Proceedings of the
11th Student Computing Research Symposium
(SCORES'25)

Ljubljana, Slovenia
September 30, 2025

Uroš Čibej
Luka Fürst
Lovro Šubelj
Jure Žabkar
(Eds.)

SCORES

https://www.scores.si

UNIVERSITY OF LJUBLJANA
FRI
Faculty of Computer and Information Science

FRI
UNIVERSITY OF LJUBLJANA
Faculty of Computer and Information Science

Proceedings of the 11th Student Computing Research Symposium (SCORES'25)

Ljubljana, Slovenia
September 30, 2025

Uroš Čibej
Luka Fürst
Lovro Šubelj
Jure Žabkar
(Eds.)

https://www.scores.si

Title	**Proceedings of the 11th Student Computing Research Symposium (SCORES'25)**
Editors	Uroš Čibej (University of Ljubljana, Faculty of Computer and Information Science) Luka Fürst (University of Ljubljana, Faculty of Computer and Information Science) Lovro Šubelj (University of Ljubljana, Faculty of Computer and Information Science) Jure Žabkar (University of Ljubljana, Faculty of Computer and Information Science)
Technical editor	Luka Fürst (University of Ljubljana, Faculty of Computer and Information Science)
Design	University of Ljubljana, Faculty of Computer and Information Science University of Maribor, Faculty of Electrical Engineering and Computer Science University of Primorska, Faculty of Mathematics, Natural Sciences and Information Technologies
Graphic material	Sources are own unless otherwise noted. The authors and Čibej, Fürst, Šubelj, Žabkar (editors), 2025
Conference	**11th Student Computing Research Symposium (SCORES'25)**
Location	University of Ljubljana Faculty of Computer and Information Science Večna pot 113, SI-1000 Ljubljana, Slovenia
Date	September 30, 2025
Program Committee	Klemen Berkovič (University of Maribor) Marko Bizjak (University of Maribor) Lucija Brezočnik (University of Maribor) Andrej Brodnik (University of Ljubljana) Uroš Čibej (University of Ljubljana) Jani Dugonik (University of Maribor) Iztok Fister Jr. (University of Maribor) Luka Fürst (University of Ljubljana) Magda Gregorova (University of Applied Sciences Würzburg-Schweinfurt) Štefan Kohek (University of Maribor) Niko Lukač (University of Maribor) Eneko Osaba (Tecnalia Research & Innovation) Jan Popič (University of Maribor) Peter Rogelj (University of Primorska) Domen Šoberl (University of Primorska) Lovro Šubelj (University of Ljubljana) Grega Vrbančič (University of Maribor) Jure Žabkar (University of Ljubljana)

Organizing Committee
Uroš Čibej (University of Ljubljana)
Luka Fürst (University of Ljubljana)
Lovro Šubelj (University of Ljubljana)
Jure Žabkar (University of Ljubljana)

Published by
Založba UL FRI
University of Ljubljana
Faculty of Computer and Information Science
Večna pot 113, SI-1000 Ljubljana, Slovenia
https://zalozba.fri.uni-lj.si/, zalozba@fri.uni-lj.si

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

University of Primorska
Faculty of Mathematics, Natural Sciences and Information Technologies
Glagoljaška ulica 8, SI-6000 Koper, Slovenia
https://www.famnit.upr.si/en, info@famnit.upr.si

Edition 1st

Publication type E-book

Published Ljubljana, Slovenia, December 2025

ISBN 978-961-7059-18-2

DOI https://doi.org/10.51939/scores25

Available at https://zalozba.fri.uni-lj.si/SCORES2025.pdf

Organizers and sponsors

© **University of Ljubljana, Faculty of Computer and Information Science**

Text © Authors & Čibej, Fürst, Šubelj, Žabkar (editors), 2025

This book is published under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). This license allows reusers to distribute, remix, adapt, and build upon the material for any purpose, even commercially, as long as attribution is given to the creator and the new creations are licensed under identical terms.

Any third-party material in this book is published under the book's Creative Commons licence unless indicated otherwise in the credit line to the material. If you would like to reuse any third-party material not covered by the book's Creative Commons licence, you will need to obtain permission directly from the copyright holder.

https://creativecommons.org/licenses/by-sa/4.0/

Price Free copy

For publisher Boštjan Slivnik, University of Ljubljana, Faculty of Computer and Information Science

Attribution Čibej, U., Fürst, L., Šubelj, L., Žabkar, J. (eds.) (2025). *Proceedings of the 11th Student Computing Research Symposium (SCORES'25).* University of Ljubljana, Faculty of Computer and Information Science. DOI: 10.51939/scores25

Kataložni zapis o publikaciji (CIP) pripravili v Narodni in univerzitetni knjižnici v Ljubljani

COBISS.SI-ID 261672195

ISBN 978-961-7059-18-2 (PDF)

Editors' Foreword

The annual Student Computing Research Symposium (SCORES), formerly known as StuCoSReC, is one of the few regular scientific gatherings explicitly targeting the student population, at least in the area of computer science. The computer science departments of the three public Slovenian universities — the University of Ljubljana, the University of Maribor, and the University of Primorska — take turns organizing this challenging but rewarding event. The 2025 edition of the symposium, SCORES'25, was prepared by and held at the University of Ljubljana, Faculty of Computer and Information Science.

At SCORES'25, we accepted 14 research papers dealing with diverse areas of computer science. With artificial intelligence booming and blooming, it is no surprise that a sizable number of contributions attempt to advance this field or employ it to advance other scientific fields. In particular, Kržan and Žabkar use machine learning methods to train a model for predicting the severity of Parkinson's disease; Loknar and Mlakar learn to distinguish apples from cherries; Šprajc *et al.* identify vineyards based on LiDAR data; Sekirnik *et al.* apply artificial intelligence approaches to 3D rendering; Stojanovič and Pucer predict missing temperature data; Stražišar *et al.* use machine learning to make contribution to chemistry; and, finally, Žabkar *et al.*, as well as Ferreira and Pucer, have found out that artificial intelligence can also be applied to more pleasurable aspects of human existence, such as playing games.

Despite the preponderance of papers related to artificial intelligence, machine learning, and pattern recognition, SCORES'25 showed that the more 'traditional' computer science fields, such as algorithmics, offer just as many opportunities to discover something new. Coincidentally, Škrbec and Fürst and Beber *et al.* tackle two distinct but highly related algorithmic problems: geometric knapsack and bin covering. The papers by Stojanović and Pangeršič and by Jug and Novak Domijan both contribute to graph theory, with the first addressing the mutual visibility problem, and the second dealing with the hitting set problem. The contributions by Anžlovar and Derganc and Urbanč apply computer science to pure mathematics. Anžlovar extends Julia sets to four dimensions and renders them in three, while Derganc and Urbanč try to numerically compute a mathematical constant for which only rough bounds were known before.

The list of people who helped make this event happen includes, but is not limited to, the members of the program committee, who produced high-quality reviews of the papers; the staff of the Faculty of Computer and Information Science in general, and the PR office in particular, who took care of all the items and services that we usually take for granted at such events but may in fact require hours of extra effort; Erik Štrumbelj and the Faculty's executive committee, who generously provided us with more than sufficient funds to organize the event; Mila Marinković, the invited speaker, who gave an engaging talk about her endeavors in applying artificial intelligence to medicine; and, last but not least, the conference attendees, who had not only enough patience to survive from the beginning to the end of the symposium but also sufficient knowledge and courage to ask challenging questions, making the event even more stimulating.

Student-centered scientific conferences are important not only because they bring together students to present and discuss challenging problems and innovative solutions but also because they may serve as a stepping stone to their future career in science. Quite a few Slovenian computing scientists, Mila from the preceding paragraph being one example, actually started their professional path by submitting a paper to SCORES. We are happy to be part of their stories, and we hope that we have just encouraged a few more people to join the ranks of scientists in the near future. With that in mind, we gladly pass the torch to the University of Primorska, the organizers of SCORES'26.

Uroš Čibej
Luka Fürst
Lovro Šubelj
Jure Žabkar

Conference Program

Session 1 **1**

1 Raymarching 3D Julia Sets
Andrej Anžlovar

5 Metaheuristic Approaches for Solving the Geometric Knapsack Problem
Nejc Škrbec, Luka Fürst

9 Empirical Analysis Of Heuristic and Approximation Algorithms For The Mutual-Visibility Problem
Vanja Stojanović, Bor Pangeršič

13 Randomness in Bin Covering
Jakob Beber, Andrej Brodnik, Rok Požar

17 Efficacy of Reduction Rules for Hitting Set Problem
Eva Jug, Jan Novak Domijan

Session 2 **21**

21 A Model for Evaluating Parkinson’s Disease Severity Through the Finger-Tapping Test
Neža Kržan, Jure Žabkar

27 Comprehensive Multi-Species Fruit Ripeness Dataset Construction: From Eight-Species Collection to Focused Apple-Cherry Detection
Maksim Loknar, Uroš Mlakar

35 Identifying Vineyards from LiDAR Data
Žan Tomaž Šprajc, Marko Bizjak, David Jesenko

39 Comparing Dimensionality Reduction Methods for Predicting Skin Deformations in 3D Models
Rok Sekirnik, Marko Bizjak, David Jesenko

43 Prompts or Policies: Which Plays Yamb Better?
Anton Žabkar, Vid Kališnik, Jure Žabkar

Session 3 **47**

47 Numeric Computations of Landau’s Constant
Matija Derganc, Luka Urbanč

51 Quantifying Player Death Impact in League of Legends
Ruben Ferreira and Jana Faganeli Pucer

55 Temperature Imputation using Graph Neural Networks
Jan Jure Stojanovič and Jana Faganeli Pucer

59 Napovedovanje optičnih lastnosti spojin za detekcijo fibril amiloida β s strojnim učenjem
Luka Stražišar, Matic Rogan, Janez Košmrlj, Jerneja Kladnik, Matevž Pesek

Index of Authors **63**

Proc. of the 11th Student Computing Research Symposium (SCORES'25), Ljubljana, Slovenia, September 30, 2025

Raymarching 3D Julia Sets

Andrej Anžlovar
aa24235@student.uni-lj.si
Faculty of Mathematics and Physics,
University of Ljubljana
Jadranska ulica 19
SI-1000 Ljubljana, Slovenia

ABSTRACT

This paper explores the technique of raymarching to render 3D Julia sets – fractal structures formed from iterative mapping of functions in quaternion space, which are then restricted to only three dimensions. Raymarching is a powerful tool well-suited for rendering mathematically simple but visually complex structures such as fractals. Unlike traditional methods, objects are depicted as signed distance fields, which allows for incredible levels of detail and accuracy. In this paper, we delve into the basics of the raymarching algorithm, some of its advantages and the calculation of a Julia set distance estimator required for raymarching.

KEYWORDS

raymarching, signed distance bound, signed distance field, sphere tracing, filled-in Julia set, distance estimator

1 INTRODUCTION

Fractals are both visually and mathematically some of the most beautiful and intriguing works one can imagine. Julia sets, being one of the most well-known and interesting family of fractals, are of course no exception to this. However, they are defined in and limited to the complex numbers, a two-dimensional set. That, of course, would make some wonder if something similar can be defined and visualized in higher dimensions, for example in three-dimensional space that we are most familiar with.

Further on, we show how the definition of Julia sets can be extended into four dimensions using quaternions and then restricted into three-dimensional space. However, these new structures are usually too complex to render using the normal means of rasterization. This is solved by instead relying on a method known as raymarching. By expressing objects not as collections of points but rather signed distance fields, raymarching gains numerous advantages over regular rasterization. For this purpose we show how to calculate a distance estimator for our three-dimensional Julia sets.

Let us first take a look at the basics of raymarching to understand what exactly is required for rendering 3D Julia sets.

2 RAYMARCHING

To render objects realistically the simplest way would be by replicating what happens in our physical world. That is to simulate rays of light hitting our object and reflecting into our eyes. But to do so, it is first necessary to know if and where exactly our rays hit the rendering target. Raymarching is just one way of computing that for objects represented with signed distance fields.

2.1 Signed distance fields

Signed distance fields are another way of representing sets in $\mathbb{R}^3$. Let $f\colon \mathbb{R}^3 \to \mathbb{R}$ be a scalar field. As mentioned in [3] and [4] f implicitly describes a set of points $A = \left\{\vec{x} \in \mathbb{R}^3 \mid f(\vec{x}) \leq 0\right\}$. With the additional conditions of $f(\vec{x}) < 0$ for all $\vec{x} \in \operatorname{Int} A$, and continuity it follows that $f^{-1}(\{0\}) = \partial A$, where $\operatorname{Int} A$ denotes the interior of A and ∂A the edge of A. This means the surface ∂A can be expressed as an implicit collection of points $\vec{x} \in \mathbb{R}^3$ that satisfy $f(\vec{x}) = 0$. Then we can properly define the signed distance field using the point-to-set distance like it can be seen in [4].

Definition 1 (point-to-set distance)**.** Let $A \subseteq \mathbb{R}^3$ be a set. The *point-to-set distance* for the set A is a function $d(\cdot, A) : \mathbb{R}^3 \to [0, \infty)$, defined as the distance from a point to its closest point in A,

$$d(\vec{x}, A) = \inf_{y \in A} ||x - y|| .$$

Definition 2 (signed distance bound/field)**.** Let $f\colon \mathbb{R}^3 \to \mathbb{R}$ be a continuous scalar field and let $A = \left\{\vec{x} \in \mathbb{R}^3 \mid f(\vec{x}) \leq 0\right\}$. Let $f(\vec{x}) < 0$ for all $\vec{x} \in \operatorname{Int} A$. Then f is a *signed distance bound* of its implicit surface $f^{-1}(\{0\}) = \partial A$ if and only if for all $\vec{x} \in \mathbb{R}^3$

$$|f(\vec{x})| \leq d(\vec{x}, f^{-1}(\{0\}))$$

and if instead the equality holds, f is called a *signed distance field.*

Generally, this means that a signed distance field f is defined in the following way:

$$f(\vec{x}) = \begin{cases} -d(\vec{x}, \partial A) & \mid \vec{x} \in \operatorname{Int} A \\ 0 & \mid \vec{x} \in \partial A \\ d(\vec{x}, \partial A) & \mid \vec{x} \in \operatorname{Ext} A \end{cases}$$

where $\operatorname{Int} A$, ∂A and $\operatorname{Ext} A$ denote the interior, edge and exterior of A respectively. As an example, using this to find the signed distance field for the unit sphere $\mathbb{S}^2$, it is easy to check that it is exactly $f(\vec{x}) = ||x|| - 1$.

Now let us take a look at how to use signed distance fields to actually render an implicit surface.

2.2 Sphere tracing

Until the end of this section, we will assume that we are trying to render an implicit surface S with its signed distance field $f\colon \mathbb{R}^3 \to \mathbb{R}$. Say we have a camera at the position $\vec{c}$, facing in the direction of the unit vector $\vec{d}$. We want to simulate the rays that come into our camera from the direction $\vec{d}$. The simplest way to do so is actually to reverse our thinking and cast the rays from our camera, computing where and how they hit S. To simplify this even further, for now, we will assume that light is coming from all directions equally, and we will ignore any reflections. Our goal is then only to find out if a

---

https://doi.org/10.51939/scores25.1

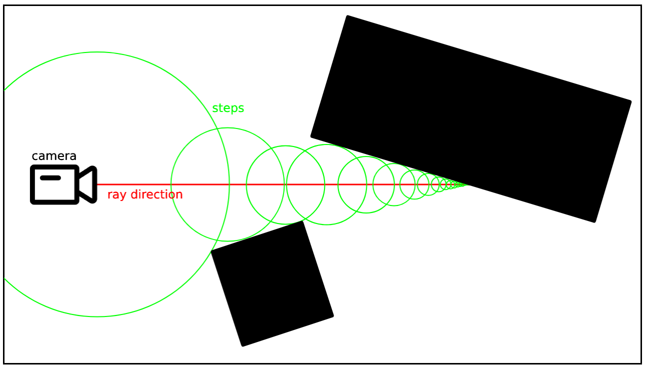


Figure 1: A simple showcase of sphere tracing.

ray from our camera at $\vec{c}$, going in the direction $\vec{d}$ would intersect S or not.

Computing this intersection is where sphere tracing comes in. The idea is as follows, we will step along the ray until we either reach S or stray too far. Let $\vec{x}$ be a variable denoting our current step position. We start with $\vec{x} = \vec{c}$ and by the definition of the point-to-set distance, the open ball with radius $t = f(\vec{x})$ does not intersect S. This means we can safely move along the ray by setting $\vec{x} = \vec{x} + t\vec{d}$ without hitting S in-between. We can now repeat this process again using the new $\vec{x}$ as the starting point Iterating this, we obtain one of two possible outcomes, either we come incredibly close to S at some point – close enough to count as a hit – or we miss S entirely. This is illustrated in algorithm 1 and figure 1.

```
Data: $D_{max}$ - maximum distance from camera, $\epsilon$ - margin of
      error considered a hit
Result: check if ray of origin $\vec{c}$ and direction $\vec{d}$ hits $S$
$t = 0$;
$d = 0$;
while $t < D_{max}$ do
    $d = f\left(\vec{c} + t\vec{d}\right)$;
    if $d < \epsilon$ then
        object was hit at $\vec{c} + t\vec{d}$;
    end
    $t = t + d$;
end
object was not hit;
```

Algorithm 1: Basic sphere tracing algorithm.

Note that, this algorithm uses our signed distance field only to ensure S is not within some open ball around a certain point. That means that replacing the signed distance field with a signed distance bound would still maintain the functionality of this algorithm, albeit making it less efficient.

This is enough to start rendering images of our signed distance field surfaces, but first how do we render a virtual image by just simulating rays? Let us imagine that slightly in front of the camera is a screen with exactly our output resolution, as if a camera was a person sitting behind a monitor. We can cast a ray from our camera in the direction of each of the pixels on this screen and check whether the ray intersects our surface S. If it does, we colour the pixel a single colour, otherwise, a different colour.

Currently, the objects we would render would not appear three-dimensional, as we have not applied any shading. This is a consequence of the simplification we made earlier, that light is coming from all directions equally. Using the methods described in [9], we can achieve more realistic lighting by computing diffusion, specular highlighting, shadows and ambient occlusion. We will not dive deeper into the algorithms and maths behind these as we could write an entire paper on these alone. But it is worth mentioning that lighting computation requires surface normals, so let us take a look at how to obtain them solely relying on the signed distance field f.

2.3 Surface normals

In general, our signed distance field f is non-differentiable, meaning the surface normal is not well-defined. However, surface normals can still be approximated. As is mentioned in [2], computing the approximation of the gradient of f works well in calculating the surface normals of S. Inigo Quillez in [8] gives us algorithm 2 for computing them.

```
Data: $\vec{x}$ - point on $S$ (the last point calculated in the sphere
      tracing algorithm 1), $h$ - a tiny increment
Result: surface normal of $S$ at $\vec{x}$
$n_x = f(\vec{x} + (h,0,0)) - f(\vec{x} - (h,0,0))$;
$n_y = f(\vec{x} + (0,h,0)) - f(\vec{x} - (0,h,0))$;
$n_z = f(\vec{x} + (0,0,h)) - f(\vec{x} - (0,0,h))$;
return normalize $(n_x, n_y, n_z)$;
```

Algorithm 2: Surface normal computation.

To understand why this works for approximating the gradient of f we write out the definition:

$$\nabla f(\vec{x}) = \left(\frac{\partial f}{\partial x}(\vec{x}), \frac{\partial f}{\partial y}(\vec{x}), \frac{\partial f}{\partial z}(\vec{x})\right),$$

and since h is small we can approximate:

$$\begin{aligned}\frac{\partial f}{\partial x}(\vec{x}) &= \lim_{k \to 0} \frac{f(\vec{x} + (k,0,0)) - f(\vec{x} - (k,0,0))}{2k} \\ &\approx \frac{f(\vec{x} + (h,0,0)) - f(\vec{x} - (h,0,0))}{2h} = \frac{n_x}{2h}.\end{aligned}$$

Similarly, we get $\frac{\partial f}{\partial y}(\vec{x}) \approx \frac{n_y}{2h}$ and $\frac{\partial f}{\partial z}(\vec{x}) \approx \frac{n_z}{2h}$, leading to $\nabla f(\vec{x}) \approx \frac{1}{2h}(n_x, n_y, n_z)$. And since we will be normalizing the vector anyway, we can forget the denominator of $2h$, giving us the exact formula in algorithm 2:

$$\text{normalize}(\nabla f(\vec{x})) \approx \text{normalize}(n_x, n_y, n_z).$$

Using this to render our sphere by adding diffusion as shown in [9], we obtain the image seen in figure 2.

Finally, we obtain something that looks like a sphere. Now that we have seen how to render using raymarching and everything we require to perform it, we can move on to trying to define and render our three-dimensional Julia sets.

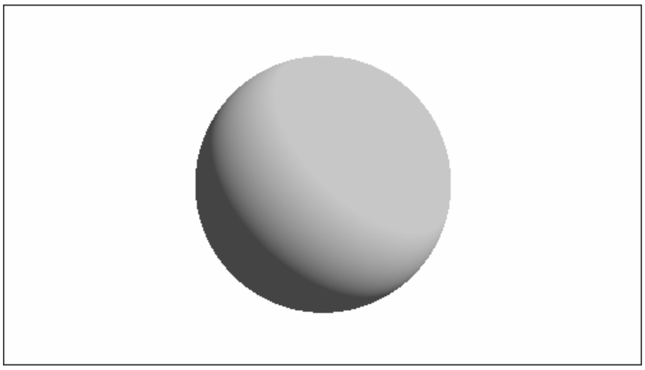


Figure 2: Raymarched sphere with diffusion.

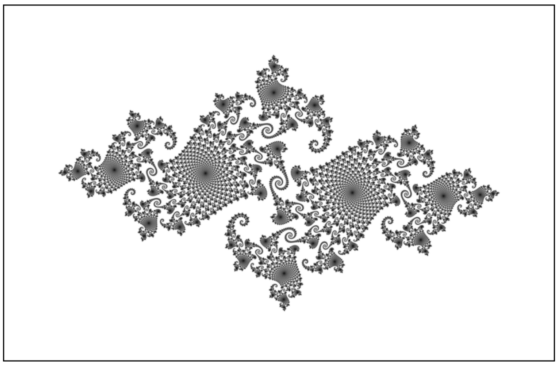


Figure 3: Example 2D Julia set.

3 JULIA SETS

To really understand how to extend Julia sets into three dimensions, we must first look at how to work with them in two-dimensional space. Some of the following formulas and algorithms are based on quite advanced mathematics stemming from complex analysis and complex dynamics. Therefore, we will only present the idea behind them and by no means provide rigorous derivations. To start with, let us define what a two-dimensional Julia set really is.

