclude that camera and world unit calibration combined with modifications of the Kalman state, can yield results comparable to state-of-the-art custom space transformation methodologies. We note however, that due to the specificity of the problem, the lack of proper datasets and the amount of external factors involved, fully testing and comparing our method to closed-source state-of-the-art is practically impossible.

In the future, we also plan to work on adaptive camera calibration for airborne vehicles with KLV gyroscopic sensor streams and fully automatic intrinsic and extrinsic parameter camera calibration using convolutional neural networks.

ACKNOWLEDGMENTS

This work was supported by DARS d.d. as a part of the project "System for short and long-term traffic prediction with the use of artificial intelligence" and funded by Slovenian Research Agency grant number P2-0041 and L7-2633.

REFERENCES

[1] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. 2009. Visual tracking with online Multiple Instance Learning. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 983–990. https://doi.org/10.1109/CVPR.2009.5206737

[2] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. 2016. Simple Online and Realtime Tracking. *arXiv* (Feb. 2016). https://doi.org/10.1109/ICIP.2016.7533003 arXiv:1602.00763

[3] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv* (April 2020). https://doi.org/10.48550/arXiv.2004.10934 arXiv:2004.10934

[4] Sedat Doğan, Mahir Serhan Temiz, and Sıtkı Külür. 2010. Real Time Speed Estimation of Moving Vehicles from Side View Images from an Uncalibrated Video Camera. *Sensors (Basel, Switzerland)* 10, 5 (May 2010), 4805–24. https://doi.org/10.3390/s100504805

[5] Huanan Dong, Ming Wen, and Zhouwang Yang. 2019. Vehicle Speed Estimation Based on 3D ConvNets and Non-Local Blocks. *Future Internet* 11, 6 (May 2019), 123. https://doi.org/10.3390/fi11060123

[6] Martin A. Fischler and Robert C. Bolles. 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM* 24, 6 (June 1981), 381–395. https://doi.org/10.1145/358669.358692

[7] Ross Girshick. 2015. Fast R-CNN. , 1440–1448 pages.

[8] Lazaros Grammatikopoulos, George Karras, and Elli Petsa. 2005. Automatic estimation of vehicle speed from uncalibrated video sequences. *ResearchGate* (Jan. 2005).

[9] Alexander Grents, Vitalii Varkentin, and Nikolay Goryaev. 2020. Determining vehicle speed based on video using convolutional neural network. *Transp. Res. Procedia* 50 (Jan. 2020), 192–200. https://doi.org/10.1016/j.trpro.2020.10.024

[10] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. 2017. Mask R-CNN. , 2961–2969 pages.

[11] Zongjian He, Jiannong Cao, and Tao Li. 2012. MICE: A Real-time Traffic Estimation Based Vehicular Path Planning Solution Using VANETs. In *2012 International Conference on Connected Vehicles and Expo (ICCVE)*. IEEE, 172–178. https://doi.org/10.1109/ICCVE.2012.39

[12] João F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. 2012. Exploiting the Circulant Structure of Tracking-by-Detection with Kernels. In *Computer Vision – ECCV 2012*. Springer, Berlin, Germany, 702–715. https://doi.org/10.1007/978-3-642-33765-9_50

[13] Stephane Hess. 2004. Analysis of the Effects of Speed Limit Enforcement Cameras: Differentiation by Road Type and Catchment Area. *Transp. Res. Rec.* 1865, 1 (Jan. 2004), 28–34. https://doi.org/10.3141/1865-05

[14] Shuai Hua, Manika Kapoor, and David C. Anastasiu. 2018. Vehicle Tracking and Speed Estimation from Traffic Videos. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 153–1537. https://doi.org/10.1109/CVPRW.2018.00028

[15] Tingting Huang. 2018. Traffic Speed Estimation From Surveillance Video Data. , 161–165 pages.

[16] Saleh Javadi, Mattias Dahl, and Mats I. Pettersson. 2019. Vehicle speed measurement model for video-based systems. *Comput. Electr. Eng.* 76 (June 2019), 238–248. https://doi.org/10.1016/j.compeleceng.2019.04.001

[17] Rudolph Emil Kalman. 1960. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering* 82, Series D (1960), 35–45.

[18] Amit Kumar, Pirazh Khorramshahi, Wei-An Lin, Prithviraj Dhar, Jun-Cheng Chen, and Rama Chellappa. 2018. A Semi-Automatic 2D Solution for Vehicle Speed Estimation From Monocular Videos. , 137–144 pages.

[19] Jing Li, Shuo Chen, Fangbing Zhang, Erkang Li, Tao Yang, and Zhaoyang Lu. 2019. An Adaptive Framework for Multi-Vehicle Ground Speed Estimation in Airborne Videos. *Remote Sens.* 11, 10 (May 2019), 1241. https://doi.org/10.3390/rs11101241

[20] David Fernández Llorca, Antonio Hernández Martínez, and Iván García Daza. 2021. Vision-based vehicle speed estimation: A survey. *IET Intel. Transport Syst.* 15, 8 (Aug. 2021), 987–1005. https://doi.org/10.1049/itr2.12079

[21] Alan Lukežič, Tomáš Vojíř, Luka Čehovin Zajc, Jiří Matas, and Matej Kristan. 2018. Discriminative Correlation Filter Tracker with Channel and Spatial Reliability. *Int. J. Comput. Vision* 126, 7 (July 2018), 671–688. https://doi.org/10.1007/s11263-017-1061-3

[22] Diogo Carbonera Luvizon, Bogdan Tomoyuki Nassu, and Rodrigo Minetto. 2016. A Video-Based System for Vehicle Speed Measurement in Urban Roadways. *IEEE Trans. Intell. Transp. Syst.* 18, 6 (Sept. 2016), 1393–1404. https://doi.org/10.1109/TITS.2016.2606369

[23] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2015. You Only Look Once: Unified, Real-Time Object Detection. *arXiv* (June 2015). https://doi.org/10.48550/arXiv.1506.02640 arXiv:1506.02640

[24] Divya Sharma, Shilpa Sharma, and Vaibhav Bhatnagar. 2022. Automated Vehicle speed Estimation and License Plate Detection for Smart Cities Development. In *2022 IEEE World Conference on Applied Intelligence and Computing (AIC)*. IEEE, 378–383. https://doi.org/10.1109/AIC55036.2022.9848890

[25] Zheng Tang, Gaoang Wang, Tao Liu, Young-Gun Lee, and Jenq-Neng Hwang. 2017. Multiple-Kernel Based Vehicle Tracking Using 3D Deformable Model and Camera Self-Calibration. *ResearchGate* (Aug. 2017).

[26] Zheng Tang, Gaoang Wang, Hao Xiao, Aotian Zheng, and Jenq-Neng Hwang. 2018. Single-Camera and Inter-Camera Vehicle Tracking and 3D Speed Estimation Based on Fusion of Visual and Semantic Features. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 108–1087. https://doi.org/10.1109/CVPRW.2018.00022

[27] Jin-xiang Wang. 2016. Research of vehicle speed detection algorithm in video surveillance. In *2016 International Conference on Audio, Language and Image Processing (ICALIP)*. IEEE, 349–352. https://doi.org/10.1109/ICALIP.2016.7846573

[28] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. 2017. Simple Online and Realtime Tracking with a Deep Association Metric. *arXiv* (March 2017). https://doi.org/10.48550/arXiv.1703.07402 arXiv:1703.07402

[29] Jianping Wu, Zhaobin Liu, Jinxiang Li, Caidong Gu, Maoxin Si, and Fangyong Tan. 2009. An algorithm for automatic vehicle speed detection using video camera. In *2009 4th International Conference on Computer Science & Education*. IEEE, 193–196. https://doi.org/10.1109/ICCSE.2009.5228496

[30] Wencheng Wu, Vladimir Kozitsky, Martin E. Hoover, Robert Loce, and D. M. Todd Jackson. 2015. Vehicle speed estimation using a monocular camera. In *Proceedings Volume 9407, Video Surveillance and Transportation Imaging Applications 2015*. Vol. 9407. SPIE, 17–30. https://doi.org/10.1117/12.2083394

[31] Xiaodong Yu, Ping Xue, Lingyu Duan, and Qi Tian. 2007. An algorithm to estimate mean vehicle speed from MPEG Skycam video. *Multimed. Tools Appl.* 34, 1 (July 2007), 85–105. https://doi.org/10.1007/s11042-006-0073-8

Empirical evaluation of time complexity of ordered dictionaries with ALGator

Jani Suban
89222015@student.upr.si
Faculty of Mathematics, Natural Sciences
and Information Technologies,
University of Primorska
Glagoljaška 8
SI-6000 Koper, Slovenia

ABSTRACT

This paper presents an empirical evaluation of different implementations of an abstract data structure, the ordered dictionary. The implemented data structures are Binary Search Tree, AVL Tree, Red-Black Tree, Zip Tree, Skip List and 2-3 Tree. The paper aims to take a step forward in evaluating of data structures with ALGator. It also compares the Zip Tree with other ordered dictionaries to encourage the use of it as a simpler and comparable implementation.

The result of the testing the data structures, both with strictly increasing (worst case scenario) and random updates and query operations, shows that the time complexity is the same as expected.

KEYWORDS

Binary Search Trees, AVL Tree, Red-Black Tree, 2-3 Tree, Zip Tree, Skip List, ALGator

1 INTRODUCTION

There are various approaches to implement the abstract data structure dictionary. But all of them try to minimise the time complexity of the operations. The first approach is to use hash tables. Hash tables have an expected time complexity $O(1)$, but at the cost of losing the order of the data.

If the goal of the data structure is to preserve the order of the data, the appropriate implementation of a dictionary is a kind of Link List where all elements to the left of element x are less than x and all elements to the right of x are greater than x. The best possible time complexity for implementing such a dictionary is $O(\log n)$[1]. This can be achieved with Balanced Binary Search Trees (BBST).

This paper presents an empirical comparison between different implementations of ordered dictionaries such as Balanced Binary Search Trees, a 2-3 Tree, a Binary Search Trees and a Skip List. Furthermore, the time complexities of all tested implementations of ordered dictionary will be empirically evacuated and compared. All data structures were tested with an open source test suite ALGator [5].

2 RELATED WORKS

The motivation for the paper, comes from the lack of data structure comparison with the test suit ALGator. The only comparison was done in [6] where the author comperes Binary and Fibonacci Heap.

The AVL Tree and the Red-Black Tree were compered in [9]. The authors found that the height of the AVL Tree is lower than that of the Red-Black Tree and therefor it is faster. In the paper [7] the author compares the Skip List with the AVL Tree and the 2-3 Tree. The author could not find any significant difference in the time needed to perform operations.

3 ORDERED DICTIONARY

A dictionary is an abstract data structure that has three operations: *find* an element in the structure, *insert* an element into the structure, and *delete* an element from the structure. In addition to these three operations, the ordered dictionary has two additional operations to go through all the elements in order. The two additional operations are *next element*, which returns the next element in order, and *has next*, which returns whether a next element exists or not. Although there are internal/passive and external/active types of iterators, in the paper we will mainly focus on the find, delete and insert operations.

Table 1: Worst time complexity of insert, delete and find for all implemented data structures

Operations	Binary Search Tree	AVL Tree	Red-Black Tree	2-3 Tree	Zip Tree	Skip List
Insert	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
Delete	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
Find	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$

There are many different ways to implement an ordered dictionary that also has a time complexity of $O(\log n)$. This paper compares the performance of the following data structures with respect to real words: Binary Search Tree, AVL Tree, Red-Black Tree, 2-3 Tree, Skip List and Zip Tree. All of the above data structures, with the exception of the Binary Search Tree, have a time complexity of $O(\log n)$, as can be seen in the Table 1. In the rest of the chapter, the most important ideas behind the implemented and tested data structures are presented and described.

3.1 Binary Search Tree

The binary search tree (BST) is the simplest of all the data structures presented in this paper. The idea behind is to split the data into two parts. The left part (left subtree) stores all nodes whose value is less than that of the current node[2], and the right part (right subtree) stores all nodes whose value is greater than that of the current node [11].

[1] $\log n$ stands for $\log_2 n$ unless otherwise specified

[2] the root node of the subtree

DOI: https://doi.org/10.26493/scores23.06

The time complexity of insertion, deletion and search operations in a binary search tree is $O(n)$. This is because the tree becomes a linked list if the data is inserted into the tree in the worst possible way, e.g. in a strictly increasing way [11]. However, if the data stored in the tree was inserted randomly, then the time complexity of all three operations is $O(\log n)$ with a high probability [8].

3.2 AVL Tree

The AVL tree is the first balanced binary search tree presented in this paper. AVL trees achieve balance by limiting the height difference between the left and right subtree to a maximum of 1, $|L.height - R.height| \leq 1$. The tree can only become unbalanced during updates (insertion and deletion). That for when the tree is updated, it checks if it is still balanced, and if not, the nodes are rotated so that the tree is balanced again [1].

The time complexity of insert, delete and find operations in an AVL tree is $O(\log n)$. This is because in the worst case is need to reach the leaves. Since the difference in height of the subtrees is at most 1, this means that all levels except the two lowest are full, so the height of the tree is at most $\log n \quad 1 = O(\log n)$ [1].

3.3 Red-Black Tree

The red-black tree is another balanced binary search tree where the leaf nodes are always *NIL* . The balance is achieved by colouring the nodes red or black. The colouring is done with the following roles:

(1) each node is either black or red,
(2) all leaf nodes are black,
(3) red nodes have no red children,
(4) each path from the root to the leaf has the same number of black nodes,
(5) the root is always black.

The balance is achieved by requiring that the black height (number of black nodes on the path from root to leaf) is the same for each leaf. This means that the lowest leaf is at most 2 times lower than the highest. This means that $n \geq 2^b$ (b is the black height and n is the number of elements in the data structure) that for $b = O(\log n)$ and also height $h \leq 2b = O(\log n)$ [2].

The time complexity of insert, delete and find operations in a Red-Black tree is $O(\log n)$. This is because, in the worst case, is needed to reach the level before the leaves, and the height of the tree is $O(\log n)$ [2].

3.4 2-3 Tree

The 2-3 tree is the only tree presented in this paper that is not a Binary Search Tree, but a B-Tree, where $B = 3$ [3]. The 2-3 Tree has two types of nodes: a 2-node, which has two children (left and middle) and one key, and a 3-node, which has three children (left, middle and right) and 2 keys. All leaf nodes are on the same level. The insertion in the 2-3 trees and also in the B trees is done in the leaf node. If the node overflows, it has 3 keys, the node is split and the middle key is inserted into the parent node. If the parent node does not exist, the middle key becomes the new root node. Deletion is done by exchanging the value to be deleted with the minimum value in the right subtree: the middle one if the left key is deleted, or the right one if the right key is deleted. If the leaf node becomes empty, the tree must be corrected by rearranging the parent node so that it becomes a 2-node, or by splitting and rotating a sibling tree [4].

This way of insert and delete operation guarantees that the leaves are always on the same level, therefor the tree is balanced. Since the tree is balanced, all three operations have the time complexity of $O(\log n)$ [4].

3.5 Skip List

A skip list is the only non-tree dictionary implementation we will examine in this paper. Skip list is a probabilistic data structure. Probability is used to define the number of pointers to the next nodes in the list. A pointer at height x always points to the next node that has at least x pointers and skips all nodes with fewer pointers. The number of pointers is calculated with the Algorithm 1. The probability that a node has one pointer is $1/2$, two is $1/4$, ... The value max in the line 3 of the Algorithm 1 is the maximum number of pointers a node can have, and it its $\log_{1/p} n = \log n$, since $p = 1/2$ is the probability that the Pseudo Random Number Generator (PRNG) returns a Boolean value $TRUE$ [7].

```
   Output: Number of pointer to the next node
1  i = 1 ;
2  k = PRNG.boolean;
3  while k ∧ i< max do
4  |   i = i + 1;
5  |   k = PRNG.boolean;
6  end
7  return i
```

Algorithm 1: Algorithm for calculating the number of pointer to the next node

Because of node skipping, the expected time complexity is $O(\log n)$ with high probability for all three operations. In the worst case, if the skipping in the skip list is small, the time complexity of all three operations is the same as for the normal link list, $O(n)$ [7].

3.6 Zip Tree

The zip tree is a probabilistic data structure. It uses the idea of the skip list to "balance the list." So each node in the tree also stores its rank value. The rank of the node is assigned in the same way as the number of pointers a node has in the skip list, that for Algorithm 1 can be used for assigning the rank of the node. A new element is always inserted as a leaf of the tree, as in all binary search trees. On the way from the leaf to the root, the newly added node becomes the root of the subtree if its rank is greater than the rank of the root of the subtree. If both nodes have the same rank, the node with the smallest value becomes the new root. Deleting an element from the tree is done by finding the correct node, removing the pointer to the node and then combining the children of the node to create a new subtree. The operation of combining subtrees is called zipping [10].

Given the ranks of the nodes, the expected time complexity is $O(\log n)$ with high probability. In the worst case, if all nodes have the same rank and the data is inserted, strictly increasing, the time complexity is $O(n)$ [10].

4 TESTING

The test method used in this paper is to measure the time duration for performing all insert, find or delete operations. Each test can have a different number of operations. The number of operations for test i can take the following forms:

$$\begin{aligned} N_i &= (n_I, n_F, n_D), \\ N_i &= (n_I, n_D) = (n_I, 0, n_D), \\ N_i &= (n_I, n_F) = (n_I, n_F, 0), \\ N_i &= (n_I) = (n_I, 0, 0), \end{aligned} \tag{1}$$

where n_I is the number of insert operations performed in the test, n_F is the number of find operations performed in the test and n_D is the number of delete operations performed in the test.

Similarly, each test has its own set of timers, one for each operation. The time taken to execute all the operations of test i can take the following forms:

$$\begin{aligned} \tau_i &= (t_I, t_F, t_D) \\ \tau_i &= (t_I, t_D) = (t_I, 0, t_D) \\ \tau_i &= (t_I, t_F) = (t_I, t_F, 0) \\ \tau_i &= (t_I) = (t_I, 0, 0) \end{aligned} \tag{2}$$

where t_I represents the time taken for all n_I insert operations, t_F represents the time taken for all n_F search operations and t_D represents the time taken for all n_D delete operations. To get only the total time of a particular operation from the timer τ_i, call $\tau_i(X)$, where X is the selected operation. For example, the timer for deletion is obtained as follows: $\tau_i(D) = t_D$.

As can be seen in the Equation 1 and the Equation 2, each test must always contain the insert operations, otherwise the delete and find operations cannot be measured in a proper way. The tests in this paper always perform two operations, either insert and find or insert and delete. The tests are divided into two parts. In the first part of the test, all values are inserted into the data structure. In the second part of the test, either the find operation or the delete operation is performed. This testing method focused on a stated simple scenario that could be real but is not a typical real world scenario.

4.1 ALGator

ALGator is used for testing the implemented ordered dictionaries (see Section 3). ALGator is a test programme introduced in the paper [5]. ALGator allows the user to implement, test and evaluate algorithms and data structures with a single tool. At the moment, only the Java programming language is supported for the implementation of the algorithms and data structures.

In ALGator, the implementation of the tested algorithms and data structures is done with the help of an abstract class in which the entire test logic and the structure of the implemented algorithms or data structures are defined and written in Java. Although the test logic is implemented in the abstract class, the tests to be performed are written in a separate file. All implemented data structures or algorithms extend the abstract class. The evaluation of the test results is done by allowing the user of ALGator to visualise the test results in a way that best represents the results. The visualisations can be saved for further use.

4.2 Testing Sequences

In this paper, two scenarios are tested. The first is the worst case, or in this case the strictly increasing sequence presented in the section 4.2.1. The second is the expected scenario or in this case the random sequence presented in the section 4.2.2.

4.2.1 Strictly Increasing Sequence. This test is to show how the data structures behave when the data is stored in the worst possible way. One way to store the data in this way is to have a strictly increasing sequence of data. The sequence is strictly increasing if for each $i, j \in \mathbb{N} \wedge i < j \wedge x_i, x_j \in \mathbb{N} \wedge x_i < x_j$, where x_i and x_j are the elements stored in the data structure.

4.2.2 Random Sequence. This test is designed to show how data structures behave when data is stored in a random manner. Data stored in this way is not stored optimally, but it represents all possible ways of storing it. In this case, for each $i, j \in \mathbb{N} \wedge i < j \wedge x_i, x_j \in \mathbb{N} \wedge x_i \quad x_j$ where x_i and x_j are the elements stored in the data structure.

5 EVALUATION

The tests were performed on a computer with an AMD Ryzen 7 2700, with 8 cores and 16 threads, 16 GB RAM and Fedora 36 with Linux kernel 6.2.14-100. The ALGator version was 0.985, created in a Docker container running Ubuntu with Java 11.0.18. The implementation of Binary Search Tree, AVL tree[3], Red-Black Tree[4], 2-3 Tree[5], Skip list[6] and Zip tree were done in Java [7].

In order to eliminate the interference caused by background operation as much as possible, all tests were carried out 5 times, although the estimate must certainly can be improved by a larger number of test runs. The tests can be divided into two groups: random insertions, deletions and find operations and strictly increasing insertions, deletions and find operations.

All tests use integers in the range $[0, 100000]$ as input for the operation. The test is performed for a different number of elements stored in the data structure. The number of elements in the data structure goes from 50 to 400 elements in steps of 50. In a next step, the implemented data structures can be generalised to a generic data type.

The small size of the test set is due to the Java stack size overflowing while testing with ALGator. The reason for the Stack Overflow error is the recursive implementation of the data structures.

5.1 Random Tests

The random tests were implemented using the Java library `java.util.Random`. The library was used to generate the pseudo-random numbers that were used as input for the insert, find and delete operations. At the beginning of each part of the test, the seed was set to 0 so that the number of variables affecting the execution time of each round was minimised.

[3] Modification of the https://www.javatpoint.com/avl-tree-program-in-java
[4] Modification of the https://algorithmtutor.com/Data-Structures/Tree/Red-Black-Trees/
[5] Modification of the https://github.com/SValentyn/2-3-tree
[6] Modification of the https://www.baeldung.com/cs/skip-lists#bd-how-to-insert-into-a-skip-list
[7] Link to the implementation https://github.com/GioGiou/BinarySearchTree

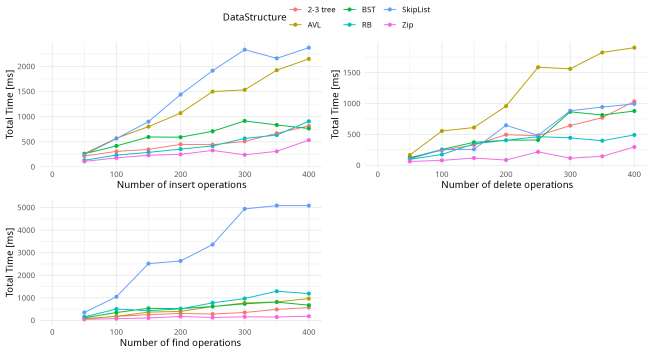


Figure 1: The time taken to perform all random insert, find and delete operations. The time is measured in milliseconds.

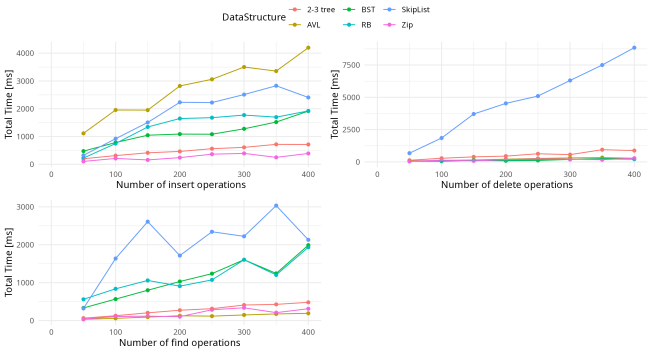


Figure 2: The time taken to perform all strictly increasing insert, find and delete operations. The time is measured in milliseconds.

The results of the testing can be seen in the Figure 1. In the top left graph it can be seen the change in total time for a given number of insertions. In the top right graph it can be seen the change in total time for performing a certain number of deletions. In the bottom left graph it can be seen the change in total time for performing a certain number of find operations. Each point in the graph represents an average of all 5 runs of the same test.

From all three graphs in the Figure1, it is evident that time $O(n \log n)$. This means that the time for all operations $O(\log n)$. This result confirms that all implemented data structures have $O(\log n)$ time complexity for insert, delete and find operations when the data is randomly inserted into the structure. The high execution time for the skip list, seen mainly in bottom left graph, is due to the probability with which the skip list is balanced, but the time for all operations is still $O(\log n)$.

5.2 Strictly Increasing Tests

The strictly increasing test is the implementation of the test method presented in the section 4.2.1. Sequential numbers were chosen over a more generic strictly increasing numbers because they are easier to implement. The result is expected to be the same as for the strictly increasing sequence, since a sequence of consecutive numbers is also a strictly increasing sequence.

The strictly increasing test consists of inserting, finding and deleting n consecutive numbers. This was implemented with a counter that starts with the value 0 and increases after each operation. The numbers that are inserted into the data structure are therefore integers from 0 to $n-1$. For the deletion, the counter was initialised with the value $n-1$ and after each deletion, the counter was decreased. If the counter for the deletion was not implemented in this way, the deletion is performed in a time of $O(1)$, because the deleted element is stored in the root of the tree each time.

The results of the testing can be seen in the Figure 2. In the top left graph it can be seen the change in total time for a given number of insertions. In the top right graph it can be seen the change in total time for performing a certain number of deletions. In the bottom left graph it can be seen the change in total time for performing a certain number of find operations. Each point in the graph represents an average of all 5 runs of the same test.

From all three graphs in the Figure 2, it is evident that time is rising $O(n \log n)$. This means that the time for all operations is $O(\log n)$. The only exception is the binary search tree, where the time increases faster, $O(n^2)$, which means that the time complexity of all tree operations is $O(n)$. This result confirms that all implemented data structures have a time complexity of $O(\log n)$ for insert, delete and find operations, except for the binary search tree, which has a time complexity of $O(n)$.