3.1 Definition

Following the definition from [1, section 2], we define our filled-in Julia sets in the following way.

Notation. Let $f : X \longrightarrow X$ be a mapping and let $n \in \mathbb{N}$ be a number. We define a new mapping $f^n : X \longrightarrow X$ as the n-fold composition of f with itself as $f^n = f \circ \cdots \circ f$.

Definition 3. Let $p \geq 2$ be an integer and let $c \in \mathbb{C}$ be any complex number. We can then define the following complex polynomial $f_c : \mathbb{C} \longrightarrow \mathbb{C}, f_c(z) = z^p + c$. The *filled-in Julia set* of f_c is the following set:

$$\mathcal{K}_c^p = \left\{ z \in \mathbb{C} \mid \text{the set } \left\{ f_c^n(z) \mid n \in \mathbb{N} \right\} \text{ is bounded} \right\}.$$

To put it in simpler terms, the filled-in Julia set is the set of complex numbers that never stray far from their origin under continuous iteration of the complex polynomial f_c. This usually forms interesting, fractal-like shapes when visualized in the complex plane. An example of which can be seen in figure 3.

Later on we will require a signed distance bound for raymarching three-dimensional variants of filled-in Julia sets. Turns out, with some quite advanced maths – illustrated in the next section – we can gain an estimate for the point-to-set distance of our filled-in Julia set.

3.2 Distance estimator

Let us assume we are working with a connected filled-in Julia set $\mathcal{K}_c^p$ of f_c, where $f_c(z) = z^p + c$. As is stated in [1, section 3.1] and [5, appendix G], it is known that there exists a unique biholomorphic mapping $\phi_c : \mathbb{C} \backslash \mathcal{K}_c^p \longrightarrow \mathbb{C} \backslash \overline{B}(0,1)$ where $\overline{B}(0,1)$ represents the closed unit disk. The following properties apply to ϕ_c:

$$\begin{gathered} \phi_c(z) = \lim_{n \to \infty} \left(f_c^n(z) \right)^{\frac{1}{p^n}} \\ \phi_c(z) \sim z \text{ as } |z| \longrightarrow \infty \\ \phi_c(f_c(z)) = (\phi_c(z))^p . \end{gathered}$$

The idea behind it is explained loosely in [7], where far from our set $\mathcal{K}_c^p$, we can approximate $f_c(z) = z^p + c \approx z^p$ since the leading term dominates. That would mean that near infinity $f_c^n(z) \approx z^{p^n}$. Setting $\phi_c(z) = \lim_{n \to \infty} \left(f_c^n(z) \right)^{\frac{1}{p^n}}$ is then almost like reversing this exponentiation near infinity, giving us some idea why $\phi_c(z) \sim z$ as $|z| \longrightarrow \infty$.

As in [1] we can then define the so-called *Green's function* of $\mathcal{K}_c^p$ $G : \mathbb{C} \longrightarrow [0, \infty)$ as

$$G(z) = \begin{cases} 0 & \mid z \in \mathcal{K}_c^p \\ \log |\phi_c(z)| & \mid z \notin \mathcal{K}_c^p \end{cases}.$$

Let us recall that we are trying to estimate the distance to our two-dimensional filled-in Julia set $\mathcal{K}_c^p$ such that it acts as a signed distance bound when we extend it into $\mathbb{R}^3$. Therefore, we would like to compute as accurate of a lower bound $f : \mathbb{C} \longrightarrow [0, \infty)$ as possible, meaning $f(z) \leq d\left(z, \mathcal{K}_c^p\right)$. That would make f a valid candidate for a signed distance bound when extended into three-dimensional space.

As is shown in [6], we can derive f in the following way. First, let $z \in \mathbb{C} \backslash \mathcal{K}_c^p$ be a complex number close to $\mathcal{K}_c^p$ and let $h \in \mathbb{C}$ be such that $z + h \in \mathcal{K}_c^p$ is the closest point to z (if one does not exist, choose one that is close enough). That means $|h| = d\left(z, \mathcal{K}_c^p\right)$ (or at least $|h| \approx d\left(z, \mathcal{K}_c^p\right)$). Since $|h|$ is small, we can write:

$$\begin{aligned} 0 = G(z+h) &= G(z) + \nabla G(z) \cdot h + O(h) \\ &\approx G(z) + \nabla G(z) \cdot h, \end{aligned}$$

where $G(z) \cdot h$ is the dot product between $G(z)$ and h. This gives us the following series of inequalities by using the triangle inequality, the real-positivity of G and the Cauchy-Schwarz inequality

$$\begin{aligned} 0 = |G(z+h)| &\approx |G(z) + \nabla G(z) \cdot h| \\ &\geq G(z) - |\nabla G(z) \cdot h| \geq G(z) - |\nabla G(z)| \, |h| . \end{aligned}$$

Rearranging the terms, it now follows that

$$d\left(z, \mathcal{K}_c^p\right) = |h| \geq \frac{G(z)}{|\nabla G(z)|},$$

making $f(z) = \frac{G(z)}{|\nabla G(z)|}$ suitable for our signed distance bound.

Interestingly, there exists a more accurate lower bound of $f(z) = \frac{\sinh(G(z))}{2e^{G(z)}|\nabla G(z)|}$ (see [1, theorem 6] for a derivation). But despite this, this tends to work worse in actual code as mentioned in [7], hence we will not be looking further into it.

To actually compute f, as we will need to when raymarching, we can first simplify the quotient. We define two sequences: $z_n = f_c^n(z)$ and $z_n' = \frac{\partial}{\partial z}\left(f_c^n(z)\right)$. Following along with the computation in [1, lemma 1] we can also check that $G(z) \approx \frac{\log|z_n|}{p^n}$ and $|\nabla G(z)| \approx \frac{|z_n'|}{p^n|z_n|}$ for sufficiently large n. Putting it together we obtain:

$$f(z) \approx \frac{|z_n| \log |z_n|}{|z_n'|} \text{ for sufficiently large } n.$$

By rewriting f using z_n and z_n', we make the computation of the distance estimate very simple using iteration. We obtain the following recursive formulas by substituting the definition of f_c:

$$z_{n+1} = z_n^p + c \text{ and } z_0 = z,$$
$$z_{n+1}' = p z_n^{p-1} z_n' \text{ and } z_0' = 1.$$

To obtain something that works even when z is far from $\mathcal{K}_c^p$, we can notice that $\mathcal{K}_c^p$ is bounded by some closed disk $\overline{B}(0, R)$. By choosing the appropriate radius R (in most cases $R = 2$ is good enough), we can define a new distance estimate

$$\tilde{f}(z) = \begin{cases} |z| - R & \mid |z| > R \\ f(z) & \mid |z| \leq R \end{cases},$$

albeit by losing continuity. Finally, all of this work can be seen in algorithm 3. The returned formula may look a bit different, but that is due to optimizing the code for performance.

```
Data: $x$ - complex number or quaternion, $n$ - the number of
      iterations, $R$ - radius of bounding disk
Result: approximate $\tilde{f}(x)$
if $|x| > R$ then
    return $|x| - R$;
end
$x_i = x$;
$x_i' = 1$;
for $i$ from 1 to $n$ do
    $x_i' = p x_i^{p-1} x_i'$;
    $x_i = x_i^p + c$;
end
return $\frac{1}{4}\sqrt{\frac{|x_i|^2}{|x_i'|^2}} \log\left(|x_i|^2\right)$;
```

Algorithm 3: Distance estimator for $\mathcal{K}_c^p$.

Now we can finally have a look at how to extend and render our three-dimensional Julia sets.

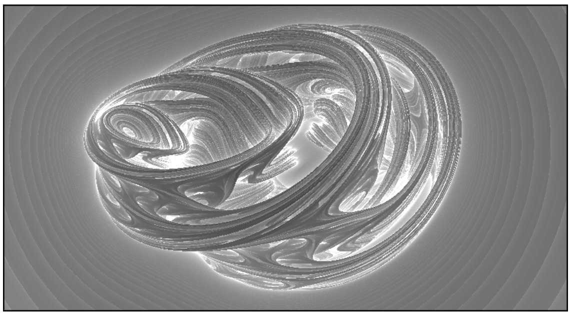


Figure 4: Example 3D Julia set

3.3 Extension into 3D

Surprisingly, extending our work so far into four dimensions is extremely simple. We redefine everything in the set of quaternions $\mathbb{H}$. This does not make any differences in our formulas and by defining $f_c \colon \mathbb{H} \to \mathbb{H}$ for $c \in \mathbb{H}$ with $f_c(q) = q^p + c$ everything else is redefined accordingly. Even algorithm 3 remains the same, the only difference being the definition of multiplication in $\mathbb{H}$.

To actually render in three dimensions, we can view $\mathbb{R}^3$ as a subspace of $\mathbb{H}$ and render only the intersection. The distance estimator computed in $\mathbb{H}$ then works as a signed distance bound in $\mathbb{R}^3$, since the distance in $\mathbb{H}$ must be smaller than in any of its subspaces. What we obtain are images similar to the one seen in figure 4.

4 CONCLUSIONS

Raymarching is a powerful tool for rendering many implicit surfaces defined using signed distance fields. It can be used to visualize even some of the most complex structures like fractals. Although not easy, it is possible to derive signed distance bounds or signed distance fields for these complicated sets and render them in 3D.

It is almost certainly possible to generalize this to even higher dimensions. We have shown how to obtain a distance estimator for filled-in Julia sets in the complex plane and quaternion space, so extending dimensions even further should work in the same way.

REFERENCES

[1] Guillaume Brouillette, Pierre-Olivier Parisé, and Dominic Rochon. 2019. Tricomplex distance estimation for filled-in julia sets and multibrot sets. *International Journal of Bifurcation and Chaos*, 29, 6, (June 15, 2019), 1950085. Publisher: World Scientific Publishing Co. DOI: 10.1142/S0218127419500858.

[2] J. C. Hart, D. J. Sandin, and L. H. Kauffman. 1989. Ray tracing deterministic 3-d fractals. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques* (SIGGRAPH '89). Association for Computing Machinery, New York, NY, USA, (July 1, 1989), 289–296. ISBN: 978-0-89791-312-6. DOI: 10.1145/74333.74363.

[3] John Hart. 1995. Sphere tracing: simple robust antialiased rendering of distance-based implicit surfaces, (Nov. 18, 1995).

[4] John C. Hart. 1996. Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12, 10, (Dec. 18, 1996), 527–545. DOI: 10.1007/s003710050084.

[5] John W. Milnor. 1990. Dynamics in one complex variable: introductory lectures. (Apr. 20, 1990). arXiv: math/9201272. DOI: 10.48550/arXiv.math/9201272.

[6] Inigo Quilez. [n. d.] Inigo quilez :: distance estimation - 2011. Retrieved Sept. 12, 2025 from https://iquilezles.org/articles/distance/.

[7] Inigo Quilez. [n. d.] Inigo quilez :: distance to fractals - 2004. Retrieved Sept. 11, 2025 from https://iquilezles.org/articles/distancefractals/.

[8] Inigo Quilez. [n. d.] Inigo quilez :: normals for an sdf - 2015. Retrieved Sept. 11, 2025 from https://iquilezles.org/articles/normalsSDF/.

[9] Lukasz Jaroslaw Tomczak. 2012. Gpu ray marching of distance fields. *Technical University of Denmark*, 8, 3.

Metaheuristic Approaches for solving the Geometric Knapsack Problem

Nejc Škrbec
Faculty of Computer and Information Science
University of Ljubljana
Ljubljana, Slovenia
nejc.skrbec@fri.uni-lj.si

Luka Fürst
Faculty of Computer and Information Science
University of Ljubljana
Ljubljana, Slovenia
luka.fuerst@fri.uni-lj.si

ABSTRACT

This paper presents solutions for the NP-hard Geometric Knapsack Problem i.e. packing rectangular items to maximize the container area. We integrate selected insertion algorithms from the literature with customized simulated annealing and genetic algorithms, in which we propose a fitness function to balance filling rate and item count. Among the insertion algorithms tested, the common perimeter heuristic algorithm achieved the highest performance. The most significant improvements across all algorithms were observed when paired with genetic algorithms, highlighting the effectiveness of this hybrid strategy.

KEYWORDS

geometric knapsack, metaheuristics, genetic algorithms, simulated annealing, hybrid optimization

1 INTRODUCTION

Packing and cutting problems (PCPs) seek geometric arrangements between containers and items, aiming to fill a container (e.g., a knapsack or sheet) with items or cut it into specified items while satisfying constraints such as non-overlapping and full containment according to the typology by Wäscher et. al. [13]. The Geometric Knapsack Problem (GNP), an extension of the classical knapsack problem, is a PCP characterized by a strongly heterogeneous item set and a limited set of containers, aiming to maximize the total value of orthogonally packed rectangular items in a two-dimensional container, where the value can be arbitrarily defined [13]. Its computational complexity makes exact solutions impractical for large instances, yet its applications in textiles (e.g., minimizing fabric waste), logistics (e.g., optimizing cargo loading), and manufacturing (e.g., efficient component placement) highlight its practical importance.

Existing heuristics often struggle to balance solution quality and computational efficiency. This paper integrates three insertion algorithms from the literature—Local Best Fit Algorithm (LBFA) [4], Constructive Heuristic Algorithm (CHA) [9], and Common Perimeter Heuristic Algorithm (CPHA) [6]—with customized simulated annealing (SA) and genetic algorithm (GA) metaheuristics. We propose a novel integration featuring a unique GA fitness function that balances filling rate and item count to enhance solution quality and efficiency. We evaluate these algorithms, both standalone and in hybrid combinations, on benchmark datasets [10], with performance metrics detailed in Table 1, providing insights into their practical applicability for industries requiring optimized packing solutions.

2 TERMINOLOGY

2.2 Item. An item p_i is a rectangle $p_i = \{(x_{i_1}, y_{i_1}), (x_{i_2}, y_{i_2})\}$ where (x_{i_1}, y_{i_1}) is the bottom-left vertex and (x_{i_2}, y_{i_2}) is the top-right vertex. The width and height are given by $w_i = x_{i_2} - x_{i_1}$ and $h_i = y_{i_2} - y_{i_1}$, respectively.

2.3 Container. A container is a rectangular area with fixed dimensions $W \times H$, anchored at the origin $(0, 0)$.

2.4 Placement. The placement operation moves an item to a new position by translating its reference vertex to a target point (x_t, y_t). The transformation is

$$p'_i = \{(x_{i_1} + \Delta x, y_{i_1} + \Delta y), (x_{i_2} + \Delta x, y_{i_2} + \Delta y)\} \tag{1}$$

where $\Delta x = x_t - x_{i_1}$ and $\Delta y = y_t - y_{i_1}$. A placement is valid when the translated item is contained inside the container and does not overlap with any of the other placed items.

2.5 Filling Rate. The filling rate is defined as the ratio of the total area of placed items to the container's area.

3 PROBLEM DEFINITION

Consider a rectangular container with dimensions $W \times H$ and a set of $N = \sum_{i=1}^{n} b_i$ rectangular items, where b_i represents the number of available copies of item i. Each item i has dimensions (w_i, h_i) and a value equal to its area, $v_i = w_i \times h_i$. Items are placed using their bottom-left vertex as the reference vertex. The objective is to maximize the total value of the placed items

$$\max \sum_{i=1}^{n} a_i v_i \tag{2}$$

where a_i represents the number of placed copies of item i

3.1 Placement validity. The placed items must be validly placed, i.e., be fully contained within the container without overlapping:

$$\begin{gathered} \forall i \in \{1, 2, \ldots, N\} : \\ \left(0 \le x_{i_1} < x_{i_2} \le W \land 0 \le y_{i_1} < y_{i_2} \le H\right) \end{gathered} \tag{3}$$

$$\begin{gathered} \forall i, j \in \{1, 2, \ldots, N\}, i \ne j : \\ \left((x_{i_1} < x_{j_2} \le x_{i_2} \lor x_{j_1} < x_{i_2} \le x_{j_2}) \Rightarrow (y_{i_2} \le y_{j_1} \lor y_{j_2} \le y_{i_1})\right) \land \\ \left((y_{i_1} < y_{j_2} \le y_{i_2} \lor y_{j_1} < y_{i_2} \le y_{j_2}) \Rightarrow (x_{i_2} \le x_{j_1} \lor x_{j_2} \le x_{i_1})\right) \end{gathered} \tag{4}$$

3.2 Placement orthogonality. The items must be placed orthogonally, i.e., with sides parallel to the container's edges, and no rotations are allowed:

$$\begin{gathered} \forall i \in \{1, 2, \ldots, N\}, \quad p_i = \{(x_{i_1}, y_{i_1}), (x_{i_2}, y_{i_2})\} : \\ \mathcal{P}(p_i) = \{(x_{i_1} + \Delta x, y_{i_1} + \Delta y), (x_{i_2} + \Delta x, y_{i_2} + \Delta y)\} \end{gathered} \tag{5}$$

https://doi.org/10.51939/scores25.2

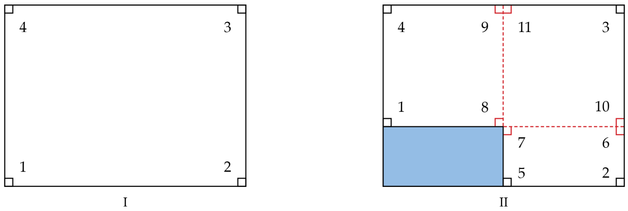


Figure 1: Example of angles in an empty container (I) and after inserting an item (II).

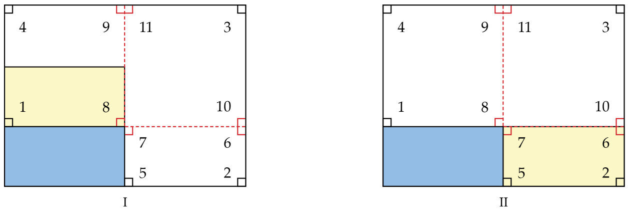


Figure 2: Example of AOPs and fit.

4 INSERTION ALGORITHMS

4.1 Local Best Fit Algorithm (LBFA)

This algorithm is based on a fit evaluation method first proposed by Chen et. al. [4], where covered angles are used as a measure of fit when inserting items into a container.

4.1.1 Angle. We consider right angles that satisfy one of the following criteria: those formed by two container sides (e.g., angle 1, Figure 1), by a container side and an item side (e.g., angle 5), by two item sides or their extensions (e.g., angles 7, 8), or by a container side and an item side extension (e.g., angles 9, 11). In our implementation, only the positions of these angles are stored, without distinguishing between angles that share the same position (e.g., angles 9 and 11, Figure 1).

4.1.2 Angle-occupying placement (AOP). An item *covers* an angle if it lies on the item's side or coincides with one of its vertices (e.g., the blue item covers angle 1, Figure 1 (II)). The algorithm considers valid placements where the placed item covers at least one angle. The fit of an inserted item is equal to the number of angles covered after its insertion (e.g., the fit of the yellow item is 2 in Figure 2 (I), and 4 in Figure 2 (II)).

4.1.3 Procedure. Given a container and an item list, the procedure generates a list of AOPs (L_a) for all items. It iteratively: (1) computes the fit for each AOP in L_a, selecting the one with the maximum fit and breaking ties by minimizing the Euclidean distance to the origin (bottom-left-first strategy) [1]; (2) inserts the corresponding item into the container; (3) updates L_a by regenerating AOPs for the unplaced items and removing those for the inserted item. The process continues until L_a is empty, returning the final layout and filling rate.

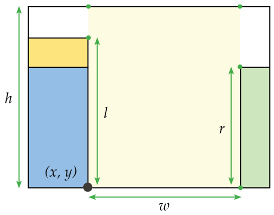


Figure 3: Available space attributes.

4.2 Constructive Heuristic Algorithm (CHA)

The constructive heuristic algorithm [9] evaluates the fit for an unplaced item by considering empty spaces between items in the current layout.

4.2.1 Available position. A point inside the container not covered by any item in the current layout.

4.2.2 Available space. A rectangular region within the container, extending to its top edge and containing no items, which may overlap with other available spaces. It is defined by its position (coordinates of the bottom-left vertex, point (x, y), Figure 3), width (horizontal distance between the left and right side, w), height (vertical distance from the bottom-left vertex to its projection on the container's top edge, h), and left/right wall lengths (distances from the bottom-left or bottom-right vertex to the topmost vertex of the nearest item on the left or right side, l and r, Figure 3).

4.2.3 Placement. Can be defined by mapping an item's vertex (e.g., bottom-left) to a corresponding space's vertex (e.g., bottom-left). The fit is computed by a function [9] returning integer values between -1 and 4, where -1 indicates the item is too large, and 4 indicates equal widths and item height matching the longer wall's length.

4.2.4 Procedure. The algorithm takes a container and an item list and initializes a list of available spaces. Next it iteratively selects the leftmost available space and computes fit for all unplaced items. It then inserts the item with the maximum fit (mapping bottom-left or bottom-right vertices based on wall lengths) and updates the list of available spaces and the item list. The loop runs until S is empty or the leftmost space's y-coordinate exceeds the container height, returning the final layout. After inserting a new item into the container, the existing unoccupied spaces may be partially or fully covered, requiring dimension updates or removal to maximize their total area and container coverage. Two edge cases are: (1) an empty container becomes an unoccupied space with no further steps needed; (2) with one item, two new unoccupied spaces are created at its top-left and bottom-right vertices. For layouts with multiple items, new unoccupied spaces are generated based on the last inserted item by: (1) updating overlapping spaces to create new non-overlapping ones as previously described; (2) expanding spaces by drawing a leftward horizontal ray from each current space's point, adding a new space where it intersects another item's edge; (3) filtering spaces by grouping those with shared x or y coordinates and selecting the largest area in each group.

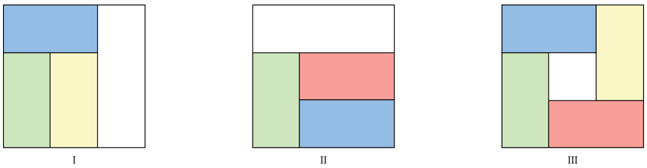


Figure 4: Layouts obtained for the same items with LBFA(I), CHA (II) and CPHA (III).

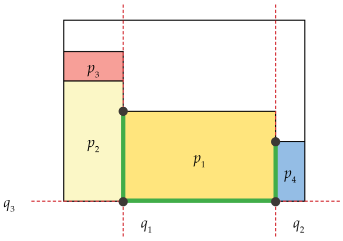


Figure 5: Common perimeter of item p_1 (green).