The reason why the time for inserting into the binary search tree is smaller than the time of the AVL tree and the Red Black tree is due to the rebalacing of the AVL and Red Black trees. For a larger test set, the time of the binary search tree will exceed the time of the AVL and Red-Black tree. The reason that the skip list with a time complexity of $O(\log n)$ performs worst is the lack of skipping, which can be seen especially in the top right graph in the Figure 2.

6 CONCLUSIONS AND FUTURE WORKS

This paper presented the empirical evaluation of time complexity for different implementations of ordered dictionaries. More specifically, balanced trees (AVL, Red-Black, Zip, 2-3 Tree), Skip List and Binary Search Tree. All data structures were tested using the ALGator test suit. Although the size of the test set was small, it can be seen from Figure 1 and Figure 2 that the time complexity of all data structures is $O(\log n)$, except for the binary search tree with strictly increasing insertion, deletion and search, where the time complexity is $O(n)$, as expected from the theoretically proven time complexity.

Our aim is to improve the outcome of this work in the future by increasing the size of the test sets. By doing so, we hope to obtain more accurate results and to better identify the differences in time complexity of all data structures. Furthermore, we plan to generalise the idea of Zip Trees from working with Binary Search Trees to working with k-ary Search Trees.

REFERENCES

[1] Georgii Maksimovich Adelson-Velskii and Evgenii Mikhailovich Landis. 1962. An algorithm for organization of information. In *Doklady Akademii Nauk*, Vol. 146. Russian Academy of Sciences, 263–266.

[2] Rudolf Bayer. 1972. Symmetric binary B-Trees: Data structure and maintenance algorithms. *Acta Informatica* 1, 4 (1972), 290–306. https://doi.org/10.1007/

bf00289509

[3] R. Bayer and E. McCreight. 1970. Organization and Maintenance of Large Ordered Indices. In *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control* (Houston, Texas) *(SIGFIDET '70)*. Association for Computing Machinery, New York, NY, USA, 107–141. https://doi.org/10.1145/1734663.1734671

[4] Thomas H. Cormen, Charles Eric Leiserson, Ronald Linn Rivest, and Clifford Seth Stein. 2009. *Introduction to Algorithms* (third ed.). MIT Press. https://mitpress.mit.edu/books/introduction-algorithms-third-edition

[5] Tomaz Dobravec. 2019. Implementation and Evaluation of Algorithms with ALGator. *Informatica (Slovenia)* 43 (2019).

[6] ALEKSANDAR GEORGIEV. 2022. *Primerjava implementacij dvojiške in Fibonaccijeve kopice.* https://repozitorij.uni-lj.si/IzpisGradiva.php?lang=slv&id=140423

[7] William Pugh. 1990. Skip Lists: A Probabilistic Alternative to Balanced Trees. *Commun. ACM* 33, 6 (jun 1990), 668–676. https://doi.org/10.1145/78973.78977

[8] Bruce Reed. 2003. The Height of a Random Binary Search Tree. *J. ACM* 50, 3 (may 2003), 306–332. https://doi.org/10.1145/765568.765571

[9] Svetlana Strbac-Savic and Milo Tomasevic. 2012. Comparative performance evaluation of the AVL and red-black trees. *ACM International Conference Proceeding Series*, 14–19. https://doi.org/10.1145/2371316.2371320

[10] Robert E. Tarjan, Caleb Levy, and Stephen Timmel. 2021. Zip Trees. *ACM Transactions on Algorithms* 17, 4 (oct 2021), 1–12. https://doi.org/10.1145/3476830

[11] P. F. Windley. 1960. Trees, Forests and Rearranging. *Comput. J.* 3, 2 (01 1960), 84–88. https://doi.org/10.1093/comjnl/3.2.84 arXiv:https://academic.oup.com/comjnl/article-pdf/3/2/84/1358330/030084.pdf

Evaluation of algorithms for finding shortest paths in a network

Dani Zugan
89212060@student.upr.si
Faculty of Mathematics, Natural Sciences and Information Technologies
Computer Science
University of Primorska
Glagoljaška 8
SI-6000 Koper, Slovenia

ABSTRACT

The paper evaluates three different algorithms for computing all-pairs shortest paths. We compare the well-known Floyd-Warshall algorithm with two simple modifications of it. The key difference lies in the fact that the relaxations are done in a smarter way. We evaluate the algorithms on three different graph models - uniform Erdős-Rényi, binomial Erdős-Rényi, and Albert-Barabási. Based on the results, we can observe that both modified algorithms outperform the Floyd-Warshall algorithm.

KEYWORDS

Tree algorithm, Hourglass algorithm, Floyd-Warshall algorithm, all-pairs shortest path.

1 INTRODUCTION

Graph theory has long grappled with the timeless challenge of finding the shortest paths within graphs, making it a classic problem in algorithmic studies. The key idea revolves around navigating a (directed) graph, where each arc carries a specific weight, in search of paths that minimize the sum of these arc weights. This fundamental problem finds applications in various real-world scenarios, including bioinformatics, logistics, telecommunications and so on [1].

Two prominent variations of this problem are the single-source shortest path and the all-pairs shortest path (APSP) problems. The single-source variant focuses on discovering paths from a fixed starting vertex to all other vertices in the graph, while the APSP entails finding the shortest path between every possible pair of vertices [4].

In this work, focus will be solely on the APSP variant, aiming to evaluate three different algorithms which offer effective solution. Generally, the APSP problem can be tackled using the technique of relaxation. The core concept of relaxation involves evaluating whether we can enhance the weight of the shortest path from vertex u to v by routing it through vertex w, updating it whenever necessary. Among the well-known relaxation-based solutions, one of the most straightforward approaches is the Floyd-Warshall dynamic programming algorithm which is among algorithms being evaluated. This algorithm boasts a time complexity of $O(n^3)$ when dealing with graphs containing n vertices. While its ease of implementation is commendable, two alternative algorithms will be evaluated which potentially improve efficiency and scalability for solving the APSP problem [4].

In this article, two modifications of the Floyd-Warshall algorithm are introduced and evaluated: the Tree algorithm and the Hourglass algorithm, described in paper Modifications of the Floyd-Warshall algorithm by Andrej Brodnik, Marko Grgurovič and Rok Požar. These modifications offer a notable difference from the original Floyd-Warshall algorithm by implementing a more intelligent approach to carrying out relaxations. This is achieved by introducing a tree-like structure that enables the algorithms to avoid unnecessary relaxations that do not contribute to the final result.

It's important to note that despite these enhancements, both the Tree algorithm and the Hourglass algorithm maintain a worst-case time complexity of $O(n^3)$, which is the same as the classic Floyd-Warshall algorithm. However, their expected performance is significantly improved in practical scenarios [3].

Algortihms are evaluated on graphs generated by using Erdős-Rényi and Albert-Barabási method [2]. Specifically graphs are directed with uniformly distributed arc weights on $[0, 1]$.

2 PRELIMINARIES

A directed graph (digraph), denoted as G, is represented as a pair (V, A), where V is a finite, non-empty set of vertices, and A is a set of arcs that are a subset of $V \times V$. V is denoted as $\{v_1, v_2, ..., v_n\}$ where n is a positive integer greater than or equal to 2 [3].

A path, denoted as P, within a digraph G connecting vertices $v_{P,0}$ to $v_{P,m}$ is a finite sequence of distinct vertices $P = v_{P,0}, v_{P,1}, ..., v_{P,m}$. It is important to note that for $i = 0, 1, ..., m1$, the pair $(v_{P,i}, v_{P,i\ 1})$ must represent an arc in the graph G. The length of a path P, represented as |P|, is simply the count of vertices along the path. A k-path refers to a path in which all intermediate vertices are selected from a specific subset $\{v_1, v_2, ..., v_k\}$, where k is a positive integer greater than or equal to 2 [3].

A weighted digraph G is defined as (V, A) with the inclusion of a weight function denoted as w. This function assigns a weight $w(a)$ to each arc a within the set A. In essence, a shortest path from vertex u to vertex v is a path in G whose total weight is the lowest among all possible paths from u to v. The distance between any two vertices, u and v, is defined as the weight of shortest path from u to v within the graph G [3].

3 ALGORITHMS

3.1 The Floyd-Warshall algorithm

The Floyd-Warshall algorithm is a well-known algorithm which uses simple dynamic programming approach to solve APSP on a weighted, directed graph.The key idea behind the Floyd-Warshall algorithm is that it considers all possible paths between pairs of vertices and gradually refines the shortest distances until it computes the shortest paths for all pairs. Its time complexity is $O(|V|^3)$ due

DOI: https://doi.org/10.26493/scores23.07

to three nested for loops and does not depend on number of arcs. The pseudocode of Floyd-Warshall algorithm is given in Algortihm 1 [3, 4].

```
1 for k := 1 to n do
2     for i := 1 to n do
3         for j = 1 to n do
4             if W_ik   W_kj < W_ij then
5                 W_ij := W_ik   W_kj;
6             end
7         end
8     end
9 end
```

Algorithm 1: Floyd-Warshall(W)

3.2 The Tree algorithm

The Tree algorithm is the first version of the modified Floyd-Warshall algorithm. Consider the k-th iteration, and let OUT_k represent a shortest path tree originating from vertex v_k. It is based on the observation that the relaxation in lines 4-5 would not always succeed in lowering the value of W_{ij}. Instead of simply looping through every vertex of V in line 3, we perform the depth-first traversal of OUT_k. This allows us to skip iterations where it's guaranteed that they won't reduce the current value of W_{ij} [3].

The pseudocode for the enhanced algorithm, known as the Tree algorithm, can be found in Algorithm 2. The first step involves the construction of the tree OUT_k using the ConstructOUT procedure outlined in Algorithm 3. Subsequently, a depth-first search operation is carried out [3].

```
1 Initialize π, an n × n matrix, as π_ij := i;
2 for k := 1 to n do
3     OUT_k := ConstructOUT_k(π);
4     for i := 1 to n do
5         Stack := empty;
6         Stack.push(v_k);
7         while Stack   empty do
8             V_k := Stack.pop();
9             forall children v_j of v_x in OUT_k do
10                if W_ik   W_kj < W_ij then
11                    W_ij := W_kj + W_kj;
12                    π_ij := π_kj;
13                    Stack.push(v_j);
14                end
15            end
16        end
17    end
18 end
```

Algorithm 2: Tree(W)

In Algorithm 2 vertices of OUT_k are visited in DFS order, which is facilitated by using the stack. We can avoid pushing and poping of

```
1 Initialize n empty trees: T_1, T_2, ..., T_n.;
2 for k := 1 to n do
3     T_1.Root := v_1;
4 end
5 for i := 1 to n do
6     if i   k then
7         Make T_i a subtree of the root of T_ki.
8     end
9 end
10 return T_k
```

Algorithm 3: $ConstructOUT_k(\pi)$

each vertex by precomputing two read-only arrays dfs and $skip$ to support the traversal of OUT_k. The array dfs comprises the vertices of OUT_k that have been visited during the depth-first search (DFS) traversal. In contrast, the array $skip$ serves the purpose of bypassing the subtree of OUT_k when relaxation of edges in that subtree does not yield successful results [3].

3.3 The Hourglass algorithm

The Tree algorithm can be further improved by using another tree. The second tree, represented as IN_k, resembles a shortest path tree, with the difference being that it serves as a shortest path "graph" for routes from v_i to v_k for every $v_i \in V$ except v_k. Precisely IN_k is not a tree, but if direction of arcs is reversed, it shifts it into a tree with v_k as the root. This is actually a substitute of the for loop on variable i in line 2 of Algorithm 1 and in line 4 of Algorithm 2. This modification of Floyd-Warshall algorithm is named the Hourglass algorithm, the name comes from placing IN_k tree atop the OUT_k tree, which gives it an hourglass-like shape, with v_k being at the neck. The pseudocode of the Hourglass algorithm is given in Algorithms 4 and 5. Additional algorithm constructs IN_k similarly to the construction of OUT_k, except that the matrix ϕ_{ik} is used instead [3].

```
1 Initialize π, an n × n matrix, as π_ij := i;
2 Initialize φ, an n × n matrix, as φ_ij := i;
3 for k := 1 to n do
4     OUT_k := ConstructOUT_k(π);
5     IN_k := ConstructIN_k(φ) forall children v_i of v_k in IN_k
       do
6         RecurseIN(W, π, IN_k, OUT_K, v_i);
7     end
8 end
```

Algorithm 4: $Hourglass(W)$

In practice, the algorithm's efficiency can be enhanced by employing an additional stack instead of recursion. This optimization significantly speeds up the implementation process. It's important to highlight that the worst-case time complexity of the Hourglass and Tree algorithm stays at $O(n^3)$. This scenario is evident when all shortest paths are direct arcs themselves, resulting in a tree structure where all leaves are children of the root, and this configuration remains unchanged throughout the algorithm's execution [3].

```
1  Stack := empty;
2  Stack.push(v_k);
3  while Stack ≠ empty do
4  |  v_x := Stack.pop();
5  |  forall children v_j of v_x in OUT_k do
6  |  |  if W_ik + W_kj < W_ij then
7  |  |  |  W_ij := W_ik + W_kj;
8  |  |  |  π_ij := π_kj;
9  |  |  |  φ_ij := φ_ik;
10 |  |  |  Stack.push(v_j);
11 |  |  else
12 |  |  |  Remove the subtree of v_j from OUT_k;
13 |  |  end
14 |  end
15 end
16 forall children v_i' of v_i in IN_k do
17 |  RecurseIN(W, π, φ, IN_k, OUT_K, v_i');
18 end
19 Restore OUT_k by reverting changes done by all iterations of
     line 12;
```

Algorithm 5: *RecurseIN*$(W, \pi, IN_k, OUT_K, v_i)$

4 EVALUATION

4.1 Testing environment

All algortihms and graph generators were implented in C and C++ and compiled using gcc version 6.3.0. The tests were ran on an AMD Ryzen 7 5700U with 16GB RAM running on Windows 11 64-bit.

4.2 Graph generation

Three different variant of random graphs were generated using Erdős-Rényi model and Albert-Barabási model[1]. At the beginning of generating each variant seed was set to 1 and was incremented by +1 for each new graph. All graphs generated underwent a DFS search to ensure strong connectivity[2].

Firstly, using binomial Erdős-Rényi model the input values to random graphs were the number of vertices, denoted as n, and probability, denoted as p. Each possible arc $(n*(n-1))$ in a directed graph is included with probability p, independently from every other arc. Weights are added uniformly from the interval [0,1]. Four different sizes of graphs (512, 1024, 2048 and 4096 vertices) and five different probabilities (0.1, 0.3, 0.5, 0.7, 0.9) were selected as input. For each instance of the input five different graphs were created, all together 100 different graphs were created using this model.

Secondly, a different variant was used, called uniform Erdős-Rényi model. Input values for creating graphs were number of vertices and number of arcs, denoted by m. Out of all $n*(n-1)$ possible arcs in a directed graph a random permutation was made to select the desired number of arcs. Weights are added uniformly from the interval [0,1]. Again the sizes of graphs were 512, 1024, 2048 and 4096 vertices and inputs for m were $5*n$, $10*n$, $20*n$, $50*n$, $100*n$. All together 100 different graphs were created using this model.

Finally, using Albert-Barabási model the input values for creating random graphs were number of vertices of final graph, number of vertices of initial graph, denoted by c, and number of arcs added in each round, denoted by m. Initially a clique of size c is created. At each step, one new vertice is added, with m new arcs to the vertices already in the graph. With preferential attachment the vertices with higher degree have a higher probability to receive new arcs. The graph constructed after $n-c$ steps is undirected. To determine the direction of each arc, a function similar to a coin flip is employed. This function randomizes the orientation of the arcs, ensuring an equal distribution where half of the arcs lie above the main diagonal in the adjacency matrix, and the other half lie below it. For this evaluation the m value was fixed at 15 and C value was fixed at 30. Again the sizes of graphs were 512, 1024, 2048 and 4096 vertices and for each instance 20 different graphs were generated (80 together). Weights are added uniformly from the interval [0,1].

4.3 Evaluation

As mentioned, Tree and Hourglass algorithm were compared with Floyd-Warshall algorithm. The results on graphs generated by binomial Erdős-Rényi model are shown in figure 1, by uniform Erdős-Rényi model are shown in figure 2 and by Albert-Barabási model are shown in figure 3. To better visualize the results, natural logarithm of arcs is used on the x axis. Both modifications performed much better than Floyd-Warshall algorithm, especially as the number of vertices gets higher. It is worth noting that, between the Tree and Hourglass algorithms, the Tree algorithm demonstrated slightly better results in the performance comparison.

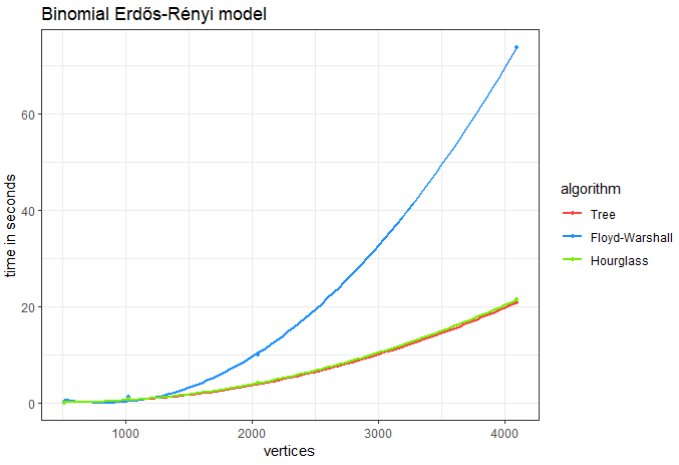


Figure 1: Binomial Erdős-Rényi model results

In figure 4 results of all three variants of graphs are shown. As expected both modifications were slower as the graph got more dense and the number of vertices stayed the same. Moreover also Floyd-Warshall algorithm was a little bit slower as graph got denser which is unexpected.

[1]Link to code for generating random graphs is available here: GitHub Repository
[2]In Albert-Barabási model some seeds did not produce strongly connected graph and therefore more than 80 seeds were used.

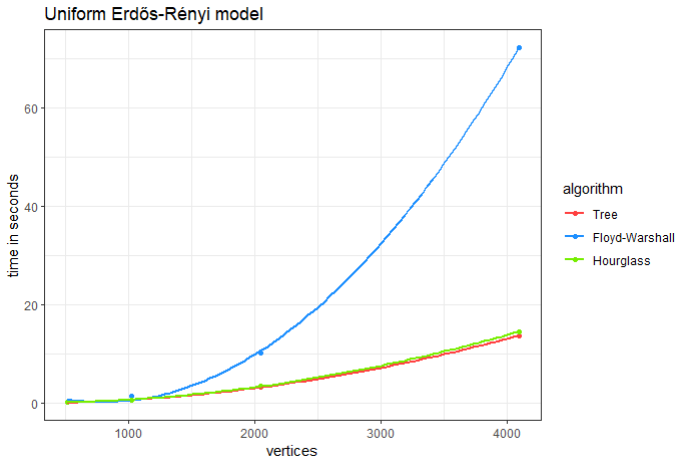


Figure 2: Uniform Erdős-Rényi model results

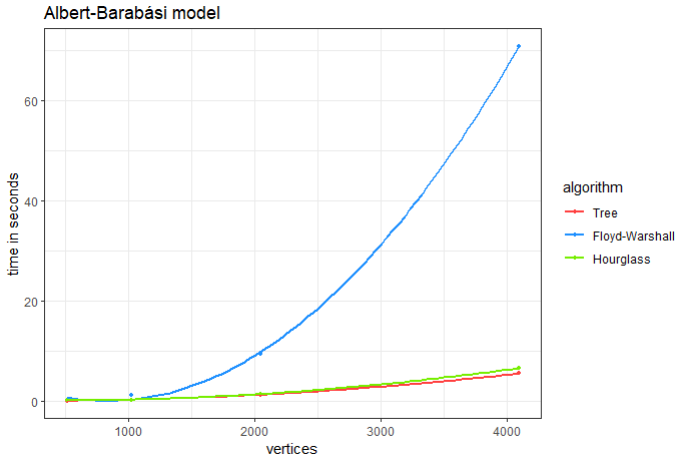


Figure 3: Albert-Barabási model results

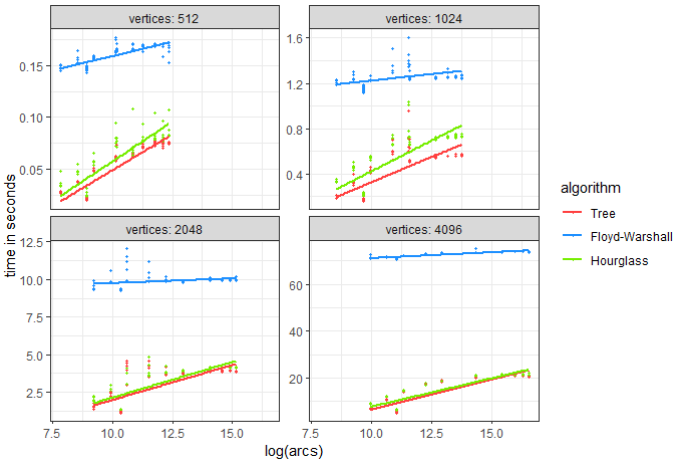


Figure 4: Combined results

5 CONCLUSIONS

In this paper, a straightforward evaluation of three algorithms for finding shortest paths in a network was presented. Random networks were generated using the Erdős-Rényi and Albert-Barabási models. The paper's results showed that both modifications of the algorithms performed better than the Floyd-Warshall algorithm, especially when the size of the graphs (vertices) was larger.

To further enhance the comprehensiveness of the evaluation, the Tree and the Hourglass algorithms will be further assessed on graphs that are not strongly connected. By incorporating non-strongly connected graphs into the assessment, deeper insights into the behavior and robustness of the algorithms in real-world scenarios, are expected to be gained.

REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Inc., USA.

[2] Albert-Laszlo Barabasi and Rita Albert. 1999. Albert, R.: Emergence of Scaling in Random Networks. Science 286, 509-512. *Science (New York, N.Y.)* 286 (11 1999), 509–12. https://doi.org/10.1126/science.286.5439.509

[3] A. Brodnik, M. Grgurovič, and R. Požar. 2021. Modifications of the Floyd-Warshall Algorithm with Nearly Quadratic Expected-Time. *ARS MATHEMATICA CONTEMPORANEA* (2021). https://amc-journal.eu/index.php/amc/article/view/2467

[4] R. W. Floyd. 1962. Algorithm 97: Shortest path. *Commun. ACM* 5 (1962), 345. https://doi.org/10.1145/367766.368168

Concurrent migration of containers in decentralized cloud computing network

Andrej Erjavec
89201090@student.upr.si
Faculty of Mathematics, Natural Sciences and Information Technologies,
University of Primorska
Glagoljaška 8
SI-6000 Koper, Slovenia

Aleksandar Tošić*
aleksandar.tosic@upr.si
Faculty of Mathematics, Natural Sciences and Information Technologies,
University of Primorska
Glagoljaška 8
SI-6000 Koper, Slovenia

ABSTRACT

In this paper we propose a new algorithm for decentralized application orchestration in Nion Network. The proposed algorithm improves on the existing implementation by performing multiple migrations between nodes such that it does not create race conditions. Moreover, we show experimental data testing various statistical measures of fitness in an effort to find the most suitable one. Our results show a significant improvement over the existing.

KEYWORDS

Decentralization, blockchain, edge computing, cloud computing, decentralized orchestration, application migration, containerization.

1 INTRODUCTION

The evolution of the internet and a growing number of its services started to reveal the shortcomings of centralized service architecture that has become unsuited to certain emerging trends [6]. Cloud computing, which is, like other internet services, highly centralized, is one of the major fields where the disadvantages of centralization are the most obvious [5]. Among these are low fault tolerance, a significant amount of processing load on data centers, and a difficult process of service migration between cloud infrastructure providers (vendor lock-in). Edge computing aims to address some of these challenges by distributing system resources as well as data across a multitude of nodes [3], but soon it was realized that edge devices are significantly underutilized in terms of system resource consumption and the coordination required for consistent system operation still relies on a centralized orchestrator. These findings have driven the development of a blockchain-based protocol for decentralized cloud computing, named Nion Network [4]. The protocol addresses multiple issues of existing solutions, the most significant change it delivers is support for fully decentralized orchestration. In addition, it aims to solve the issue of edge device heterogeneity by enabling their homogeneous operation that is independent of hardware configuration. The latter is achieved by packing applications into containers and performing migrations of containers during run-time. The migrations are validated by a consensus mechanism and are recorded on the blockchain in order to provide a verifiable and transparent log of system state changes.

*Also with InnoRenew CoE.

DOI: https://doi.org/10.26493/scores23.10

2 MOTIVATION

The migrations of containers in Nion Network are determined by a deterministic algorithm that produces a migration plan based on system resource statistics of nodes inside a network. These are collected and propagated to a block producer in each slot when a new block is created. The main goal of migrations is to evenly distribute a resource load across the network. However, the current implementation of the algorithm only supports up to one migration to be performed in each block, which significantly increases the time needed for the network to stabilize and hence reduces its overall performance. The operation of the algorithm is based on the concept of minimizing the load of most loaded nodes while maximizing the load of least loaded nodes to achieve an even distribution of system resources. Each migration can be represented as a vector containing three values:

$$\{containerToMigrate, sourceNode, destinationNode\}$$

The algorithm always takes the most loaded node as a source and the least loaded node as a destination. Among containers from the most loaded node, the one with the highest CPU usage is selected to be migrated. In addition, the impact of the proposed migration is calculated before the execution to evaluate whether the migration would improve the state of the system. As a measure of system stability, the difference between the most and the least loaded node is taken and compared before and after migration. For simplicity, the CPU load is expressed in percentage where we assume the value is normalized by hardware performance.