4.3 Common Perimeter Heuristic Algorithm (CPHA)

This algorithm uses the common perimeter heuristic, first proposed by Hadjiconstantinou and Iori [6]. It mitigates the bias of the bottom-left insertion policy, which prioritizes placements near the coordinate origin and restrict certain types of layouts. To further illustrate CPHA's advantage, consider a container with side length $3x$ and items $I = \{(x, 2x), (2x, x), (x, 2x), (2x, x)\}$. LBFA places three items: the first at $(0, 0)$, the second above it covering four angles, and the third beside the first, but cannot place the fourth (Figure 4 (I)). CHA places three items: the first at $(0, 0)$, the second in a space's bottom-right corner at $(x, 0)$, and the fourth at (x, x), but cannot place the third (Figure 4 (II)). CPHA, by maximizing the common perimeter, can place all four items, achieving a layout unattainable by LBFA and CHA, as shown in Figure 4 (III).

4.3.1 Contact. The situation when sides of two items (p_1 and p_2) or an item and the container lie on a common line (e.g., line q_1, Figure 5). The shorter of both sides is the contact between them.

4.3.2 Common perimeter. This is the sum of the lengths of all contacts of an item with other items and the container in the current layout. The common perimeter value of an item after the insertion into the current layout is what determines the fit value for this algorithm.

4.3.3 Procedure. The algorithm initializes an empty set of unpackable items and processes the item list sequentially. For each item: (1) if marked unpackable, skip it; (2) generate all valid placements by mapping item vertices to container or layout vertices; (3) compute the common perimeter for each placement; (4) if no valid placements exist, mark the item unpackable; (5) otherwise, select the placement with the maximum fit (earliest if tied), insert the item, remove it from the item list, and update the layout. The process continues until all items are processed, returning the final layout and filling rate.

5 METAHEURISTICS

5.1 Simulated annealing (SA)

Our SA-based method builds upon the approach proposed by Leung et. al. [9] but is modified to accept any insertion algorithm H. Our cooling schedule employs geometric cooling, defined as $T' \leftarrow \alpha \times T$, where α is the cooling factor. Our neighborhood function generates new solutions by swapping two random items in the input item list and applying H. Our probability function accepts a layout if it improves the filling rate or satisfies $e^{\frac{-\Delta}{T}} > r$, where Δ is the filling rate difference and r is a random number between 0 and 1. The algorithm takes a container, item list, insertion algorithm H, initial temperature T_0, minimum temperature T_{min}, and cooling factor α. After initializing a solution using H, it generates a neighboring solution using the neighborhood function and accepts it based on the probability function. Then it cools the temperature and repeats the process. It halts when the temperature falls below T_{min} or the maximum number of iterations is reached. It returns the final layout.

5.2 Genetic algorithm (GA)

In our GA-based method, each individual consists of an item list (genome) and a container. The population, a set of such individuals, evolves over generations through selection, crossover, and mutation. We introduce a novel fitness function:

$$\texttt{fitness} = \texttt{filling\_rate} - k \times \texttt{num\_items}, \qquad (6)$$

where k is a coefficient balancing space maximization with minimizing item count. However, this fitness function may bias toward solutions with fewer and larger items, limiting the inclusion of smaller items that could enhance filling rate.

Our algorithm creates a fixed-size population of size s is used and initializes it with random permutations of the input item list. For each permutation, the heuristic H computes an initial layout, and the fitness is evaluated using the fitness function. Individuals are sorted in a descending order by fitness to prioritize high-quality solutions. The algorithm applies selection, crossover, and mutation in each iteration to evolve the population. Selection employs roulette selection with linear scaling [12], transforming fitness values via $f' = a \times f + b$, where a and b are computed to align the population's average fitness to a target value. Crossover uses the OX3 operator [6], generating two offspring by copying a random segment from one parent and filling remaining positions with items from the other. The offspring with higher fitness is added to the population. Mutations, applied with probability p, utilize M1 and M3 operators with equal 50% probability [8]. M1 removes a random number of items and reinserts them using H, with the maximum removal count decreasing over iterations as $n_{\max} = \frac{(i_{\max} - i) \times M}{i_{\max}}$, where i is the current iteration, $i_{\max}$ is the maximum number of iterations, and M is the item list length. M3 locally optimizes layouts by removing items, selecting a placement point with minimal coordinates, and reinserting items to maximize the filling rate.

The GA takes as input a container, item list, fitness function C, heuristic H, selection S, crossover R, mutation M, mutation probability p, and population size s. It iteratively applies S, R, and M, updating the population to maintain size s. If a solution's filling rate exceeds the current best, it updates the final layout and filling rate, ultimately returning the optimal layout and its filling rate.

Table 1: Performance of LBFA, CHA, and CPHA, standalone and with SA and GA, across test cases.

Test Case	Items	LBFA		LBFA+SA		LBFA+GA		CHA		CHA+SA		CHA+GA		CPHA		CPHA+SA		CPHA+GA	
		Fill (%)	Time (s)	Fill (%)	Time (s)	Fill (%)	Time (s)	Fill (%)	Time (s)	Fill (%)	Time (s)	Fill (%)	Time (s)	Fill (%)	Time (s)	Fill (%)	Time (s)	Fill (%)	Time (s)
B-1	9	50	0.15	97	8.63	97	1052.96	96	0.23	97	5.53	97	477.59	97	0.17	97	10.36	97	760.50
OKP-1	15	60	0.58	67	43.68	78	685.03	69	0.23	87	11.70	93	546.68	76	0.36	93	19.19	93	1269.79
HT-1	16	95	1.21	100	4.19	100	11.08	86	0.35	95	16.38	100	93.15	90	0.56	97	41.06	100	163.55
GCUT-1	17	77	1.95	80	105.26	84	1987.33	75	0.42	77	22.58	92	670.53	78	0.92	86	63.59	92	1057.54
GCUT-2	17	60	1.87	69	92.72	80	2211.82	71	0.52	79	28.16	93	691.25	83	0.62	94	48.74	95	1811.76
GCUT-3	17	77	1.82	82	105.45	82	1776.31	59	0.54	80	34.58	85	720.54	77	1.02	85	56.66	95	1233.49
GCUT-5	17	72	1.84	82	98.05	90	2262.95	75	0.65	85	32.75	87	698.22	83	0.86	87	60.82	93	1675.32
B-2	18	80	1.83	97	95.94	97	1624.21	79	0.35	92	24.37	97	663.68	93	0.98	94	54.31	97	1346.44
C-1	19	79	0.25	79	16.91	91	865.88	83	0.05	96	2.44	96	490.28	88	0.16	93	15.03	97	938.39
C-2	20	88	0.28	95	17.85	95	755.15	93	0.05	93	3.09	95	499.20	86	0.21	96	19.42	95	1059.25
C-3	20	88	0.68	90	46.78	90	1018.30	86	0.05	90	6.51	95	529.60	86	0.33	90	30.39	95	1327.91
C-5	20	81	1.38	84	88.86	84	1366.51	73	0.24	85	12.92	92	581.70	86	0.69	92	45.33	95	1560.41
HT-8	21	84	3.93	88	190.68	88	3510.65	71	0.69	81	31.27	88	706.08	71	1.45	88	85.35	88	1663.66
B-3	23	82	3.93	91	247.57	91	3210.18	67	0.90	87	47.93	92	872.27	78	1.68	90	114.18	94	2405.63
HT-4	26	85	8.13	88	493.21	91	5235.39	65	0.31	76	35.00	87	844.95	85	2.55	91	178.11	91	2917.94
OKP-2	30	92	3.40	92	209.48	95	1435.24	86	0.13	93	8.05	97	609.93	86	1.05	88	55.05	96	1805.79
OKP-3	30	74	2.19	80	114.39	91	1235.54	81	0.24	90	11.02	96	614.02	84	0.59	92	39.23	96	1397.44
OKP-4	33	72	0.77	93	105.89	93	1533.79	90	0.19	93	8.99	98	610.56	85	0.83	92	50.83	97	1410.79
B-5	39	86	30.28	86	1928.39	91	26494.40	68	0.80	85	82.98	85	1397.89	88	8.65	91	502.58	91	5272.81
HT-10	47	86	68.22	90	3872.35	92	31020.87	69	1.43	80	150.92	89	1746.68	94	10.89	94	679.01	96	10445.71

6 EXPERIMENTAL EVALUATION

6.1 Setup

We evaluated our insertion algorithms (LBFA, CHA, and CPHA) on their own and in combination with custom SA and GA using test cases from benchmark datasets [10], which demonstrate significant variation in container and item characteristics. The datasets include **GCUT** [2] with square containers and varied item shapes, **B** [3] with square and taller containers, **C** [11] with diverse container sizes, **HT** [7] with small to elongated containers, and **OKP** [5] with square containers and elongated items. All algorithms were implemented in Python using *NumPy*, *Shapely*, and *Matplotlib* libraries. Tests were run on a 2.6 GHz 6-core Intel i7 with 16 GiB RAM. We set the following parameters for SA: $T_0 = 1000$, $T_f = 0.01$, $\alpha = 0.8$, 10k iterations. For GA, the parameters are as follows: population=30, $p_m = 0.1$, 100 iterations.

6.2 Results

We evaluated LBFA, CHA, and CPHA both standalone and with SA and GA, on selected test cases from previously mentioned datasets. Performance metrics are detailed in Table 1. Standalone algorithms yielded lower filling rates due to their dependence on initial item order and fixed strategies. LBFA and CHA, constrained by bottom-left bias, often produced suboptimal layouts. CPHA outperformed them by maximizing common perimeter, particularly on square containers such as in test cases B-1 and HT-10. SA improved filling rates across all algorithms by exploring neighboring solutions, with moderate execution times. GA achieved the highest filling rates, but incurred significant computational costs, especially for LBFA and CPHA on large instances such as HT-10 (over 10,000 seconds). CHA was the fastest, leveraging efficient empty-space updates, while LBFA's angle calculations led to longer times, scaling with item count. CPHA with GA consistently delivered the best filling rates, excelling on datasets with diverse item sizes (e.g., OKP, C), due to its flexibility in vertex selection. However, identical filling rates in some cases (e.g., B-1 across CPHA approaches) suggest local maxima. An anomaly occurred in one case where CPHA+GA underperformed CPHA+SA. The lack of item rotation likely limited filling rates, particularly for elongated containers (e.g., HT-4, HT-8).

7 CONCLUSION

This paper highlights the efficacy of approaches combining metaheuristic with insertion algorithms for solving the GNP. The CPHA approach, paired with GA, achieved the highest filling rates, despite longer execution times. These results enhance the understanding of GNP optimization and provide directions for future research.

REFERENCES

[1] Brenda S. Baker, Jr. Coffman, E. G., and Ronald L. Rivest. 1980. Orthogonal Packings in Two Dimensions. *SIAM J. Comput.* 9, 4 (1980), 846–855. https://doi.org/10.1137/0209064

[2] J. E. Beasley. 1985. Algorithms for Unconstrained Two-Dimensional Guillotine Cutting. *The Journal of the Operational Research Society* 36, 4 (1985), 297–306. http://www.jstor.org/stable/2582416

[3] Edmund Burke, Graham Kendall, and Glenn Whitwell. 2004. A New Placement Heuristic for the Orthogonal Stock-Cutting Problem. *Operations Research* 52 (08 2004), 655–671. https://doi.org/10.1287/opre.1040.0109

[4] Mao Chen, Chao Wu, Xiangyang Tang, Xicheng Peng, Zhizhong Zeng, and Sanya Liu. 2019. An efficient deterministic heuristic algorithm for the rectangular packing problem. *Computers Industrial Engineering* 137 (2019), 106097. https://doi.org/10.1016/j.cie.2019.106097

[5] Sándor P. Fekete and Jörg Schepers. 2004. A Combinatorial Characterization of Higher-Dimensional Orthogonal Packing. *Mathematics of Operations Research* 29, 2 (2004), 353–368. http://www.jstor.org/stable/30035687

[6] Eleni Hadjiconstantinou and Manuel Iori. 2007. A hybrid genetic algorithm for the two-dimensional single large object placement problem. *European Journal of Operational Research* 183, 3 (2007), 1150–1166. https://doi.org/10.1016/j.ejor.2005.11.061

[7] E. Hopper and B.C.H. Turton. 2001. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research* 128, 1 (2001), 34–57. https://doi.org/10.1016/S0377-2217(99)00357-4

[8] Igor Kierkosz and Maciej Łuczak. 2013. (12 2013), 1–20.

[9] Stephen C.H. Leung, Defu Zhang, Changle Zhou, and Tao Wu. 2012. A hybrid simulated annealing metaheuristic algorithm for the two-dimensional knapsack packing problem. *Computers Operations Research* 39, 1 (2012), 64–73. https://doi.org/10.1016/j.cor.2010.10.022 Special Issue on Knapsack Problems and Applications.

[10] Oscar Oliveira, Dorabela Gamboa, and Elsa Silva. 2019. RESOURCES FOR TWO-DIMENSIONAL (AND THREE-DIMENSIONAL) CUTTING AND PACKING SOLUTION METHODS RESEARCH. 131–138. https://doi.org/10.33965/ac2019_201912L016

[11] Elsa Silva, Filipe Alvelos, and José Carvalho. 2010. An integer programming model for two- and three-stage two-dimensional cutting stock problems. *European Journal of Operational Research* 205 (09 2010), 699–708. https://doi.org/10.1016/j.ejor.2010.01.039

[12] University of Nevada, Reno, Department of Computer Science. 2004. Scaling in Genetic Algorithms. https://www.cse.unr.edu/~sushil/class/gas/notes/scaling. Accessed: 2025-06-24.

[13] Gerhard Wäscher, Heike Haußner, and Holger Schumann. 2007. An improved typology of cutting and packing problems. *European Journal of Operational Research* 183, 3 (2007), 1109–1130. https://doi.org/10.1016/j.ejor.2005.12.047

Proc. of the 11th Student Computing Research Symposium (SCORES'25), Ljubljana, Slovenia, September 30, 2025

Empirical Analysis Of Heuristic And Approximation Algorithms For The Mutual-Visibility Problem

Vanja Stojanović*
vs66277@student.uni-lj.si
Faculty of Mathematics and Physics,
University of Ljubljana
Jadranska ulica 21
SI-1000 Ljubljana, Slovenia

Bor Pangeršič
bp52226@student.uni-lj.si
Faculty of Mathematics and Physics,
University of Ljubljana
Jadranska ulica 21
SI-1000 Ljubljana, Slovenia

ABSTRACT

The NP-complete mutual-visibility (MV) problem currently lacks empirical analysis on its practical behaviour despite theoretical studies. This paper addresses this gap by implementing and evaluating three distinct algorithms – a direct random heuristic, a hypergraph-based approximation, and a genetic algorithm – on diverse synthetic graph datasets, including those with analytically known $\mu(G)$ values and general graph models. Our results demonstrate that for smaller graphs, the algorithms consistently achieve MV set sizes aligning with theoretical bounds. However, for larger instances, achieved solution sizes notably diverge from theoretical limits; this, combined with the absence of tight bounds, complicates absolute quality assessment. Nevertheless, validation on known optimal graphs showed the Genetic Algorithm and other heuristics empirically performing best among tested methods.

KEYWORDS

Graph Theory, Mutual-Visibility Problem, Greedy Heuristic, Genetic Algorithm, Empirical Analysis

1 INTRODUCTION

All code and datasets for this work are available at our GitHub repository.

The mutual-visibility (MV) problem was introduced by Di Stefano et al. in [4] and was subsequently generalized and extended in [3]. Given a subset $X \subseteq V(G)$, we say that vertices $u, v \in X$ are *X-visible* if there exists a shortest path P such that $V(P) \cap X \subseteq \{u, v\}$. Such a set X is called a *mutual-visibility set* if every pair $u, v \in X$ are X-visible. A μ-set is a maximum mutual-visibility set (i.e., no larger mutual-visibility set exists). When we only require that no proper superset is an mutual-visibility set, we speak of a maximal MV-set. Let $\mu(G)$ denote the cardinality of a μ-set of graph G.

The mutual-visibility decision problem is stated as follows; given a graph $G = (V, E)$ and set $P \subseteq V(G)$, is P a mutual-visibility set? Di Stefano derived and proved a polynomial-time BFS-based algorithm for solving this problem. However, in general, finding the largest such set, i.e. finding the *mutual-visibility number* of a graph, is proved to be NP-complete [4].

For the mutual-visibility and related problems, Bilò et al. proved strong inapproximability results, demonstrating that it is not approximable within $n^{1/3-\epsilon}$ for graphs of diameter at least 3, and is APX-Hard for graphs of diameter 2. While exact solutions for some specific graph classes are known, there is also a polynomial-time approximation algorithm for general graphs [1]. Its precise lower bound on the size of the found set is discussed in detail in the Approximation Algorithm Using Hypergraphs section.

1.1 Motivation

Despite the theoretical analysis of the MV problem, there have been no public attempts to empirically analyse its behaviour in practice or to systematically implement and evaluate various heuristic and approximation algorithms. This paper addresses this gap by conducting a comprehensive empirical analysis of the MV problem. Our primary motivations are threefold:

1. To gain practical insights into the difficulty of approximating the mutual-visibility number, moving beyond theoretical bounds to observe real-world performance.
2. To empirically verify if established theoretical upper and lower bounds are observed in practice.
3. To comparatively evaluate the performance of different algorithmic approaches.

To achieve these objectives, we develop and analyse three distinct algorithms: a direct random heuristic, an approximation algorithm based on a hypergraph transformation, and a metaheuristic approach using a Genetic Algorithm. Our empirical evaluation is conducted on a dataset that includes graphs with analytically known mutual-visibility numbers (for validation) as well as other synthetic graph models.

2 ALGORITHMS AND HEURISTICS

2.1 Direct Random Algorithm

We construct a simple random algorithm for finding the largest mutual-visibility set within a graph G. The algorithm works by random sampling; by first choosing a subset size k uniformly from $1..n$, then sampling a k-subset of $V(G)$, and checking for the mutual-visibility property of the set. As a default the number of trials T was set to $10,000$. The algorithm is described in Algorithm 1. Note that the procedure $\mathrm{MV}(G, S_{candidate})$ is defined in [4, p. 7] as Procedure MV.

2.2 Genetic Algorithm (GA)

We adapt a GA approach to find a large mutual-visibility set, drawing inspiration from its application to the General Position Problem in [5].

Each candidate MV set is represented as an **individual** in the form of a binary characteristic vector of size n, where a '1' at index i indicates the inclusion of node i, and '0' indicates exclusion. The

---

*Both authors contributed equally to this research.

https://doi.org/10.51939/scores25.3

```
Input: Graph $G$, number of trials $T$
Output: Best mutual-visibility set $S^*$
$n \leftarrow |V(G)|$
$S^* \leftarrow \emptyset$
for $i \leftarrow 1$ to $T$ do
    $k \leftarrow$ random integer in $[1, n]$
    $S \leftarrow$ uniformly random sample of $k$ nodes from $V(G)$
    if $|S| > |S^*| \wedge \mathrm{MV}(G, S)$ then
        $S^* \leftarrow S$
    end
end
return $S^*$
```

Algorithm 1: Random Sampling by Subset Size for Mutual-Visibility

algorithm's **fitness function** evaluates each individual S using the formula: $\text{Fitness}(S) = |S| - M \cdot f$. Here, $|S|$ is the number of selected vertices, M is a large penalty constant, and f represents the number of violations. A violation occurs if a pair of vertices within S are not mutually visible (i.e., there exists no shortest path between them free of other vertices in S).

The algorithm begins with an **initial population** of individuals generated randomly (the solutions are not necessarily feasible). In each generation, **parent solutions** are randomly selected, and a **crossover operator** is applied (a bitwise OR operation). A **mutation operator** works by randomly selecting two gene positions (vertices) in an individual's characteristic vector and swapping their values. The **population update** phase merges the current population with the newly generated offspring and mutated individuals, sorts them by fitness, and truncates the population to maintain the predefined size, forming the next generation. The **stopping criterion** is defined as a maximum number of iterations.

The general structure of the adapted GA is outlined in Algorithm 2. The parameters p_c and p_m were both chosen simply as 1.0.

```
Input: Graph $G$; maximum iterations $T$; population size $N$;
       crossover probability $p_c$; mutation probability $p_m$; penalty
       parameter $M$
Output: Best mutual-visibility set $S^*$ found
Initialize a population $\mathcal{P}$ of $N$ individuals
Evaluate the fitness of each individual in $\mathcal{P}$
for $t \leftarrow 1$ to $T$ do
    Select parents from $\mathcal{P}$
    Generate offspring via crossover with probability $p_c$
    Apply mutation to offspring with probability $p_m$
    Form a new population by merging parents, offspring, and
      mutants
    Evaluate fitness of all individuals in the new population
    Select the best $N$ individuals to form the next generation $\mathcal{P}$
end
$S^* \leftarrow$ individual in $\mathcal{P}$ with the highest fitness
return $S^*$
```

Algorithm 2: Genetic algorithm for approximating a mutual-visibility set

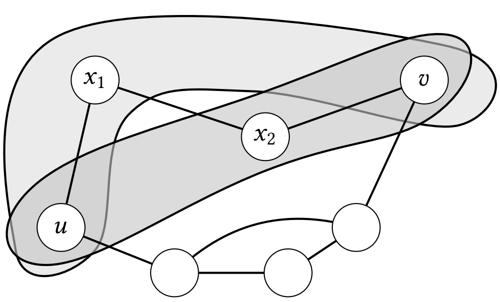


Figure 1: Generation of the hypergraph

2.3 Approximation Algorithm Using Hypergraphs

The approximation algorithm leverages a transformation of the MV problem into finding an independent set in a 3-uniform hypergraph, as described in [1]. The hypergraph H is constructed with the same set of vertices as the original graph G, i.e., $V(H) = V(G)$. For any unordered pair of distinct vertices $\{u, v\} \subseteq V(G)$, a shortest path $\langle u, x_1, x_2, \ldots, x_k, v \rangle$ is considered. The set of internal vertices on this path is denoted $B(\{u, v\}) = \{x_1, \ldots, x_k\}$, which can be empty if $k = 0$ (i.e., u and v are adjacent). A hyperedge $\{u, v, x\}$ is then added to H for each such pair $\{u, v\}$ and for every internal vertex $x \in B(\{u, v\})$. See Figure 1 for an example.

An independent set S of this hypergraph H is a subset of $V(G)$ such that no hyperedge e (e.g., $\{u, v, x\}$) is entirely contained within S. Notice, an independent set S of H is exactly a μ-set of G. That is, if $u, v \in S$ are two distinct vertices, it must be the case that no vertex $x \in B(\{u, v\})$ is also in S, as H would otherwise contain the hyperedge $\{u, x, v\}$. The methodology assumes that for graphs with at most 6 vertices, a μ-set can be computed by brute force.

Once the hypergraph H is constructed, the algorithm proceeds to compute an independent set S of H using a greedy selection strategy defined in [2]. The general approach for this phase is outlined in Algorithm 3.

```
Input: Graph $G$
Output: Approximated mutual-visibility set $S$
// Phase 1: Hypergraph Construction
Construct a hypergraph $\mathcal{H}$ from $G$ as described in the text
// Phase 2: Greedy Independent Set Selection
$S \leftarrow V(H)$
$H' \leftarrow H$
while There exists a vertex $y \in V(H')$ such that its degree
  $deg_{H'}(y) \geq 1$ do
    Select $y^* \in V(H')$ s. t. $deg_{H'}(y^*)$ is maximum
    $S \leftarrow S \setminus \{y^*\}$
    Remove $y^*$ from $V(H')$
    Remove all edges incident to $y^*$ from $E(H')$
end
return $S$
```

Algorithm 3: Greedy algorithm for approximating an MV set via hypergraph independent set

The approximation algorithm is particularly useful, because it is guaranteed to find a mutual-visibility set of size $\Omega\left(\sqrt{n/\overline{D}}\right)$, where $\overline{D}$ is the average distance between vertices of a graph, rigorously defined in (1). The proof of the bound lower is provided in [1].

$$\overline{D} = \frac{2}{n(n-1)} \sum_{\{u,v\}\in\binom{V(G)}{2}} d(u,v) \qquad (1)$$

3 EXPERIMENTAL METHODOLOGY

The algorithms will be executed on graph instances from the 'n10' and 'n100' size categories within the curated dataset. For each graph instance, both the Direct Random Heuristic, the Approximation Algorithm Using Hypergraphs, and the Genetic Algorithm will be run multiple times, with average results recorded.

3.1 Data Sets

The empirical analysis is conducted on a single, dataset of synthetically generated graphs. It comprises various graph classes, each selected to serve specific purposes in our analysis. Table 1 provides a detailed overview of these classes, including their key properties, the parameters varied during their generation, and their expected $\mu(G)$ values for validation purposes. The graph classes are broadly categorized as follows:

1. **Category 1: Analytical Solutions.** This category includes graphs where $\mu(G)$ can be calculated directly using well-known formulas. It also features more complex graphs for which tighter bounds were obtained in recent research.
2. **Category 2: General Graph Models.** This category includes less predictable graph structures, namely Generalized Petersen Graphs and connected Erdős-Rényi graphs. For these models, $\mu(G)$ is generally unknown or computationally intractable to determine exactly.

The reason for choosing these general graph categories is due to the simplicity of generating Erdős-Rényi graphs, and the still analytically unknown solutions to the Generalized Petersen graphs. We hope to provide empirical insight into the structure of the MV problem on such variety of graphs.

3.2 Performance Metrics and Analysis

The primary metrics collected for each algorithm on every graph instance will include approximated MV set size $|S|$ as well as execution time.

For graph classes in Category 1 (where $\mu(G)$ is known analytically), the **approximation ratio** ($|S|/\mu(G)$) will be calculated.

The analysis will involve:

- **Comparative Performance Across Scales:** Evaluating how the average approximation ratio and runtime of each algorithm vary across different size categories ('n10', 'n100') and graph types (e.g., Complete, Tree, Grid, Mycielskian types, Erdős-Rényi).
- **Bound Verification:** For graph classes with known theoretical bounds, plotting the achieved $|S|$ against the theoretical lower bounds (e.g., $\sqrt{n/\overline{D}}$, $\omega(G)$) to visually assess if these bounds are met or exceeded in practice.

4 RESULTS AND ANALYSIS

4.1 Overall Performance Overview

We first provide a high-level summary of the algorithm's average performance for Category 1 of the dataset. Notice that in larger graph instances, the average performance of GA can occasionally suffer due to the algorithm getting stuck in local maxima. This results in a few individual cases exhibiting very poor approximation ratios, while other individual runs on similar instances might still perform well. These findings are visible in Tables 2, where $\overline{\alpha}$ represents *average ratio for the approach*, and 3, where $\overline{T}$ represents *average runtime* for the approach. Notice that the genetic algorithm has $\overline{\alpha}$ over 1 in some cases, this is due to its indeterministic nature, where it can choose a bigger S than $\mu(G)$, and it does not optimize it in the specified maximum iterations.

Table 2: Average Ratio by Algorithm, Graph Type, and Size Category.

Graph Type	Size Category	$\overline{\alpha}$ Random	$\overline{\alpha}$ Hyper	$\overline{\alpha}$ Genetic
Complete	n10	1.000	1.000	1.000
Complete	n100	1.000	1.000	1.000
Grids	n10	0.892	0.675	0.767
Grids	n100	0.822	0.707	3.897
Mycielskian	n10	0.917	0.889	0.919
Mycielskian	n100	0.587	0.867	0.815
Trees	n10	1.000	1.000	1.024
Trees	n100	0.547	1.000	2.573

Table 3: Average Performance by Algorithm, Graph Type, and Size Category.

Graph Type	Size Category	$\overline{T}$ Random [s]	$\overline{T}$ Hyper [s]	$\overline{T}$ Genetic [s]
Complete	n10	0.014	0.000	0.015
Complete	n100	0.709	0.024	7.000
Grids	n10	0.280	0.001	0.228
Grids	n100	1.998	3.206	10.503
Mycielskian	n10	0.196	0.001	0.213
Mycielskian	n100	4.758	1.085	29.246
Trees	n10	0.129	0.000	0.025
Trees	n100	1.471	19.924	7.170

4.2 Approximation Quality Analysis

Our empirical analysis reveals that for graphs with a small number of vertices (e.g., $n = 10$), the obtained mutual-visibility set sizes consistently approximate or closely align with the established theoretical lower and upper bounds. However, for larger graph instances (eg. $n = 100$), this relationship becomes less evident. The random-sampling algorithm provides a natural baseline against which to compare more complex methods for the mutual-visibility problem. Empirical results show that, with $T = 10\,000$ trials, it often yields near-optimal estimates of the mutual-visibility number $\mu(G)$ on small (e.g. $n = 10$) and even on certain larger graphs of certain classes. However, as the order of the graph increases, the quality of the approximation deteriorates unless the number of random trials is increased proportionally. Another observation is that the lower bound obtained from the hypergraph approximation method is on average lower than the general lower bound provided by the largest degree and hence not very useful in providing tighter bounds. For

Table 1: Graph Classes for Empirical Evaluation.

Graph Class	**Key Properties / Purpose**	**Parameters Varied**	**Expected $\mu(G)$ (for validation)** [4],[6]
Category 1: Analytical Solutions			
These classes have simple, direct formulas for $\mu(G)$, or known tighter bounds obtained in recent research.			
Complete Graphs (K_n)	Densest, fully connected.	n	$\|V\|$
Trees	Sparse, acyclic, general case.	n	$\|L\|$ (set of leaves)
Grid Graphs ($\Gamma_{m,n}$)	Regular 2D lattice.	m, n	$2 \cdot \min(m, n)$ for $m, n > 3$
Tori Graphs ($T_{m,n}$)	Regular 2D lattice with wrap-around.	m, n	$\leq 3 \cdot \min(m, n)$ (exact for $m = 12, 15$)
Mycielskian of Cycles ($M(C_n)$)	Mycielskian operation on cycle graphs.	n (base)	$n + \lfloor \frac{n}{4} \rfloor$ for $n \geq 8$ (and $n + 2$ for $4 \leq n \leq 7$)
Mycielskian of Paths ($M(P_n)$)	Mycielskian operation on path graphs.	n (base)	$n + \lfloor \frac{n+1}{4} \rfloor$ for $n \geq 5$ (and 6 for $n = 4$)
Mycielskian of Star Graphs ($M(K_{1,k})$)	Mycielskian on stars (base has universal vertex).	$k + 1$ (base nodes)	$2k + 1$ for $k \geq 1$ (from $2n - 1$ for base n)
Category 2: General Graph Models			
Generalized Petersen Graphs ($G(n, k)$)	Specific highly symmetric cubic graphs.	n, k	N/A (generally unknown/no simple formula for $\mu(G)$)
Erdős-Rényi ($G(n, p)$)	Varied density, random edge distribution.	n, p	N/A (generally unknown/probabilistic)

the genetic algorithm approach the relation between the achieved μ-set size vs. general lower bound ($\Delta(G)$) can be observed from the graph in Figure 2. Note that the blue coloured data points represent analysis on general Petersen graphs, while orange data points represent Erdős-Rényi graphs.

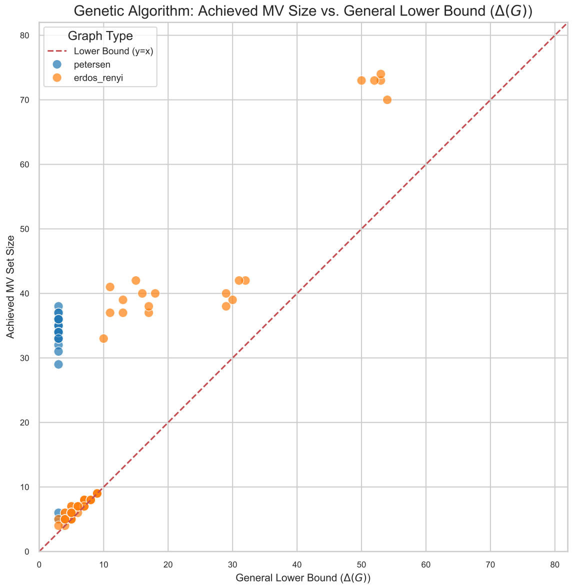


Figure 2: μ-set size vs. general lower bound for genetic algorithm approach.

5 CONCLUSION

This paper presented an empirical analysis of three distinct algorithmic approaches– a direct random heuristic, a hypergraph-based approximation, and a Genetic Algorithm– for solving the NP-complete MV problem. Our evaluation across various graph datasets revealed that for larger graph instances, the achieved solution sizes notably diverge from theoretical limits which, in the absence of tight bounds or known optimal values for larger graphs, definitively assessing the absolute quality of the found solutions becomes challenging. Nevertheless, from the validation performed on graph classes with analytically known $\mu(G)$, the Genetic Algorithm, and potentially other general heuristic approaches, empirically demonstrated superior performance among the tested methods.

6 FUTURE WORK

From the initial insights into the practical hardness of the MV problem, several promising directions for future research emerge.

Other than including other graph instances and a broader range of sizes to evaluate, a promising direction involves a deeper investigation into the correlation between specific graph properties and algorithmic performance. Beyond the general graph types studied here, future work could examine how various graph properties such as density, diameter, and average degree influence the achieved MV set size and runtime across different graph classes.

Furthermore, while this paper focused on a direct random heuristic, a hypergraph-based approximation, and a Genetic Algorithm, future research could explore other metaheuristics like Simulated Annealing, or exact methods for smaller instances, such as Integer Linear Programming, as demonstrated by authors in [5].

Finally, moving beyond theoretical graphs, a compelling avenue involves applying these algorithms to more real-world examples. This directly links back to the problem's initial motivation in robotic navigation, where algorithms for repositioning robots on a graph while maintaining mutual visibility are crucial [4].

ACKNOWLEDGMENTS

We thank Assoc. Prof. PhD Csilla Bujtás for her guidance during the initial stages of this research.

We also extend our gratitude to Assist. Prof. Dr. Uroš Čibej for his valuable advice and support throughout the preparation of this paper.

REFERENCES

[1] Davide Bilò, Alessia Di Fonso, Gabriele Di Stefano, and Stefano Leucci. 2024. On the approximability of graph visibility problems. arXiv:2407.00409 [cs.CC] https://arxiv.org/abs/2407.00409
[2] Yair Caro and Zsolt Tuza. 1991. Improved lower bounds on k-independence. *Journal of Graph Theory* 15, 1 (1991), 99–107. https://doi.org/10.1002/jgt.3190150110 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/jgt.3190150110
[3] Serafino Cicerone, Gabriele Di Stefano, Lara Drožđek, Jaka Hedžet, Sandi Klavžar, and Ismael G. Yero. 2023. Variety of mutual-visibility problems in graphs. *Theoretical Computer Science* 974 (2023), 114096. https://doi.org/10.1016/j.tcs.2023.114096
[4] Gabriele Di Stefano. 2022. Mutual visibility in graphs. *Appl. Math. Comput.* 419 (2022), 126850. https://doi.org/10.1016/j.amc.2021.126850
[5] Zahra Hamed-Labbafian, Narjes Sabeghi, Mostafa Tavakoli, and Sandi Klavžar. 2025. Three algorithmic approaches to the general position problem. arXiv:2503.19389 [math.CO] https://arxiv.org/abs/2503.19389
[6] Dhanya Roy, Sandi Klavžar, and Aparna Lakshmanan. 2024. Mutual-visibility and general position in double graphs and in Mycielskians. arXiv:2403.05120 [math.CO] https://arxiv.org/abs/2403.05120

Randomness in Bin Covering

Jakob Beber
89242105@student.upr.si
University of Primorska
Faculty of Mathematics, Natural Sciences and Information Technologies
Koper, Slovenia

Andrej Brodnik
andrej.brodnik@upr.si
University of Primorska
Faculty of Mathematics, Natural Sciences and Information Technologies
Koper, Slovenia

Rok Požar
rok.pozar@upr.si
University of Primorska
Faculty of Mathematics, Natural Sciences and Information Technologies
Koper, Slovenia

ABSTRACT

In the paper, we study the online version of the bin covering problem and the possible strategies for solving it. We later focus on the Dual Next Fit strategy, where we argue that the worst-case analysis is overly pessimistic for practical applications. To explore this, we developed a web platform for implementing and testing bin covering strategies against various sequence generators. In our empirical results, we highlight the gap between theoretical worst-case and "average-case" performance for DNF strategy.

KEYWORDS

online algorithms, bin covering, dual next fit, competition ratio

1 INTRODUCTION

In the *bin covering problem*, we are given a sequence of items with sizes between 0 and 1, which we must distribute into bins to cover as many of them as possible. A bin is considered covered if the sum of the sizes of the items placed within it is at least 1.

The content of this work is presented through a fictional problem, which will make it easier to understand the purpose, research, and results of this paper.

1.1 Motivational Example

In the village of Kosevnice, the villagers have built a special supercomputer. The computer is special in that it can save a vast amount of energy if as many of its processors as possible are running. For a processor to operate, it must have at least a minimum load. All the villagers want to use the supercomputer, but at the same time, they want to pay the lowest possible price for its use. Therefore, they have authorized the mayor to create a program that will schedule work processes on the computer. The mayor's program must distribute the work processes across the supercomputer in such a way that at any given moment, the maximum number of processors are active. This way, the mayor will ensure low computer usage costs for all villagers.

This fictional problem also models real-world problems well, where we need to distribute data among the maximum possible number of processors, where each processor requires a minimum amount of data to operate.

2 PROBLEM DEFINITION

Let us formally define the *bin covering problem*, which we are solving in the motivational example.

In the bin covering problem, we are given a sequence $\sigma = (v_1, v_2, \ldots, v_i)$ of items $v_l \in (0,1)$. We place the items v_l from the sequence σ into bins $K_1, K_2, ..., K_j$, where $1 \leq j < |\sigma|$. The fill level $\text{fill}(K)$ of a bin K is the total size of the items within it:

$$\text{fill}(K) = \sum_{v \in K} v.$$

If the fill level of a bin is at least 1, the bin becomes *covered*.

A covered bin means that a processor on the supercomputer has at least a minimum amount of work, and the mayor wants as many processors as possible to be active. The problem we are solving in the *bin covering problem* is to arrange the items into bins in such a way that the number of covered bins is maximized.

For every input sequence σ of items, there exists a solution that arranges the items into bins such that the number of covered bins is maximal. Such a solution is called an *exact solution*, and an algorithm that could achieve this is called an *optimal* algorithm, denoted by OPT. Due to the computational difficulty of the problem, the optimal algorithm is unattainable. The reason for this is that the bin covering problem has been proven to be NP-hard [1]. This complexity forces us to seek alternative approaches, such as heuristic strategies, to find good solutions efficiently.

2.1 Heuristic Strategy

A heuristic strategy is an algorithm designed to find a proficient solution. This type of strategy does not guarantee an optimal result, but rather aims to find a high-quality solution in a practical amount of time.

2.2 Competitive Ratio

To evaluate a heuristic strategy, we introduce the competitive ratio. This ratio tells us how good our strategy is compared to the optimal strategy.

Let us denote our strategy by A. The number of covered bins that strategy A returns for a given sequence σ is denoted by $A(\sigma)$.

We evaluate strategy A by its *competitive ratio* R, which is the value that satisfies the inequality

$$A(\sigma) \geq R \cdot OPT(\sigma) - C, \tag{1}$$

for all finite sequences σ and some constant C. If $C = 0$, we say the competitive ratio is *strict*, otherwise it is *asymptotic*. The competitive ratio is a positive real number $R \leq 1$, where $R = 1$ means that strategy A is optimal.

2.3 Online Problem

The mayor's program in the motivational problem must immediately assign work processes to a specific processor on the computer. If items arrive one after another and we must place each item into

---

https://doi.org/10.51939/scores25.4

a bin as soon as it arrives, we are talking about the *online version* of the bin covering problem. We add three methods to the sequence σ: 'openSequence', 'nextItem', and 'endOfSequence'. With the 'openSequence' method, we prepare the sequence for reading; the 'nextItem' method gives us the next item in the sequence; and the 'endOfSequence' method signals if we are at the end of the sequence. Unlike the offline version, in the online version, we must first open a bin if we want to place items into it. When we open a bin, it has a fill level of 0, and we say it is *active*. We must place an item into an active bin before the next item arrives. A bin is active until we close it. We close it when it becomes covered. A closed bin cannot be opened again. Items placed in bins cannot be moved.

In the remainder of this paper, we will focus only on the *online version* of the problem.

2.4 Oracle

In the online version of a problem, a strategy is inherently at a disadvantage because it must make decisions without knowing what inputs will arrive in the future. This can be seen as playing a game against an all-powerful *adversary* that can craft the input sequence in the worst possible way for the given strategy. To balance the playing field, the model of computation of this problem is sometimes augmented with an *oracle* [2].

An *oracle* is a theoretical entity that has complete knowledge of the entire input sequence. It can pass that knowledge, in a form of "advice", to the online strategy by writing it on a *advice tape.* The strategy is free to read from this tape at any point during its execution to inform its decisions. The number of bits the strategy reads from the tape is defined as the *advice complexity* [3]. This model allows us to determine how much information about the future is necessary, to achieve a better competitive ratio.

3 STRATEGIES

In the motivational example, the mayor must create a program that will best schedule the incoming work processes onto the supercomputer's processors. We present three different strategies, from simple ones to a strategy that uses advice.

3.1 Dual Next Fit

The Dual Next Fit (DNF) strategy was the first proposed strategy for the bin covering problem. It originates from its dual, Next Fit, which was used in the bin packing problem. Assmann *et al.* [1] proved that the competitiveness of this strategy is 1/2. We could have developed the strategy ourselves, as its derivation is very straightforward.

3.1.1 *Strategy Description.*

The operation of the strategy on an input sequence σ is described in the following steps:

```
First, openSequence $\sigma$.
Then, until endOfSequence $\sigma$, do the following:
  (1) Take the nextItem $v$ from the sequence $\sigma$.
  (2) If there are no active bins, open a new bin.
      Place item $v$ into the active bin and close
      it if it becomes covered.
```

We notice that with this strategy, we always have exactly one active bin.

3.2 Dual Harmonic

The Harmonic Strategy (Dual Harmonic or DH) is the next strategy we consider. The idea behind the strategy is that we can classify incoming items by size. We choose an arbitrary parameter k, which divides the items by size into k groups:

$$\left(1, \frac{1}{2}\right], \left(\frac{1}{2}, \frac{1}{3}\right], \ldots, \left(\frac{1}{k-1}, \frac{1}{k}\right], \left(\frac{1}{k}, 0\right) \qquad (2)$$

The competitiveness of the strategy is $1/2$, as proven by Epstein *et al.* [4].

3.2.1 *Strategy Description.*

We receive an item from the sequence σ and classify it into the corresponding group based on its size. During operation, we always have exactly k active bins, one for each group. We only fill a specific bin with items from the same group. For filling the bins within each group, we use the DNF strategy—we place an item into a bin, and when the bin is covered, we close it and open a new one for that group. We note that a bin for the group

$$\left(\frac{1}{i-1}, \frac{1}{i}\right], \quad 1 < i \leq k,$$

is full precisely when it contains i items, whose total size is at most $i/(i-1)$. Consequently, the surplus is at most

$$\frac{i}{i-1} - 1 = \frac{1}{i-1} \ .$$

3.3 Strategy with Precise Advice

To construct this strategy, we introduce the notion of $openSequence \rightarrow \langle advice \rangle$, a form of help to the strategy, which we defined as *advice on a tape.*

3.3.1 *Advice on a Tape.*

The strategy specifically handles items in the group $(1, 1/2]$ with the *Oracle's* help. It is sensible for half of the largest items to be in bins by themselves, rather than in pairs as the DNF strategy might suggest. Thus, for an input sequence σ, the *Oracle* provides the strategy with advice on a tape $< m, x_m >$. The *Oracle* counts all items in the sequence with a size greater than $1/2$, and half of this total count is represented by m. The size of the smallest of these top m items is represented by x_m. By identifying only half of all items larger than $1/2$, we enable the strategy to always cover at least m bins with the other half of these large items.

3.3.2 *Strategy Description.*

The strategy uses $< m, x_m >$ to open m bins, which we call *critical bins.* Initially, we set a *virtual fill* of size x_m for these critical bins. We can only place one item of size $\geq x_m$ into each critical bin. When such an item is placed in a critical bin, the bin's virtual fill is updated to the actual size of the placed item. We also choose an arbitrary parameter k, which opens k bins with the same distribution of items by size into k groups as the Harmonic strategy does in (2). The process continues as follows:

```
First, openSequence $\sigma \rightarrow < m, x_m >$.
Then, until endOfSequence $\sigma$, do the following:
  (1) Take the nextItem $v$ from the sequence $\sigma$.
  (2) If $v$ is greater than or equal to $x_m$, place
      item $v$ into the next critical bin that does
      not yet contain such an item and update
      the virtual fill of the bin.
```

(3) If v is smaller than x_m and greater than or equal to $1/k$, place the item into its corresponding size-class bin using the DNF strategy.
(4) If v is smaller than $1/k$, place the item into a critical bin whose virtual fill is not yet greater than 1, and update the virtual fill. If all critical bins have a virtual fill of at least 1, place the item in bin k using the DNF strategy.

We repeat this until we reach the ‘endOfSequence‘ on the input sequence.

This presented strategy distributes incoming items more effectively and thus manages to cover a larger number of bins. The competitiveness of the described strategy A is presented in the form of inequality (1), which was proven by Brodnik *et al.* [3]

$$A(\sigma) \geq \frac{2}{3} \cdot OPT(\sigma) - \frac{173}{60}.$$

4 DNF ANALYSIS

The DNF strategy is extremely simple, but due to its poor competitiveness, it is often overlooked. Let us examine how an adversary can force the DNF strategy to achieve a competitive ratio of 1/2, and how we can explore the competitiveness more thoroughly if we limit the adversary.