The migration is executed only in case the difference is lower after execution. In case the migration is performed, its goal is reached since the migration contributes to a more even distribution of resource consumption. However, when observing the possible scenarios, we noticed that the proposed migration might not be ideal since only one container is considered in the process of selection. There might exist another migration between the same nodes that would produce a lower difference once executed. Additionally, by not considering all possible containers from the most loaded node, there is a possibility of migration not being performed in the current slot due to a calculated negative impact on system stability, which leaves the system deprived of a possible improvement.

3 CONTRIBUTION

In the practical part of this study, we implement an algorithm that addresses the issues of the current implementation. The problem

```
Data: BlockData
Result: Migration plan
if !AppQueue.isEmpty() then
    while !AppQueue.isEmpty() do
        Min ⇐ FindMinLoadedNode(BlockData);
        Min.addApp(AppQueue.dequeue());
    end
else
    AppToMigrate ⇐ Max.MaxLoadApp;
    DeltaScore ⇐ (Max.score − Min.score);
    NextDeltaScore ⇐
     (Max.score − AppToMigrate.score) − (Min.score
     AppToMigrate.score);
end
if Math.abs(DeltaScore > NextDeltaScore) then
    Migrate(AppToMigrate, Min);
end
```

Algorithm 1: Deterministic migration algorithm

we are dealing with is similar to process scheduling with the goal of minimizing the total time required to complete all tasks. In this context, optimization algorithms like Longest-processing-time-first (LPT) and Shortest-processing-time-first (SPT) are often used [2]. However, these algorithms are designed for systems where the number and difficulty of all tasks are known prior to execution making them unsuitable for a dynamic network like Nion where the load distribution is constantly changing with new applications entering the network in each slot.

3.1 Implementation

The proposed solution is designed to compute a migration plan based on the current state of the network. To provide a separate testing environment for new solutions, we implemented a simplified simulation of the Nion protocol, which tries to mimic the dynamic nature of the real network to ensure similar conditions. The implementation is executed in three steps where each aims to improve a different aspect of the algorithm, while the main focus is reducing the time needed for the network to stabilize.

3.1.1 Improvement 1: Enabling multiple migrations per block. First, we focus on solving the most significant limitation of the algorithm while keeping the existing concept for container selection. As a measure of system stability, the difference between the most and the least loaded node is taken as in the case of the original algorithm. Multiple migrations per block are enabled by building a migration plan inside a loop, which iterates as long as the migration proposed in the current iteration still improves the system stability. We also observed that once the local minimum of a difference is reached, at the same time the global minimum for that slot is reached as well which allows us to break the loop and return the produced migration plan.

3.1.2 Improvement 2: Improved procedure of container selection. The previous change to the algorithm enables multiple migrations per block but still leaves room for improvement. In this step, we address the problem of non-ideal migrations that can slow down the process of load balancing between nodes or miss a chance for stability improvement. The goal of this step is to consider all containers from the most loaded node in the process of selection, not only the one with the highest CPU usage. Instead of directly selecting the most loaded container, We first evaluate the impact of migration for each of these containers and select the one that has the greatest impact on system stability when migrated. As in the previous improvement, the migrations are being added to the plan as long as the migration in the current iteration produces a lower difference compared to the previous iteration.

3.1.3 Improvement 3: Use of standard deviation as a measure of system stability. Since the problem involves evenly distributing system resources, the use of standard deviation as a measure of system stability seems reasonable. With this approach, network stability is obtained by calculating the standard deviation of CPU usage across all containers on the network using the following formula:

$$stdDev = \sqrt{\frac{1}{|N|} * \sum_{n \in N} (n_{cpu} \overline{N}_{cpu})^2} \tag{1}$$

In the previous versions of an algorithm, the execution of migration is decided based on comparing the difference between the most and the least loaded node in the current and the previous iteration. However, these nodes are most likely to be different once a migration is performed which makes such a comparison method inappropriate and may result in sooner than desired termination of the algorithm. The use of standard deviation enables us to evaluate the impact of individual migrations not only on nodes involved in migration but on the network as a whole. The migration plan is still generated inside a loop with the difference in standard deviation now taking the role of a measure. Similar to previous cases the loop continued as long as the standard deviation was lower than the one in the previous iteration. In the previous cases, we were able to evaluate the impact of migration before the execution since only the involved nodes were considered. In the case of standard deviation, however, the same can not be achieved since the calculation of standard deviation involves all nodes and containers on the network. It is therefore not possible to make an evaluation before the migration is executed. For this reason, the migration has to be simulated prior to calculation in order to assess its impact on the network. An important note is that the integration of such an algorithm into an actual network would require the block producer node to have the ability to perform a simulation of migration. That can be achieved by implementing a system able of making a copy of the network state based on received resource statistics and using this model as a simulation environment.

This step also completes the final version of our solution (Algorithm 2).

4 RESULTS

In this section, we present the results obtained during the testing of improved versions of the migration algorithm. The tests are done for each of the mentioned improvements to compare its performance against previous implementations. Each test is performed based on average and worst-case scenarios. In the average case, a

```
   Data: BlockData
   Result: Migration plan[]
 1 MigrationPlan ⇐ [];
 2 DstCandidates ⇐ BlockData;
 3 StdDev ⇐ GetStdDev();
 4 StdDevPrev ⇐ StdDev;
 5 while StdDev <= StdDevPrev & DstCandidates.size > 0
   do
 6     Max ⇐ FindMaxLoadedNode(BlockData);
 7     Min ⇐ FindMinLoadedNode(DstCandidates);
 8     AppToMigrate ⇐ Max.MaxLoadedApp;
 9     Containers ⇐ Max.getContainers();
10     Containers.sort(CPU);
11     LoadDelta ⇐ Max.cpu − Min.cpu;
12     for Container : Containers do
13         NextDeltaScore ⇐
            (Max.cpu − AppToMigrate.cpu) − (Min.cpu
            AppToMigrate.cpu);
14         if NextLoadDelta > Delta then
15             AppToMigrate ⇐ Container;
16             break;
17         end
18         LoadDelta ⇐ NextLoadDelta;
19     end
20     SimulateMigration(AppToMigrate, Max, Min);
21     DstCandidates.remove(Min);
22     StdDevPrev ⇐ StdDev;
23     StdDev ⇐ GetStdDev();
24     if StdDev < StdDevPrev then
25         Migration ⇐ {AppToMigrate, Min, Max};
26         MigrationPlan.add(Migration);
27     end
28 end
```

Algorithm 2: Final version of migration algorithm

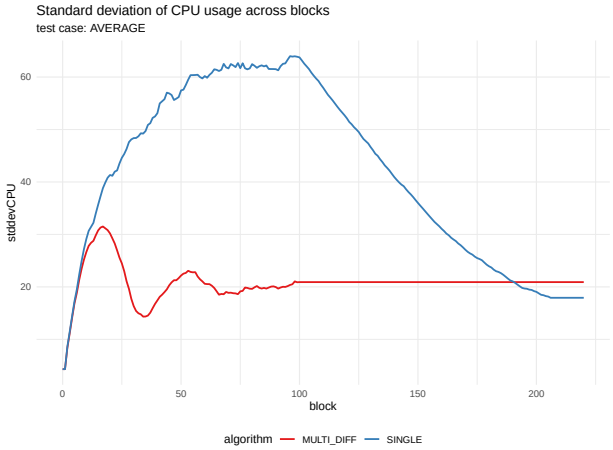


Figure 1: Comparison of performance between single and multiple migrations per block

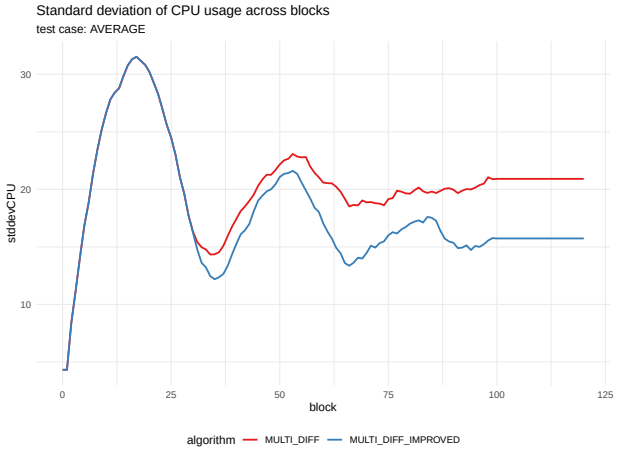


Figure 2: The effect of improved container selection on standard deviation

fixed number of containers is added to a randomly selected node in each iteration while in the worst case, the containers are always added to the same node. When a desired number of containers on the network is reached, the adding stops. We take the standard deviation of CPU load across the network as a measure of performance for all tests. The testing is performed on a simulated network with 1000 containers and 256 nodes.

After the first improvement, which enabled multiple migrations per block, we see significant improvement in terms of time required for stabilization (Figure 1). While it took around 200 blocks for the original algorithm to complete the stabilization, the new version required around 100 blocks, which makes it reach the minimum standard deviation almost two times faster. As shown in the figure, the standard deviation starts to decline much sooner when performing multiple migrations per block. However because of rapid stabilization in the early phases of the process when new blocks are still entering the network, the standard deviation is not in constant decline. When a stable-enough state is reached, new migrations can not improve the stability for the next sequence of blocks making the standard deviation rise again. The rising ends when enough new containers enter the network and an algorithm is again able to produce migrations that start improving the stability.

The next version of the algorithm where we implemented an improved routine for container selection shows an improved performance when compared to the initial multi-migration-per-block algorithm (Figure 2). Since the selection is such that the proposed migration has the largest impact on the stability, we are able to reach a lower standard deviation as in the previous case when a more naive approach for selection was used.

Changing the method for measuring stability with the use of standard deviation as a measure further contributed to reaching better stability of the network (Figure 3). The cause for the improvement is widening the scope on which the new algorithm is able

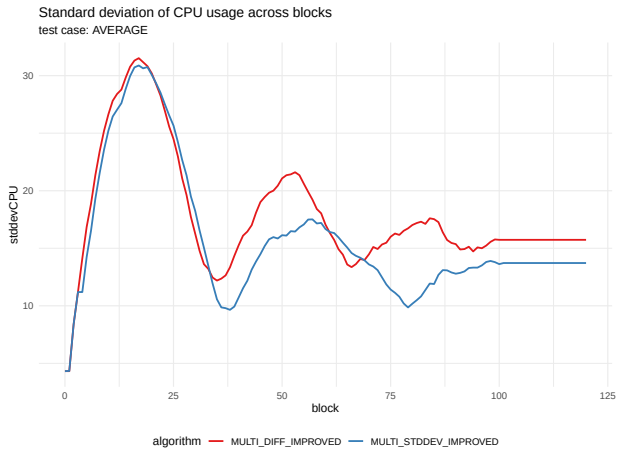


Figure 3: The effect of improved container selection on standard deviation

to evaluate the impact of migrations. This contributes to a higher number of migrations per block as we show in Table 1 resulting in a lower standard deviation reached.

Another important indicator, that can be used to assess the performance of our algorithm, is the number of migrations per block an algorithm is able to produce. We observe that the last improvement where the standard deviation is used in combination with improved container selection produces the highest number of migrations.

Table 1: Migrations per block based on algorithm

	number of migrations per block		
algorithm	min	mean	max
SINGLE	1	1	1
MULTI DIFF	1	2.79	9
MULTI DIFF IMPROVED	1	2.75	9
MULTI STDDEV	1	2.66	10
MULTI STDDEV IMPROVED	1	4.60	18

4.1 Quality of proposed solution

In the end, we want to measure the quality of our proposed solution. In this case, the algorithms for process scheduling represent a good reference for comparison. Even though their usage is unsuitable for our use case, the fact they provide an optimal distribution of resources enables us to make an evaluation of how close our solution is to being optimal. We took the Longest processing time algorithm (LPT) [1] as a reference since it provided the finest results. We tested the final version of our algorithm and collected the results of 20 tests where a new simulation network was generated in each test to provide variability in conditions. We present the results in Table 2.

As expected there is a gap in performance between algorithms, but considering the different use cases these algorithms are intended for, we conclude that the outcome generated by our solution is acceptable.

Table 2: Comparison of our algorithm with LPT

	std. dev. of CPU usage (%)	
case/algorithm	our implementation	LPT
average case	13.85	2.31
worst case	16.09	4.07

5 CONCLUSION

In this paper, we presented the main operational concept of the Nion protocol and highlighted the main drawbacks of its migration algorithm. We extended our findings by proposing an improved version of the algorithm that helps to achieve faster network stabilization times. Our solution enables multiple migrations per block and improves the concept of selecting the container to be migrated.

In the future, we could continue our work on improving the algorithm by implementing more advanced optimization techniques to further improve its performance and tune its operation to more specific requirements. One possible step in the evolution of our algorithm would be the use of multi-criteria optimization where multiple parameters are considered as opposed to the current solution where decisions are made based on CPU usage only. By setting priorities on the parameters we would get an algorithm that is more flexible and adjustable to current network needs and features.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the European Commission for funding the InnoRenew CoE project (H2020 Grant Agreement #739574) and the PHArA-ON project (H2020 Grant Agreement #857188) and the SRC-EDIH project (DIGITAL-2021-EDIH-01 call, project number: 101083351) and the Republic of Slovenia (Investment funding of the Republic of Slovenia and the European Union of the European Regional Development Fund), as well as the Slovenian Research Agency (ARRS), for supporting project number J2-2504.

REFERENCES

[1] Asep Anwar, Didit Damur Rochman, and Rendiyatna Ferdian. 2021. Parallel Machine Scheduling with Shortest Processing Time (SPT) and Longest Processing Time (LPT) TO Minimize MAKESPAN at PT. ABC. *Geographical Education (RIGEO)* 11, 6 (2021), 403–407.

[2] Jun-Ho Lee and Hoon Jang. 2019. Uniform Parallel Machine Scheduling with Dedicated Machines, Job Splitting and Setup Resources. *Sustainability* 11, 24 (2019). https://doi.org/10.3390/su11247137

[3] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646. https://doi.org/10.1109/JIOT.2016.2579198

[4] Aleksandar Tošić, Jernej Vičič, Michael Burnard, and Michael Mrissa. 2022. A Blockchain-based Edge Computing Architecture for the Internet of Things. (2022). https://doi.org/10.20944/preprints202111.0489.v2

[5] Magnus Westerlund and Nane Kratzke. 2018. Towards Distributed Clouds: A Review About the Evolution of Centralized Cloud Computing, Distributed Ledger Technologies, and A Foresight on Unifying Opportunities and Security Implications. In *2018 International Conference on High Performance Computing Simulation (HPCS)*. 655–663. https://doi.org/10.1109/HPCS.2018.00108

[6] Wenli Yang, Erfan Aghasian, Saurabh Garg, David Herbert, Leandro Disiuta, and Byeong Kang. 2019. A Survey on Blockchain-Based Internet Service Architecture: Requirements, Challenges, Trends, and Future. *IEEE Access* 7 (2019), 75845–75872. https://doi.org/10.1109/ACCESS.2019.2917562

How to Set the Maximum Number of Function Evaluations for the L-SHADE Algorithm with the AS^3D Approach?

Jana Herzog
jana.herzog1@um.si
Faculty of Electrical Engineering and Computer Science, University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

Janez Brest
janez.brest@um.si
Faculty of Electrical Engineering and Computer Science, University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

Borko Bošković
borko.boskovic@um.si
Faculty of Electrical Engineering and Computer Science, University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

ABSTRACT

The control parameter, maximum number of function evaluations plays two important roles during the optimization process. It can determine the population size of some evolutionary algorithms and it also serves as a stopping condition of an optimization process. In this paper, we focus on setting the value of the control parameter for the L-SHADE algorithm for a chosen large-scale benchmark function. For this purpose we utilized a recently proposed approach AS^3D, which enables us to predict a stopping condition with a certain probability for a given solver and optimization problem, while using the fixed-target approach.

KEYWORDS

stopping condition, evolutionary algorithm, target approach

1 INTRODUCTION

Recently proposed algorithms, such as jSO [6], L-SHADE [19], iL-SHADE [5] and MadDE [4] use the maximum number of function evaluations (*maxFEs*) as a control parameter. This has two roles during an optimization process. It influences the population size and it serves as a stopping criteria. When a specific value of function evaluations as a stopping condition is set, this is the fixed-budget approach [13].

In competitions, such as CEC [1], the maximum number of function evaluations is set as a stopping condition. The value of it is predetermined or set according to the past years competitions. However, it can occur that the improper setting of the stopping condition can affect the performance of the evolutionary algorithms. It can happen that the budget is too small, which can result in premature convergence and the solution which is not of optimal quality. Some mishaps during a new experiment can happen if the predetermined budget is different from the one used in the original paper. This can affect the comparison of the algorithms, since a bigger budget can help the algorithm reach a solution of a better quality. However, the stopping criteria does not only terminate the algorithm, but it also plays an important role in the analysis and comparison of the evolutionary algorithms. The stopping condition can produce a significant differences in the ranking of evolutionary algorithms [18]. Before an experiment, it is crucial to set the correct stopping condition due to all aforementioned points. It is unknown how many number of function evaluations an algorithm will need to reach the solution of a wanted quality. The question which we propose in this paper is: how to determine a maximum number of function evaluations for a chosen evolutionary algorithm and problem?

For this purpose, we focus on setting a stopping condition/control parameter for a given evolutionary algorithm on a chosen benchmark function. We chose a 7-nonseparable, 1-separable Shifted and Rotated Elliptic Function from CEC'2013 Large-Scale Global Optimization benchmark functions [16] and L-SHADE. L-SHADE is considered as the state-of-the-art algorithm from the previous CEC competitions with the success-history based parameter adaptation and linear population size reduction [19]. To be able to set the value of a control parameter (*maxFEs*), we will utilize a recently proposed approach AS^3D (Analysis of the Stochastic Solvers based on the Statistical Distribution) [12]. This approach is based on the statistical distribution and parameters of the observed variable. In our case, this is the number of function evaluations needed to reach a specific target. With this approach, we will not only set a stopping condition/control parameter for the higher dimensions of the given optimization problem, but also provide these values with a specific probability. Identifying the statistical distribution and its parameters enables us to establish a predictive model. The predictive model helps in estimating the stopping condition according to the characteristics of the chosen evolutionary algorithm and optimization problem.

The paper is organized as follows. In Section 2, the related work is described. In Section 3, the experiment and analysis are provided. Section 4 concludes our paper.

2 RELATED WORK

In this paper, we focus on three aspects of the evolutionary computation: on setting the stopping condition, the analysis and comparison of evolutionary algorithms, and the statistics behind it all.

Firstly, we focus on setting the control parameter of the optimization process for a specific evolutionary algorithm and optimization problem. In our case, when analyzing the L-SHADE algorithm, the control parameter serves also as a stopping condition. This is the fixed-budget approach [3]. This means that the value of it is predetermined and the algorithm stops, when the budget is spent. Contrary to the fixed-budget approach, with the fixed-target approach [10], we set a certain quality of solutions, which should be reached by the evolutionary algorithm. In this case we observe the number of function evaluations or runtime needed to reach this quality of solutions.

Setting the stopping condition represents a demanding task. In [14], authors argue that setting a higher number of function evaluations as a stopping condition may not present a higher computational cost and should be considered in benchmarking. They

DOI: https://doi.org/10.26493/scores23.11

examine the effect of a higher evaluation budget on the performance, mean, convergence of the algorithms, and population diversity. In [18], the authors show that using different stopping criteria produces different results while comparing the state-of-the-art algorithms. They argue that this fact is often overlooked by the researchers.

One of the most recent approaches, which offers several aspects of the algorithm's comparison and analysis is AS^3D approach, which will be utilized in this paper. This approach does not only rely on statistics, but it takes into consideration the characteristics of an algorithm and also of an optimization problems. It offers a deeper insight into the performance of an algorithm based on the statistical distribution of the observed variables [12]. It does not only provide a statistical distribution of the observed variable as it is described in [17], but also establishes predictive models with which we can predict stopping conditions with any wanted probability.

However, one should not neglect the basis for every fair comparison of the evolutionary algorithms: the parametric and non-parametric statistical tests [8], [9]. Several statistical approaches have been proposed as an answer to only using the statistical tests. Those are the following [7], [15], [20] and [9].

3 EXPERIMENT

In this paper, we focused on analysing the L-SHADE algorithm on a 7-nonseparable, 1-separable Shifted and Rotated Elliptic Function. The main intention is to show how to set the control parameter for L-SHADE for a larger dimension of the chosen benchmark function.

We made 100 independent runs for each chosen dimension $D = \{5, 10, ..., 40\}$ of the large-scale function. We applied the target-approach with the optimal quality of solutions. The proposed approach AS^3D requires that the given evolutionary algorithm reaches the target in all runs, so that the hit ratio is 100%. We show how to set/predict the control parameter/stopping condition *maxFEs* for the dimension $D = 50$ based on the model established from the smaller dimensions. Then we will empirically validate the results by running the L-SHADE for $D = 50$ and comparing the empirical and predicted values.

Firstly, we analysed the statistical distribution of 100 independent runs of each dimension $D = \{5, 10, ..., 40\}$. For this purpose, we used the Shapiro Wilk's statistical test, where the p-value needs to be less than 0.005 ($p < 0.005$). We observed how many number of function evaluations *NFEs* are needed that L-SHADE reaches the set optimal solution in each of the independent runs. We show that the statistical distribution is normal for each of the chosen dimensions. The Fig. 1 depicts the normal distribution for the dimension $D = 10$. Histogram can be a good visualization tool for determining the statistical distribution of the given sample [11].

Normal distribution has two parameters: mean and standard deviation, which are calculated as shown in Eq. (1) and Eq. (2).

$$\bar{x} = \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right) \tag{1}$$

The x in Eqs. (1) and (2) represents the *NFEs*. The n represents the sample size (number of independent runs), which is in this case 100.

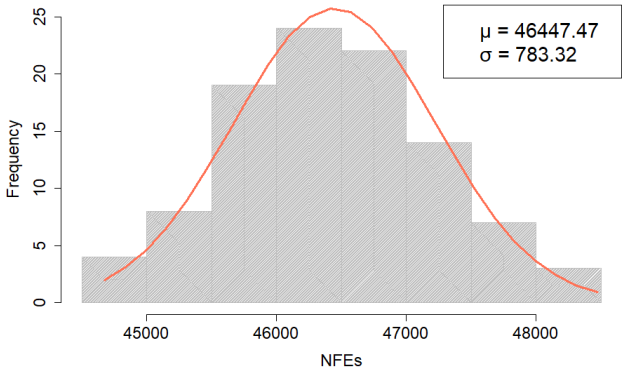


Figure 1: The normal distribution for the dimension $D = 10$.

$$\sigma = \sqrt{\frac{\Sigma(x_i - \mu)^2}{n}} \tag{2}$$

To be able to predict the stopping condition/the control parameter for $D = 50$, we need to establish the predictive model based on the parameters of the statistical distribution. The parameters follow a trend, in our case they follow a polynomial trend shown in Eq. (3). The a, b, and c are the real numbers. To be able to correctly determine, which trend line is being followed by the data we use the R^2 value. This serves as a valuable indicator to determine the optimal curve fit for the provided parameters [2]. If the R^2 is close to 1, this indicates a very good fit. In our case, R^2 was 0.9931. In Eq. (4) the predictive model for the parameter μ is established. The established predictive model is shown in Fig. 2. Eq. (5) shows the trend line fitted to the data and will be used for further calculations.

$$y_{solver} = a \cdot x^2 \quad b \cdot x \quad c \tag{3}$$

$$\mu_{L-SHADE}(D) = 426.31 \cdot D^2 - 2382 \cdot D \quad 24{,}716 \tag{4}$$

The second parameter of the statistical distribution is the standard deviation (σ). We also established a predictive model for (σ) following the same procedure for $D = \{5, 10, ..., 40\}$. The predictive model is shown in Eq. (5) and in Fig. 3.

$$\sigma_{L-SHADE}(50) = 87.774 \cdot D^2 - 2146.2 \cdot D \quad 11{,}957 \tag{5}$$

Firstly, we will predict μ and σ for the $D = 50$. The calculations are shown in Eqs. (6) and (7).

$$\mu_{L-SHADE}(50) = 426.31 \cdot 50^2 - 2382 \cdot 50 \quad 24{,}716 = 971{,}391 \tag{6}$$

$$\sigma_{L-SHADE}(50) = 87.774 \cdot 50^2 - 2146.2 \cdot 50 \quad 11{,}957 = 112{,}136 \tag{7}$$

We predicted both parameters μ and σ. However, to determine the stopping condition, we also need to know with what probability do we want the optimal solutions to be reached. Here, we will need the Z-score table [11], with which we can estimate the probability. In our case, we are calculating the control parameter/stopping

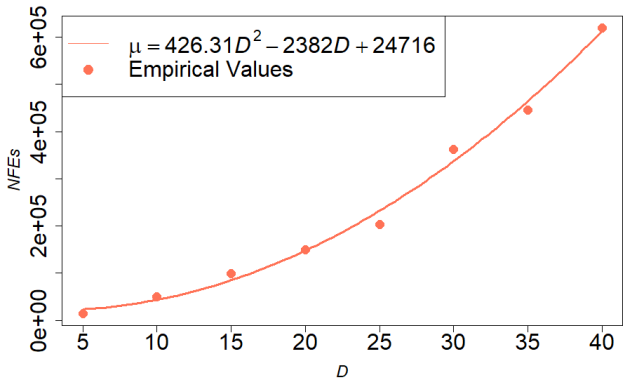


Figure 2: Prediction model for the mean (μ) of $NFEs$ for L-SHADE on observed dimensions $D = \{5, 10, ..., 40\}$. The R^2 is 0.99.