4.1 Adversarial Sequence

If, for any generated sequence, we sort the items in descending order of size, we can force the DNF strategy into its worst-case competitive performance for that specific sequence.

Since the competitiveness of strategies is measured as the minimum across all possible sequences σ, we must follow the following constraints when designing such a sequence:

(1) The sequence σ has length $|\sigma| = i$, where there are i items.
(2) We have $i/2 = n$ items of size $1/n$. We call these *small items.*
(3) Simultaneously, we have $i/2 = n$ items of size $1 - 1/n$, called *large items.*

The optimal solution for such a sequence is $OPT(\sigma) = n$. If the sequence is sorted from largest to smallest, the DNF strategy achieves a competitive result of $DNF(\sigma) = n/2$.

4.2 Random Sequences

In real-world cases, item sequences are often more random. It is unlikely that we will receive only large items for a while, followed by only the smallest ones. These items are typically intermixed.

Furthermore, restrictions based on the size of small items are unrealistic. When placing items into bins sequentially, we do not know the length of the sequence, so in practice, the size of small bins should not depend on the length of the sequence. In empirical scenarios, we define small items as those with sizes between 0 and 0.3, thus decoupling from the sequence length.

The distribution of small and large items in the “bad” sequence is modeled to disadvantage the strategy, but such distributions are rarely seen in real life. This results in sequences that are more common in nature and therefore worth analyzing in more detail.

5 PLATFORM

To test the presented strategies and arbitrary sequences on real examples, we developed a web platform [5] called “Bin Covering Interactive”[1]. The web platform allows testing or implementation of strategies, a sequence generator, or any custom script. To use or develop on the platform, knowledge of the Python programming language and understanding of the platform architecture are required.

5.1 Platform Architecture

The platform provides two base classes: `Strategy` and `Generator` (see Figure 1). Both classes are only skeletons on which any strategy or sequence generator can be built. Each class contains four mandatory methods: `init()`, `start()`, `next()`, and `stop()`. These public methods allow the two classes to communicate without revealing internal details.

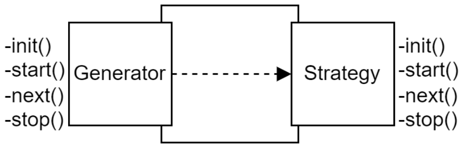


Figure 1: Operation of the Dual Next Fit strategy.

We implemented the DNF and DH strategies ourselves, along with a special item generator for which we know the result of the $OPT(\sigma)$ strategy. Thanks to the generator and the platform itself, we conducted a comprehensive comparison of the competitive ratios of the DNF strategy on different input sequences. The results are presented in Chapter 6.

5.2 DNF Strategy Implementation

The pseudocode for implementing the strategy is shown in Algorithm 1. We use the `next()` method from the `Generator` class to provide the next item from the sequence σ. We place items into `bin_load`. Each time we place an item, we check whether `bin_load` has reached the threshold to cover a bin. If so, we increment the `full_bins` counter and reset `bin_load` to 0. With the `stop()` method, we terminate the process and return the number of covered bins.

5.3 Item Generator Implementation

The implementation of the generator is available on the platform. In pseudocode, we focus on the most important method in the class, `generate_numbers`, presented in Algorithm 2. The method takes the parameter `bins_covered`, which defines how many bins the $OPT(\sigma)$ strategy will cover for the generated sequence. Items are generated randomly and subtracted from a full bin. When the bin's remaining space reaches a predefined `threshold`, we determine the size of the last item to exactly fill it. The generator sends one

---

[1] The project can be publicly accessed on http://4xww0fuh2a1u96pm.myfritz.net:5000/

```
Input: Class $Generator$
Output: Number of full bins
Function init():
    Data: $bin\_load \leftarrow 0$
    Data: $full\_bins \leftarrow 0$
Function start($generator$):
    $self.gen \leftarrow generator$
Function next():
    $bin\_load \leftarrow bin\_load + item$
    if $bin\_load > BIN\_COVER\_LOAD$ then
        $full\_bins \leftarrow full\_bins + 1$
        $bin\_load \leftarrow 0$
    end
Function stop():
    return $full\_bins$
```

Algorithm 1: Generator Class Implementation

item at a time using the `Generator.next()` method. Once all items have been sent, we call `Generator.stop()` to receive the number of bins covered.

```
Input: Integer $bins\_covered$
Output: List of generated numbers $numbers$
Function generate_numbers($bins\_covered, \_max\_load$):
    $numbers \leftarrow []$
    $i \leftarrow 0$
    $threshold \leftarrow 0.3$ ;                    // fixed threshold
    while $i < bins\_covered$ do
        $random\_number \leftarrow$ get_number($(0.3 * (\_max\_load - 1)), \_max\_load$)
        $bin\_load \leftarrow bin\_load - random\_number$
        if $bin\_load < (threshold * \_max\_load)$ then
            $numbers$.append($random\_number$)
            $numbers$.append($bin\_load$)
            $bin\_load \leftarrow \_max\_load$
            $i \leftarrow i + 1$
        end
        else
            $numbers$.append($random\_number$)
        end
    end
    return $numbers$
```

Algorithm 2: Generate Numbers Function

6 RESULTS

In this work, we focused on how the DNF strategy behaves on random input sequences σ. Table 1 shows the configuration options for generating items used in the `Generator` class. Using these options, we created input sequences and then applied transpositions to them. Transpositions were performed in such a way that no item remained in its original position.

Table 1: Item size selection probabilities during sequence generation

Item size in %	Selection probability in %
0 - 30	50
30 - 70	8.3
70 - 100	41.7

In our experiments, we observed how the average competitive ratio changes as the length of the sequence $|\sigma|$ increases. The results (see Table 2) were somewhat surprising. Despite increasing $|\sigma|$ and the number of permutations applied to the sequence, the competitiveness of the strategy remained relatively stable. We also observed that the DNF strategy performs significantly better on random input sequences.

Table 2: Results of the DNF strategy compared to $OPT(\sigma)$

$\lvert\sigma\rvert$	Runs	Seeds	$OPT(\sigma)$	$avg(DNF(\sigma))$	Avg in %
841	100	1	300	228.42	76.14
1682	100	1	600	458.35	76.39
3393	100	1	1200	915.67	76.31
6713	100	1	2400	1828.57	76.19
13517	100	1	4800	3655.13	76.15
14192	100	1	5000	3814.55	76.29
28434	100	1	10000	7635.69	76.36
28301	1000	1	10000	7630.39	76.30
28419	1000	1	10000	7634.68	76.35
56607	100	1	20000	15263.58	76.32
56462	1000	1	20000	15248.22	76.24
113218	100	1	40000	30524.96	76.31
142030	1000	1	50000	38170.94	76.34
226920	100	1	80000	61071.49	76.34
283457	100	1	100000	76336.49	76.34
283717	1000	1	100000	76340.72	76.34
850230	100	1	300000	228969.36	76.32
1416167	1000	1	500000	381605.29	76.32

7 CONCLUSION

Despite our empirical findings showing a much better competitive ratio for random sequences, the mathematical proof of tighter bounds for non-adversarial inputs remains an open problem. The web platform offers us a valuable tool to implement and evaluate, both existing and novel strategies under various sequence distributions.

REFERENCES

[1] Susan F Assmann, David S. Johnson, Daniel J. Kleitman, and JY-T Leung. 1984. On a dual version of the one-dimensional bin packing problem. *Journal of algorithms* 5, 4 (1984), 502–525.

[2] Joan Boyar, Lene M Favrholdt, Shahin Kamali, and Kim S Larsen. 2021. Online bin covering with advice. *Algorithmica* 83 (2021), 795–821.

[3] Andrej Brodnik, Bengt J Nilsson, and Gordana Vujovic. 2023. Online bin covering with exact parameter advice. *arXiv preprint arXiv:2309.13647* (2023).

[4] Leah Epstein, Lene M Favrholdt, and Jens S Kohrt. 2012. Comparing online algorithms for bin packing problems. *Journal of Scheduling* 15 (2012), 13–21.

[5] J. Beber. 2025. Bin Covering. https://github.com/jakobb01/BinCovering/tree/main/web_interface. Accessed: 2025-06-28.

Proc. of the 11th Student Computing Research Symposium (SCORES'25), Ljubljana, Slovenia, September 30, 2025

Efficacy of Reduction Rules for Hitting Set Problem

Eva Jug
Jan Novak Domijan
UL FRI

Abstract

We give a brief overview of the hitting set problem and propose a kernelization algorithm based on five reduction rules. We present the implementation of the proposed algorithm and analysis of the performance of each reduction rule on a set of problem instances provided by the PACE challenge. Lastly, we compare the runtime of our implementation with that of a state–of–the–art SAT solver.

Keywords

Hitting set, hypergraph, kernelization, python, algorithm

1 Introduction

The *hitting set problem* is a generalization of one of the most famous NP-hard problems, the vertex cover problem. It can be stated as such: given a collection S of sets $S_1, \ldots, S_m$, what is the smallest set that intersects every set of S? In this text we will use the graph theoretic formulation of the same problem where S defines the hypergraph $G = (V, E)$, $V = \bigcup_{i=1}^{m} S_i$, $E = \{S_1, \ldots, S_m\}$, and we want to find the smallest set of vertices T such that $T \cap S_i \neq \emptyset$ for all $S_i \in S$. The size of an instance of the hitting set problem is the number of vertices of the hypergraph.

While the problem is NP-hard, it is feasibly solvable for small enough instances, and some larger instances can be reduced using *kernelization algorithms.* A kernelization algorithm can very broadly be defined as a polynomial time process that takes an instance of a problem and produces a smaller instance with the same solution or gives some additional information about the solution. It is commonly made up of multiple *reduction rules* that aim to achieve a similar purpose as a kernelization algorithm but usually have no proper theoretical size guarantee on their own. In this paper we present a kernelization algorithm made up of five reduction rules which we implemented. We then used the implementation to analyze the performance of different combinations of reduction rules on a test set provided by the PACE challenge.

2 Reduction rules

We first added a very basic rule that we called *the loop rule* which would add any vertex v to the cover if it was the only vertex in some hyperedge. The rule is essential for the soundness of the other reduction rules. On top of that, we decided to implement the following four rules.

2.1 Vertex domination rule

Definition 1. *We define edge-set for vertex v in a hypergraph as $ES(v) = \{e \in E \mid v \in e\}$.*

The vertex domination rule [2] says that we remove a vertex $v \in V$ if $\exists v' \in V : ES(v) \subset ES(v')$. This stems from the idea that if we add v' to our cover, we will cover **at least** all those hyperedges that would be covered by adding v to our cover.

Definition 2. *We define an injective numbering function $f_v : V \longrightarrow \{1, \ldots, n\}$ that maps every vertex in a hypergraph to a distinct natural number where $n = |V|$.*

Now we can also take into account the special case in which $\exists v, v' \in V : ES(v) = ES(v')$. It is obvious that in this case we need only keep one of the vertices v, v' in our hypergraph but to be consistent we always remove the vertex with the smaller f_v value, i. e. v such that $\exists v' \in V : ES(v) = ES(v') \wedge f_v(v) < f_v(v')$. This way we guarantee that at least one such vertex will always remain in the hypergraph.

2.2 Hyperedge dominating rule

Definition 3. *We define an injective numbering function $f_e : E \longrightarrow [m]$ that maps every hyperedge in a hypergraph to a distinct natural number where $m = |E|$.*

The hyperedge dominating rule [2] says that we remove hyperedges which are a superset of some other hyperedge. To take into account hyperedges that describe the same set of vertices, we use the f_e value. We remove any hyperedge $e \in E$ such that $\exists e' \in E : e' \subset e \vee e' = e \wedge f_e(e') < f_e(e)$ holds. This is well-defined since covering the hyperedge e' means adding a vertex v that is in both hyperedges e and e'.

2.3 Degree-2 hyperedge branching rule

The degree-2 hyperedge branching rule [7] is a rule that adds a vertex to the cover if the edges of degree 2 that contain it form a suitable substructure.

Define the set $IN(v) = \{v' \mid v' \in V \wedge \exists e \in E : v, v' \in e \wedge |e| = 2\}$ for any $v \in V$. Given a vertex $v \in V$, we can categorize hyperedges that contain v into three distinct types:

- **Type 1**: $e \in E : v \in e \wedge |e| = 2$,
- **Type 2**: $e \in E : |e| > 2 \wedge \exists v' \in IN(v) : v, v' \in e$,
- **Type 3**: $e \in E : |e| > 2 \wedge \forall v' \in IN(v) : v \in E \wedge v' \notin E$.

Note that we can ignore hyperedges of size 1 containing v since it will be categorized by our algorithm as a *loop* and the vertex v will automatically be added to the cover.

We can now count the number of the first three types of hyperedges:

- for **Type 1** we use the notation $\delta^2(v)$ *(as is used in [7]),*
- for **Type 2** we use the notation k,
- for **Type 3** we use the notation ℓ.

If we don't add the vertex v to the cover, we must add all $v' \in IN(v)$ to the cover in order for all hyperedges of **Type 1** to be covered. In this case, we cover all **Type 1** and **Type 2** hyperedges, as well as some m hyperedges e for which it holds that $v \notin e \wedge \exists v' \in IN(v) : v' \in e$. We denote this as $m = |\{e \in E \mid v \notin e, \exists v' \in IN(v) : v' \in e\}|$. This means that we have covered exactly $\delta^2(v) + k + m$ hyperedges by adding $\delta^2(v)$ vertices to the cover.

---

https://doi.org/10.51939/scores25.5

Alternatively, if we add v to the cover, we have covered exactly $\delta^2(v) + k + \ell$ hyperedges by adding 1 vertex to the cover. In order for this coverage to be comparable to the previous case, we have to add **at most** m vertices to the cover to cover **at least** $\delta^2(v) + k + \ell + m$ hyperedges. Note that we may choose the exact right m vertices to cover the exact m hyperedges from the previous case. This means that we have covered **at least** as many hyperedges with $m + 1$ vertices as we did with $\delta^2(v)$.

Ergo, if $m + 1 \leq \delta^2(v)$ or simplified $m < \delta^2(v)$, without loss of generality, we can add the vertex v to the cover.

2.4 Crown decomposition

Crown decomposition, first introduced by [4], is a kernelization technique that uses independent sets to reduce the number of vertices in the input graph.

A crown decomposition is defined as a triple (I, H, M) where:

- I is an independent set,
- H is the neighbourhood of I in G,
- M is a maximal matching on the bipartite graph $G_{I,H}$ with vertices $I \cup H$ and edges $(u, v) \in E \cap (I \times H)$.

Lemma 1. *Let G be a graph with a crown decomposition $C = (I, H, M)$. If $|M| = |H|$, then the vertices in H can be added to the cover and the vertices in I can be removed from the graph.*

Proof. For each edge of M at least one endpoint must be contained in a minimal cover. That endpoint can either be a member of I or a member of H. No matter the choice of endpoint, H is covered as $|M| = |H|$. By choosing an endpoint v in I we cover only edges between I and H, but those edges would also be covered if we only chose endpoints of H as $I \subseteq N(H)$. Thus, by choosing to add H to the cover we cover all the edges between I and H and possibly some additional edges between H and $N(H) \setminus I$. As $N(I)$ is covered, the vertices I can be removed from the graph. □

The concept of a crown decomposition can be generalized to hypergraphs by altering the definition of the set H.

A crown decomposition for a hypergraph $G = (V, E)$, is a triple (I, H, M), where:

- I is an independent set,
- H is the set $\{h \mid \exists x \in I.\, h \cup \{x\} \in E\}$,
- and M is a maximal matching of the bipartite graph $G_{I,H}$ with $V(G_{I,H}) = I \cup H$ and $E(G_{I,H}) = \{(x, h) \in I \times H \mid \{x\} \cup h \in E\}$.

Lemma 2. *Let G be a hypergraph with a crown decomposition $C = (I, H, M)$. If $|M| = |H|$, then the vertices in I can be removed from the graph and the edge set of I can be replaced by H.*

The proof is similar to the previous proof except H is added to the edge set of G instead of the current cover. Both definitions are equivalent for regular graphs if the loop rule is employed as H would constitute only singleton sets.

While the definitions outline the structure of a crown decomposition we still need a constructive way of finding large enough independent sets.

The authors of [2] used the concept of *weakly related edges* to find an independent set on hypergraphs which can be used to produce a crown decomposition. We present a brief overview of their construction and results.

Definition 4. *Two hyperedges are weakly related if they intersect in at most $d - 2$ elements. A hyperedge e and a set W are weakly related if each element of W is weakly related to e.*

A maximal set W of weakly related edges is constructed by first adding all edges of size less than $d - 2$ and then adding each remaining edge if it is weakly related to W.

Then to bound the number of vertices in W apply the rule defined by Algorithm 1.

Algorithm 1: High Occurrence rule

```
for $i = d - 2, \dots, 1$ do
    for $e \subseteq E : E \in W, |e| = i$ do
        $W_e = \{S \in W \mid e \subseteq S\}$;
        if $|W_e| > k^{d-1-i}$ then
            $W = W \setminus W_e$;
            $E(G) = E(G) \setminus W_e$;
            $W = W \cup \{e\}$
```

For a proof of soundness refer to [2]. The key detail is that after applying the high occurrence rule $|W| \leq k^{d-1}$.

To construct a crown decomposition define the sets $I = V \setminus V(W)$ and $H = \{h \mid x \in I \wedge \{x\} \cup h \in E\}$. Observe that I is an independent set. If there existed an edge $e \notin W$ that connected two vertices u, v of I, then the edge would contain 2 vertices of I and at most $d - 2$ vertices of $V(W)$ but then e could be added to W which contradicts its maximality.

For the last step find a maximal matching M on $G_{I,H}$ using an appropriate matching algorithm. For more details on finding a suitable matching on $G_{I,H}$ bipartite graph refer to [2] and [1].

Theorem 1. *After applying the crown decomposition rule G contains at most $O(k^d)$ vertices.*

Proof. The set H contains only subedges of size $d - 1$ otherwise the corresponding edge could be added to W which would contradict it maximality. Additionally, any edge that intersects I must have an edge of size d in W that intersects it in $d - 1$ elements. It follows that $|V(H)| \leq dk^{d-1}$ as W has at most k^{d-1} edges of size d and each of those edges has d subsets of size $d - 1$.

Now if $|I| \leq |H|$, then $|V(G)| = |I| + |V(W)|$ where $|I| \leq |H| \leq |V(H)|$. Consequently $|V(G)| \leq |V(H)| + |V(W)| \leq dk^{d-1} + k^{d-1}$. But if $|I| \geq |H|$ then I is removed and H is added. Hence $|V(G)| = |V(H)| + |V(W)| \leq dk^{d-1} + k^{d-1}$. In both cases $|V(G)| \in O(k^d)$. □

3 Algorithm

The algorithm proceeds as follows:

1. Return None if the current cover is larger than the smallest valid cover found so far.
2. Apply each rule until no rules can be applied and only apply the crown decomposition rule when d is changed.
3. Repeat (1).
4. Choose a vertex of maximum degree v and recursively solve the instances $(G \setminus v, c + v)$ and $(G \setminus v, c)$.

(5) Return the smallest valid cover found by the recursive calls, if both calls return None then return None.

Essentially, each rule is applied and then two instances are recursively solved: one where the highest degree vertex is added to the cover and the second where it is removed from the vertex set and every edge. The algorithm is among the simpler algorithms for solving the hitting set problem. For an overview of more complex algorithms refer to [6]. Correctness of the algorithm comes from the correctness of each rule individually.

Implementation details

Our implementation is written in the python programming language and uses the numpy library for storing and manipulating hypergraphs and the scipy library for calculating a maximum matching on bipartite graphs. Hypergraphs are stored in a $|V| \times |E|$ boolean matrix with columns representing edges and rows representing the edge set of each vertex. The matrix will in most cases be quite sparse and could benefit from the use of a sparse matrix representation. But we opted for a dense representation as only $O(|V|)$ matrices are held in memory at any given time leading to a space complexity of $O(|E||V|^2)$ which is manageable given our test set.

Storing graphs as a boolean matrix allows for efficient implementation of common graph operations using boolean operators: the union of two edges can be obtained using the bitwise and operator, the intersection can be obtained using the bitwise or operator, equality can be checked using the bitwise xor operator etc. Additionally, using rows and columns of the matrix as boolean masks allows for efficiently calculating the edge set of vertices, the vertex set of edges and the neighbourhood of a set of vertices. All three operations are used extensively in the crown decomposition rule and the degree two rule.

We also considered two possible improvements to the algorithm that were later cut due to time constraints. The first is parallelization. The problem lends itself well to parallelization as each instance is essentially independent. A simple way to achieve parallelization would be to implement a queue with atomic operations which would hold instances in the form (G, c, b), where G is a hypergraph, c the current cover and b a flag indicating that d was changed during branching. To further speed up execution an atomic counter could be introduced for the current smallest size of cover. If a thread finds a valid cover of size k, it can update the counter, and any instance whose cover size is greater than k can be discarded. Note that such a counter would not impact the soundness of the algorithm.

The second was using a different branching rule. In its current state the algorithm chooses a vertex of highest degree and either adds it to the cover or removes it. But the correctness of the algorithm does not depend on the order of vertices chosen. We considered two additional branching rules but were unable to test their efficacy due to time constraints.

4 Methodology

Our starting point for this project was the *PACE Challenge*[1]. They had outlined the problem and provided some basic resources which we could then work off of. The challenge for the hitting set problem was split into two tracks, the exact track and the heuristic track. Here we focus on the former.

The *PACE Challenge* also provided links to test sets appropriate for every given track. We made use of two different test sets. We used the verifier with a small test set[2] to ensure every rule in our algorithm worked as it should and used the public instances for the Hitting Set Problem[3] to compare the efficiency of our algorithm to that of others. We will refer to the former as **testset1** and the latter as **testset2**.

testset1 is comprised of simpler problems with at least 6 and at most 311 vertices and at least 5 and at most 309 hyperedges. Meanwhile **testset2** is made up of problems containing at least 200 and at most 8340 vertices and at least 546 and at most 8340 hyperedges. Note that, being the verifier, **testset1** is the only one problem set with public solutions.

At first, we measured how much time it took to solve a given problem from the chosen test set. We then also chose to count the number of vertices and edges each rule removed as runtime by itself does not give a good overview of how much each individual rule contributed. When counting with larger instances we gave a set amount of time before ending computation.

Before testing various subsets of rules we considered some hypotheses. The simple rules — the vertex domination, the edge domination and the loop rule — would reduce total runtime. The crown decomposition rule would increase total runtime with the exception of larger hypergraphs. The degree-2 rule would rarely be used as it has strict conditions which must be met.

5 Comparison to standard SAT solver

We also used the Python library *PySAT* and the solver *cd* to see how well our algorithm compares to the current standard. Note that *PySAT*'s module *solvers* is a Python wrapper for the code originally implemented in the C/C++ language[4].

Current runtime for **testset1**[5] for the standard SAT solver is less than a second (around 0.273 s). The current runtime for **testset2**[6] for the standard SAT solver is unknown due to the algorithm taking more time than expected. It managed to solve the first two problems in under 2 s but got stuck on the third problem. Both runtimes also include print statements for each problem.

The best current runtime for the category **Hitting Set - Exact** from the *PACE Challenge*[7] is 1, 696.90 s[8].

6 Results

We present our results in the form of two tables. Both tables use data from **testset1** as both algorithms observed in our experimentation had difficulty solving all the problems of **testset2**.

Table 1 represents how many vertices were removed by the vertex domination rule, loop rule and degree-2 rule, how many edges were removed by the edge domination rule and how many

---

[1] https://pacechallenge.org/

[2] https://github.com/MarioGrobler/hs_verifier

[3] https://github.com/MarioGrobler/PACE2025-instances/tree/master/hs

[4] https://pysathq.github.io/docs/html/

[5] https://github.com/MarioGrobler/hs_verifier/tree/main/Hitting%20Set%20Verifier/src/test/resources/testset

[6] https://github.com/MarioGrobler/PACE2025-instances/tree/master/hs/exact

[7] https://pacechallenge.org/2025/hs/

[8] https://optil.io/optilion/problem/3221#tab-4

	Rules				
Test problem	Vertex dom.	Edge dom.	Loop	Degree-2	Crown decomp.
test	1	4	1	1	0
bs20	39	48	27	2	0
bs50	8333	9354	6928	257	0
bs100	226281	267243	210006	6586	0
bs150	220144	264828	204523	6426	0
bs200	33165	42359	33356	688	0
bs250	8	10	2	0	0
bs300	21	23	4	0	0
Total	487992	583869	454847	13960	0

Table 1: Number of times each rule was successfully used in 15 minutes

	Subset of Rules				
Test problem	No rules	Simple rules	Degree-2	Crown	Baseline SAT solver
test	0.0021	0.001	0.0011	0.001	0.001
bs20	27.9421	0.051	0.0797	0.5564	0.002
bs50	TO	25.2136	25.9055	26.7124	0.004
bs100	TO	TO	TO	TO	0.008
bs150	TO	TO	TO	TO	0.0162
bs200	TO	TO	TO	TO	0.0116
bs250	TO	TO	TO	TO	0.0745
bs300	TO	TO	TO	TO	0.1557
Total	TO	TO	TO	TO	0.273

Table 2: Run time in seconds for different subsets or rules

vertices or edges were removed by the crown decomposition rule. It is clear from our testing that the three simple rules — vertex domination, edge domination and loop rule — are used frequently along with the degree-2 rule. We also observed that the crown decomposition rule removed no vertices or edges. Given the results we more closely analyzed why no valid crown decomposition was found. During all tests only two scenarios were accounted for: either $W = V(G)$ and I was empty, or a crown decomposition was found but $|H| > |I|$. This would indicate that the values of k and d were large enough to not exclude any instances where $|W| > k^{d-1}$ and that the graphs did not contain a suitable substructure to apply the crown decomposition rule.

If we take a look at the instances bs200, bs250 and bs300 we see that very few rules were applied. This is because most of the time was spent on the crown decomposition rule which when combined with a timeout window (TO) of 15 minutes meant that the other rules were applied less. A possible solution to improving the performance of the crown decomposition rule would be to only run it when a sufficiently small cover has already been found.

Table 2 shows the time it took for our algorithm to solve a given problem using different subsets of rules. The following subsets of rules were used:

- *No rules*: our algorithm with nothing but basic branching,
- *Simple rules*: our algorithm with the vertex domination, edge domination and loop rules,
- *Degree-2*: basic rules with added degree-2 rule,
- *Crown*: basic rules with added degree-2 rule and crown decomposition.

We also compared these times to the time it took for the PySAT solver to solve the problems.

We can see that the simple rules significantly reduce run time. The crown decomposition rule strictly increases runtime which is sensible as it removes no vertices or edges. The degree-2 rule does reduce run time in instance bs50 but it marginally increases runtime in bs20. We also observe that the PySAT solver was significantly faster than our implementation. It was able to solve all instances in testset1 while our implementation only solved the three smallest ones.

7 Conclusion

In conclusion, the simpler rules that focused on smaller substructures of the hypergraph significantly decreased total runtime while the crown decomposition rule remained unused despite its strong size guarantee. Another observed downside of the crown decomposition rule was its complexity and large run time.

In terms of efficiency, our implementation ended up being significantly slower than a state–of–the–art SAT solver. Despite the relative speed neither solver finished a single problem when run on the more complex instances of the PACE challenge.

The next step of our research would be to include a larger number of reduction rules along with different base algorithms. While there are not as many reduction rules specific to the hitting set problem compared to the vertex cover problem, any reduction rule for the vertex cover problem could be adapted to the hitting set problem.

References

[1] Faisal Abu-khzam, Michael Fellows, Michael Langston, and W. Suters. 2007. Crown Structures for Vertex Cover Kernelization. *Theory Comput. Syst.* 41 (10 2007), 411–430. https://doi.org/10.1007/s00224-007-1328-0

[2] Faisal N. Abu-Khzam. 2010. A kernelization algorithm for d-Hitting Set. *J. Comput. System Sci.* 76, 7 (2010), 524–531. https://doi.org/10.1016/j.jcss.2009.09.002

[3] C. Berge. 1984. Hypergraphs: combinatorics of finite sets, Vol. 45. *Elsevier* (1984).

[4] Benny Chor, Mike Fellows, and David Juedes. 2005. Linear Kernels in Linear Time, or How to Save k Colors in O(n2) Steps. In *Graph-Theoretic Concepts in Computer Science*, Juraj Hromkovič, Manfred Nagl, and Bernhard Westfechtel (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 257–269.

[5] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. 2019. *Kernelization: Theory of Parameterized Preprocessing.* Cambridge University Press.

[6] Andrew Gainer-Dewar and Paola Vera-Licona. 2016. The minimal hitting set generation problem: algorithms and computation. arXiv:cs.DS/1601.02939 https://arxiv.org/abs/1601.02939

[7] Lei Shi and Xuan Cai. 2010. An exact fast algorithm for minimum hitting set. In *2010 Third International Joint Conference on Computational Science and Optimization*, Vol. 1. IEEE, 64–67.

Proc. of the 11th Student Computing Research Symposium (SCORES'25), Ljubljana, Slovenia, September 30, 2025

A Model for Evaluating Parkinson's Disease Severity Through the Finger-Tapping Test

Neža Kržan
nk08882@student.uni-lj.si
Faculty of Mathematics and Physics,
University of Ljubljana
Jadranska cesta 21
SI-1000 Ljubljana, Slovenia

Jure Žabkar
jure.zabkar@fri.uni-lj.si
Faculty of Computer and Information Science,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

ABSTRACT

We developed a classification model based on the criteria described in the Movement Disorder Society–Unified Parkinson's Disease Rating Scale (MDS-UPDRS). Using the descriptions provided in the MDS-UPDRS scale, we generated synthetic finger-tapping signals to train deep learning models. These models were then applied to real data obtained from video recordings, where hand movements were tracked using the MediaPipe Hand tool. The scores define three key anomalies: decrease in amplitude, decrease in speed, and signal interruptions. We mathematically defined these anomalies and modeled each of them separately using three simple autoencoders. The latent spaces of the autoencoders were used as input to the k-nearest neighbors (kNN) method, which determined the severity level for each anomaly. For new cases, no additional training is required. Instead, the observed finger-tapping signal is compared to a set of synthetically generated signals, and the kNN method identifies the closest predefined signal, enabling an assessment consistent with the MDS-UPDRS criteria. The highest accuracy was achieved by the model predicting the decrease in speed, although defining this anomaly numerically is challenging due to its purely textual description in the MDS-UPDRS scale. Overall, the MDS-UPDRS score classifier achieved an accuracy of 87%.

KEYWORDS

Parkinson's disease, Finger-Tapping test, Autoencoder, Classification

1 INTRODUCTION

Parkinson's disease is a chronic neurodegenerative disorder of the central nervous system that affects between 1%-2% of the global population [27]. Despite extensive research, its origin and effective treatment remain unclear. The disease causes weakening, damage, and death of nerve cells (neurons) in parts of the brain that are crucial for controlling motor functions. As a result, patients experience characteristic symptoms such as movement difficulties, tremors, muscle rigidity, and balance disorders, which significantly hinder daily life [18].

To diagnose Parkinson's disease, several diagnostic tests are used today, among which one of the most commonly applied is the finger tapping test. This test is designed to assess motor functions, particularly movement speed, coordination, and bradykinesia. During the test, the participant touches their thumb with their index finger, then moves it as far away as possible before returning it to the starting position, with each such movement representing one cycle. The goal is either to perform as many cycles as possible within a specific time period (e.g., 10 seconds) or to complete a predetermined number of cycles (e.g., 10 to 15) with maximum speed and accuracy. The evaluation considers speed (number of taps within a given time), consistency of movements, the presence of abnormalities (such as slowness, interruptions, or irregular movements), and decrease in movement amplitude [6]. Individual anomalies in tapping performance are scored on the Movement Disorder Society-Unified Parkinson's Disease Rating Scale (MDS-UPDRS [19]), ranging from 0 to 4:

(0) No problems.
(1) Any of the following: a) the regular rhythm is broken with one or two interruptions or hesitations of the tapping movement; b) slight slowing; c) the amplitude decrements near the end of the 10 taps.
(2) Any of the following: a) 3 to 5 interruptions during tapping; b) mild slowing; c) the amplitude decrements midway in the 10-tap sequence.
(3) Any of the following: a) more than 5 interruptions during tapping or at least one longer arrest (freeze) in ongoing movement; b) moderate slowing; c) the amplitude decrements starting after the 1st tap.
(4) Cannot or can only barely perform the task because of slowing, interruptions, or decrements.

Although the test is simple and widely used, its main limitation is subjectivity, as the final assessment depends on the neurologist, who must carefully observe the participant's movements and identify any abnormalities; otherwise, there is a risk of an incorrect evaluation.

Our goal was to develop an approach that would enable a better interpretation of the model's classifications based on finger movement and to define the relationship between signal features and the MDS-UPDRS rating scale. By analyzing finger movement signals, we aim to determine whether the curves can capture anomalies (such as changes in amplitude, speed, and signal interruptions) that can be mathematically described. By manipulating and generating signals containing these anomalies, we create a larger dataset, which is then used for training models. In this way, we aim to establish a connection between the textual description of the score and the numerical MDS-UPDRS rating, as well as to provide an explanation of why a specific score is assigned to a given signal.

https://doi.org/10.51939/scores25.6

2 RELATED WORK

Studies on Parkinson's disease often focus on various symptoms, with finger tapping being one of the most commonly analyzed tasks [13]. The main research goals include disease severity assessment (MDS-UPDRS), comparing patients with healthy controls, and analyzing specific symptoms such as tremor or bradykinesia [2].
Data collection methods vary, ranging from keyboard tapping [1, 20] to wearable sensors like accelerometers [14, 23], or video analysis of finger tapping [15–17, 21], often using tools such as MediaPipe Hand [7, 10, 26], OpenPose [10, 11, 21] or DeepLabCut [15].
Models are built either on manually extracted features based on MDS-UPDRS criteria [3, 7, 10–12, 26] or using deep learning methods [4, 9]. Demographic factors like age and gender are frequently included [17].
Most studies use a single multi-class classifier predicting MDS-UPDRS scores [3, 8–12, 15, 21, 26], while others apply sets of binary classifiers or two-stage approaches, where the first model distinguishes healthy individuals from patients, and the second estimates the score [7, 16, 17, 25]. The most common algorithms include SVM [3, 8, 17, 26], Random Forests [3, 8, 9, 16, 17, 21, 26], Decision Trees [15, 17, 26] and k-Nearest Neighbors [3, 8, 17].

3 METHODOLOGY

By manipulating anomalies in finger-tapping signals and generating new signals, we created a dataset that captures different types of anomalies and their possible ranges. This process also provided anomaly coefficients, which were used for classifying scores according to the MDS-UPDRS scale. Using three autoencoders, we compressed the generated signals into a latent space, which was then used as input data for predicting the anomaly coefficients. In this way, we obtained the expressed intensity of each anomaly, providing an explanation for the assigned MDS-UPDRS score.

3.1 Dataset

We had 183 videos from 91 different participants. The level of motor impairment in the videos was rated on the MDS-UPDRS scale from 0 to 4 by a neurologist. For each participant, two videos were recorded: one of the left-hand tapping and one of the right-hand tapping. When determining the level of motor impairment based on the finger-tapping test, the dominant hand was evaluated [28]. Videos were obtained for all five MDS-UPDRS scores.

Table 1: Number of finger-tapping videos by MDS-UPDRS scores and their percentage.

MDS-UPDRS	0	1	2	3	4	SUM
Number of videos	49	51	53	23	7	183
Percentage	26%	28%	29%	13%	4%	100%

The videos were processed using the MediaPipe Hand tool. From the extracted skeletons, we calculated the Euclidean distances between the fingertips of the thumb and index finger, as well as between the joints at the tips of the thumb and index finger, thereby obtaining the raw finger-tapping signals (amplitudes) for each video. Missing values in the signals were interpolated if both preceding and following values existed, with the interpolated value calculated as the average of the neighboring points. Since MediaPipe Hand does not always detect all 21 key points accurately, the skeleton can shift between frames, introducing noise into the data. Additionally, individuals with Parkinson's disease exhibit tremor during finger tapping, which also introduces noise. The noise was removed, and the signals were smoothed using a moving average method. All signals were then normalized based on their minimum and maximum values, mapping them onto the interval $[0, 1]$. The signals were truncated to 15 ([6]) repetitions or, in some cases, to their maximum length.

3.2 Data Generation

We mathematically described three main types of anomalies: the number of interruptions in the signal, the decrease in amplitude at a specific repetition based on the score, and the decrease in tapping speed. For each anomaly type, we generated separate sets of signals — signals containing only amplitude decrease, only speed decrease, or only interruptions.
First, we created a signal simulating a basic finger-tapping signal without noise or tremor, modeled as a sinusoidal function

$$\text{signal} = \frac{sin(2\pi \cdot f \cdot t) + 1}{2}, \tag{1}$$

where f is the base frequency and t is time (in seconds). The generated signals are 10 seconds long, corresponding to the standard test duration, during which a participant typically performs 10–15 repetitions [6]. The initial tapping frequency was randomly selected from the interval $[1.5, 5]$ ($f \sim U([1.5, 5])$ Hz); for healthy older adults (approximately 50 years or older), the average tapping frequency ranges between 3.8 Hz and 4.7 Hz [22], whereas for individuals with Parkinson's disease, the frequency can decrease to approximately 1.5–3 Hz, depending on disease severity [24].
The amplitude decrease was modeled using an exponential decay after a specified repetition. The speed decrease was defined as a gradual reduction in tapping frequency. Interruptions were added as periods (minimum 0.3 seconds [5, 9]) where the signal drops to zero, with randomly selected durations and occurrence counts.
For each anomaly type, we used extreme boundary values to cover the widest possible range of anomalies. During signal generation, we also get the coefficients of individual anomalies (e.g., the repetition at which amplitude or speed decrease begins, and the number of interruptions), which were normalized based on the minimum and maximum possible values for each anomaly type. We generated a dataset of $n = 100,000$ signals for each anomaly type.

3.3 Autoencoders

We built an autoencoder with a simple architecture for signal reconstruction. Each type of signal (signal with amplitude decrease, signal with speed decrease, and signal with interruptions) was treated separately, meaning that a separate model with the same architecture was trained for each signal type. The basic idea of the autoencoder was to project the input data into a latent space and then reconstruct the input signal from it. In this way, the model learned the characteristics of the anomalies, compressed them into the latent space, and this latent representation was later used as the input for predicting the coefficients of individual anomalies.
The autoencoder (architecture shown in the Figure 1) consists of

an encoder and a decoder. The encoder is composed of a sequence of fully connected linear layers with ReLU activation functions and intermediate Batch Normalization (`BatchNorm1d`, BN). The decoder has a similar architecture to the encoder, except for the last layer, which uses a sigmoid activation function because the signals were normalized to the range $[0, 1]$.

The latent space has a dimensionality of 16 ($\mathbb{R}^{16}$), chosen empirically through experiments with different latent space sizes. Dimensions below 16 resulted in noticeably poorer signal reconstruction, especially for signals containing interruptions, while higher dimensions did not significantly improve reconstruction.

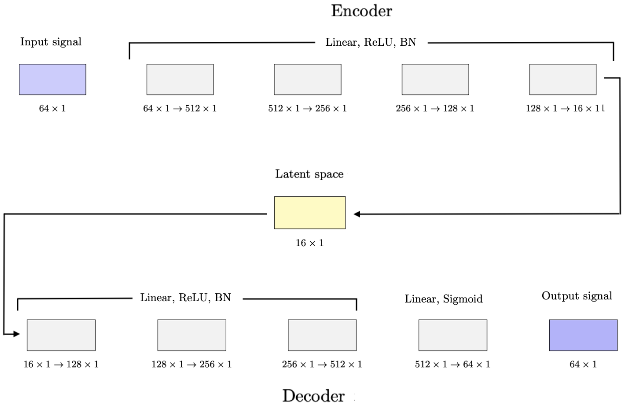


Figure 1: Autoencoder architecture.

The input data consisted of time series signals generated using the method described in Section 3.2. The signals were segmented into time windows using a sliding window approach with a window size of 64 and a step size of 16, meaning the windows overlapped to better capture the temporal structure of the signal. Segmenting the signals allowed the model to handle signals of varying lengths without any issues.

The training process involved minimizing the Mean Squared Error (MSE) between the input and reconstructed signals. The Adam optimizer with a learning rate of $\eta = 10^{-3}$ was used because of its robustness and fast convergence. The network was trained for 100 epochs on a training dataset of $n = 100,000$ signals.

The model's performance was evaluated using MSE. The average MSE for reconstructing signals with amplitude decrease was 0.0006, indicating a very good fit of the model to the input data. Similarly, the average MSE for signals with speed decrease was 0.0004. The average MSE for signals with interruptions was slightly higher, 0.0025, which may indicate the greater complexity of this type of data.

3.4 The k-Nearest Neighbors Method (kNN)

To predict the anomaly coefficients from the latent vectors of the autoencoders, we used the k-nearest neighbors (kNN) regression method. For each of the three analyzed signal types, we first standardized the latent vectors. Three separate kNN models were built, one for each type of anomaly, taking as input the corresponding latent vectors produced by the autoencoder for that anomaly type. The prediction was performed using locally weighted regression, where the weights were determined by an exponentially decaying Gaussian function. This ensures that the closest neighbors have the highest influence on the prediction, while more distant points receive negligible or zero weights. The models were trained on a training set and tested on a separate test set, with a data split of 80% for training and 20% for testing.

To evaluate the prediction quality, we used the mean squared error (MSE). The results confirm the usefulness of the autoencoder's latent vectors for subsequent predictive analysis and enable reliable interpretation of the extracted features: amplitude decrease: MSE = 0.0032; speed decrease: MSE = 0.0015; interruptions: MSE = 0.0163.

3.5 MDS-UPDRS Score classifier

We developed a classifier that can estimate the clinical significance of an observed anomaly in a real signal based on the MDS-UPDRS scale, using the output data from the kNN regressors. Each anomaly was evaluated separately and assigned a corresponding score. To ensure meaningful comparison and effective training of the regression models, the numerical coefficients of each anomaly were normalized to the interval $[0, 1]$. The boundaries for amplitude decrease and interruptions were defined without difficulty, while the decrease in tapping speed is not well-defined in the evaluation scale, so we opted for a continuous partitioning of the $[0, 1]$ interval.

The classifier outputs three separate scores: one considering only the amplitude decrease, one considering only the decrease in tapping speed, and one considering only the interruptions. This provides insight into which signal anomalies are more prominent and explains the reasoning behind the overall impairment score. The final score is determined as the maximum of these three scores, as the criteria in the scale are disjoint, meaning that fulfilling any single condition is sufficient.

Table 2: Intervals for mapping the anomaly coefficient to a score according to the MDS-UPDRS scale.

MDS-UPDRS	Interruptions	Amplitude	Speed
0	$[0, 0.143)$	$[0.95, 1]$	$[0, 0.2)$
1	$[0.143, 0.286]$	$[0.90, 0.95)$	$[0.2, 0.4)$
2	$(0.286, 0.714]$	$[0.45, 0.9)$	$[0.4, 0.8)$
3	$(0.714, 0.857]$	$[0.09, 0.45)$	$[0.8, 0.9]$
4	$(0.857, 1]$	$(0, 0.09]$	$[0.9, 1]$

4 EVALUATION

The accuracy of the classifier is evaluated on generated test signals (signals, which were not used for training either the autoencoders or the kNN), which are generated as described in Section 3.2, except that a signal may contain more than one anomaly (e.g., amplitude decrease and interruption). We generated a set of one hundred test signals, on which the classifier achieved 87% accuracy. The classifier was not evaluated on real signals because our dataset was assessed by only one neurologist (while in related studies, the signals were evaluated by at least three neurologists [13]), and there is no publicly available test set of finger-tapping signals from patients with Parkinson's disease.

The use of the classifier was illustrated using both generated test signal (Figure 2) and real signal (Figure 3).

Table 3: Anomaly coefficients and predicted coefficients along with the corresponding MDS-UPDRS scores for signal on figure 2.

	Amplitude	Speed	Interruption
Anomaly coeff.	0.63	0.00	0.00
Predicted coeff.	0.59	0.03	0.00
MDS-UPDRS	2	0	0

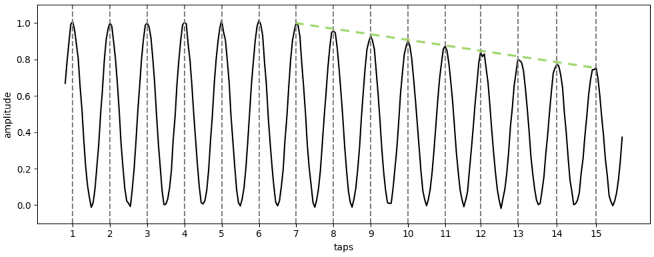


Figure 2: Test signal with a MDS-UPDRS 2, containing an amplitude decrease at the 7th tap (marked with a green line).