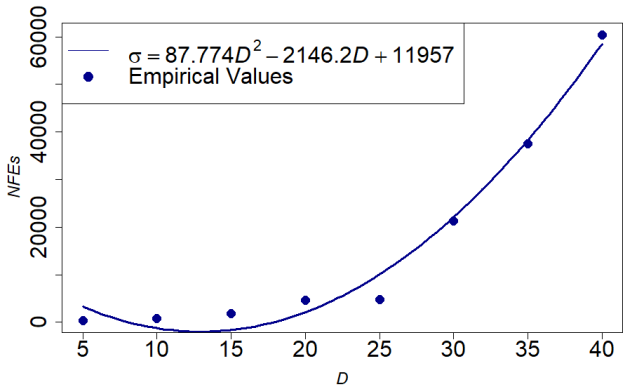


Figure 3: Prediction model for the standard deviation (σ) of $NFEs$ for L-SHADE on observed dimensions $D = \{5, 10, ..., 40\}$. The R^2 is 0.98.

condition, so we will use the Eq. (8). We want to predict the stopping condition with 99% probability. For this purpose, we will use the Z-score table and check what the value is for 99% probability. The value at 99% probability is 3.1, so we will use this value for our further calculations.

$$Z = \frac{maxFEs - \mu}{\sigma} \tag{8}$$

$$maxFEs(99\%) = \mu \quad Z \cdot \sigma = 971{,}391 \quad 3.1 \cdot 112{,}136 = 1{,}319{,}012 \tag{9}$$

In Eq. (9), we show the prediction of the stopping condition/control parameter *maxFES*. The stopping condition *maxFES* to reach the optimal solutions with L-SHADE on the given benchmark function and 99% probability is 1,356,045 *maxFES*.

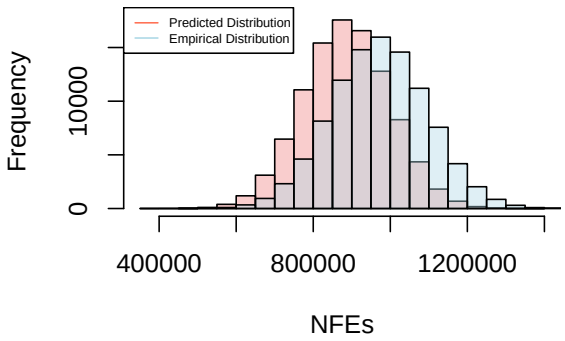


Figure 4: The comparison of the empirical and predicted statistical distribution for $D = 50$.

To empirically validate these results, we measured the hit ratio. Hit ratio represents the relationship between the number of successful runs and all runs. With a 99% hit ratio, we obtained 1,118,419 *maxFEs*. With this, we show the usefulness of the proposed approach. The gap between the empirical and predicted values is 15%. We can apply this approach for any probability. For this purpose, we also show predicted and empirical results for the probability of 50%. The predicted and empirical*maxFEs* for the probability/hit ratio 50% for $D = 50$ are 971,391 and 877,252. The gap between the empirical and predicted value is 10%. In our examples, it is evident that the predicted values are higher than the empirical ones. This shows that our predictive model is slightly pessimistic.

To show how well the predicted and empirical mean (μ) of the *NFEs* match, we will utilize another aspect of the AS^3D approach. Since we predict the parameters of the statistical distribution, we also predict the statistical distribution. In Fig. 4, we show how well the predicted and empirical statistical distributions match.

This experiment indicates that the control parameter for L-SHADE on the chosen benchmark function can be predicted by utilizing the AS^3D approach. However, this approach also enables us to:

- Estimate the probability with which a (sub)-optimal solution can be reached according to the preset stopping conditions.
- Analyze and compare the chosen evolutionary algorithms and optimization problems.

Still, we need to take into the account some limitations of the approach. Firstly, the hit ratio needs to be 100%. This means that the algorithm reaches a given quality of solutions for each independent run. Since the predictive model is established on smaller dimension of the optimization problem and the prediction is made for the larger dimensions, the chosen optimization problem needs to be multidimensional. It can also occur that the parameters of the statistical distribution do not follow any recognizable trend line. This means that the prediction cannot be made.

Overall, the control parameter/stopping condition can be set by the proposed approach.

4 CONCLUSION

In conclusion, the control parameter maximum number of function evaluations (*maxFEs*) holds a significant role in the optimization process. It impacts the parameter population size of an evolutionary algorithm and it serves as a stopping criteria. This paper focused on setting the control parameter *maxFEs* for the L-SHADE algorithm and selected Large-Scale benchmark function. Through the experiment, we show that the recently proposed approach is appropriate for predicting the control parameter of L-SHADE by establishing a predictive model based on smaller dimensions of the chosen benchmark function. The stopping condition is predicted by considering the 99% probability. The calculations were empirically validated by comparing the empirical and predicted values of the control parameter.

In conclusion, by using this approach, we not only set stopping condition, but also provide probability with which the chosen quality of solution can be reached.

ACKNOWLEDGEMENT

This work was supported by the Slovenian Research Agency (Computer Systems, Methodologies, and Intelligent Services) under Grant P2-0041.

REFERENCES

[1] A. W. Mohamed, A. A. Hadi, A. K. Mohamed, P. Agrawal, A. Kumar, P. N. Suganthan. December 2021. *Problem Definitions and Evaluation Criteria for the CEC 2022 Special Session and Competition on Single Objective Bound Constrained Numerical Optimization.* Technical Report. Nanyang Technological University, Singapore. https://github.com/P-N-Suganthan/2022-SO-BO

[2] Sandra Arlinghaus. 1994. *Practical handbook of digital mapping terms and concepts.* CRC Press.

[3] Anne Auger and Nikolaus Hansen. 2023. An Introduction to Scientific Experimentation and Benchmarking. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation.* 854–877.

[4] Subhodip Biswas, Debanjan Saha, Shuvodeep De, Adam D Cobb, Swagatam Das, and Brian A Jalaian. 2021. Improving Differential Evolution through Bayesian Hyperparameter Optimization. In *2021 IEEE Congress on Evolutionary Computation (CEC).* IEEE, 832–840.

[5] Janez Brest, Mirjam Sepesy Maučec, and Borko Bošković. 2016. iL-SHADE: Improved L-SHADE algorithm for single objective real-parameter optimization. In *2016 IEEE Congress on Evolutionary Computation (CEC).* IEEE, 1188–1195.

[6] Janez Brest, Mirjam Sepesy Maučec, and Borko Bošković. 2017. Single objective real-parameter optimization: Algorithm jSO. In *2017 IEEE Congress on Evolutionary Computation (CEC).* IEEE, 1311–1318.

[7] Borja Calvo, Ofer M Shir, Josu Ceberio, Carola Doerr, Hao Wang, Thomas Bäck, and Jose A Lozano. 2019. Bayesian performance analysis for black-box optimization benchmarking. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion.* 1789–1797.

[8] Joaquín Derrac, Salvador García, Daniel Molina, and Francisco Herrera. 2011. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation* 1, 1 (2011), 3–18.

[9] Salvador García, Alberto Fernández, Julián Luengo, and Francisco Herrera. 2010. Advanced Nonparametric Tests for Multiple Comparisons in the Design of Experiments in Computational Intelligence and Data Mining: Experimental Analysis of Power. *Information sciences* 180, 10 (2010), 2044–2064.

[10] Nikolaus Hansen, Anne Auger, Dimo Brockhoff, and Tea Tušar. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Transactions on Evolutionary Computation* 26, 6 (2022), 1293–1305.

[11] Homer Thornton Hayslett. 2014. *Statistics.* Elsevier.

[12] Jana Herzog, Janez Brest, and Borko Bošković. 2023. Analysis based on statistical distributions: A practical approach for stochastic solvers using discrete and continuous problems. *Information Sciences* 633 (2023), 469–490. https://doi.org/10.1016/j.ins.2023.03.081

[13] Thomas Jansen. 2020. Analysing stochastic search heuristics operating on a fixed budget. *Theory of evolutionary computation: recent developments in discrete optimization* (2020), 249–270.

[14] Anezka Kazikova, Michal Pluhacek, and Roman Senkerik. 2021. How Does the Number of Objective Function Evaluations Impact Our Understanding of Metaheuristics Behavior? *IEEE Access* 9 (2021), 44032–44048. https://doi.org/10.1109/ACCESS.2021.3066135

[15] Antonio LaTorre, Daniel Molina, Eneko Osaba, Javier Poyatos, Javier Del Ser, and Francisco Herrera. 2021. A Prescription of Methodological Guidelines for Comparing Bio-inspired Optimization Algorithms. *Swarm and Evolutionary Computation* (2021). https://doi.org/10.1016/j.swevo.2021.100973

[16] Xiaodong Li, Ke Tang, Mohammad N Omidvar, Zhenyu Yang, Kai Qin, and Hefei China. 2013. Benchmark Functions for the CEC 2013 Special Session and Competition on Large-scale Global Optimization. *gene* 7, 33 (2013), 8.

[17] Mattos, David Issa and Bosch, Jan and Olsson, Helena Holmström. 2021. Statistical Models for the Analysis of Optimization Algorithms With Benchmark Functions. *IEEE Transactions on Evolutionary Computation* 25, 6 (2021), 1163–1177. https://doi.org/10.1109/TEVC.2021.3081167

[18] Miha Ravber, Shih-Hsi Liu, Marjan Mernik, and Matej Črepinšek. 2022. Maximum number of generations as a stopping criterion considered harmful. *Applied Soft Computing* 128 (2022), 109478. https://doi.org/10.1016/j.asoc.2022.109478

[19] Ryoji Tanabe and Alex S. Fukunaga. 2014. Improving the search performance of SHADE using linear population size reduction. In *2014 IEEE Congress on Evolutionary Computation (CEC).* 1658–1665. https://doi.org/10.1109/CEC.2014.6900380

[20] Niki Veček, Marjan Mernik, and Matej Črepinšek. 2014. A chess rating system for evolutionary algorithms: A new method for the comparison and ranking of evolutionary algorithms. *Information Sciences* 277 (2014), 656–679.

Group Contribution Cooperative Co-Evolution Framework for CEC'2013 Large-Scale Optimization Problems

Klemen Berkovič
klemen.berkovic1@um.si
Faculty of Electrical Engineering and Computer Science, University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

Borko Bošković
borko.boskovic@um.si
Faculty of Electrical Engineering and Computer Science, University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

Janez Brest
janez.brest@um.si
Faculty of Electrical Engineering and Computer Science, University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

ABSTRACT

Solving black-box large-scale global optimization problems presents a significant challenge for conventional optimization algorithms. In this article, we introduce a new cooperative co-evolution framework and a modified component grouping algorithm. The new framework employs a divide-and-conquer strategy based on the modified component grouping algorithm. The modified component grouping algorithm breaks down a complex problem into manageable groups. These smaller groups are then optimized efficiently using an algorithm from the NiaPy framework. The proposed framework is highly versatile, it is capable of utilizing various optimization and component grouping algorithms. To assess its performance, we conducted a comparative study against multiple algorithms. We used fifteen benchmark functions from the CEC'2013 large-scale global optimization benchmark. Our findings highlight the potential of our framework in enhancing optimization algorithms' effectiveness when dealing with black-box large-scale global optimization problems.

KEYWORDS

Large-scale global optimization, black-box optimization, CEC'2013 LSGO, component grouping, cooperative co-evolution

1 INTRODUCTION

In the field of optimization, we encounter intricate challenges, demanding inventive solutions for real-world issues. Four interconnected domains stand out: 1) Black-Box (BB) optimization, 2) Large-Scale Global Optimization (LSGO), 3) Cooperative Co-evolution (CC) frameworks and 4) Component Grouping (CG) methods.

BB optimization deals with a hidden formulation of a fitness function, so it relies exclusively on evaluations of the fitness function, for finding solutions. This mirrors complex real-world scenarios where traditional methods have difficulties. LSGO tackles the complexities of a problem that has a high number of components and is very time-consuming to optimize. The high number of components need to be optimized in search of a good solution to the problem. LSGO problems can be found in fields like urban planning [2] and healthcare [10]. CG methods provide a crucial thread, disentangling a complex problem with overlapping components. Our goal is to combine the CG method within the CC framework to effectively optimize the BB LSGO problems. We will use the idea of recursive differential grouping as a CG method to generate groups that represent sub-problems for easier problem-solving. The proposed framework is implemented for the NiaPy optimization framework [11], which includes many nature-inspired algorithms. Our goal is to be able to use any optimization algorithm from that framework within the proposed CC framework.

The rest of the paper is organized as follows: The related work on used algorithms, CC and CG methods are introduced in Section 2. Then, the proposed CC framework and modified CG algorithm are described in Section 3. Experimental studies are presented in Section 4. Finally, conclusions and future work are described in Section 5.

2 RELATED WORK

In the field of tackling BB optimization challenges, CC frameworks have emerged as a powerful tool. In [7] authors used the combined strengths of CC framework with genetic algorithm (GA) to navigate the complexities inherent in such problems. The CCGA-1 algorithm was developed and tested on known optimization functions, where the number of components was set to 0 and 0. They showed that the CCGA-1 algorithm was better than the original GA algorithm.

The CG method focuses on efficient problem decomposition through problems' components interaction understanding. An effective CG method minimizes inter-group and maximizes intra-group interactions [12]. Two primary methods for detecting these interactions are: 1) non-monotonicity [5] and 2) non-linearity [9] detection.

Authors in [6] introduced a CC algorithm with a differential CG method for LSGO, laying the foundation for integrating a differential CG method as a central mechanism in the CC framework. They introduced the Differential Grouping algorithm, which uses non-linearity components interaction detection. With automated decomposition of the problem through CG, this pioneering approach set a precedent for subsequent research. They showed that a near-optimal CG can significantly improve the solution quality for the BB LSGO problems.

In [8] a recursive CG method for large-scale continuous optimization was introduced, called the Recursive Differential Grouping 3 (RDG3) algorithm. The RDG3 algorithm uses non-linearity detection to identify components interactions by detecting changes in a fitness function when perturbing components. The RDG3 algorithm showed that a recursive method can break down a complex optimization problem into more manageable sub-problems, without explicitly examining all pairwise components interactions. Notably, the RDG3 algorithm reduces the CG time complexity of n-dimensional problem to (n n). To validate the effectiveness

DOI: https://doi.org/10.26493/scores23.15

of the RDG3 algorithm, a CC algorithm with the RDG3 algorithm was used on CEC'2013 LSGO benchmark.

In summary, these works collectively illustrate the evolution of CC algorithms. They showcase the role of the CG in addressing the challenges of BB LSGO problems. Approaches like recursive CG, adaptive parameter estimation and handling overlapping components underline the versatility and effectiveness of the CC framework. Therefore CC frameworks are highly desirable for optimizing complex systems across various real-world applications.

3 METHODOLOGY

In this section, we describe our Group Contribution CC (GCCC) framework and the Modified Recursive Differential Grouping 3 (MRDG3) algorithm. The advantage of the MRDG3 algorithm is that it has fewer parameters than the RDG3 algorithm. The advantages of the GCCC framework are as follows: 1) it can use any optimization algorithm implemented in the Python programming language for the NiaPy optimization framework, 2) it is capable of using any CG method implemented in the Python programming language for the NiaPy optimization framework and 3) based on previous advantages it enables the rapid testing of new optimization and CG algorithms within the GCCC framework.

3.1 Modified Recursive Differential Grouping 3 algorithm

The MRDG3 algorithm was implemented in the Python programming language for the NiaPy optimization framework and is shown in Algorithm 1. The MRDG3 algorithm has four input parameters: 1) BB fitness function f, 2) lower bound of the search space , 3) upper bound of the search space and 4) threshold value ϵ_n. The function ℓ returns the number of components in a vector or a set. MRDG3 starts by checking interactions with component x , recursively identifying interacting components with algorithm `Interact` [8]. A threshold ϵ_n controls the group size for optimization, which can be seen in line 6. When all interactions between component x and the remaining components in set are identified, MRDG3 continues with the first remaining component in set , what is seen in line 12. Finally, the MRDG3 outputs set with separable components and set with non-separable component groups, what is seen in line 16.

Input: f, , , ϵ_n

1 , , $\mathbf{x}$, , $\mathbf{x}_{,}$, $y_{l,l} \leftarrow$ [], [], [1], [1], , f( );
2 $\mathbf{x}$ $\leftarrow$ [**for** i = **to** ℓ( ) **do** i];
3 **while** ***not*** *empty* **do**
4 $\leftarrow$ [];
5 $\mathbf{x}^*$ $\leftarrow$ `Interact`(f, , , $\mathbf{x}$, $\mathbf{x}$, $\mathbf{x}_{,}$, $y_{l,l}$,);
6 **if** $\ell(\mathbf{x}^*) < \epsilon_n$ ***and*** $\ell(\mathbf{x}^*)$ $\ell(\mathbf{x}$) **then**
7 $\mathbf{x}$, $\mathbf{x}$ = $\mathbf{x}^*$, ;
8 **if** ℓ( ) = 0 **then** `Append`(, $\mathbf{x}$) and **break**;
9 **else**
10 **if** $\ell(\mathbf{x}^*)$ = **then** `Append`(, $\mathbf{x}^*$) **else** `Append`(, $\mathbf{x}^*$);
11 **if** ℓ( ) > **then**
12 $\mathbf{x}$ $\leftarrow$ [[0]];
13 `Delete`(, *[0]*);
14 $\mathbf{x}$ $\leftarrow$;
15 **else if** ℓ( ) = **then** `Append`(, $\mathbf{x}$) and **break**;
16 **return** , ;

Algorithm 1: Modified Recursive Differential Grouping 3 (MRDG3) algorithm.

Input: f, , , ϵ_n

1 $\leftarrow$ `MRDG3`(f, , , ϵ_n); `// Algorithm 1`
2 , $\leftarrow$ [**foreach** g ***in*** **do** Initialize algorithm a with starting population based on group g];
3 * $\leftarrow$ Get best individuals from initialized populations for each algorithm based on fitness values;
4 $\mathbf{x}^*$ $\leftarrow$ Get best individual from * based on fitness values;
5 **while** *stopping condition meet* **do**
6 **for** i = **to** ℓ( ) **do**
7 [i], $\mathbf{x}^*$ $\leftarrow$ `RunGeneration`([i], [i], f, ,);
8 **if** $f(\mathbf{x}^*) < f($ $^*[i]$) **then**
9 $^*[i] \leftarrow \mathbf{x}^*$;
10 **if** $f(\mathbf{x}^*) < f(\mathbf{x}^*)$ **then** $\mathbf{x}^* \leftarrow \mathbf{x}^*$;
11 **if** *Any group found new local best individual* **then**
12 **foreach** g ***in*** **do** $\mathbf{x}^*[g] \leftarrow$ $^*[g]$;
13 **if** $f(\mathbf{x}^*) < f(\mathbf{x}^*)$ **then** $\mathbf{x}^* = \mathbf{x}^*$;
14 **return** $\mathbf{x}^*$, $f(\mathbf{x}^*)$;

Algorithm 2: Group Contribution Cooperative Co-evolution (GCCC) framework.

3.2 Group Contribution Cooperative Co-evolution framework

As the CG method aims to decompose and divide the BB LSGO problem into smaller subgroups for smarter optimization, this part describes the GCCC framework. GCCC uses decomposition information for optimization algorithms initialization, algorithms population initialization, and for running a generation of an algorithm. Our proposed GCCC framework's pseudocode is shown in Algorithm 2. GCCC was implemented in the Python programming language for the NiaPy optimization framework.

The GCCC framework works as follows. First, groups of components for independent optimizations are retrieved with the help of the CG method, seen in line 1. In line 2, algorithms and starting populations are initialized based on groups . All algorithms initialize their initial population, which is used and updated by the algorithms during the execution of one generation of the algorithm. In line 3 for each initialized population local best individual is found so that every algorithm has it's own local best individual. Then global best individual is found from all the local best individuals. Search for global best is faster because we are using only a small part of all individuals, which is depicted in line 4.

Our main novelty of the GCCC framework is seen when the optimization stage begins in line 5. In line 7, we perform only one generation of the algorithm a. After the algorithm a ends a generation, a new population for the algorithm a is returned with the new local best individual $\mathbf{x}^*$. Line 10 checks if the new local best individual is the new global best individual and updates the global best individual if that is true. After all algorithms in finish their own generation, line 11 checks if the new individual has to be constructed. If so, the new individual is composed of all local

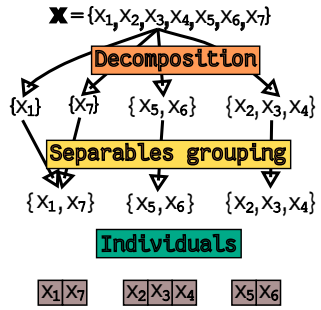


Figure 1: Components grouping and individual construction for Equation 1.

best individuals * components values, which is seen in line 12. Line 13 updates the global best individual if the composed individual is better than the current global best individual. After the GCCC returns to line 5 and starts the new iteration of the algorithm and conducts previous steps until the stopping condition is not achieved.

To demonstrate how the GCCC constructs the individuals of optimization algorithms and how it evaluates an individual, let's take the optimization problem formulated as:

$$f(\mathbf{x}) = x \quad (x \ -x\ ) \quad (x \ -x\ ) \quad (x \quad x\ ) \quad x\ . \qquad (1)$$

We will optimize Equation 1 as a BB problem with a lower bound equal to − and an upper bound equal to for all seven components. First, the GCCC decomposes the problem into four groups and then joins separable components into one group. So after the decomposition stage, we have three groups. This is depicted in Figure 1. In the initialization stage, three algorithms are initialized with initial populations. Each algorithm in initialized individuals with the length of one individual equal to the number of components in a group that the algorithm is going to optimize. When the algorithm a in runs one generation of the optimization algorithm, fitness function f, fills the missing components with the lower bound of the search space. In this example, for the first algorithm, which is optimizing group containing components x and x , components x , x , x , x and x are filled with value − . In line 12 the GCCC constructs an individual $\mathbf{x}^*$ that has all seven components filled. So when evaluating an individual $\mathbf{x}^*$ at line 13, there is no need to fill any components.

4 EXPERIMENT

In this section, the results of the GCCC framework with two different optimization algorithms from the NiaPy framework are presented. Test functions were taken from the CEC'2013 LSGO benchmark competition [3] and are implemented in the C++ programming language. Python code of the performed experiments in this work is available via a connection[1]. For comparison, we used algorithms PSO [1] and SCA [4], which are implemented in the NiaPy framework. Algorithms' parameters are presented in Table 1 and were set based on experimental results on test functions that are part of the NiaPy framework.

In Table 2, the obtained results of CG for each test function in the CEC'2013 LSGO benchmark are shown. The results show that

[1]https://github.com/kb2623/scores23

Table 1: Algorithms' parameters for conducted experiment.

Algorithm	Parameter values
MRDG3	$\epsilon_n =$ 0
SCA	= , $a =$, r = 0, $r_x =$
PSO	= , c = , c = , $w = 0.$, v = − . , $v_x =$.

Table 2: Results of the CG for the MRDG3 algorithm on the CEC'2013 LSGO benchmark functions.

	Number of groups	Number of separable components	Number of evaluations
f	1	0	5992
f	0	1000	2998
f	2	1	5987
f	1	0	5992
f	6	800	8287
f	5	750	8569
f	1	0	5992
f	1	0	5992
f	1	0	5992
f_0	21	151	19357
f	1	0	5992
f	1	0	5992
f	1	0	5992
f	1	0	5992
f	1	0	5992

the number of evaluations is fairly small. MRDG3 does not need n number of evaluations to decompose a BB problem, where n represents the number of components and for the CEC'2013 LSGO benchmark $n =$ 000. For fully separable functions, only f is 00 correct. The result of decomposition on function f is missing seven groups. MRDG3 has incorrect results on function f_0 because this function has no separable sub-components. CG results for functions f , f and f are 00 correct, because functions have overlapping sub-components. The result of decomposition for the function f is 00 correct because that function is fully non-separable.

For each function from the CEC'2013 LSGO benchmark, we performed 50 runs for each algorithm. Results are shown in Table 3 where we reported three values for each benchmark function. The top value in each cell represents the minimum, the middle value is the median and the bottom value is the standard deviation. Underlined values represent better results when comparing the basic algorithm to its CC-based algorithm. For each of the algorithms used with the GCCC framework, we added a sign that indicates if the GCCC framework is better ( ), similar ( ), or worse (−). This assumption was made based on the Wilcoxon signed-rank test from SciPy[2]. We used $\alpha = 0.0$ for detecting statistical differences. The only similarity we detected was between algorithm SCA and SCA-GCCC on function f . From the last row in Table 3, we can see that SCA-GCCC compared to SCA is better on eleven functions, worse on three functions, and one function had a similar result. When comparing PSO-GCCC to PSO, we detected that PSO-GCCC was better on twelve functions, and on three functions was worse than PSO.

[2]https://scipy.org/

Table 3: Algorithms results for the CEC'2013 LSGO benchmark.