Table 4: Predicted coefficients along with the corresponding MDS-UPDRS scores for signal on figure 3.

	Amplitude	Speed	Interruption
Predicted coeff.	0.15	0.81	0.47
MDS-UPDRS	3	3	2

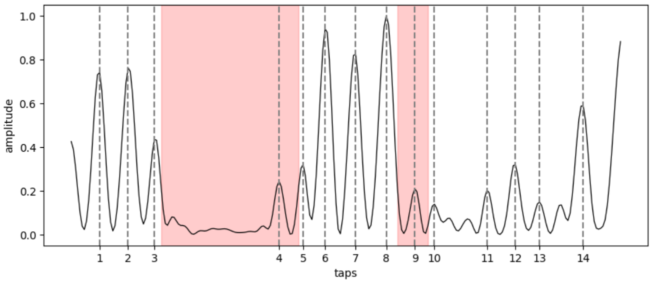


Figure 3: Real signal with a MDS-UPDRS 3.

Table 5: Anomaly coefficients and predicted coefficients along with the corresponding MDS-UPDRS scores for signal on figure 4.

	Amplitude	Speed	Interruption
Anomaly coeff.	1.00	0.00	0.43
Predicted coeff.	0.49	0.69	0.47
MDS-UPDRS	2	2	2

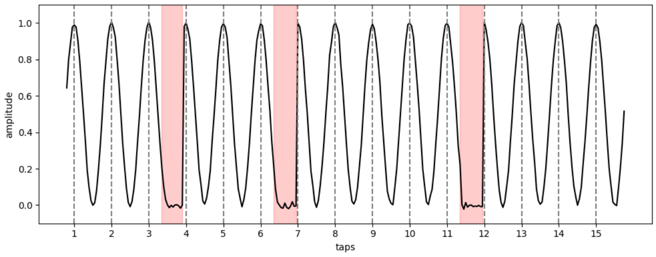


Figure 4: Test signal with an MDS-UPDRS score of 2, containing three interruptions marked in red.

Most interruptions or freezing episodes occur at the local minima of the signal; therefore, the model also detects a more significant decrease in amplitude. At the same time, it detects a decrease in speed (or frequency), as the signal becomes practically constant at the moment of interruption. This is expected since, during a movement freeze, speed cannot be measured. The presence of interruptions in the signals thus affects the estimation of the speed decrease coefficient. Even though the tapping frequency itself does not decrease, the model interprets the interruption as a greater drop in speed because a temporary halt in movement indicates a decrease in speed. An example of this is shown in Figure 4, with the corresponding anomaly coefficients and predictions presented in Table 5.

5 CONCLUSION

We developed a method to improve the interpretability of finger-tapping classification for Parkinson's disease assessment. The approach detects anomalies in signals, such as amplitude changes, speed decrease, and interruptions, and links them to MDS-UPDRS scores. Separate autoencoders captured features of each anomaly type, and k-nearest neighbors (kNN) were used to predict anomaly coefficients, achieving high accuracy. The main challenge was handling signals of varying lengths, solved through segmentation, though this may have led to a loss of temporal details. Building a classifier for MDS-UPDRS scoring was difficult due to limited and imbalanced real data, as well as the textual description of certain anomalies like speed decrease. The biggest limitation is the lack of real data. This issue highlights that the generated dataset does not cover the entire latent space, which can result in unreliable MDS-UPDRS estimates for real signals. The autoencoders and kNN methods contribute to understanding the relationship between mathematically defined anomalies in finger movement signals and the MDS-UPDRS scores.

REFERENCES

[1] Warwick R. Adams. 2017. High-accuracy detection of early Parkinson's Disease using multiple characteristics of finger movement while typing. *PLoS ONE* 12, 11 (2017), e0188226. https://doi.org/10.1371/journal.pone.0188226

[2] J. Amo-Salas, A. Olivares-Gil, Á. García-Bustillo, D. García-García, Á. Arnaiz-González, and E. Cubo. 2024. A Method for Automatic and Objective Scoring of Bradykinesia Using Orientation Sensors and Classification Algorithms. *Healthcare* 12, 439 (2024).

[3] G. Amprimo, I. Rechichi, C. Ferraris, and G. Olmo. 2023. Objective Assessment of the Finger Tapping Task in Parkinson's Disease and Control Subjects using Azure Kinect and Machine Learning. *IEEE 36th International Symposium on Computer-Based Medical Systems* (2023), 640–645.

[4] Eric Y. Chiang, Qingkai Kong, Timmy Siauw, and Alexandre M. Bayen. 2019. Discrete Fourier Transform.

[5] Arnaud Delval, Luc Defebvre, and Céline Tard. 2017. Freezing during tapping tasks in patients with advanced Parkinson's disease and freezing of gait. *PLoS ONE* 12, 9 (2017).

[6] C. G. Goetz, B. C. Tilley, S. R. Shaftman, G. T. Stebbins, S. Fahn, P. Martinez-Martin, W. Poewe, C. Sampaio, M. B. Stern, R. Dodel, et al. 2008. Movement Disorder Society-sponsored revision of the Unified Parkinson's Disease Rating Scale (MDS-UPDRS): scale presentation and clinimetric testing results. *Movement Disorders* 23, 15 (2008).

[7] D. L. Guarín, J. K. Wong, N. R. McFarland, and A. Ramirez-Zamora. 2024. Characterizing Disease Progression in Parkinsons Disease from Videos of the Finger Tapping Test. *IEEE transactions on neural systems and rehabilitation engineering* 32 (2024), 2293–2301.

[8] Z. Guo, W. Zeng, T. Yu, Y. Xu, Y. Xiao, Z. Cao, and X. Cao. 2022. Vision-Based Finger Tapping Test in Patients With Parkinsons Disease via Spatial-Temporal 3D Hand Pose Estimation. *IEEE Journal of bimedical and health informatics* 26, 8 (2022), 3848–3859.

[9] M. S. Islam, W. Rahman, A. Abdelkader, S. Lee, P. T. Yang, J. L. Purks, J. L. Adams, R. B. Schneider, E. R. Dorsey, and E. Hoque. 2023. Using AI to measure Parkinsons disease severity at home. *NPJ Parkinson's disease* 156, 6 (2023).

[10] H. Li, X. Shao, C. Zhangc, and X. Qian. 2021. Automated assessment of Parkinsonian finger-tapping tests through a vision-based fine-grained classification model. *Neurocomputing* 441 (2021), 260–271.

[11] Z. Li, I. Kang, M. Cai, X. Liu, Y. Wang, and J. Yang. 2022. An Automatic Evaluation Method for Parkinson's Dyskinesia Using Finger Tapping Video for Small Samples. *Jurnal of Medical and Biological Engineering* 42 (2022), 351–363.

[12] M. Lu, Q. Zhao, K. L. Poston, E. V. Sullivan, A. Pfefferbaum, M. Shahid, M. Katz, L. M. Kouhsari, K. Schulman, A. Milstein, J. C. Niebles, and V. W. Henderson. 2021. Quantifying Parkinsons disease motor severity under uncertainty using MDS-UPDRS videos. *Medical Image Analysis* 73 (2021).

[13] O. Martinez-Manzanera, E. Roosma, M. Beudel, R. W. Borgemeester, T. van Laar, and N. M. Maurits. 2016. Computer Vision for Parkinson's Disease Evaluation: A Survey on Finger Tapping. *IEEE Transactions on Biomedical Engineering* 63, 5 (2016), 1016–1024.

[14] A. Yu. Meigal, S. M. Rissanen, M. P. Tarvainen, S. D. Georgiadis, P. A. Karjalainen, O. Airaksinen, and M. Kankaanpää. 2012. Linear and nonlinear tremor acceleration characteristics in patients with Parkinson's disease. *Physiological Measurement* 33, 3 (2012), 395–412. https://doi.org/10.1088/0967-3334/33/3/395

[15] G. Morinan, Y. Dushin, G. Sarapata, S. Rupprechter, Y. Peng, C. Girges, M. Salazar, C. Milabo, K. Sibley, T. Foltynie, I. Cociasu, L. Ricciardi, F. Baig, F. Morgante, L. A. Leyland, R. S. Weil, R. Gilron, and J. OKeeffe. 2022. Video-Based Automated Assessment of Movement Parameters Consistent with MDS-UPDRS III in Parkinsons Disease. *Journal of Parkinsons Disease* 2211–2222, 12 (2022).

[16] G. Morinan, Y. Dushin, G. Sarapata, S. Rupprechter, Y. Peng, C. Girges, M. Salazar, C. Milabo, K. Sibley, T. Foltynie, I. Cociasu, L. Ricciardi, F. Baig, F. Morgante, L. A. Leyland, R. S. Weil, R. Gilron, and J. OKeeffe. 2023. Computer vision quanti cation of whole-body Parkinsonian bradykinesia using a large multi-site population. *NPJ Parkinson's disease* 1, 9 (2023).

[17] A. Moshkova, A. Samorodov, N. Voinova, A. Volkov, E. Ivanova, and E. Fedotova. 2024. Parkinsons Disease Detection by Using Machine Learning Algorithms and Hand Movement Signal from LeapMotion Sensor. , 321–327 pages.

[18] National Institutes of Neurological Disorders and Stroke. 2024. Parkinson's Disease. https://www.ninds.nih.gov/health-information/disorders/parkinsons-disease

[19] International Parkinson and Movement Disorder Society. 2008. MDS-UPDRS: Movement Disorder Society–sponsored revision of the Unified Parkinson's Disease Rating Scale (English final version). PDF.

[20] Atemangoh B. Peachap, Daniel Tchiotsop, Valérie Louis-Dorr, and Didier Wolf. 2020. Detection of early Parkinson's disease with wavelet features using finger typing movements on a keyboard. *SN Applied Sciences* 2, 10 (2020), 1–10. https://doi.org/10.1007/s42452-020-03444-4

[21] G. Sarapata, Y. Dushin, G. Morinan, J. Ong, S. Budhdeo, B. Kainz, and J. OKeeffe. 2023. Video-Based Activity Recognition for Automated Motor Assessment of Parkinsons Disease. *IEEE Journal of bimedical and health informatics* 27, 10 (2023), 5032–5041.

[22] Isao Shimoyama, Toshiaki Ninchoji, and Kazuo Uemura. 1990. The Finger-Tapping Test: A Quantitative Analysis. *Archives of Neurology* 47, 6 (1990), 681–684.

[23] J. Stamatakis, J. Ambroise, J. Crémers, H. Sharei, V. Delvaux, B. Macq, and G. Garraux. 2013. Finger Tapping Clinimetric Score Prediction in Parkinson's Disease Using Low-Cost Accelerometers. *Computational Intelligence and Neuroscience* 2013 (2013).

[24] Stefan Williams, David Wong, Jane E. Alty, and Samuel D. Relton. 2023. Parkinsonian Hand or Clinician's Eye? Finger Tap Bradykinesia Interrater Reliability for 21 Movement Disorder Experts. *Journal of Parkinson's Disease* 13, 4 (2023), 525–536. https://doi.org/10.3233/JPD-223256

[25] J. Yang, S. Williams, D. C. Hogg, J. E. Alty, and S. D. Relton. 2024. Deep learning of Parkinson's movement from video, without human-defined measures. *Journal of the Neurological Sciences* 4463 (2024).

[26] T. Yu, K. W. Park, M. J. McKeown, and Z. J. Wang. 2023. Clinically Informed Automated Assessment of Finger Tapping Videos in Parkinsons Disease. *Sensors* 9149, 23 (2023).

[27] J. Zhu, Y. Cui, J. Zhang, R. Yan, D. Su, D. Zhao, A. Wang, and T. Feng. 2024. Temporal trends in the prevalence of Parkinson's disease from 1980 to 2023: a systematic review and meta-analysis. *The Lancet Healthy Longevity* 5, 7 (2024), 464–e79. https://doi.org/10.1016/S2666-7568(24)00094-1

[28] M. Zupanic. 2024. Avtomatsko ocenjevanje testa tapkanja s prsti pri bolnikih s Parkinsonovo boleznijo. Magistrska naloga, Fakulteta za računalništvo in informatiko.

Proc. of the 11th Student Computing Research Symposium (SCORES'25), Ljubljana, Slovenia, September 30, 2025

Comprehensive Multi-Species Fruit Ripeness Dataset Construction: From Eight-Species Collection to Focused Apple-Cherry Detection

Maksim Loknar
maksim.loknar@student.um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

Uroš Mlakar
uros.mlakar@um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

ABSTRACT

This paper presents a comprehensive framework for constructing a large-scale multi-species fruit ripeness detection dataset and validates it through focused instance segmentation on apple and cherry subsets. We develop a systematic methodology for capturing and annotating fruit images across five primary species (apples, cherries, tomatoes, strawberries, cucumbers), each annotated with three distinct ripeness states (ripe, unripe, spoiled) plus an obscured category for partially visible fruits. Our complete dataset encompasses 486 high-resolution images with 6,984 polygon-based annotations collected under diverse garden conditions, with a 15-fold augmentation strategy expanding the apple-cherry training subset to 2,240 images. To validate our dataset construction methodology, we train and evaluate a Mask R-CNN model with ResNet-50 backbone specifically on the apple and cherry subsets (184 images with 3,551 annotations), achieving 22.49% AP@0.5 and 60.63% COCO mAP. The framework addresses key challenges including handling significant occlusion (53.8% of annotations), temporal sampling across different growing seasons, class imbalance mitigation, and scalable annotation protocols.

KEYWORDS

Multi-species dataset construction, agricultural computer vision, fruit ripeness classification, instance segmentation, apple detection, cherry detection, precision agriculture

1 INTRODUCTION

The development of robust computer vision systems for agricultural applications requires comprehensive datasets that capture the full diversity of crop species and conditions encountered in real-world farming environments. While significant advances have been made in single-species fruit detection [9], the construction of large-scale, multi-species datasets with consistent ripeness annotations remains a critical challenge for advancing precision agriculture technologies [7].

Creating effective multi-species datasets presents unique challenges beyond those encountered in single-crop systems. Different fruit types exhibit vastly different morphologies, ripening patterns, and growing seasons, requiring flexible yet consistent annotation protocols [8]. The continuous nature of ripeness progression varies significantly between species—cherries may transition from unripe to overripe within days, while apples maintain harvestable ripeness for weeks. Existing datasets either focus on single species like MinneApple [2] or provide limited ripeness granularity like Fruits-360 [12], failing to capture the complexity needed for practical agricultural deployment.

This work presents a comprehensive framework for constructing a large-scale multi-species fruit detection dataset encompassing eight primary fruit types (apples, pears, cucumbers, cherries, raspberries, peppers, tomatoes, strawberries) with ongoing expansion to include plums. Each species is annotated with three ripeness states, creating a rich taxonomic structure of 25+ categories. Our methodology addresses critical gaps in existing agricultural datasets by establishing consistent annotation protocols across diverse fruit morphologies while maintaining species-specific ripeness criteria.

To validate our dataset construction approach, we focus our initial model development on apple and cherry subsets, representing complementary challenges in agricultural computer vision. Apples exhibit gradual color transitions and significant size variation, while cherries present challenges of small fruit clustering and rapid ripeness progression. This focused validation demonstrates the dataset's utility while establishing baselines for future multi-species model development.

The primary contributions of this work include: (1) a comprehensive dataset of eight fruit species with consistent ripeness annotations, (2) scalable annotation protocols accommodating diverse fruit morphologies, (3) rigorous quality control measures ensuring cross-species consistency, and (4) experimental validation through focused instance segmentation on representative species subsets achieving practical detection performance. By establishing clear protocols for multi-species dataset construction, this work provides essential infrastructure for advancing agricultural computer vision toward practical deployment across diverse crop types.

2 RELATED WORK

2.1 Single-Species vs. Multi-Species Datasets

Existing agricultural datasets largely fall into two categories: high-quality single-species datasets and diverse but inconsistently annotated multi-species collections. Single-species datasets like MinneApple [2] provide detailed pixel-level annotations for apples but lack generalizability to other fruits. The CitDet dataset [1] focuses on citrus with precise bounding boxes but excludes ripeness classification entirely.

---

https://doi.org/10.51939/scores25.7

Multi-species datasets attempt broader coverage but sacrifice annotation quality or consistency. The ACFR dataset [8] includes apples, mangoes, and almonds across multiple modalities but provides only coarse bounding boxes. Fruits-360 [12] offers 90,000 images across 131 fruit types but uses controlled studio conditions that fail to represent agricultural environments. The Fresh and Stale Classification dataset [14] addresses quality assessment but lacks spatial localization necessary for harvesting applications.

Recent work by Sa et al. [13] demonstrates that models trained on diverse fruit types achieve better generalization than single-species models, with performance improvements of 15-20% when deployed on novel fruit types. However, their dataset lacks the ripeness granularity necessary for quality assessment applications.

2.2 Challenges in Multi-Species Annotation

Annotation consistency across diverse fruit types presents significant challenges. Bargoti and Underwood [2] show that inter-annotator agreement decreases from 0.92 for single species to 0.71 for multi-species tasks, primarily due to inconsistent ripeness criteria application. Different fruits require distinct visual cues—color for tomatoes, size for cucumbers, texture for strawberries—complicating unified annotation protocols.

The VGG Image Annotator [3] addresses multi-attribute annotation through hierarchical labeling systems, enabling species-specific attributes while maintaining dataset-wide consistency. Recent studies emphasize the importance of domain expert involvement, with agricultural specialists achieving 25% higher annotation accuracy than general crowd workers for ripeness classification tasks.

Temporal consistency presents particular challenges in multi-species contexts. While apples and pears follow similar seasonal patterns, berries and vegetables have multiple growing cycles per year. Zhang et al. [15] demonstrate that temporal stratification in dataset construction improves model robustness by 18% compared to random sampling approaches.

2.3 Instance Segmentation in Agriculture

Instance segmentation has emerged as the preferred approach for agricultural applications requiring precise fruit localization. He et al. [5] introduced Mask R-CNN, which has become the standard architecture for agricultural instance segmentation. Agricultural adaptations include modified anchor sizes for small fruits like berries and multi-scale feature fusion for handling size variation within species.

Recent agricultural applications demonstrate strong performance: Zhang et al. [15] achieve 89% mask AP for apple detection, while cherry detection studies report 85% accuracy despite challenging clustering patterns. However, these successes remain limited to single-species or simplified two-class (ripe/unripe) problems. Multi-species, multi-ripeness instance segmentation remains largely unexplored, motivating our comprehensive dataset construction effort.

3 COMPREHENSIVE DATASET CONSTRUCTION

3.1 Data Collection Strategy and Sources

Our dataset construction employed a dual-source approach combining internet-sourced imagery with personally collected agricultural data. The primary annotated dataset comprises 486 high-resolution images spanning five fruit species (apples, cherries, strawberries, tomatoes, cucumbers) sourced from publicly available images of gardens, orchards, and fruit trees worldwide. These images were specifically selected to represent diverse environmental conditions, lighting scenarios, and fruit maturity stages encountered in real agricultural settings.

Additionally, we collected 7,432 unannotated images from personal agricultural sources, including planted trees and home gardens. This supplementary dataset encompasses six species (apples, cherries, tomatoes, cucumbers, plums, raspberries) and was captured under varied temporal conditions to ensure comprehensive lighting variation. Images were collected during early morning hours from 6-8 AM featuring soft, directional lighting with long shadows, midday periods from 12-2 PM with harsh overhead lighting and minimal shadows, and late evening sessions from 6-8 PM displaying warm-toned lighting with color temperature shifts. These images are reserved for future expansion through pseudo-labeling once the model achieves sufficient accuracy for semi-automated annotation.

3.2 Annotation Methodology and Quality Control

The 486 images underwent meticulous manual annotation using the VGG Image Annotator (VIA) [3], with polygon-based instance segmentation for precise fruit boundary delineation. Each fruit instance was annotated with species identification and ripeness classification (ripe, unripe, spoiled), with an additional "obscured" category for partially visible fruits—a critical real-world challenge affecting 53.8% of all annotations.

Our annotation protocol ensured consistency through flexible vertex counts ranging from 3 to 126 vertices, averaging 14.1 vertices per polygon for accurate boundary representation. The hierarchical labeling system progressed from plant type to individual instance to ripeness state and visibility status. We systematically handled occlusions with visibility flags for partially hidden fruits, achieving comprehensive coverage with 6,984 total annotations across 486 images.

The annotation process yielded detailed instance-level segmentation masks with accompanying metadata including bounding boxes averaging 71.4 × 78.8 pixels, area calculations with a mean of 12,450 pixels^2, and occlusion indicators. From the complete annotated dataset, we selected apple and cherry subsets for initial model validation, representing 184 base images with 3,551 annotations that form the foundation of our training dataset.

3.3 Unified Annotation Framework

3.3.1 Taxonomic Structure. Our classification system employs a hierarchical structure with 25 primary categories consisting of 8 species multiplied by 3 ripeness states yielding 24 fruit categories,

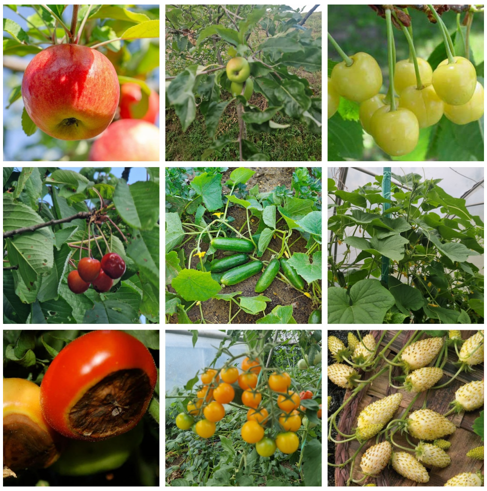


Figure 1: Example images from the annotated dataset

plus one background category. The structure is expandable to accommodate new species, with plums currently being added to the dataset.

3.3.2 Species-Specific Ripeness Criteria. While maintaining three universal ripeness categories, we developed species-specific determination criteria. For color-based determination used with apples, tomatoes, peppers, and cherries, fruits are classified as ripe when showing greater than 75% characteristic color coverage, unripe when displaying less than 25% color development, and spoiled when exhibiting brown or black discoloration or visible mold.

Size-based determination applies to cucumbers and pears, where ripe fruits fall within 80-100% of variety-specific size range, unripe specimens measure less than 80% of expected size, and spoiled fruits show yellowing, soft spots, or wrinkled skin.

Texture-based determination is used for strawberries and raspberries, with ripe fruits displaying uniform color with firm but yielding texture, unripe fruits showing white or green areas with hard texture, and spoiled specimens exhibiting soft spots, mold, or collapsed structure.

3.3.3 Annotation Process and Quality Control. We employed the VGG Image Annotator (VIA) [3] with custom configurations for multi-species annotation. The process included polygon-based instance segmentation with a minimum of 8 vertices, a hierarchical attribute system progressing from species to ripeness to quality flags, occlusion handling with visibility percentage annotation, and edge case documentation with a photographic reference library.