	SCA	SCA-GCCC		PSO	PSO-GCCC	
	3.2116e+11	2.0983e+11		2.2449e+11	1.8236e+11	
f	3.5997e+11	2.0983e+11		2.5767e+11	2.0881e+11	
	1.7041e+10	6.1654e-05		1.4521e+10	4.4700e+09	
	1.0988e+05	4.7598e+04		1.8498e+04	2.0991e+04	
f	1.1765e+05	4.7598e+04		2.0399e+04	2.4994e+04	–
	4.0256e+03	7.3498e-12		9.9743e+02	1.7148e+03	
	2.1625e+01	2.1625e+01		2.1540e+01	2.1514e+01	
f	2.1660e+01	2.1659e+01		2.1565e+01	2.1559e+01	
	1.0164e-02	1.0796e-02		1.4204e-02	1.8726e-02	
	6.7784e+12	1.4182e+13		4.5338e+12	4.1230e+12	
f	3.0918e+13	3.4167e+13	–	7.5541e+12	6.9831e+12	
	9.6588e+12	1.1140e+13		1.2660e+12	1.6166e+12	
	5.4753e+07	4.8419e+07		5.5953e+06	5.1143e+06	
f	7.0849e+07	4.8419e+07		8.9464e+06	8.2784e+06	
	5.2834e+06	0.0000e+00		1.9627e+06	1.7852e+06	
	1.0688e+06	1.0688e+06		1.0530e+06	1.0518e+06	
f	1.0717e+06	1.0704e+06		1.0587e+06	1.0594e+06	–
	1.8314e+03	6.2956e+02		1.8514e+03	2.4651e+03	
	5.3201e+14	9.9382e+14		1.5502e+14	1.0261e+14	
f	8.3849e+15	9.9382e+14		4.3684e+14	3.4803e+14	
	6.1855e+15	0.0000e+00		2.3930e+14	1.8312e+14	
	9.6025e+17	9.6025e+17		1.5852e+17	1.0169e+17	
f	1.8327e+18	1.9011e+18	–	3.6707e+17	3.0508e+17	
	5.9387e+17	5.6790e+17		1.0251e+17	8.8785e+16	
	4.0665e+09	4.0665e+09		4.6629e+08	5.0571e+08	
f	5.7516e+09	5.6652e+09		6.8898e+08	7.7170e+08	–
	7.2037e+08	5.1872e+08		1.3255e+08	1.4362e+08	
	9.4583e+07	9.4583e+07		9.2361e+07	9.2574e+07	
f_0	9.6401e+07	9.5732e+07		9.3658e+07	9.4114e+07	–
	4.8204e+05	1.8366e+05		4.2994e+05	4.4469e+05	
	6.3357e+16	6.3357e+16		1.0835e+16	8.1720e+15	
f	6.4911e+17	1.0331e+17		3.7431e+16	3.5241e+16	
	4.4896e+17	6.2676e+15		1.5817e+16	2.1938e+16	
	7.8795e+12	1.7039e+12		5.4019e+12	1.7039e+12	
f	8.3439e+12	1.7039e+12		6.1512e+12	1.7039e+12	
	2.1357e+11	0.0000e+00		2.8899e+11	0.0000e+00	
	5.0621e+16	6.3339e+16		1.1161e+16	1.0884e+16	
f	7.5321e+17	9.4042e+16		3.8680e+16	3.1150e+16	
	4.7974e+17	5.2137e+15		1.7411e+16	1.3835e+16	
	9.3147e+16	9.3147e+16		1.2127e+16	2.1257e+16	
f	1.0716e+18	1.1121e+18	–	6.5235e+16	5.3469e+16	
	7.2936e+17	7.6183e+17		3.2750e+16	2.3430e+16	
	1.4057e+17	5.6572e+11		3.7352e+16	5.6572e+11	
f	7.7989e+17	5.6572e+11		6.9690e+16	5.6572e+11	
	3.1321e+17	0.0000e+00		2.1668e+16	0.0000e+00	
Overall / –:		11/1/3			12/0/3	

5 CONCLUSIONS

We tackled the BB LSGO problems with overlapping components from the CEC'2013 LSGO benchmark suit, using a divide-and-conquer method. We used the GCCC framework, that we implemented in the Python programming language for the NiaPy optimization framework. We demonstrate that the GCCC can be used with many existing algorithms from the same optimization framework. For the decomposition of overlapping problems, we used the MRDG3 algorithm, that we implemented in the Python programming language for the NiaPy optimization framework. To systemically evaluate the efficacy of our CC algorithm, we used multiple algorithms and tested them on the CEC'2013 LSGO benchmark which consists of fifteen test functions. Experimental results showed that the GCCC framework facilitated problem-solving, and outperformed its base versions of used algorithms on some of the test functions.

For the feature work, we suggest designing more benchmark problems with a more flexible variable interaction structure and richer sources of overlap. Developing a CC framework that has an option of sharing the same population between all algorithms that are working cooperatively in solving BB LSGO problems.

ACKNOWLEDGMENTS

This work was supported by the Slovenian Research Agency (Computer Systems, Methodologies, and Intelligent Services) under Grant P2-0041.

REFERENCES

[1] J. Kennedy and R. Eberhart. 1995. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, Vol. 4. IEEE, 1942–1948 vol.4. https://doi.org/10.1109/ICNN.1995.488968

[2] Douglass B. Lee. 1994. Retrospective on Large-Scale Urban Models. *Journal of the American Planning Association* 60, 1 (1994), 35–40. https://doi.org/10.1080/01944369408975549

[3] Xiaodong Li, Ke Tang, Mohammad N Omidvar, Zhenyu Yang, Kai Qin, and Hefei China. 2013. Benchmark functions for the CEC 2013 special session and competition on large-scale global optimization. *gene* 7, 33 (2013), 8.

[4] Seyedali Mirjalili. 2016. SCA: A Sine Cosine Algorithm for solving optimization problems. *Knowledge-Based Systems* 96 (2016), 120–133. https://doi.org/10.1016/j.knosys.2015.12.022

[5] Masaharu Munetomo and David E. Goldberg. 1999. Linkage Identification by Non-monotonicity Detection for Overlapping Functions. *Evolutionary Computation* 7, 4 (1999), 377–398. https://doi.org/10.1162/evco.1999.7.4.377

[6] Mohammad Nabi Omidvar, Xiaodong Li, Yi Mei, and Xin Yao. 2014. Cooperative co-evolution with differential grouping for large scale optimization. *IEEE Transactions on evolutionary computation* 18, 3 (2014), 378–393. https://doi.org/10.1109/TEVC.2013.2281543

[7] Mitchell A Potter and Kenneth A De Jong. 1994. A cooperative coevolutionary approach to function optimization. In *International conference on parallel problem solving from nature*. Springer, 249–257. https://doi.org/10.1007/3-540-58484-6_269

[8] Yuan Sun, Xiaodong Li, Andreas Ernst, and Mohammad Nabi Omidvar. 2019. Decomposition for large-scale optimization problems with overlapping components. In *2019 IEEE congress on evolutionary computation (CEC)*. IEEE, 326–333. https://doi.org/10.1109/CEC.2019.8790204

[9] Masaru Tezuka, Masaharu Munetomo, and Kiyoshi Akama. 2004. Linkage Identification by Nonlinearity Check for Real-Coded Genetic Algorithms. In *Genetic and Evolutionary Computation – GECCO 2004*, Kalyanmoy Deb (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 222–233. https://doi.org/10.1007/978-3-540-24855-2_20

[10] Jerry D VanVactor. 2011. Cognizant healthcare logistics management: ensuring resilience during crisis. *International Journal of Disaster Resilience in the Built Environment* 2, 3 (2011), 245–255. https://doi.org/10.1108/17595901111167114

[11] Grega Vrbančič, Lucija Brezočnik, Uroš Mlakar, Dušan Fister, and Iztok Fister Jr. 2018. NiaPy: Python microframework for building nature-inspired algorithms. *Journal of Open Source Software* 3 (2018). Issue 23. https://doi.org/10.21105/joss.00613

[12] Tian-Li Yu, David E. Goldberg, Kumara Sastry, Claudio F. Lima, and Martin Pelikan. 2009. Dependency Structure Matrix, Genetic Algorithms, and Effective Recombination. *Evol. Comput.* 17, 4 (dec 2009), 595–626. https://doi.org/10.1162/evco.2009.17.4.17409

Retrieving deleted records from Telegram

Žan Počkar
zp68409@student.uni-lj.si
Faculty of Computer and Information Science,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

Tom Sojer
ts22842@student.uni-lj.si
Faculty of Computer and Information Science,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

ABSTRACT

Mobile applications utilize their own mechanism of storing data on a local mobile device. One interesting instant messaging platform that provides a mobile client is Telegram. In this paper, we focus on some of the research [5] done on the matter of how Telegram encodes and stores data. We present results of a forensic analysis made on two devices that used the application.

KEYWORDS

Mobile forensics, Telegram, SQLite, Write-Ahead log

1 INTRODUCTION

Telegram is a cloud based messaging service. It was first released for Android in 2010 and by now works on every major operating system. By far the most usage represents Android, as 85% of all the users use this OS. It supports different communication options, such as so-called normal chats, secret chats, video calling, chat rooms, etc. Normal chats do not support end-to-end encryption, but instead use an internal protocol MTProto that encrypts data on Telegram's servers. For end-to-end encryption, the secret chat method is mostly used.

As claimed by Telegram in June of 2022, they were one of the top 5 downloaded apps of that year with more than 700 million monthly users. This is why research is crucially needed to catch any potential flaw that could be exploited in a malicious manner.

In our article, the following will be presented: the background section (2), where we will go into details of Telegram Messenger and how the app stores data, then we will present some related work (section 3). The follows a review of how the authors of the discussed article prepared for the investigation (subsection 4.1) and analysis (subsection 4.2). We will show forensic tools useful for research (section 5) and discuss methods utilized in the reference article (section 6).

2 BACKGROUND

2.1 Database

To understand the behavior of the app's storage, we need to go into details about how it handles the database. The examined application version was 7.9.3, released in August, 2021. Telegram Messenger client stores data with SQLite. This database's method of storing is using a Write-Ahead log file (WAL) with a checkpoint of 1000 pages (a database in SQLite is divided into pages, each of size at least 512 bytes). Using the WAL file is a method of ensuring that data doesn't get lost. Before anything is written to a database, it is first logged. This ensures that any transaction can be repeated if some failure appears. The checkpoint of 1000 pages notes that all data before it reaches that point has been written to disk. In case of a crash, the recovery procedure would include finding the checkpoint and recovering anything until reaching the checkpoint. Meanwhile the main database is left intact and changed after that point has been reached. This means that recently deleted data can be found in the WAL file, but not in the main database file.

2.2 Telegram Data Structure

While Telegram stores some local data in clear text, such as the name of the sender/recipient, more volatile data is stored into complex data structures that appear in a binary serialized form. Fields in the structure are stored as a sequence of bytes, where they appear in specific positions. Retrieving information from such a structure demands first to deserialize it and then decode each useful field. The main problem is that only for a limited set of such structures, the structure and serialization scheme is known. The exact decoding scheme is not made public by the brand, which is why the process of figuring out the correct structure is left to the community. The structures are constantly redefined or removed after new features are released, which makes it harder to maintain tools that decode all the different variations.
Since a part of the application is open source, Anglano et al. [2] performed a source code analysis to identify all the different data structures used by the app as well as to reconstruct the structure and serialization schemes. The cited article named these structures *Telegram Data Structures* or *TDSs*. We will take a look at TDSs later in the analysis.

3 RELATED WORK

Although many different tools and solutions were used in this paper, individual use cases, justifications for the choices of tools and procedural workflows were not documented in detail. The related work section helps to explain some of those uncertainties as it lists works that depict guidelines, techniques, protocols, and standardized procedures in the field of mobile forensics, records retrieval, and data carving. They also credit works explaining (then relevant) Telegram's folder and database structure and the possible forensic analysis approaches, as well as works on similar messaging applications.

Some articles have already dealt with TDSs on other platforms. One instance is a paper, written by Gregoio et al. [3], which deals with and investigates the app on a Windows phone. The importance of forensic analysis has also been introduced on other messaging platforms such as WhatsApp [1] and WeChat [6].

DOI: https://doi.org/10.26493/scores23.03

As it was mentioned in the paper, with each version of Telegram released, the structure of its database, data structures, objects and, in general, the way data is handled and processed by the app, change as a result of added or removed features. This means that most, if not all, of the previously designed protocols and tools will have reduced success rates of analyzing the data. There has however been some previous research on the SQLite structures for data carving [4].

Despite the app's rapidly changing internal structure, older tools, documentation and works remain relevant due to the cross validation of the results which has proven itself to be essential. Using a plethora of tools and various versions of the same tools could yield significantly better results.

4 METHODOLOGY

4.1 Preparation and collection

To prepare for an examination and analysis and get a sense of how deleted records are handled by the application using different forensic tools, an investigative environment was first established. Two Android phones were examined: an LG G6 with Android 9 and Samsung A50 with Android 11. Both phones had a SIM card activated and Telegram Messenger client installed. Root access was gained on both devices in order to collect data used by the app. All data on the two phones that was unrelated to the observed app was deleted. This allowed the researchers to focus on relevant data only.

After finishing with the experimental setup, messages were exchanged between the clients. The preparation for the analysis included the exchange of media files, such as images and videos. All of the media were later deleted both from the app and the trash bin folder. In total, around 2200 messages were exchanged during this process [5].

The research focused on retrieving data in different scenarios. Criteria included (1) elapsed time since the device had been acquired, (2) whether the device is powered on, off or in airplane mode and the state of the application (3), i.e. if messages are created or deleted. As many different conditions were tested, several images of the devices were produced. For instance, in one of the images acquired, the phone was powered off and the researchers gained data from that state. By comparing different criteria, some potential real world scenarios were simulated.

4.2 Examination and analysis

As part of the forensic analysis, some open source and commercial forensic tools were used. This includes software for SQLite analysis and image acquisition software. We will discuss other tools later in the article.

There are a few forensically relevant data sources on the device. Firstly, the main local database, named cache4.db. In one of the scenarios, the size of cache4.db was 1336 kB, while the related WAL file had the size of 4475 kB. This is a consequence of using the Write-Ahead log. The local database included 52 tables, containing user data, messages, contacts and phone numbers. There is also a user configuration file that stores details of the account on the device, profile photos of user's contacts and also copies of files the user interacted with. The latter is stored in the media folder. From the normal chat messages table, several table entries could be read in clear text, such as the name of sender/receiver, date sent, the message status and direction. Info and body entries of this table were in the form of the Telegram Data Structure, which as previously defined, is in a binary form. Using forensic tools, some data was queried and the initial conclusions were:

- The same message may appear in the WAL file several times.
- The Auto-delete feature of Telegram, that removes messages after a certain period, did not affect the ability to recover messages.
- The body of messages is in a TDS form and not easily interpreted.

Experiments concluded that while different states of the device result in the cache4.db and WAL file changes, they do not alter the data inside of the messages table. For instance, when a phone has a network connection, both files might be altered after a short amount of time. They will remain in the same state if a phone is in airplane mode or the SIM card is removed. While cache4.db and the WAL file change very frequently, the messages tables inside doesn't. The researchers then observed events when the messages table was modified, and this happened when:

- a message was created, even if not successfully sent
- a message was received
- a message was opened
- a message was deleted

Hex editor was used to retrieve information. Now let's describe some other observations. Firstly, there was more storage needed to record an event when an outgoing message appeared compared to an incoming message.
Secondly, the main database file does not have the auto vacuum feature enabled which keeps the file size at the minimum. This potentially means that some of the deleted records might be retrieved. This was tested by the examiners on both devices. They deleted hundreds of records and attempted to recover them. Immediately after deletion, most of the messages were recoverable, but after new messages were created, all the deleted messages were not recoverable anymore. This was verified in the database file and the WAL file using a hex editor. Records are also not recoverable after the user has logged out.
Here, we can also make an observation with the WAL file. Since the file makes a commit to the database after a 1000 pages have been reached, a lot of data can be restored for a while if the commit doesn't occur for a longer period of time. In this sense, deleted records may be retrievable for a while, but here an element of luck is involved.
After deletion of media files that were used in any message exchange, some artifacts were still available on both the devices. This was observed by first deleting pictures and by comparing their artifacts with the records in cache4.db.

Lastly, we can ask ourselves and observe when records are not available anymore. This is the case on two occasions:

- when a user logs out and the database is deleted
- when the local database is deleted in the settings menu

5 TOOLS

Tools used for this research can be split into forensic and non-forensic tools. Non-forensic tools were used mainly for preparing the white box environment for the experiment, screen capturing and monitoring purposes. On the other hand, forensic tools were used mainly for extraction and analysis of both devices and Telegram app data.

5.1 Non-forensic tools

Among the non-forensic tools used for setting up the devices Magisk Manager was used for rooting and Team Win Recovery Project - TWRP for creating a custom recovery menu. In addition to the third-party solutions, an inbuilt android feature was used for creating additional user on each of the two devices. Dual apps or Dual messenger are features of contemporary Android devices which enable creating a virtual copy of an app that can then be used with a separate account as an independent user with their own file system, databases and permissions.

5.2 Forensic tools

A total of 26 tools were used, of which 13 were commercial use tools, 9 either open source or free and 4 were commercial tools with some form of a free version. As it was previously mentioned, the version of the tool used often significantly impacted the quality of the obtained results. Which is why in addition to exploring different software solutions, a combined use of multiple versions of the same tools was utilized.

All of the tools used were categorized according to their use case into forensic analysis, SQLite analysis and image acquisition tools. But not all tools were equally successful. Due to the rapid changes to the Telegram Data Structure, most tools were not up to date with the latest changes and could not decode any data. Some tools were successful, but not in all cases, some could successfully decode only the normal chat messages while some worked only with the secret messages. As the changes in the TDS are unlikely to backtrack, newer versions of forensic tools had higher success rates as they were more compatible with the latest TDS encoding. Examples of the above mentioned cases are Cellebrite UFED Physical Analyzer 7.50, Oxygen Forensics 14.1, and AXIOM v5.7 and v6.0.

Cellebrite UFED Physical Analyzer is a commercial tool used for uncovering digital evidence, trace events, and examine digital data. It allows for import and decoding of a specific application through its selective decoding, and it speeds up many aspects of the investigation including automatized report generation. Despite all the quality of life features, in this case, it was able to decode only normal messages.

Oxygen Forensics tools used in this research are marketed as all-in-one solutions for extraction, decoding, and analysis of both data and artifacts for both computer and mobile forensic investigations. Although convenient, these tools were able to decode only the secret messages.

When it came to AXIOM, built for remote acquisitions, collection and evidence analysis from different sources including mobile devices, the older version could not decode any data while the newer could decode only normal messages.

On the other hand, tools on which the researchers had relied the most were SQLite Analysis tools. Especially tools for inspecting, querying, carving and parsing the database. Although these tools also followed the trend of newer versions yielding better results.

6 DISCUSSION

This attempt to retrieve the telegram data can't be considered as nothing but a proof of concept and even that done in extreme laboratory conditions. While the article uses some novel approaches to retrieve stored data and does achieve some success. We must raise some questions about the methodology and the fine result of retrieving the data. We can generally split our questions into two categories. The first categories deal with the methodology of the used device and acquired data. The second category deals with the final results and readability of said data.

We first must raise questions about the devices used. The team decided to use two completely wiped devices on which the first installed a custom recovery menu and gained root access. Secondly, they installed the same version of the telegram app, a version which is now discontinued. Lastly, while the team did simulate conversation with the telegram app no other application was running on the devices prior, during or after communications.

It is true that the article itself freely admits that the device setup wasn't forensically sound but more of a whitebox proof of concept. And yet even after this question arise if the test wasn't done in too many perfect conditions and if such results even carry any weight in real world practice.

Let's begin with the devices themselves. The rationale for not using other applications to reduce during the test is sound but in our opinion flawed. Multiple times the data was retrieved only because the team was able to identify changes after the messages were exchanged, in addition the article states that the data is in certain instances volatile and can be lost if the changes occur on the operating device. It is true the team did test three different changes such as turning the device on and off that did not change the data. But the same test did show that actions inside the applications lead to such change. The question arises if the running of similar applications at the same time could lead to trigger such changes. In this vein of questioning is the decision to use just a limited amount of accounts while we understand that because of limited time and money for any such research smaller sample sizes must be used. During the research, the team has shown that a message from any source using the telegram application will modify the already stored data. That means that an average user's message would be much harder to retrieve especially if the user is an active user of the application.

While the usage of the same version of the application is ideal, this is a likely event for users which use the application regularly so we don't see much problem with this approach even if it is a little idealistic. Another problem with the usage of this version of application is the fast change rate for the application. As many application telegram changes quickly and frequently this means that approaches that work on one version may not work on newer versions.

And lastly gaining root access in advance and installing custom recovery software before installing the telegram application is a

problem in the methodology. As we can't be sure how gaining root access or installing the recovery menu, could change the data we are trying to access but this could be mitigated through a usage of multiple copies used in any forensic investigation even if it is extremely time consuming.

Gaining root access may also raise extra problems regarding who and why it is being done. Gaining access during a lawful police investigation with proper warrants is no concern, but gaining root access by a private individual for other purposes may be illegal and subject to punitive actions from local authorities.Meaning that any method that requires such access can't always be used reliably in all circumstances.

The other major category of the question raised is pertaining to the results themselves. They have had moderate success in retrieving different data from the telegram application, they freely admit that they have much greater success with text messages in comparison to the pictures. But the thing one must question is if the success from retrieving text data shouldn't be classified more as a partial or limited success. The problem which is already mentioned in the articles is the format of encoding used in telegram text data. While the team was on multiple occasions able to retrieve the raw data they couldn't read large or even in some cases whole portions of retrieved data as data was parsed inside the TDSs structure. One must then ask themselves if retrieved data that can't be decoded in reasonable timeframes can't even be considered retrieved or must it at most be just used as proof of communication between parties. The whole question of usability of the retrieved data does not only rests with the team conducting the research as the research has shown that most modern and available current forensic tools aren't capable of decrypting the data. In the future, if better tools for decrypting the retrieved data are developed, such a method for retrieval may be used for more than just proof of communication.

7 CONCLUSION

We find that the article has a straight forward goal, clear methodology and a concise plan for proving its findings. In the article, the team thoroughly presents the Telegram applications, the methodology they used, their process and their results.

Conclusions regarding volatility of cache4.db database and the Write-Ahead log file along with the data collected from other articles on Telegram could be used in further investigations. In conclusion, we believe this article is an important first step at looking into retrieving the deleted records from the Telegram Messenger client. Further analysis would be needed in regards to discovering possible vulnerabilities of the application as too many tradeoffs and ideal conditions were used in the primary research for this article.

REFERENCES

[1] Cosimo Anglano. 2014. Forensic analysis of WhatsApp Messenger on Android smartphones. https://www.sciencedirect.com/science/article/pii/S1742287614000437. *Digital Investigation* 11, 3 (2014), 201–213. Special Issue: Embedded Forensics.

[2] Cosimo Anglano, Massimo Canonico, and Marco Guazzone. 2017. Forensic analysis of Telegram Messenger on Android smartphones. https://www.sciencedirect.com/science/article/pii/S1742287617301767. *Digital Investigation* 23 (2017), 31–49.

[3] J. Gregorio, A. Gardel, and B. Alarcos. 2017. Forensic analysis of Telegram Messenger for Windows Phone. https://www.sciencedirect.com/science/article/pii/S1742287617301032. *Digital Investigation* 22 (2017), 88–106.

[4] Dirk Pawlaszczyk and Christian Hummert. 2021. Making the Invisible Visible – Techniques for Recovering Deleted SQLite Data Records. https://conceptechint.net/index.php/CFATI/article/view/17. *International JOURNAL of Cyber Forensics and Advanced Threat Investigations* 1, 1-3 (2021).

[5] Alexandros Vasilaras, Donatos Dosis, Michael Kotsis, and Panagiotis Rizomiliotis. 2022. Retrieving deleted records from Telegram. https://www.sciencedirect.com/science/article/pii/S2666281722001287. *Forensic Science International: Digital Investigation* 43 (2022), 301447.

[6] Songyang Wu, Yong Zhang, Xupeng Wang, Xiong Xiong, and Lin Du. 2017. Forensic analysis of WeChat on Android smartphones. https://www.sciencedirect.com/science/article/pii/S1742287616301220. *Digital Investigation* 21 (2017), 3–10.

Slovenian command word speech recognition using transfer learning

Blaž Kovačič
blaz.kovacic@student.um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

Borko Bošković
borko.boskovic@um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

ABSTRACT

In this work, we present a speech recognition system using convolutional neural networks that are able to differentiate between four different Slovenian command words: naprej, nazaj, levo, desno. In neural network training, we intentionally limited ourselves to only 20 audio recordings per command word from a single speaker. We used transfer learning in conjunction with audio augmentation on a pre-trained model that we previously trained on 10 different English words with a total of about 25000 recordings from the Google Speech Commands Dataset v0.01. Using the transfer learning approach, we achieved highly accurate results on the test set with an accuracy of 0.988. Additionally, we demonstrate the soundness of our approach by comparing the results with those obtained when training an empty model from scratch, observing a substantial difference in results.

KEYWORDS

Speech recognition, transfer learning, human-computer interaction

1 INTRODUCTION

Speech recognition is an important interdisciplinary field that is present at every step of our everyday lives. It is used in home automatization with virtual assistants, in the transcription of medical documentation, in smart telephone answering machines, in the automatic generation of subtitles for deaf and hard of hearing, and in the speech controlled assistive technologies for disabled people.

Speech recognition technology goes back to the 50s of the previous century [5] when Bell Laboratories developed a system that was able to recognize spoken digits for a single speaker. We can generally differentiate between continuous speech recognition and isolated-word speech recognition, where the latter task is concerned with the recognition of a single word at a time. Today the technology in use for continuous speech recognition is mostly based on deep neural networks, convolutional neural networks (CNNs), and statistical models such as the Hidden Markov Model [1].

In our work, we focus on isolated-word speech recognition using CNNs, which means we recognize a single word in isolation. When building customized speech-controlled systems, such as an assistive technology solution for a specific person with special needs, we may be faced with limited availability of training data for our speech recognition system. This is especially true for the Slovenian language, where, to the best of our knowledge, doesn't exist a large database containing recordings of short command words. We address this issue in our work, by limiting ourselves to only 20 recordings per command word from a single speaker, and then use transfer learning and data augmentation techniques to develop a highly accurate speaker-dependent Slovenian command word speech recognition system.

The first step we take to develop such a system is to build a base model using large amounts of one-second-long recordings (about a total of 25000 recordings) using the Speech Commands Dataset v0.01 by Google [7]. We then use the technique of transfer learning on this model, by freezing all neural network layers except for the layers in last 5 blocks (corresponding to 14 out of a total of 27 layers) and then training those top layers using the limited data that we have. We demonstrate that using such an approach yields very favorable results and allows the model to generalize much better to the test data.

2 RELATED WORK

In the related work [6] on which we build upon and extend it using the idea of transfer learning, the authors used a deep learning approach for isolated-word speech recognition using CNNs. The authors recognized 10 different keywords such as "left", "right", "on", "off". They compared different approaches of neural network training: In the first approach they used raw audio data in conjunction with 1D CNN, whereas in the other two approaches they used Mel-spectrograms and Mel-frequency cepstral coefficients (MFCCs) as features that were input into a 2D CNN, where the latter approach yielded the best results (accuracy of 0.9619).

In [2], the authors used transfer learning for an automated speech recognition task, adapting a Wav2Letter CNN. They used an English base model to train a German model that outperformed the German baseline model trained from scratch. This can be especially helpful when limited amounts of training data and limited GPU memory are available; in addition, the training time is significantly reduced, and the final model accuracy is much higher.

In [3] the authors worked on a problem of keyword detection (similar to "Ok Google" detection in Google Assistant) where they compared three approaches: neural network with one hidden layer (a so-called "vanilla" neural network), a deep neural network with three hidden layers, and a CNN. The first approach is very fast, but such oversimplified network architecture did not yield optimal results; using the second approach the authors obtained an accuracy of 0.719, whereas using CNN yielded an accuracy of 0.945.

In [4] the authors worked on a task of large-vocabulary continuous speech recognition. Using raw audio signals, the authors trained a CNN to extract features (using convolutional filters) of basic word

DOI: https://doi.org/10.26493/scores23.04

units - phonemes. They relayed the output of the CNN to the next part of the neural network containing linear units that returned conditional probabilities of phonemes for each frame of speech recording. The authors then decoded this sequence of phonemes using a hidden Markov model into final words.

3 EXPERIMENTAL SETUP

3.1 Environment

We performed the experiment on Windows 11 using the GeForce RTX 3070 GPU with 8GB VRAM and central processing unit AMD Ryzen 7 5800H, 3.20GHz. We installed the TensorFlow 2.10 package for Python and executed code on WSL2 Ubuntu (Windows Subsystem for Linux). We used librosa library to obtain MFCC features, and audiomentations library for audio data augmentation.

3.2 Base model design

In the first part of the experiment, we build upon the work of Soliman, et. al. [6]. We use the Speech Commands Dataset v0.01 which contains one-second-long recordings of short English words such as "Happy" or "Marvin". For each word class there are a total of about 2500 recordings from various speakers. Just as authors in [6], we limit ourselves to the following 10 word classes: "yes" (2377), "no" (2375), "up" (2375), "down" (2359), "left" (2353), "right" (2367), "on" (2367), "off" (2357), "go" (2372) and "stop" (2380). In addition to these classes, we include the class for "Silence" ($6 \times 60s$ split into 360 one-second-long recordings) and "Other". The class "Silence" includes recordings of environmental noise, whereas the class "Other" contains 1000 one-second-long recordings of words from 10 classes other than those we trained it upon, such as "Happy" or "nine"; we used 100 recordings for each of 10 out-of-vocabulary classes. In the implementation, we use the 2D CNN architecture from [6] which is shown in Table 1. We sample the audio using a 16 kHz sampling frequency. We first put each audio recording through a pre-processing phase where we use a pre-emphasis high-pass filter to put more emphasis on higher frequencies of the audio, after which we z-normalize the audio by subtracting the mean and dividing each audio sample by the standard deviation. We then proceed with feature extraction where we use Mel-frequency cepstral coefficients (MFCCs) as features. For each time window, we extract 40 MFCC coefficients from the audio, using a window length of 25 ms and a hop length of 10 ms. The resulting CNN input shape is (40, 101, 1). We split data into training, validation and test sets using the ratios of 70:10:20. The neural network was trained in 200 epochs with a batch size of 64.

Block	Layer Type	Units	Kernel Size
1	Conv2D, BN, DO	32	(3, 3)
2	Conv2D, BN, MP, DO	32	(3, 3)
3	Conv2D, BN, DO	64	(3, 3)
4	Conv2D, BN, DO	64	(3, 3)
5	Conv2D, BN, MP, DO	64	(3, 3)
6	Conv2D, BN, DO	128	(3, 3)
7	Conv2D, BN, MP, DO	128	(3, 3)
8	Flatten	-	-
	Dense	1000	-
9	Dense	L2 (Softmax)	-

Table 1: Base model CNN architecture based on [6], where Conv2D corresponds to 2D convolutional layer, BN is Batch Normalization layer, MP is the Max Pooling 2D layer, and DO is 0.5 Dropout layer

3.3 Transfer learning approach

We recorded and classified one-second-long audio recordings into five classes that correspond to command words spoken in Slovenian language, namely: "naprej" (forward), "nazaj" (backward), "levo" (left), "desno" (right), and *Silence* class. The Silence class is composed of one-second-long recordings of environmental noise.

By design, we limited our training and validation data to 20 unique recordings per class, recorded with a smartphone for a single speaker. Additionally, we expanded the training and validation data (but not the test data) using audio augmentation technique, generating 100 additional recordings for each class. We used the audiomentations Python library to augment the recordings, varying the pitch by -2 to 2 semitones with a probability of 0.5, time stretching the audio by a rate between 0.8 and 1.25 with a probability of 0.5, and shifting the audio recordings by a fraction between -0.3 and 0.3 with a probability of 0.5. We split the resulting 120 recordings per class in ratio 85:15 among training and validation sets (giving us 102 samples/class in the training set, and 18 samples/class in the validation set).

The test data was recorded by the same speaker, but from a laptop microphone and at a different distance. In addition, the test data contains unique (not augmented) recordings, namely 100 unique recordings per class, each containing a range of natural variations in pitch and speed of pronunciation, which was done to obtain a sure estimate of model accuracy.

We then used the pre-trained base model and replaced the last softmax layer with a new one (for our 4+1 classes), and froze all layers in blocks 1-4 (see Table 1), training only layers from last 5 blocks (layers from *Conv2D 64* to *Dense 5 Softmax*). We decided upon training exactly 5 blocks empirically, as much fewer or much more than that had a negative affect on accuracy. In our experimental results, we demonstrate the significant difference that this approach makes in comparison to training all layers of a pre-trained model, as well as in comparison to model training from scratch.

For training, we used the Adam optimizer with a learning rate of 1e-3, batch size of 64, an early stopping callback with patience of 10, that monitors validation loss and prevents overfitting.

4 EXPERIMENTAL RESULTS

4.1 Base model

Training the base model persisted until the 105-th epoch, when it was interrupted by an early stopping mechanism, resulting in a training accuracy of 0.973, validation accuracy of 0.950, and a test accuracy of 0.954. The average class F1 measure on the test set was 0.956. Model accuracy with respect to the epoch number is given in Figure 1, and the confusion matrix is shown in Figure 2.

The obtained results are very much in line with those of the reference paper [6], and serve as a good basis on which to build upon using transfer learning.

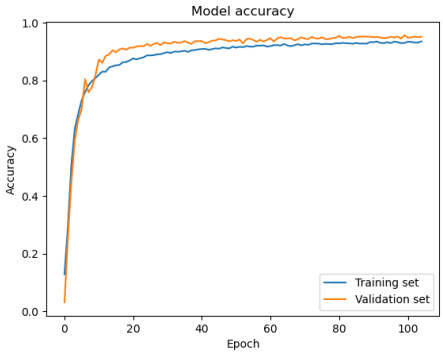


Figure 1: Base model train and validation accuracy with respect to the epoch number for 2D CNN using MFCC features.

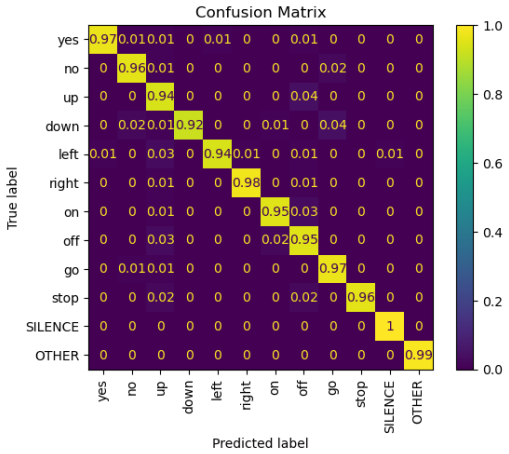


Figure 2: Base model confusion matrix for 2D CNN using MFCC features.

4.2 Transfer learning approach

We demonstrate the soundness of our approach by first displaying results of transfer learning when freezing all layers except for those in the last 5 blocks of the pre-trained base model, and then comparing that with the results obtained by training *all* layers of the pre-trained base model, which results in a worse model that exhibits lower accuracy on the test data. Finally, we demonstrate the unfeasibility of the approach where the model is trained without transfer learning using uninitialized weights, showing a substantial difference in accuracy.

4.2.1 Results when training last 5 blocks of the pre-trained model. The pre-trained base model was used and all layers except for those in last 5 blocks were frozen. The training was early-stopped after 99 epochs and lasted less than a minute. Training and validation accuracy of 1.00 were obtained, with a test accuracy of 0.988. The average class F1 score on the test set was 0.988. Model accuracy with respect to the epoch number is shown in Figure 3, and the confusion matrix is shown in Figure 4.

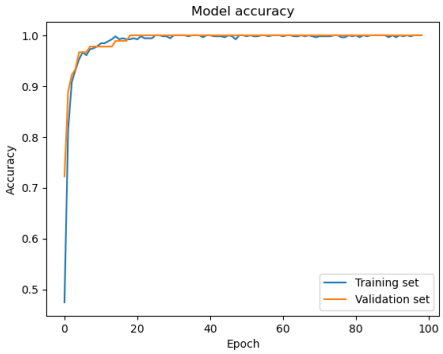


Figure 3: Transfer learning model (trained on layers from last 5 blocks) train and validation accuracy with respect to the epoch number for 2D CNN using MFCC features.

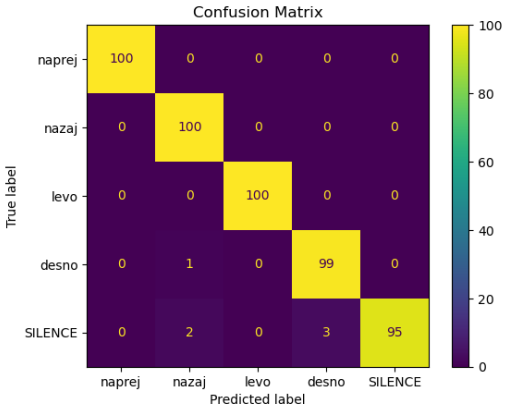


Figure 4: Transfer learning model (trained on layers from last 5 blocks) confusion matrix.

4.2.2 Results when training all layers of a pre-trained model. The pre-trained base model was used and all layers were trained. The training was early-stopped after 125 epochs. The obtained test accuracy of 0.930 was lower than when using the previous approach. The average class F1 score on the test set was 0.928. The confusion matrix for this setting is shown in Figure 5, demonstrating a somewhat degraded accuracy when compared to the model that was trained only on layers from the last 5 blocks.

4.2.3 Results without transfer learning. The original model architecture was used and all layers were trained from scratch. It was observed from high variations in validation accuracy with respect to the epoch number, that overfitting occurred, and that the model didn't generalize to the test data, which was observed from train accuracy being 0.9, validation accuracy of 0.92, and a test accuracy of 0.418, with an average F1 score on test set being 0.352. The large difference between the test and validation accuracy is most likely due to the fact that the test set was generated in a different way than the train and validation sets, and was thus more challenging.

To further test the feasibility of training from scratch with such a small dataset, we decided to implement a much smaller model with

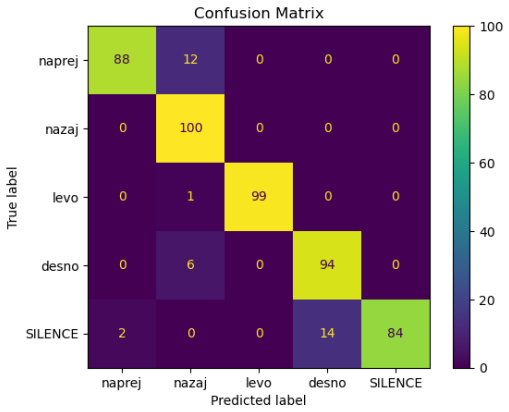


Figure 5: Transfer learning model (trained on all layers) confusion matrix.

the architecture consisting of Conv2D layer with 32 units, followed by MaxPolling2D layer and by Conv2D with 64 units, followed by MaxPooling2D layer, and then by Flattening and Dense 64 layer, and finally a Softmax layer with 5 units. The model was trained for 106 epochs; from the learning curve it was observed that model validation accuracy converged quickly and without large variations. Training accuracy and validation accuracy were both 1.0, whereas test accuracy was merely 0.695, and the average F1 score of 0.696 on the test set. Even though the validation accuracy was very high, the model wasn't able to predict the test data very well.

5 DISCUSSION

In summary, we can see that it is possible to make use of transfer learning in combination with audio data augmentation to train a convolutional neural network with very small amounts of training data, yielding a useful model that can be utilized in a command word speech recognition system with favorable accuracy.

We could see that by freezing an optimal amount of bottom CNN layers, which contain filters that detect less complex features than top layers, we can fine tune the base model and improve accuracy.

The approach, however, is not without limitations. One limitation that we observed was that the choice of command words that we decided to recognize could influence the system's ability to differentiate between them. Additionally, it was observed that the neural network seems to overgeneralize, assigning overly confident classification probability scores to erroneous classes when presented with "out-of-vocabulary" words, although this was observed to occur with the base model as well. This leaves room for further research into methods of out-of-vocabulary word detection when utilizing CNNs for spoken word command recognition.

6 CONCLUSION

We presented a speaker-dependent speech recognition system that is successfully able to differentiate between four different Slovenian command words (including a class for *silence*), trained on very limited amounts of training data, namely 20 recordings per word class. We achieved a high recognition accuracy of 0.988 by making use of the transfer learning approach, in combination with the audio data augmentation technique.

First we augmented the 20 recordings in each class by additional 100 recordings, by synthetically varying the pitch, time stretching and time shifting the audio recordings, giving a total of 120 recordings for each class in the training and validation sets. For training, we made use of a pre-trained convolutional neural network model which we previously trained on large amounts of recordings of 10 different word classes from the Google Speech Commands Dataset, using MFCCs as features. Obtained model accuracy was 0.954, which was very much in line with results of the reference paper [6]. We then used this model by training only the layers from its last 5 blocks (14 layers out of a total of 27 layers). To obtain a realistic classification accuracy score, we built a separate test set, using a different microphone and a different recording distance than in recordings utilized for the training and validation sets. This test set consisted of 100 unique recordings for each class, from a single speaker, and was not augmented.

We demonstrated that freezing the optimal amount of bottom layers yields better training results (accuracy of 0.988) than when training all layers of a pre-trained model (accuracy of 0.930). We have also shown that when using our small dataset, model training without transfer learning yielded much lower test accuracy scores, namely, 0.418 and 0.695 for the larger and smaller models, respectively.

We can conclude that by making use of the transfer learning in combination with data augmentation, we can compensate for small amounts of training data, allowing us to build a potentially useful system that can respond to the Slovenian spoken word commands from a single speaker. In future work we would like to research and compare methods that would allow for accurate detection of "out-of-vocabulary" words, such as the use of Mahalanobis distance or Bayesian neural networks. Additionally, we would like to research the model prediction accuracy and the optimal amount of recordings that would be required under the condition that same commands are spoken by different speakers.

REFERENCES

[1] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. 2012. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97. https://doi.org/10.1109/MSP.2012.2205597

[2] Julius Kunze, Louis Kirsch, Ilia Kurenkov, Andreas Krug, Jens Johannsmeier, and Sebastian Stober. 2017. Transfer learning for speech recognition on a budget. *arXiv preprint arXiv:1706.00290* (2017).

[3] Xuejiao Li and Zixuan Zhou. 2017. Speech command recognition with convolutional neural network. *CS229 Stanford education* (2017), 31.

[4] Dimitri Palaz, Mathew Magimai.-Doss, and Ronan Collobert. 2015. Convolutional Neural Networks-based continuous speech recognition using raw speech signal. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 4295–4299. https://doi.org/10.1109/ICASSP.2015.7178781

[5] Melanie Pinola. 2011. *Speech Recognition Through the Decades: How We Ended Up With Siri.* https://www.pcworld.com/article/477914/speech_recognition_through_the_decades_how_we_ended_up_with_siri.html Accessed on 2023-03-19.

[6] Aljenan Soliman, Salah Mohamed, and Iman Abuelmaaly Abdelrahman. 2021. Isolated Word Speech Recognition Using Convolutional Neural Network. In *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*. 1–6. https://doi.org/10.1109/ICCCEEE49695.2021.9429684

[7] Pete Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *CoRR* abs/1804.03209 (2018). arXiv:1804.03209 http://arxiv.org/abs/1804.03209

Identifying communities and ranking the drivers' performance in Formula One

Matej Horvat
mh3418@student.uni-lj.si
Faculty of Computer and Information Science, University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

Lan Sovinc
ls7312@student.uni-lj.si
Faculty of Computer and Information Science, University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

Domen Grzin
dg6781@student.uni-lj.si
Faculty of Computer and Information Science, University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

ABSTRACT

This paper explores an approach for ranking the performance of Formula One drivers across different eras using network analysis methods, namely PageRank combined with community detection algorithms. Typically, attempts to rank driver performance have been limited, focusing solely on the evolution of technology and rules in the sport, and attempting to find the best individual driver, ignoring the fact that numerous good drivers never competed directly. To address these challenges, we collected data from all Formula One races since its inception in 1950 through 2022 and used network analysis to create communities of drivers based on their direct competition with each other. Within these communities, we then applied the PageRank algorithm to rank the drivers. Our approach has been shown to produce more meaningful and relevant driver rankings compared to the full graph PageRank results, and effectively recognises the dominance of drivers in their respective eras. Our methodology provides the basis for more sophisticated performance comparisons in sports with long histories and changing conditions.

KEYWORDS

Network Analysis, Formula One, Community Detection, Performance Comparison, PageRank, Leiden Algorithm

1 INTRODUCTION

The FIA Formula One World Championship was introduced in 1950. The word "Formula" in the name refers to the set of rules laid down by the FIA (Fédération Internationale de l'Automobile), to which all cars must conform. Over the years, the championship has been subject to numerous changes in order to keep it at the cutting edge of technology, maintain popularity and spectator numbers, and remain relevant in the world of motorsport.

Technology, engineering and rule changes play a very important role in the sport, but it is often said that the driver makes the biggest difference in the final performance of a constructor. Fans of the sport often compare their favourite drivers to others and try to make them seem better than others, or even hail them as the best ever. Since the beginning of the championship, there have been more than 800 drivers. We believe that the talent and performance of different drivers over the years cannot be objectively compared. Their results can only be compared with the results of those who have raced alongside them. There are some eras in the sport that are not formally defined, which refer to certain technological changes or changes in the rules of the sport. The results of the most successful drivers can often be linked to these eras, but the results still depend too much on the work of engineers and race car designers, rather than on the performance of the individual driver. To solve this problem, we used the principles of network analysis to find an objective way to classify the performance of drivers using the help of Leiden algorithm and node importance classification algorithm.

2 RELATED WORK

Network analytic methods have become a valuable tool in sports research [12]. Their use spans several areas of sports research; in sports management, they can be used to study the relationships between teams and their sponsors [5]. They can help understand the success and performance of teams in competitions [9] or social structures of sports organisations, e.g., what characteristics cause a tennis match to be abandoned [2].

One of the most important methods in network analytics are centrality measures. They provide quantitative metrics for assessing the importance and centrality of nodes in a network. In a network of passes between members of a basketball team, centrality measures rank players according to their importance to the team [6]. We focused our research on PageRank [3]. PageRank was originally developed by Google to rank web pages. It focuses on the idea that important nodes in a network are mainly connected to other important nodes. This is achieved by counting the number and quality of edges of a node, which estimates the importance of the node. Another important method of network analysis is community detection. With these methods, the network is divided into multiple node communities based on criteria such as similarity measures, modularity, or optimization algorithms. In a network of basketball players, players can be grouped based on their performance [4]. Discovering the optimal communities in a network (modularity optimization) is a well-known NP-hard problem [1], so when we talk about community discovery algorithms, we talk about heuristic algorithms. As such, they differ in their speed and correctness of detected communities [8]. We decided to use the **Leiden** modularity optimization algorithm since it gives excellent results and computes them quickly [11].

In recent years, the use of modularity to improve the results of centrality measures has received some attention [7]. Since most real-world networks are modular, this logically leads to better results. There is much work demonstrating the usefulness of such centrality measures, including in social networks, e.g., for identifying influential spreaders during an epidemic.

DOI: https://doi.org/10.26493/scores23.08

3 RESULTS

3.1 Graph without communities

Table 1: Top drivers by Pagerank

Driver	Active years	Titles won	Wins	Podiums
Juan Fangio	1950 – 1958	5	24	35
Graham Hill	1958 – 1975	2	14	36
Jack Brabham	1955 – 1970	3	14	31
Stirling Moss	1951 – 1961	0	16	24
Bruce McLaren	1958 – 1970	0	4	27
Lewis Hamilton	2007–	7	103	192
Alberto Ascari	1950 – 1955	2	13	17
Nino Farina	1950 – 1956	1	5	20
Alain Prost	1980 – 1993	4	51	106
Niki Lauda	1971 – 1985	3	25	54

Looking at the results of running the PageRank algorithm on the graph without communities, we see some very high-performing drivers in the table 1. 8 of the top 10 nodes represent drivers who have won at least one drivers' championship title in their time on the circuit. We can quickly see that many very good drivers are missing from the list (the most prominent example being the legendary Michael Schumacher). The second obvious problem is that we are comparing drivers who raced in completely different eras of the sport. The list includes some of the earliest winners, like Juan Fangio, and some of the most recent, like Lewis Hamilton. The amount of information we can extract from this list is relatively small and is the reason we prefer to analyse PageRank results for each of the communities we detected.

3.2 Detected communities

The Leiden community detection algorithm has identified 6 communities of drivers that have raced with each other the most. Applying the PageRank algorithm to each community yields promising results compared to the data without communities. Results of the PageRank algorithm and detected communities are shown on figure 1.

3.2.1 Community 1. The first community identified contains the drivers who raced from the beginning of Formula One in the 1950s to the 1970s. Of the first 10 top drivers by PageRank, 6 out of 10 drivers became drivers' world champions at least once.

The top 3 drivers from this community were:

- Juan Manuel Fangio: The driver dominated in the early days of Formula One, winning the drivers' championship five times in the 1950s. His record remained undefeated until 2003, when Michael Schumacher became world champion for the sixth time.
- Jack Brabham: Three-time world champion in the late 1950s and 1960s who was also knighted for his racing successes.
- Graham Hill: Two-time world champion in the 1960s and the only driver to win the Triple Crown of Motorsport (the 24 Hours of Le Mans, Indianapolis 500 and Monaco Grand Prix).

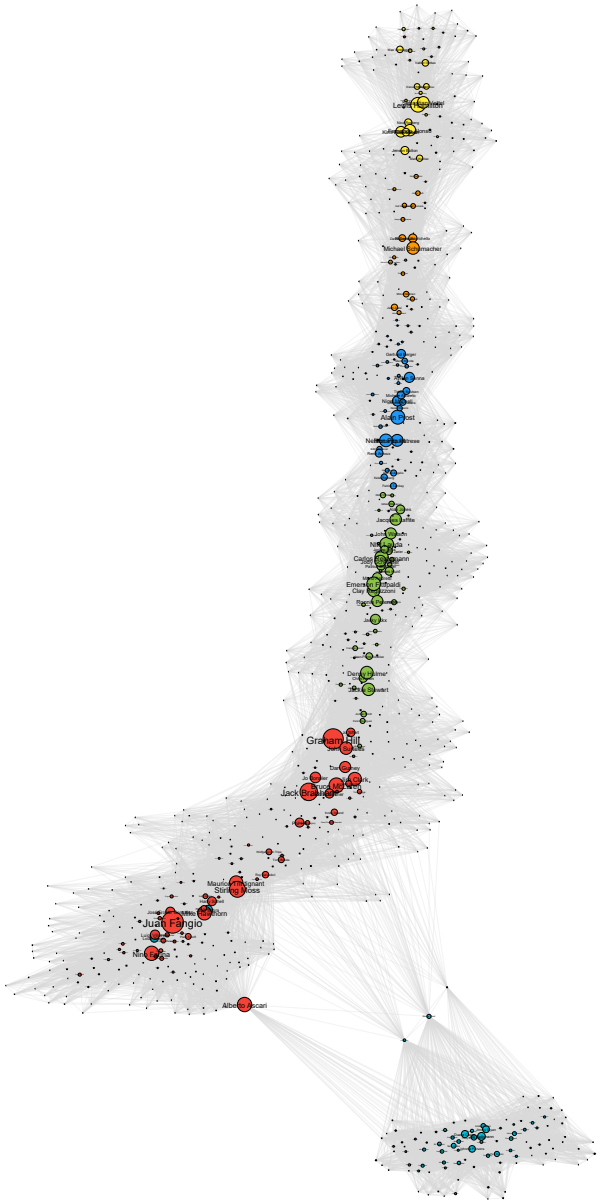


Figure 1: Formula One drivers performance network. The bigger the node, the better the driver's performance. Colored by the detected communities.

The other top finishers include some of the most successful drivers of the era. While not all drivers became world champions, they won many times, scored many points in their careers, and contributed greatly to the sport.

3.2.2 Community 2. The second community contained nodes of drivers that competed from the 1970s to the early 1980s (table 2). 4 of the top 10 nodes represent the drivers, who became world champions. Interestingly, only one of the first three drivers on the list was a world champion, showing that our approach does not

identify the drivers performance only by the number of titles they won. However, it should be noted that the results of the first 5 drivers in the ranking are very similar.

Table 2: Top drivers by Pagerank in community 2

Driver	Active years	Titles won	Wins	Podiums
Emerson Fittipaldi	1970 – 1980	2	14	35
Carlos Reutemann	1972 – 1982	0	12	45
Clay Regazzoni	1970 – 1980	0	5	28
Niki Lauda	1971 – 1985	3	25	54
Jody Scheckter	1972 – 1980	1	10	33
Ronnie Peterson	1970 – 1978	0	10	26
John Watson	1973 – 1985	0	5	20
Jacques Laffite	1974 – 1986	0	6	32
James Hunt	1973 – 1979	1	10	23
Patrick Depailler	1972 – 1980	0	2	19

The top 3 drivers in the list:

- Emerson Fittipaldi: One of the most famous drivers of this era and 2-time champion.
- Carlos Reutemann: Never became world champion, but finished at the top of the championship in the late 1970s and early 1980s, winning 12 Grand Prix.
- Clay Regazzoni: The driver won 5 times, but never became champion. His other achievements include 28 podiums and 209 points.

Also notable are three-time world champion Niki Lauda and one-time world champion James Hunt, who were known for their rivalry in the 1970s.

3.2.3 Community 3. This group consists of drivers from the 1980s to early 1990s and includes many icons of motorsport. In this period there were 6 different world champions and 4 of them are the first 4 on the ranking. The other two were Keke Rosberg and Niki Lauda, which the algorithm assigned to the previous community.

The top finishers were:

- Alain Prost: He was a four-time world champion and also held the record for most Grand Prix wins until 2001.
- Ayrton Senna: One of the sport's greatest legends and a three-time world champion. He is also known for his rivalry with Prost.
- Nelson Piquet: Also won the drivers' championship three times and is considered one of the best drivers of the era.

3.2.4 Community 4. The fourth community is very interesting because it contains drivers from the 1950s and 1960 who competed in Formula One only for the Indianapolis 500 race, which is now exclusively part of the IndyCar Series. The race was on the Formula One calendar from 1950 to 1960. Visualizing the network those drivers stand out by creating a lot of edges between each other and very little between "regular" F1 drivers as they only raced with them once a year. Also, none of those racers ever won the drivers' championship.

The top performers of the community are:

- Jim Rathmann: Winner of the Indianapolis 500 in 1960.
- Duane Carter: Scored a third place in the Indianapolis 500 in 19533.
- Jimmy Bryan: Winner of the Indianapolis 500 in 1958.

3.2.5 Community 5. This community includes drivers from the recent Formula One era (table 3). We have 6 world champions in the top 10 places.

Table 3: Top drivers by Pagerank in community 5

Driver	Active years	Titles won	Wins	Podiums
Lewis Hamilton	2007–	7	103	192
Sebastian Vettel	2007 – 2022	4	53	122
Fernando Alonso	2001–	2	32	102
Kimi Räikkönen	2001 – 2021	1	21	103
Valtteri Bottas	2013–	0	10	67
Max Verstappen	2014–	2	38	82
Nico Rosberg	2006 – 2016	1	23	57
Felipe Massa	2002 – 2017	0	11	41
Daniel Ricciardo	2011 – 2022	0	8	32
Sergio Pérez	2011–	0	6	30

In the top 3 we find:

- Lewis Hamilton: Considered one of the greatest drivers in Formula One, he is a seven-time world champion. He holds many records, including sharing the record for the most drivers' world titles with Schumacher and the most total Grand Prix victories.
- Sebastian Vettel: Four-time consecutive world champion, dominating in the early 2010s.
- Fernando Alonso: The driver with the longest racing experience in the sport and a two-time world champion who regularly scores points in races.

Other important names are Max Verstappen with 2 world titles in 6th place and Valtteri Bottas and Sergio Perez in 5th and 10th place. These two are interesting because they have never been world champions, but have competed for teams that have won many constructors' titles in recent years.

3.2.6 Community 6. The last community identified contains drivers whose careers span from the 1990s to the early 2000s (table 4). There are only 3 world champions in the top 10, but this is mainly due to the dominance of Michael Schumacher.

Table 4: Top drivers by Pagerank in community 6

Driver	Active years	Titles won	Wins	Podiums
Michael Schumacher	1991 – 2012	7	91	55
David Coulthard	1994 – 2008	0	13	62
Rubens Barrichello	1993 – 2011	0	11	68
Mika Häkkinen	1991 – 2001	2	20	51
Ralf Schumacher	1997 – 2007	0	6	27
Giancarlo Fisichella	1996 – 2009	0	3	19
Jacques Villeneuve	1996 – 2006	1	11	23
Jean Alesi	1989 – 2001	0	1	32
Heinz-Harald Frentzen	1994 – 2003	0	3	18
Eddie Irvine	1993 – 2002	0	4	26

The 3 most powerful drivers were:

- Michael Schumacher: For many, Formula One's greatest legend, sharing the record for most world titles won with Lewis Hamilton. He won his first two titles in the nineties, but his dominance on the track was the greatest in the early 2000s, when he won five more in a row.
- David Coulthard: Never won the title, but finished in the top three five times between 1995 and 2001.
- Rubens Barrichello: Also never won the drivers' title, but finished in the top 5 every year between 2000 and 2004.

Other important drivers on the list include two-time world champion Mika Häkkinen and one-time title winner Jacques Villeneuve.

4 METHODS

The research was started by collecting the drivers data. We acquired the open source dataset from the platform Kaggle [10]. It contained data about all the Formula One races, drivers, constructors, qualifying, circuits, lap times, pit stops, championships from 1950 till the present day.

The dataset was composed of 14 tables in the form of .csv text files, where we decided to only include completed championships so we cut out data from the year 2023 onwards. We processed the raw data using Python and the data manipulation and analysis library pandas, using the results of all provided races. For each race we calculated the number of drivers, as some earlier races had multiple drivers racing with the same vehicle.

Then the graphs and networks library NetworkX was used to construct a directed multigraph. We wanted to create a graph that represented who competed with whom, for how long and how good their relative performance was. The graph was composed of nodes, where each node represented a driver. Edges between the drivers represented their ranking in the championship. For each race of every season we made edges based on the following principle: If driver A had a better final position than driver B, then we formed a directed edge from the node representing driver B to A. Each edge has a weight associated with it, depending on the final position in the race of the driver forming the directed edge. We normalised the weights to equalise the effect of each race and prevent some unexpected outliers. This means that the last driver formed directed edges to every other driver that competed in that race, the second last to every other except the last and so forth until the first driver, who did not form any edges.

After creating the graph we continued by detecting communities. For the proof of concept we used the built-in function of NetworkX to detect communities using the Louvain algorithm (nx.community.louvain_communities). We later switched to the Ledien algorithm in the final implementation using the library cdlib.

Once we gathered the detected communities of the drivers who competed together, we continued by classifying driver node performance by using the Pagerank algorithm, for which we used the NetworkX built-in function (nx.pagerank). We also ran the algorithm on the graph without communities to get a baseline performance for each.

To visualise the communities created by the Leiden algorithm, we used the library igraph to create a plot to represent the outcome. We used the Fruchterman-Reingold force-directed graph drawing for the plot, where the node size represents the performance of the node representing the driver and color represents the comunity the driver belongs to.

5 CONCLUSION

Many times, PageRank has proven to be one of the most successful methods of determining the importance of nodes. However, in the case of Formula One, which has been held for decades, the algorithm was insufficient because the result of the calculation was a list of drivers from different eras, and many of the top candidates have never participated in a race. Such ranking of drivers is largely unfounded. We solved the problem by identifying communities. In layman's terms, the approach to grouping drivers into communities is to create communities based on technical improvements or rule changes. Our approach was more systematic, as we calculated communities algorithmically based on the number of races the drivers raced against each other. This allowed us to compare the relative performance of each driver to those they actually raced against. The Leiden algorithm also detected the one special group of racers who only raced at a single event each year, which was not explicitly noted anywhere in the dataset. When we applied PageRank to each of the calculated communities, the results were much more meaningful than the results of the first algorithm run. Each of the communities contains mainly world champion drivers and their close competitors. Most importantly, the drivers in the list all raced together and their careers overlapped to a large extent. This allowed us to create a fair driver ranking, and the amount of knowledge we could extract from the data increased significantly.

In the future, the ranking system could be improved even more. At the moment, we do not compare all drivers directly with each other, but only the drivers in detected communities. After evaluating the performance of the drivers for every community, the analysis could continue. Ideally, a method that compares the relative performance of the top ranked nodes in each community should be developed to compare the drivers in absolute terms.

REFERENCES

[1] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. 2006. On Modularity - NP-Completeness and Beyond.

[2] Kristijan Breznik and Vladimir Batagelj. 2012. Retired Matches Among Male Professional Tennis Players. *Journal of sports science & medicine* 11 (06 2012), 270–8.

[3] S. Brin and L. Page. 1998. The anatomy of a large-scale hypertextual Web search engine. *Comput. Networks ISDN* 30, 1-7 (1998), 107–117.

[4] Alessandro Chessa, Pierpaolo D'Urso, Livia De Giovanni, Vincenzina Vitale, and Alfonso Gebbia. 2022. Complex networks for community detection of basketball players. *Annals of Operations Research* (08 2022).

[5] Joe Cobbs. 2011. The dynamics of relationship marketing in international sponsorship networks. *Journal of Business & Industrial Marketing* 26 (10 2011), 590–601.

[6] Jennifer Fewell, Dieter Armbruster, John Ingraham, Alexander Petersen, and James Waters. 2012. Basketball Teams as Strategic Networks. *PloS one* 7 (11 2012), e47445.

[7] Zakariya Ghalmane, Mohammed El Hassouni, Chantal Cherifi, and Hocine Cherifi. 2019. Centrality in modular networks. *EPJ Data Science* 8, 1 (may 2019).

[8] Gnce Keziban Orman, Vincent Labatut, and Hocine Cherifi. 2011. On Accuracy of Community Structure Discovery Algorithms. *Journal of Convergence Information Technology* 6, 11 (nov 2011), 283–292.

[9] Luca Pappalardo and Paolo Cintia. 2017. Quantifying the relation between performance and success in soccer. *Advances in Complex Systems* 21 (05 2017).

[10] Rao R. 2023. Formula 1 World Championship (1950 - 2023). Retrieved from https://www.kaggle.com/datasets/rohanrao/formula-1-world-championship-1950-2020. Accessed on May 9, 2023.

[11] V. A. Traag, Ludo Waltman, and Nees Jan Van Eck. 2019. From Louvain to Leiden: Guaranteeing well-connected communities. *Sci. Rep.* 9 (2019), 5233.

[12] Hagen Wäsche, Geoff Dickson, Alexander Woll, and Ulrik Brandes. 2017. Social network analysis in sport research: an emerging paradigm. *European Journal for Sport and Society* 14 (05 2017), 1–28.

Sensitivity Analysis of Named Entity Extraction based on Deep Learning

Lea Roj
lea.roj1@student.um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

Štefan Kohek
stefan.kohek@um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

Aleksander Pur
aleksander.pur@policija.si
Ministry of the Interior,
Štefanova ulica 2
SI-1000 Ljubljana, Slovenia

Niko Lukač
niko.lukac@um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

ABSTRACT

In the ever-evolving landscape of data visualization and extraction, deep learning techniques have become increasingly pivotal. Addressing this, our paper conducts a sensitivity analysis of Named Entity Extraction on text related to the infamous Panama Papers. We combine the Rebel and Natural Language Processing models to extract entities and relations. To visualize the extracted knowledge, we employ the NetworkX library to transform this data into intuitive graphs. These are then compared to a predefined expected graph to assess the influence of various parameters during the creation process. To ensure an accurate comparison, the graphs are transformed into vectors using the `Graph2Vec` method after undergoing pre-processing tasks, such as the removal of self-loops, isolated nodes, and node renaming. The similarity with our benchmark graph is determined using Euclidean distance metrics. Results highlight the influence of span length, length penalty, and the number of beams on graph generation. Notably, span length emerges as the most impactful factor, in determining graph detail and complexity.

KEYWORDS

Entity Extraction, Relation Extraction, Graph Generation, Embeddings, Graph Similarity

1 INTRODUCTION

Graphs provide an intuitive, more appealing visualization of data, revealing patterns and correlations that are not instantly visible from raw data. An alternative to graphs are relational and non-relational databases. Relational databases store data in structured tables and use primary and foreign keys to interconnect data. Through these connections, relational databases can generate valuable insights by merging tables [10]. On the other hand, non-relational databases, often called NoSQL databases [18], are non-tabular databases and excel in horizontal scalability, allowing more machines to be easily integrated. Their maintenance is cheap as they do not depend on expensive hardware and the accommodation of changes is easier. Moreover they are designed to store simple data structures as a key and a value.

Graphs, relational databases, and NoSQL databases are fundamental in data management but differ significantly. In graphs, data is stored as nodes, edges, and properties, offering flexibility when adding new relationships or nodes. In contrast, relational databases use tables with rows and columns, potentially requiring substantial redesigns when introducing new data relationships and data models. On the other hand NoSQL databases are more adaptable and can easily handle changes and a variety of data types, making them a scalable solution. While graphs employ query languages like Neo4j's Cypher [20], relational databases predominantly use SQL [14], and NoSQL databases use a variety of query languages. Extracting semantic relationships using relational databases for documents is challenging. Representing data with graphs emerges as an effective solution. A primary challenge is determining the optimal parameters for graph construction, which is the focus of our paper.

The concept of a "Named Entity" and relation extraction was first put forward during the Sixth Message Understanding Conference [5] in 1995. Recent advancements were described by Lung-Hao et al. in [11]. Insights from the publications indicate the development and evaluation of the capability of a Chinese healthcare NER recognizer. The best results were achieved by the MIGBaseline team using the BERT-BiLSTM-CRF model [3].

While Named Entity Extraction (NEE) identifies entities, Relation Extraction (RE) seeks to determine relationships between these identified entities. In the article from 2023, Moscato et al. [15] presented a multi-task framework for biomedical relation extraction using a transformer-based model trained on three publicly available datasets related to drug, protein, and medical relationships. Expanding on this theme of advancements in RE, the Collaborative Oriented Relation Extraction (CORE) system offers another noteworthy contribution. Introduced in 2023 by Marchesin et al. [13],

DOI: https://doi.org/10.26493/scores23.12

the CORE system was specifically designed to extract multi-aspect relationships from scientific texts.

In this paper, we introduce a novel workflow for testing hyperparameters and performing sensitivity analysis in the context of entity resolution, using Wikipedia summaries. Our approach employs Term Frequency - Inverse Document Frequency (TF-IDF) vectorization combined with cosine similarity in the specific context of entity resolution. While contemporary methods, such as Word2Vec [4] and BERT, provide more advanced ways to compute word similarity by capturing semantic meanings of the words, we opted to use the TfidfVectorizer [16] to transform textual data into a TF-IDF representation. Even though it is not the state-of-the-art it is effective and simple to use, due to its simplicity.

We perform a detailed comparison by converting these summaries into TF-IDF vector representations and subsequently calculating their cosine similarity. This strategy enables us to efficiently describe entities that various names could refer to while essentially denoting the same concept. Furthermore, by embedding the cosine similarity as an attribute of the relationship, we introduce a direct metric for quantifying the relevance between linked entities. Our method combines advanced NLP methods. The process begins with tokenization to prepare the text, followed by model-based generation to structure the information. A distinctive feature of our approach is the overlap strategy we have adopted in tandem with relation extraction. By segmenting long texts with overlaps, we maintain context and ensure relationships are not missed or fragmented.

The structure of this paper spans four sections. After the introduction, the following section delineates the methods and functions employed. The third section presents the results, including a sensitivity analysis concerning different parameters and their impact on NEE. The last section concludes the paper.

2 METHODOLOGY

The following subsections will provide a comprehensive breakdown of important methods implemented in the project.

2.1 Named entity extraction

Firstly, the input text is initially processed using the `en_core_web_lg`, an English language model, from the `spacy` library [7]. This pretrained model supports various NLP tasks such as NER, RE, and tokenization. Tokenization involves separating the raw text into smaller units known as tokens. Each token ID represents a unique word in the text, while same words share the same ID. By dividing text into manageable spans, determining the start and end tokens of those spans based on span and overlap length, this process converts the human-readable text into a format that machine-learning models can understand and prepares the text for further handling.

The span length determines the size of the processed text segments and is associated with the number of words. Opting for a lower number results in a more detailed graph, because more individual spans are analyzed. However, selecting an optimal span length is crucial because if the value is too small, the text can become too fragmented, rendering the represented nodes meaningless. The influence of span length is exemplified in Fig. 1. The graph in Fig. 1a contains 9 entities, whereas the graph in Fig. 1b has 35 entities. As can be seen, in graph Fig. 1a, an increased span length results in fewer spans being analyzed. Consequently, it may overlook some entities or relations that might be present in smaller spans.

On the other hand, the overlap length ensures a certain degree of continuity between consecutive text segments, preserving context. Through this overlap, we minimize the risk of losing essential contextual information at segment boundaries. The value represents the number of words from the previous span, that are also used at the beginning of the next one.

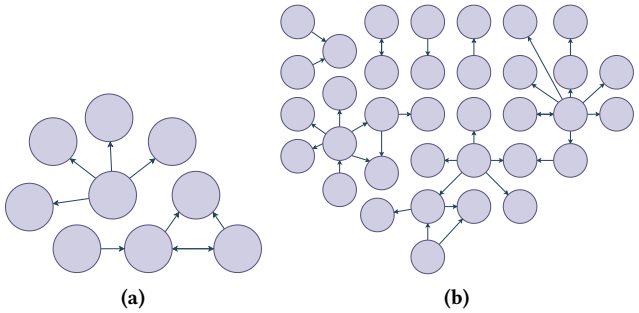


Figure 1: Impact of span length on graph generation when a) Span length = 120 and b) Span length = 50.

For Named Entity Recognition (NER), RE, and knowledge base creation [19] the rebel-large [9] model is used. It is a sequence-to-sequence (seq2seq) architecture, which is based on BART [12]. The seq2seq design enables the model to perform end-to-end RE tasks. It takes an input sequence, processes it, and produces an output sequence. In the context of "rebel-large," these output sequences represent triplets, where each triplet consists of a "head," a "tail," and the "relation" between them. The rebel-large model is prevalent in deep learning applications as it can identify and categorize named entities from raw text [1]. In conjunction with the model, initializing a corresponding tokenizer is critical, as it transforms raw text into a format the model can effectively process.

Handling vast amounts of data effectively is critical in optimizing the performance of deep learning models. The approach we took enables models to manage varying sizes of data by configuring specific parameters, such as input text, span length, overlap length, maximum length, length penalty, number of beams, number of return sequences, and the tokenizer's maximum length. Length penalty plays a pivotal role in sequence range determination. A positive value encourages longer sequences, a zero value maintains neutrality, and a negative value leans towards shorter sequences [21]. The number of beams influences the quality of the results by determining how many top sequences undergo exploration at each decision point or iteration during the search process. By employing the beam search algorithm, the number of return sequences determines how many top sequences are retrieved. The parameter maximum length defines the upper limit for the length of processed sequences, while the maximum length tokenizer indicates the maximum allowed length of text following tokenization. Given

the length of the processed text, span, and overlap, the text is divided into manageable segments, with each segment subsequently undergoing tokenization.

2.2 Relation Classification (RC)

When provided with a span of text, which was pre-processed using a NLP function, the rebel-large model predicts potential relationships, which are represented in token sequences. These sequences are subsequently decoded back into textual representations using the tokenizer. Within these textual representations, specific tokens ("<triplet>", "<subj>", "<obj>") help delineate and structure the relationship. We represent these potential relationships with "head", "tail", and "type". The "head" corresponds to the starting entity, while the "tail" refers to the ending entity of the relationship. The "type" characterizes how the two entities are related. For instance, as illustrated in Fig. 2, the "type" serves as the label on the relationship's edge.

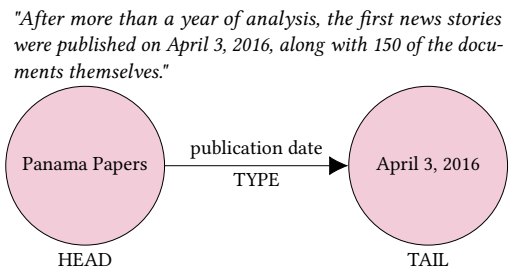


Figure 2: Illustration of relationship extraction from text.

2.3 Entity Filtering and Normalization

Entities undergo validation on Wikipedia via an HTTP GET request targeting a specific entity through Wikipedia's API. In response, the API returns data in JSON format, which includes the official title or name of the entity as recognized by Wikipedia, a link leading to the detailed Wikipedia page for the entity, and a brief overview of the entity. If entity data is found, then the entity title is the same as the one on the Wikipedia page. On the other hand, if data is not found, then it uses the provided entity name.

Cosine similarity is a common metric for measuring document similarity. By comparing summaries of entities from Wikipedia, we determine the degree of similarity between them. Using the TfidfVectorizer [16], textual data is transformed into a TF-IDF representation. Subsequently, the function `find_similar_entity` computes the cosine of the angle between the given summary and the Wikipedia summary of each stored entity, and returns the entity with the highest similarity score. Both summaries are vectors, representing the weighted word frequencies in a document [8]. A cosine similarity of 1 indicates that the vectors are identical, while a cosine similarity of 0 indicates that the vectors are completely dissimilar. Using Eq. 1 we solved the challenge of multiple entities sharing common names or terms. For example "Napoleon Bonaparte" and "Napoleon I" are two names for the same entity. Cosine Similarity is calculated as:

$$\frac{\vec{A} \cdot \vec{B}}{||\vec{A}||||\vec{B}||}, \tag{1}$$

where $\vec{A}$ and $\vec{B}$ present TF-IDF vectors of each summary.

2.4 Similarity calculation

The objective was to identify graphs resembling a predefined or expected graph. Before calculating similarity using embeddings from Graph2Vec [17], we preprocess each graph to ensure uniformity. This process includes removing self-loops, eliminating isolated nodes, and renaming nodes based on their sorted order in the node list of the graph. Subsequently, the `get_embedding` function obtains the desired embeddings.

Representing each graph as a vector in high-dimensional space enables more efficient graph comparison and analysis. Embeddings can be plotted in a coordinate system, where the proximity between points corresponds to their similarity. The Euclidean distance offers a measure of this distance by calculating the vector length between two points [2].

Using the graph embeddings, we define the Euclidean distance as:

$$d_i = \sqrt{\sum_{j \in I-i} (A_j - B_{i,j})^2}, \tag{2}$$

where A_j denotes the expected graph's embedding and $B_{i,j}$ represents the embeddings from other graphs. I is the set of all dimensions or indices in the high-dimensional space.

3 RESULTS

The analysis was conducted using a range of fixed and variable parameters. Those were chosen by heuristically testing parameters, based on which had the greatest influence on the graph generation and its quality. The variable parameters include span length, length penalty, and number of beams. On the other hand, the fixed parameters were set as:

- overlap length = 25
- maximum length = 130
- tokenizer's maximum length = 512

Table 1: Parameter configurations and similarity.

Span length	Length penalty	Number of beams	Similarity
90	-3	12	81.05%
60	-5	6	68.46%
50	0	6	66.73%
60	0	8	54.63%
90	0	6	52.20%
40	-5	6	46.60%
60	0	12	46.57%
50	5	12	45.88%
60	-1	6	41.78%
60	1	6	36.94%
40	-3	6	36.28%
40	1	6	35.12%
60	3	6	29.66%
60	5	6	21.40%
30	0	6	0.00%

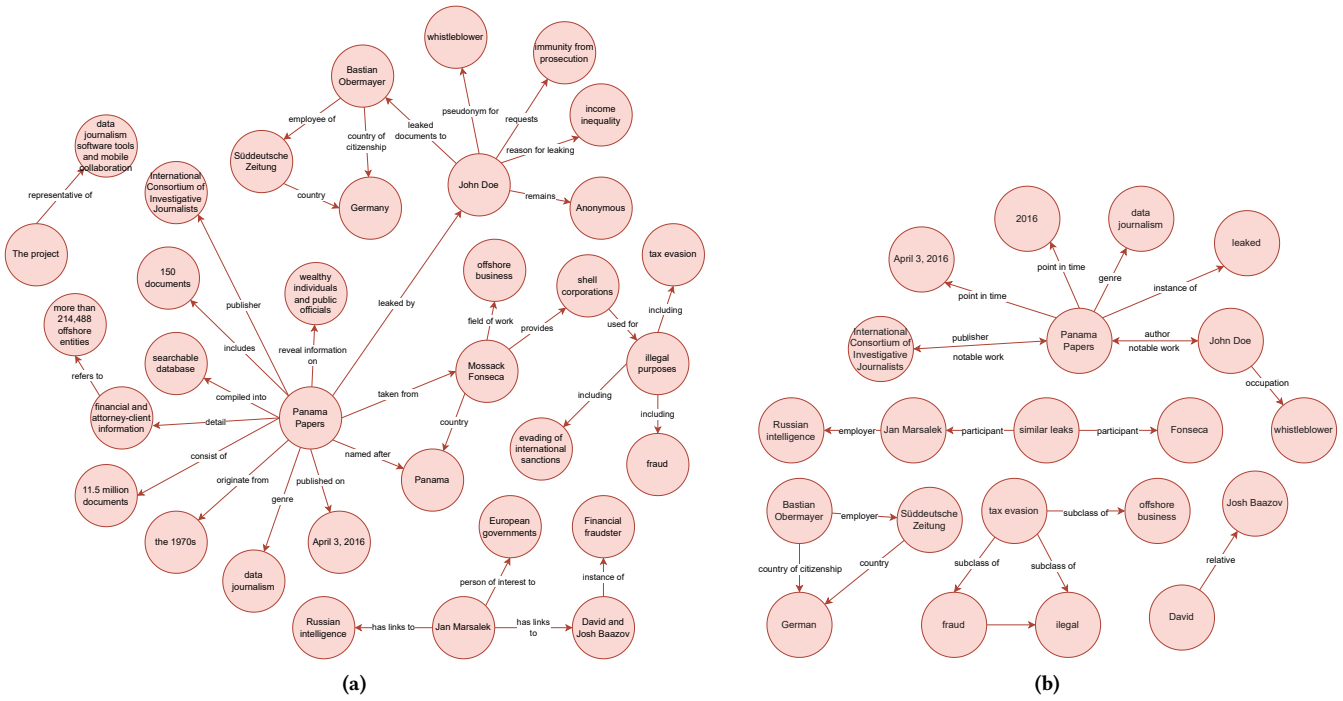


Figure 3: Graph comparison: a) Ground truth and b) Graph with similarity score 81.05%.

By exploring given variations of different parameters, we aim at analyzing sensitivity impact of them on generated graphs. In the table 1 are represented main results in terms of similarity. The table doesn't show systematic way of changing the parameters values because there was a lot of duplication and minor differences. Based on the Panama Papers text, the predefined graph in Fig. 3a consists of 34 nodes and 33 edges. This graph was constructed manually and contains a detailed representation of the data. All graphs were visualized using NetworkX [6].

The analysis was conducted on a system with Intel Core i7-13700K CPU, GeForce RTX 3070 Ti GPU, and 32 GB of RAM.

Upon comparison of the most similar graph (Fig. 3b) and the predefined graph (Fig. 3a), distinct parallels emerge. Both contain pivotal entities such as "Panama Papers", "John Doe", "Bastian Obermayer", and "Jan Marsalek". Other entities are mostly grouped around those main nodes. However, distinctions are evident. The generated graph explores diverse relationship types, featuring categories like "relative" and "subclass of". In contrast, the ground truth graph has more descriptive relationships, including terms such as "originate from", "compiled into", "leaked documents to", "reason for leaking", and "representative of". The predefined graph also contains more detailed entities, for example "11.5 million documents" and provides specific information behind John Doe's leaks ("income inequality"). The generated graph has a broader focus, considering relationships between various entities and their relevance to the Panama Papers. Meanwhile, the predefined graph delves deeper into specific details, like the services "Mossack Fonseca" provides and the nature of "shell corporations".

4 CONCLUSION

Named entity extraction is essential for extracting specific details from large sets of documents, making them clearer and easier to understand. Such extraction techniques can be used in journalism or intelligence systems to automatically identify pivotal events and their associated key entities. The proposed solution is presented on one dataset, but the same methodological approach could be applied to different data. This forms the foundation for the use of optimization algorithms and for selecting optimal hyper-parameters of the graph. The generated graphs, as observed, offer a panoramic view of the textual data and a quick overview, in contrast to predefined graph that delves into specifics.

Future advancements in this domain hold great promise. Considering a potential node coloring based on NER categories and introduction of more detailed relations, which would provide us with more information. For instance, a model trained on medical data, would not perform well on financial data without additional training. Therefore, training models to specific domains can vastly improve extraction accuracy. Ultimately, while current NER systems provide invaluable insights and streamline information processing, continuous improvement and domain-specific adaptations will be the cornerstone for achieving remarkable precision and clarity in the future.

ACKNOWLEDGMENTS

The authors acknowledge joint financial support from the Slovenian Research Agency and Slovenian Ministry of the Interior (target research program No. V2-2260).

REFERENCES

[1] Tareq Al-Moslmi, Marc Gallofré Ocaña, Andreas L. Opdahl, and Csaba Veres. 2020. Named Entity Extraction for Knowledge Graphs: A Literature Overview. *IEEE Access* 8 (2020), 32862–32881. https://doi.org/10.1109/ACCESS.2020.2973928

[2] Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*. Vol. 4. Springer.

[3] Zhenjin Dai, Xutao Wang, Pin Ni, Yuming Li, Gangmin Li, and Xuming Bai. 2019. Named Entity Recognition Using BERT BiLSTM CRF for Chinese Electronic Health Records. In *2019 12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. 1–5. https://doi.org/10.1109/CISP-BMEI48845.2019.8965823

[4] Yoav Goldberg and Omer Levy. 2014. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722* (2014).

[5] Ralph Grishman and Beth Sundheim. 1996. Message Understanding Conference-6: A Brief History. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*. https://aclanthology.org/C96-1079

[6] Aric Hagberg, Pieter J. Swart, and Daniel A. Schult. [n. d.]. Exploring network structure, dynamics, and function using NetworkX. ([n. d.]). https://www.osti.gov/biblio/960616

[7] Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. *To appear* 7, 1 (2017), 411–420.

[8] Anna Huang, David Milne, Eibe Frank, and Ian H Witten. 2008. Clustering documents with active learning using wikipedia. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 839–844.

[9] Pere-Lluís Huguet Cabot and Roberto Navigli. 2021. REBEL: Relation Extraction By End-to-end Language generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*. Association for Computational Linguistics, Punta Cana, Dominican Republic, 2370–2381. https://aclanthology.org/2021.findings-emnlp.204

[10] Nishtha Jatana, Sahil Puri, Mehak Ahuja, Ishita Kathuria, and Dishant Gosain. 2012. A survey and comparison of relational and non-relational database. *International Journal of Engineering Research & Technology* 1, 6 (2012), 1–5.

[11] Lung-Hao Lee, Chao-Yi Chen, Liang-Chih Yu, and Yuen-Hsien Tseng. 2022. Overview of the ROCLING 2022 Shared Task for Chinese Healthcare Named Entity Recognition. In *Proceedings of the 34th Conference on Computational Linguistics and Speech Processing (ROCLING 2022)*. The Association for Computational Linguistics and Chinese Language Processing (ACLCLP), Taipei, Taiwan, 363–368. https://aclanthology.org/2022.rocling-1.46

[12] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. (2019).

[13] Stefano Marchesin, Laura Menotti, Fabio Giachelle, Gianmaria Silvello, and Omar Alonso. 2023. Building a Large Gene Expression-Cancer Knowledge Base with Limited Human Annotations. *Database* (2023), 1–19.

[14] Jim Melton and Alan R Simon. 1993. *Understanding the new SQL: a complete guide*. Morgan Kaufmann.

[15] Vincenzo Moscato, Giuseppe Napolano, Marco Postiglione, and Giancarlo Sperlì. 2023. Multi-task learning for few-shot biomedical relation extraction. *Artificial Intelligence Review* (2023), 1–21.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[17] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. 2020. Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*. ACM, 3125–3132.

[18] Christof Strauch, Ultra-Large Scale Sites, and Walter Kriha. 2011. NoSQL databases. *Lecture Notes, Stuttgart Media University* 20, 24 (2011), 79.

[19] Milena Trajanoska, Riste Stojanov, and Dimitar Trajanov. 2023. Enhancing Knowledge Graph Construction Using Large Language Models. *arXiv preprint arXiv:2305.04676* (2023).

[20] Aleksa Vukotic, Nicki Watt, Tareq Abedrabbo, Dominic Fox, and Jonas Partner. 2015. *Neo4j in action*. Vol. 22. Manning Shelter Island.

[21] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).

Human-assisted reinforcement learning demonstrated on the Flappy Bird Game

Jana Ristovska
89191025@student.upr.si
Faculty of Mathematics, Natural
Sciences and Information Technologies
University of Primorska
Glagoljaška ulica 8
SI-6000 Koper, Slovenia

Domen Šoberl
domen.soberl@famnit.upr.si
Faculty of Mathematics, Natural
Sciences and Information Technologies
University of Primorska
Glagoljaška ulica 8
SI-6000 Koper, Slovenia

ABSTRACT

In model-free reinforcement learning, the agent usually starts learning by blind exploration, which can take a significant amount of time before starting to experience positive reinforcement that drives further progress. In this paper, we address the following question: Can the learning time be reduced if the agent first observes successful behaviors demonstrated by humans, before commencing its independent learning? We propose an adaptation of the traditional Q-learning algorithm, so that it can gradually integrate recorded demonstrations into the learning process, and demonstrate this method on the well-known Flappy Bird game. We recorded 1496 gameplays of Flappy Bird played by 22 volunteers, and selected 941 successful recordings to assist the Q-learning algorithm. The results show that such human assistance speeds up the learning process in a logarithmic manner, which means that the biggest gain is made in the initial stages of learning and becomes almost negligible later on. We show experimentally that in the initial stages of learning, human demonstrations contain more useful information than the agent can acquire independently.

KEYWORDS

Reinforcement learning, Q-learning, Learning by demonstration, Imitation learning

1 INTRODUCTION

Using reinforcement learning [9], computers can learn, by trial and error, how to play simple video games. The learning process typically takes several hours, after which the computer can reach, or even surpass the human level of playing [6]. However, an interesting question arises: Can the learning time be reduced if the computer is given the opportunity to observe a number of successfully played games by humans, before commencing its independent learning? Incorporating player gameplay data has already been considered as a potential way to enhance the learning process in the domain of video games. In this paper, we aim to research possible improvements in the training time of the traditional Q-learning algorithm when integrating into the learning process a successfully played game by humans. We demonstrate our approach on the well-known Flappy Bird Game. The Flappy Bird Game serves as a suitable platform to investigate this research question due to its simplicity and well-defined gameplay mechanics. We study if and how the training improves with the quality of the recorded playthroughs i.e. the number of scores reached by the human. Throughout the paper, we will use the term *human-assisted reinforcement learning* to label any approach in the field of machine learning, specifically reinforcement learning, where human demonstrations are incorporated into the learning process to aid the reinforcement learning agent's decision-making, although similar ideas have been tried under various names.

2 RELATED WORK

Reinforcement learning has garnered significant interest in the domain of playing video games. Traditional approaches like Q-learning and Deep Q-Networks (DQNs) have demonstrated the ability to train agents to achieve superhuman performance in various games. More recent advancements, such as Proximal Policy Optimization (PPO) [8] and Asynchronous Advantage Actor-Critic (A3C) [2], have shown improved stability and sample efficiency. These algorithms have been employed to train agents in a range of games, including Atari 2600 games, Gymnasium environments, and custom game simulations [6, 9].

The concept of human-assisted reinforcement learning, where agents learn from demonstrations provided by humans, has gained traction in recent research. One approach in this research area is called "Imitation Learning" or "Behavioral Cloning", where the agent tries to mimic the expert's behavior directly [11]. However, this approach can suffer from issues like compounding errors, resulting in sub-optimal performance.

Another prominent approach is "Inverse Reinforcement Learning" (IRL), where the agent attempts to infer the underlying reward function from expert demonstrations [1]. This allows the agent to learn the task's structure without requiring exact expert actions. However, Inverse Reinforcement Learning can be challenging due to the need to model the reward function accurately.

Researchers have applied human-assisted reinforcement learning to various tasks, such as robotic manipulation [12], autonomous driving [4, 13], and game playing [8]. Studies have shown that leveraging human demonstrations can lead to faster convergence and better performance compared to training from scratch [3, 8]. In the context of video games, human demonstrations have been used to teach agents to play games like Super Mario Bros, Dota 2, and Montezuma's Revenge, showing promising results in reducing learning time and improving performance [5]. Similar research employing learning from demonstrations in combination with transfer learning has been showcased in a soccer simulator called Keepaway [10], investigating the effects of both the quality of recorded

DOI: https://doi.org/10.26493/scores23.13

demonstrations and the integration of demonstrations from multiple instructors.

3 IMPLEMENTATION

As the training environment, we adapted an existing Flappy Bird implementation, which was written in Python and released under the MIT license [7]. This open-source implementation offers a simplified version of the classic Flappy Bird game, to which we added a state discretization feature and a reward function. The state is composed of the horizontal distance from the bird to the right side of the closest lower pipe and the vertical distance from the bird to the top side of the closest lower pipe. We discretize the state by splitting the horizontal distance into 10 and the vertical distance into 25 equidistant intervals. This way we obtain 250 discrete states, which we uniquely map into their integer representations $\{0, \ldots, 249\}$. The rewarding system returns the negative reward of -1000 when the bird dies (which also terminates the episode), and the positive reward of +10 points at every step when the bird stays alive. This reward system was influenced by similar research in the field.

We implemented two types of Q-learning algorithms: the traditional Q-learning algorithm with discrete states and actions to serve as the performance baseline, and our proposed human-assisted Q-learning algorithm that can integrate recorded gameplay data into the learning process.

Figure 1 shows the data flow with the traditional (baseline) Q-learning approach. Based on the current state, as received from the environment, the agent selects and executes the most prominent action according to the current Q-table policy. When the action is executed, a new state (state') is received from the environment, together with the obtained reward. The previous state, the executed action, the reward, and the new state are then used to update the Q-table using the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) \quad \alpha \cdot \left(r \quad \gamma \cdot \max_{a'} Q(s',a') \right), \tag{1}$$

where s is the current state, a the executed action in state s, r the received reward, s' the next state, a' a future action, α the learning rate and γ the discount factor. We set the learning rate α to 0.45, and the discount factor γ to 0.9. We selected these values through meticulous fine-tuning, aiming to achieve the best training results we could.

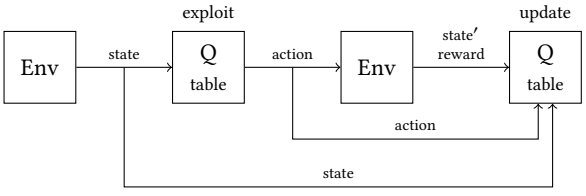


Figure 1: The learning process of the baseline Q-learning algorithm.

The human-assisted Q-learning algorithm that we propose in this paper works as shown in Figure 2. We use two separate Q-tables to implement the policy that is used by the intelligent agent during the Q-learning process: the (trainable) Q-table Q, which is the same as the Q-table used by the traditional Q-learning algorithm, and the user Q-table Q_U, which is populated with the recorded human gameplays.

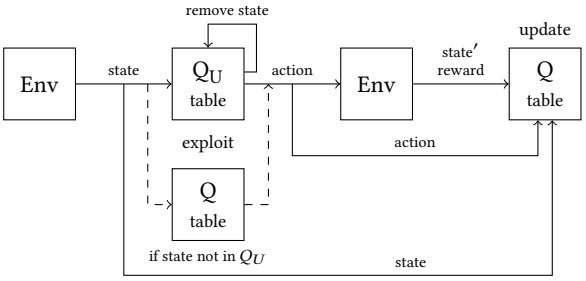


Figure 2: The learning process of our human-assisted Q-learning algorithm.

The process of populating the Q_U table is as follows: For every step of the recorded game, we discretize the game state as described above, read the action that the user made in that state, the obtained reward and the consequent state, which we also discretize. We then compute the $Q(s,a)$ value similarly as in the online training using Equation 1, however, there is no interaction with the environment in the process. The computed $Q(s,a)$ value is inserted into the Q_U table at the $[s,a]$ position. Because the computation relies on the existing Q_U entries for s' and a', the process is repeated two times for the same recording. Our experiments showed that doing more than two passes does not improve the training performance.

During the online training, the Q_U table is utilized as follows: If an entry for the current state s exists in the Q_U table, the action is decided based on this entry, after which the entry for state s is removed from Q_U. If there is no entry for s in Q_U, table Q is used as with the traditional Q-learning algorithm. We may say that the knowledge from the recorded human gameplay is gradually being integrated into the learned policy. When the executed action is based on the values from the user Q-table Q_U, the agent has followed the human's strategy. Presuming that the human player played a feasible strategy, such a choice should indicate a certain advantage over blind exploration.

4 OBTAINING HUMAN GAMEPLAY DATA

In order to obtain a pool of recorded gameplays, we got 22 volunteers to play the game of Flappy Bird. They were instructed to play the game as many times as they want and to aim to score as many points as they can. They played 1496 games in total, 555 of which scored 0. Since our methodology is based on providing successful human demonstrations, we only used the recordings with at least 1 point reached, which was 941 gameplays. The maximum score reached was 53 and the second best was 33. The distribution of all the recorded gameplays is shown in Figure 3.

A gameplay recording is composed of a tuple (h, v, a, r, s) for each consecutive game step. Variables h and v represent the horizontal and the vertical distances in pixels from the next pipe. Values a, r and s respectively represent the executed action, the obtained reward and the current score. The resulting state can be obtained from the subsequent tuple in the recording.

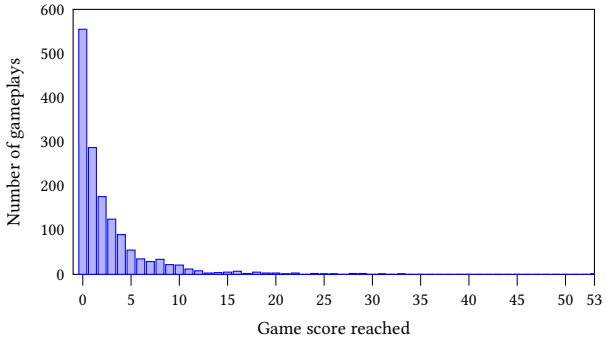


Figure 3: Distribution of the recorded human gameplays according to the reached game score.

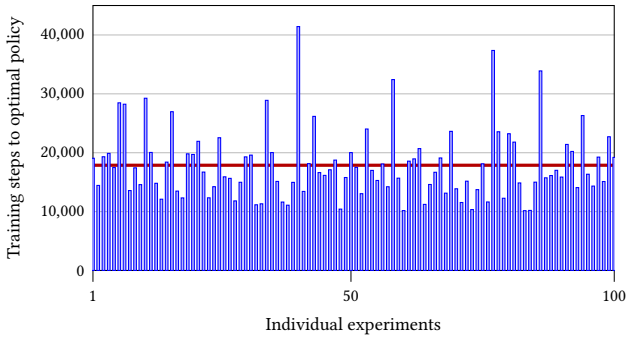


Figure 4: The results of 100 experiments using the traditional Q-learning algorithm.

5 EXPERIMENTAL RESULTS

In reinforcement learning, the usual approach to measuring the performance is to count the number of episodes over which the learning curve converges. In our case, an episode ends only with the bird's death, so the episodes vary in length considerably. Moreover, after the agent learns how to play optimally, an infinitely long episode follows until terminated manually. Therefore, as a measure of performance, we count the number of training steps needed for the agent to reach a specific score, after which the training is concluded. The initial experiments showed that the agent always learned an optimal policy before reaching 60 points, therefore we set the training goal to reaching 100 points.

We conducted a single experiment as follows. We ran the learning algorithm and let it play for as many training episodes as needed, until the bird scored 100 points. Afterward, the training was stopped and we recorded the total number of training steps taken during all the episodes together.

5.1 Traditional Q-learning

We conducted 100 independent experiments using our implementation of the traditional Q-learning algorithm, which we took as the baseline method. The average number of training steps to learn an optimal policy was 17872.18, with the standard deviation of 5912.22. The maximum number of training steps was 41426 and the minimum was 10158. To put this into perspective, with the frequency of 60 frames (actions) per second, the average training time took 296.4 seconds, which is 4 minutes and 56.4 seconds. Individual results are shown in Figure 4, where the red line denotes the average number of training steps. Different outcomes are due to the fact that the pipes are generated randomly by the environment.

5.2 Human-assisted Q-learning

When assisting the learning algorithm with a single recorded human gameplay, the duration of that gameplay is of crucial importance. Not only more data is provided by a longer gameplay, the playing skills and therefore the employed strategy might also be better, since the player managed to 'survive' longer. In our case, the duration of the game is strongly correlated with the final score achieved in that game, since the speed of the bird is constant and the pipes are spaced equally. We therefore conducted the experiments as follows: We grouped the recorded gameplays according to their duration — gameplays with the total score of 1 were put into the first group, gameplays with the total score of 2 into the second group, etc. This way we formed 30 groups of recorded gameplays with the same distribution as shown in Figure 3 (note that for some scores in the range of 1 to 53, no gameplay was recorded). We then ran the learning algorithm independently 100 times for each group, each time selecting randomly a gameplay from the current group to assist in training. We measured how much the training time decreased on average for each group in relation to the average training time of the baseline learning algorithm. We were interested in whether and in which cases the decrease in the training time would be greater than the duration of the assisted human gameplay.

The left plot in Figure 5 shows the average decrease in training time for all 30 gameplay groups, which are denoted with red dots. Gameplay durations are averages for each group, although variations within a group are negligible (up to the amount of time needed to travel between two adjacent pipes). The improvement increases with the duration of the used gameplay recording in a near-logarithmic manner. We can notice that the improvement rate slows down significantly when the duration of the gameplay exceeds 30 seconds. This indicates that the majority of the human skill is encoded into the first 30 seconds of gameplay, after which the same strategy is very likely repeated, with less probability of novelty in the recorded data.

The right plot in Figure 5 shows the same data from a different perspective. Instead of the absolute reduction in the training time, the difference between the reduced time and the average duration of the used recordings is shown along the y-axis. A positive value indicates that the training time was reduced more than the duration of the recording. This means that there was more useful information in the recorded gameplay than could be obtained by independent training in the same amount of time. Shorter recordings show a higher ratio, the highest being with the first group of recordings, where the human player managed to pass only one obstacle and lasted for about 3 seconds. The ratio is dropping with longer games and reaches 0 at about 15 seconds mark.

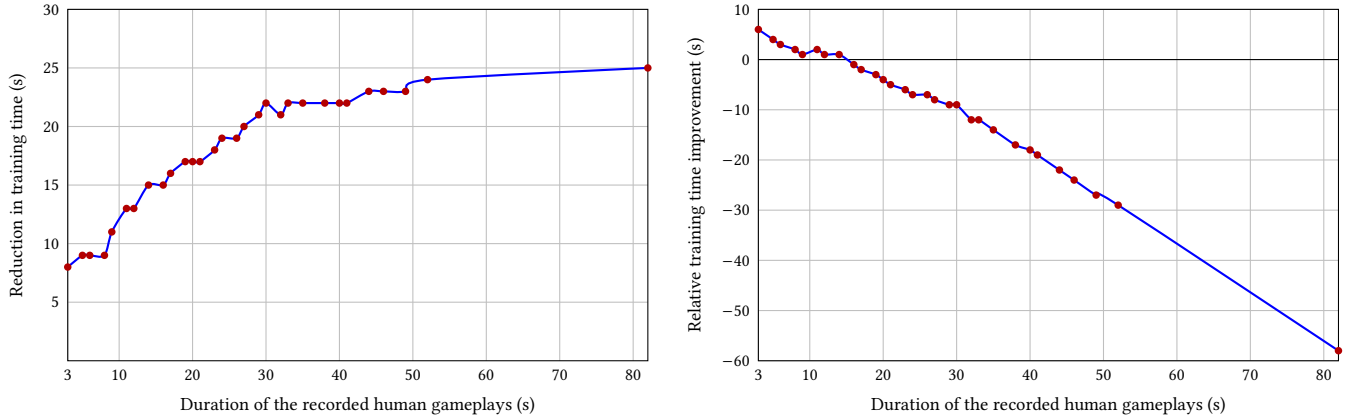


Figure 5: Improvement in the training time when using human gameplay recordings of different durations.

6 CONCLUSION

The results of our study demonstrate the potential of human-assisted reinforcement learning in improving the performance of training an agent. The main premise of this paper is that a human demonstration of a successful playing strategy contains more useful information than an agent can in the same amount of time obtain independently. We verified this idea on the well-known Flappy Bird game by first implementing a traditional Q-learning algorithm, which was able to learn an optimal strategy in about 5 minutes on average. We then proposed an adaptation to the traditional Q-learning algorithm that can integrate gameplay recordings into the learning process. We obtained 941 useful gameplay recordings from 22 volunteers that we used with our adapted learning algorithm and compared the reduction of the training time in comparison to the traditional Q-learning approach.

The results confirmed our hypothesis and showed that the most useful information regarding a successful Flappy Bird playing strategy is encoded in the first 30 seconds of human gameplay, after which the improvement still somewhat increases, but at an insignificant rate. We may conclude that in those first 30 seconds, a human player encounters most of the possible situations, after which it mostly repeats already seen playing maneuvers.

Such absolute training time improvement alone may suffice in cases where offline data is abundant and online training is expensive. However, to assess whether a human demonstration actually contains more information than blind exploration, we analyzed relative improvement in the training time, which showed that on average, the first 15 seconds of human demonstration provided more useful information than could independently be obtained by the agent.

The main limitation of our research is the simplicity of the chosen training environment. After discretization, we obtained a state space spanning 250 possible states, which can fairly quickly be explored by the traditional Q-learning algorithm, as we have also demonstrated. This leaves little room for improvement through human assistance. We believe that in more complex domains, where it takes an agent hours or even days to train, human assistance could prove much more beneficial. Moreover so when the reward is sparse and the agent must rely on pure luck to obtain the reward for the first time. Human demonstrations of obtaining a reward could significantly speed up this initial part of the training.

Our approach is currently limited to discrete states and actions, which is the limitation of the traditional Q-learning algorithm with tabulated Q-values. As a part of our future work, we aim to move towards continuous environments and explore the possibilities to integrate human assistance into modern deep Q-learning architectures. This way we may be able to tackle more complex environments and even real-world control problems.

REFERENCES

[1] Pieter Abbeel and Andrew Y Ng. 2004. Apprenticeship Learning by Inverse Reinforcement Learning. In *Proceedings of the Twenty-First International Conference on Machine Learning (ICML-2004)* (Banff, Alberta, Canada). ACM, New York, NY, USA, 1–8.

[2] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz. 2016. Reinforcement Learning through Asynchronous Advantage Actor-Critic on a GPU.

[3] P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei. 2018. Deep reinforcement learning from human preferences. arXiv:1706.03741 [stat.ML]

[4] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy. 2018. End-to-end Driving via Conditional Imitation Learning.

[5] J. Leike, M. Martic, and S. Legg. 2017. Learning through human feedback. DeepMind Blog. https://www.deepmind.com/blog/learning-through-human-feedback

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning.

[7] Russs123. 2020. Flappy Bird. https://github.com/russs123/flappy_bird.

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG]

[9] R. S. Sutton and A. G. Barto. 2018. *Reinforcement learning: An introduction* (2 ed.). The MIT Press, Cambridge, MA, USA.

[10] M. E. Taylor, H. B. Suay, and S. Chernova. 2011. Integrating reinforcement learning with human demonstrations of varying ability. In *10th International Conference on Autonomous Agents and Multiagent Systems 2011, AAMAS 2011*, Vol. 1. FAAMAS, Richland, SC, 617–624.

[11] F. Torabi, G. Warnell, and P. Stone. 2019. Recent Advances in Imitation Learning from Observation. arXiv:1905.13566 [cs.RO]

[12] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller. 2017. Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards. arXiv:1707.08817 [cs.RO]

[13] J. Wu, Z. Huang, C. Huang, Z. Hu, P. Hang, Y. Xing, and C. Lv. 2021. Human-in-the-Loop Deep Reinforcement Learning with Application to Autonomous Driving. arXiv:2104.07246 [cs.RO]

Index of Authors