Quality control measures ensured cross-species consistency through species-specific training modules for annotators, 25% overlap for inter-annotator agreement calculation, Cohen's Kappa scores exceeding 0.85 for species identification and 0.80 for ripeness classification, weekly calibration sessions reviewing problematic cases, and expert agricultural review for a 10% random sample.

3.4 Dataset Composition and Statistics

The complete annotated dataset of 486 images contains 6,384 individual fruit instances across five primary species. The dataset exhibits significant variation in annotation density, reflecting the natural clustering patterns of different fruit types. Table 1 presents the detailed distribution, while Table 2 shows the ripeness classification breakdown.

Table 1: Annotated Dataset Composition by Species

Species	Images	Annotations	Avg/Image
Apple	98	2,582	26.3
Cherry	86	969	11.3
Tomato	94	1,572	16.7
Strawberry	111	1,397	12.6
Cucumber	97	464	4.8
Total	**486**	**6,984**	**14.4**

The annotation density varies significantly across species, with apples showing the highest average of 26.3 annotations per image due to their tendency to cluster in orchard settings, while cucumbers show the lowest at 4.8 per image reflecting their more dispersed growth pattern. Notably, a significant portion of annotations (53.8%) were marked as "obscured," representing fruits partially hidden by leaves, branches, or other fruits—a critical challenge for real-world detection systems.

Table 2: Ripeness Distribution Across Species (Excluding Obscured)

Species	Ripe	Unripe	Spoiled	Total*
Apple	669 (79.7%)	91 (10.8%)	78 (9.3%)	838
Cherry	232 (49.6%)	175 (37.4%)	61 (13.0%)	468
Tomato	271 (43.7%)	322 (51.9%)	27 (4.4%)	620
Strawberry	382 (42.3%)	485 (53.7%)	37 (4.1%)	904
Cucumber	199 (77.1%)	51 (19.8%)	8 (3.1%)	258
Combined	**1,753**	**1,124**	**211**	**3,088**

*Excludes 3,896 obscured annotations for clarity

For the focused model training on apple and cherry subsets, the combined dataset provides 3,551 total instances comprising 2,582 apples and 969 cherries, with 1,306 instances having clear visibility (838 apples and 468 cherries). The ripeness balance includes 901 ripe, 266 unripe, and 139 spoiled fruits, while the occlusion challenge encompasses 2,245 obscured instances representing 63.2% of the total.

Annotation complexity metrics reveal the detailed nature of our ground truth. The average polygon contains 14.1 ± 9.8 vertices per instance, with a vertex range spanning from 3 to 126 vertices accommodating simple to complex fruit boundaries. The mean annotation area measures 12,450 pixels^2 with high variance (α=73,323), while bounding box dimensions average 71.4 × 78.8 pixels.

3.5 Data Augmentation and Dataset Expansion

To address the limited size of our annotated dataset and improve model generalization, we implemented comprehensive augmentation strategies. Each of the 140 apple and cherry images underwent 15-fold augmentation (original plus 15 augmented versions), expanding our training dataset to 2,240 images. Combined with validation splits, the total training corpus reached 7,792 images, though only the apple and cherry subsets were utilized for the model presented in this work.

Our augmentation pipeline applied both geometric and photometric transformations while maintaining biological plausibility. Geometric augmentations included RandomRotation of ±15° to simulate varied camera angles, RandomHorizontalFlip with 50% probability reflecting natural symmetry, RandomAffine translation up to 10% simulating viewpoint changes, and scale variations from 0.8 to 1.2× maintaining realistic fruit sizes.

Photometric augmentations consisted of ColorJitter with brightness, contrast, and saturation adjustments of ±30% and hue shifts of ±0.1, Gaussian blur with radius 0.5-2.0 pixels simulating focus variations, and noise injection using Gaussian noise with α=0.02 for sensor noise simulation. The augmentation strategy specifically addressed agricultural imaging challenges including varying lighting conditions from our tri-temporal collection approach, natural fruit orientations on branches, and partial occlusions from leaves and neighboring fruits. Critically, we avoided transformations that would create biologically implausible scenarios, such as extreme rotations that would place fruits in impossible orientations relative to gravity.

For validation processing, we applied minimal transformations to ensure consistency, using only ToTensor conversion to tensor format and normalization with ImageNet statistics (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]).

This dataset is publicly available at https://huggingface.co/datasets/TheCoffeeAddict/SmartHarvest.

4 FOCUSED VALIDATION: APPLE AND CHERRY DETECTION

4.1 Model Architecture and Training Implementation

We employed Mask R-CNN [4] with ResNet-50 backbone [6] and Feature Pyramid Networks [10] for multi-scale feature extraction. The architecture was specifically adapted for our seven-class problem encompassing apple-ripe, apple-unripe, apple-spoiled, cherry-ripe, cherry-unripe, cherry-spoiled, plus background.

Key architectural modifications included a custom FastRCNN predictor with dropout regularization at rate 0.3 to prevent overfitting, a MaskRCNN predictor with 256 hidden units for fine-grained segmentation, modified anchor scales of [8, 16, 32, 64, 128] pixels optimized for fruit size distribution, and non-maximum suppression with IoU threshold 0.5 to handle clustered fruits.

The training configuration utilized MS-COCO pre-trained weights [11] for initialization, SGD optimizer with momentum 0.9 and weight decay 5×10^-4, initial learning rate of 0.002 with 5-epoch warmup followed by StepLR decay with γ=0.5 every 20 epochs, batch size of 2 due to GPU memory constraints, gradient clipping with maximum norm 10.0 for stability, early stopping with patience of 15 epochs and δ=0.0001, and a maximum of 100 epochs.

The model optimizes a combined loss function:

$$L_{total} = L_{cls} + L_{box} + L_{mask} \tag{1}$$

where L_{cls} is cross-entropy loss for classification, L_{box} is smooth L1 loss for bounding box regression, and L_{mask} is binary cross-entropy for pixel-wise segmentation. Additional regularization through dropout (0.3) in the classification head and L2 weight decay (5×10^-4) helped prevent overfitting on our limited dataset.

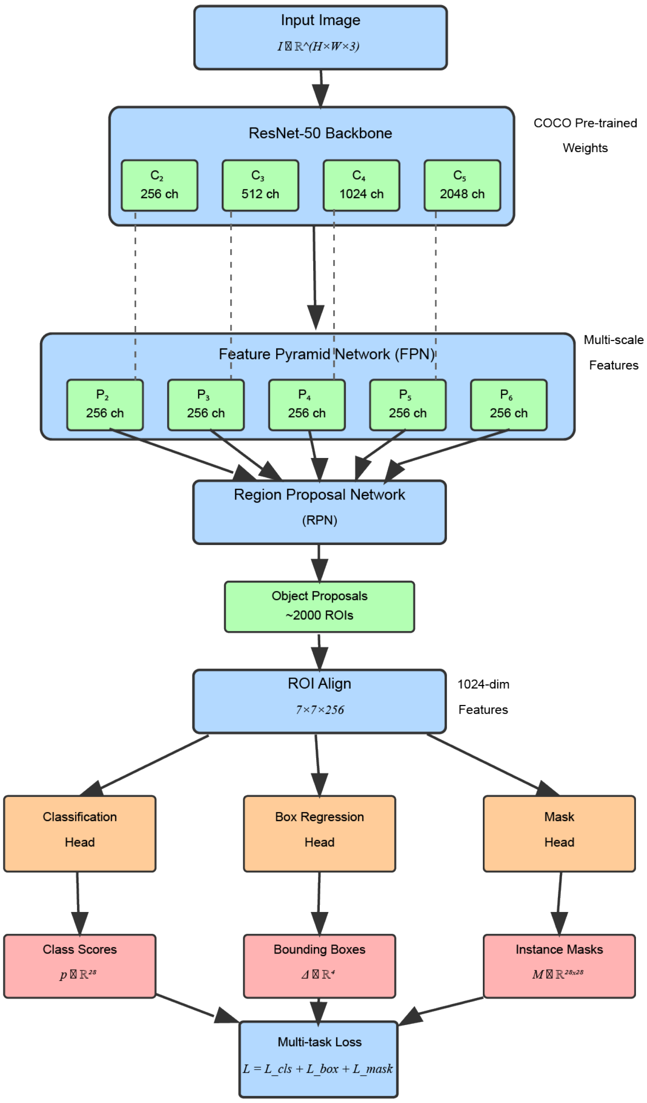


Figure 2: Mask R-CNN architecture adapted for agricultural fruit detection and segmentation, showing the ResNet-50 backbone, FPN, and specialized heads for multi-class fruit ripeness classification.

The model employs several regularization techniques to prevent overfitting including dropout regularization applied in the

FastRCNN predictor head with rate 0.3, weight decay providing L2 regularization with coefficient $5 \times 10^{-}4$, gradient clipping with maximum gradient norm of 10.0 to prevent exploding gradients, early stopping monitoring validation loss with patience of 15 epochs, and extensive data augmentation through photometric and geometric transformations. The training employs a learning rate warmup strategy over the first 5 epochs to stabilize early training, followed by exponential decay with factor 0.5 every 20 epochs.

This code is publicly available at https://github.com/Maksim3l/SmartHarvest.

4.2 Results and Analysis

We evaluate performance using standard COCO metrics [11] adapted for agricultural contexts, including detection metrics (AP@IoU=0.5, AP@IoU=0.75, AP@IoU=0.5:0.95), segmentation metrics (Mask IoU at multiple thresholds), classification metrics (per-class precision, recall, F1-score), and agricultural metrics (counting accuracy, ripeness classification accuracy).

4.2.1 Overall Detection Performance. The trained model achieved encouraging results on the apple-cherry validation subset, demonstrating the viability of our dataset construction approach:

Table 3: Instance Segmentation Performance on Apple-Cherry Subset

Metric	Value	Description
AP@0.5	0.2249	Average precision at IoU threshold 0.5
AP@0.75	0.0798	Average precision at IoU threshold 0.75
COCO mAP	0.6063	Mean average precision across IoU thresholds
Precision@0.5	0.2333	325 true positives out of 1393 detections
Precision@0.75	0.1070	149 true positives out of 1393 detections

The model demonstrates reasonable detection capability with 22.49% AP@0.5, indicating successful fruit localization in approximately one-quarter of cases at the standard IoU threshold. The higher COCO mAP of 60.63% suggests the model performs better across varying IoU thresholds, particularly at lower thresholds where precise boundary delineation is less critical.

4.2.2 Per-Class Ripeness Detection Performance. Analysis of per-class performance reveals interesting patterns in ripeness detection capabilities:

Table 4: Per-Class Average Precision at IoU=0.5

Apple Classes	AP@0.5	Cherry Classes	AP@0.5
Apple-Ripe	0.1045	Cherry-Ripe	0.1820
Apple-Unripe	0.2500	Cherry-Unripe	0.1710
Apple-Spoiled	0.3260	Cherry-Spoiled	0.3156
Average	**0.2268**	**Average**	**0.2229**

Notable observations from the per-class analysis show that spoiled fruit detection achieves the highest AP within both species, with apple-spoiled at 32.60% and cherry-spoiled at 31.56%, likely due to more distinctive visual features such as discoloration and texture changes. The balanced species performance between apples (22.68%) and cherries (22.29%) validates our balanced dataset construction approach with nearly identical average precision. Ripeness detection challenges are evident in ripe fruit classes showing lower performance, with apple-ripe at 10.45% and cherry-ripe at 18.20%, potentially due to subtle visual differences from unripe states and higher within-class variation.

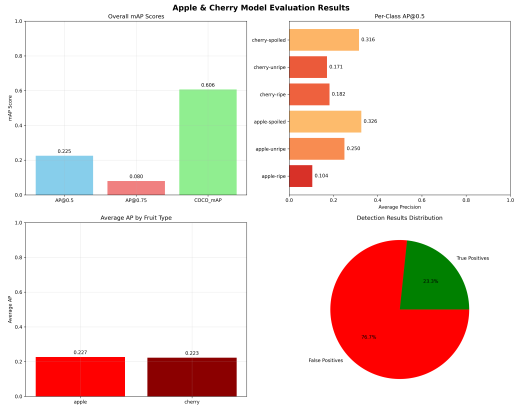


Figure 3: Evaluation results visualization showing per-class performance metrics and detection statistics for the apple-cherry model.

The model generated 1,393 total detections across the validation set, achieving 325 true positives at IoU≥0.5 for 23.3% precision and 149 true positives at IoU≥0.75 for 10.7% precision. The high detection rate indicates sensitivity to fruit presence, though there remains room for improvement in reducing false positives.

While the current performance falls below commercial deployment thresholds (typically requiring >80% AP), the results provide valuable insights. The balanced performance across species and systematic performance patterns across ripeness states validate our annotation protocols and dataset construction methodology. These results establish realistic baselines for multi-species, multi-ripeness detection, highlighting the increased complexity compared to single-species studies. The performance analysis identifies specific areas for enhancement including increased training data particularly for underperforming classes, class-weighted loss functions to address imbalance, agricultural-specific pre-training rather than MS-COCO initialization, and enhanced augmentation strategies for ripeness variation.

5 DATASET EXPANSION AND FUTURE DEVELOPMENT

The modular framework facilitates systematic expansion with plums currently being collected targeting 60 images and 800 instances, and planned additions including grapes, peaches, lettuce, and carrots. Each new species informs annotation guideline updates through protocol refinement.

Building on current model performance, we are developing an iterative improvement strategy utilizing confidence thresholding with high-confidence detections above 0.7 for initial annotations, active learning to prioritize annotation of uncertain samples with confidence between 0.3 and 0.7, class-specific models training specialized models for challenging ripeness states, and quality assurance maintaining human verification for all automated annotations.

Based on performance analysis, planned enhancements include targeted data collection focusing on underperforming classes particularly ripe fruits, balanced sampling ensuring equal representation across ripeness states, multi-modal integration adding NIR channels for improved ripeness discrimination, and temporal sequences capturing daily progression for ripeness continuity.

6 DISCUSSION

This work establishes several key contributions to agricultural computer vision. The comprehensive dataset represents the first multi-species dataset with consistent ripeness annotations across eight fruit types, providing a valuable resource for developing generalized agricultural vision systems. The hierarchical annotation structure supports both species-specific and cross-species model development.

Our scalable methodology successfully accommodates diverse fruit morphologies while maintaining consistency. The framework's successful expansion to plums demonstrates scalability to additional species. By achieving 22.49% AP@0.5 and identifying specific performance patterns, we provide honest baselines and clear improvement paths for the research community through realistic validation.

Several insights emerged from our dataset construction and model training process. The ripeness complexity is evident in the performance variation across ripeness states ranging from 10.45% to 32.60% AP, highlighting the challenge of capturing continuous biological processes in discrete categories. Dataset scale requirements suggest that while 140 images proved sufficient for initial validation, achieving commercial-grade performance likely requires 10× more data per class. Despite 8-fold augmentation, real-world variation in ripeness appearance requires actual temporal sampling across growing seasons, revealing augmentation limitations.

Our framework and results have implications for precision agriculture. Current 23.3% precision enables semi-automated workflows where models provide initial detections for human refinement. Published baselines enable research benchmarking for comparative evaluation of future agricultural vision methods. Performance analysis guides efficient data collection focusing on challenging classes through dataset prioritization.

7 CONCLUSION

This paper presents a comprehensive framework for constructing large-scale, multi-species fruit detection datasets with consistent ripeness annotations. Our dataset of eight fruit and vegetable species with 6,784 annotated instances provides a valuable resource for advancing agricultural computer vision beyond single-species limitations.

The systematic methodology we developed successfully addresses key challenges in multi-species dataset construction by maintaining annotation consistency across diverse morphologies, establishing species-specific yet comparable ripeness criteria, and creating scalable protocols for dataset expansion. Model validation on apple and cherry subsets achieved 22.49% AP@0.5 and 60.63% COCO mAP, establishing realistic baselines for multi-species, multi-ripeness detection tasks.

Key contributions include the comprehensive annotated dataset spanning eight species, detailed protocols enabling reproducible dataset expansion, quality control metrics ensuring annotation consistency, and realistic performance baselines with clear improvement paths. The identification of systematic performance patterns—particularly the superior detection of spoiled fruits and balanced performance across species—validates our dataset construction approach while highlighting areas for focused improvement.

Future work will focus on expanding both dataset scale and model capabilities. Immediate priorities include reaching 500 images per species, implementing class-balanced sampling strategies, developing agricultural-specific pre-training approaches, and exploring temporal continuity in ripeness progression. The current model's ability to detect approximately one-quarter of fruits at standard IoU thresholds provides a foundation for semi-automated annotation workflows that can accelerate dataset expansion.

This work establishes critical infrastructure for agricultural computer vision research, supporting global efforts toward sustainable and efficient food production. By providing both data resources and honest performance baselines, we enable the research community to tackle increasingly complex agricultural challenges through computer vision.

ACKNOWLEDGMENTS

The authors would like to thank Jan Popič for his help with establishing the environment for experiment work on the FERI research computer.

REFERENCES

[1] Suchet Bargoti and James Underwood. 2016. Deep Fruit Detection in Orchards. *arXiv preprint arXiv:1610.03677* (2016).

[2] Suchet Bargoti and James Underwood. 2016. Image Segmentation for Fruit Detection and Yield Estimation in Apple Orchards. *To Appear in Journal of Field Robotics* (2016).

[3] Abhishek Dutta and Andrew Zisserman. 2019. VGG Image Annotator (VIA). https://www.robots.ox.ac.uk/~vgg/software/via/. University of Oxford.

[4] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2961–2969. https://doi.org/10.1109/ICCV.2017.322

[5] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask R-CNN. In *Proceedings of the IEEE international conference on computer vision*. 2961–2969.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[7] Andreas Kamilaris and Francesc X Prenafeta-Boldú. 2018. Deep learning in agriculture: A survey. *Computers and electronics in agriculture* 147 (2018), 70–90.

[8] Thomas Kirk, Christopher McCool, Inkyu Sa, Chris Lehnert, Tristan Perez, and Ben Upcroft. 2021. A dataset for multi-modal perception of fruit trees in orchards. *Robotics and Autonomous Systems* 134 (2021), 103661.

[9] Anand Koirala, Kerry B Walsh, Zhenglin Wang, and Cheryl McCarthy. 2019. Deep learning for real-time fruit detection and orchard fruit counting: A review. *Computers and Electronics in Agriculture* 159 (2019), 107–117.

[10] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. 2017. Feature Pyramid Networks for Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2117–2125. https://doi.org/10.1109/CVPR.2017.106

[11] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. In *European Conference on Computer Vision (ECCV)*. Springer, 740–755. https://arxiv.org/abs/1405.0312

[12] Horia Mureșan and Mihai Oltean. 2018. Fruit recognition from images using deep learning. *Acta Universitatis Sapientiae, Informatica* 10, 1 (2018), 26–42.

[13] Inkyu Sa, Zongyuan Ge, Feras Dayoub, Ben Upcroft, Tristan Perez, and Christopher McCool. 2016. DeepFruits: A fruit detection system using deep neural networks. In *Sensors*, Vol. 16. MDPI, 1222.

[14] Swoyam. 2022. Fresh and Stale Classification Dataset. https://www.kaggle.com/datasets/swoyam2609/fresh-and-stale-classification/data. Accessed: 2025-06-23.

[15] Xiaolei Zhang, Yun Qiao, Rui Meng, Jiayi Fan, and Tianyi Li. 2020. Robust fruit detection using improved Mask R-CNN under complex orchard conditions. *Computers and Electronics in Agriculture* 176 (2020), 105672.

Proc. of the 11th Student Computing Research Symposium (SCORES'25), Ljubljana, Slovenia, September 30, 2025

Identifying vineyards from LiDAR data

Žan Tomaž Šprajc
zan.sprajc@student.um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

Marko Bizjak
m.bizjak@um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

David Jesenko
david.jesenko@um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

ABSTRACT

In this paper we present three approaches to identifying vineyard rows from dense drone and sparse plane-based LiDAR data. The implemented solutions address row segmentation using clustering and line fitting algorithms, whose capabilities are enhanced with the addition of normals, orientated bounding boxes and search algorithms. A ground truth, constructed manually based on insights from our preprocessing step, is then used to evaluate the segmentation results. Finally, each solution and its components are evaluated in light of sparse and dense datasets, with each type of dataset highlighting an optimal solution.

KEYWORDS

vineyards, viticulture, LiDAR, DBSCAN, RANSAC, normals, oriented bounding boxes

1 INTRODUCTION

Slovenian viticulture is experiencing a period of rapid decline, with existing vineyards being either reduced in scope or abandoned in their entirety by those who inherit them. The said decline is reported by alarming census data, which are accumulated by the Statistical Office of the Republic of Slovenia. Their 2000 vineyard census reported a total land area of 16,602 ha, while their 2024 census reported 14,102 ha, a stark reduction of 15% in just 24 years [7]. Furthermore, reports on wine self-sufficiency from the Slovenian Statistical Office indicate a decrease in wine self-sufficiency from 95% to 74% from 2022 to 2023 [1]. These reports of viticulture decline at the national level call for action, with potential interventionist policies that require a strong statistical foundation. Our work represents an initial step towards annual automated remote sensing-based accumulation of vineyard census data, with a special focus on the detection of individual vineyard rows, whose yield potential and grapevine integrity are crucial in curbing Slovenia's wine self-sufficiency decline.

The feasibility of using active remote sensing, specifically LiDAR data collected by planes to detect vineyards, was explored by Matthews & Jensen [11], whose 2012 study sought to identify vineyards using unsupervised classification. The latter relied on the repetitive and unique layout of the vineyards, whose grapevine growth is based largely on standardised vine management, laid out in parallel rows with significant spacing between the rows. In addition, the trellising structures are largely constant in height, whose uniformity is distinguished from that of the surrounding area. This height identifier was crucial to the delineation of vineyard parcels from the surrounding area, with the study's method resulting in 96% of vineyard borders being distinguished from non-vineyard data.

The alternative to detecting vineyards at high altitudes lies in the collection of drone-based LiDAR data, which was carried out by Vélez et al. [13]. Their 2023 study collected and made publicly available a total of 10 datasets based on two Spanish vineyards. These datasets were collected over the course of 2 years with a DJI M300 drone, whose primary sensor was a DJI Zenmuse L1 LiDAR. The drone was set to fly at altitudes of 20, 30, and 50 metres, with a 50% side and 80% frontal overlap. The collected LiDAR data were joined by embedded RGB information, giving researchers multiple avenues with which to detect and explore vineyards. Further dataset enrichment, conducted by the same team in 2024 [14], added photographic imagery of the observed vineyards, thus providing additional insights and possible bases for vineyard and grapevine detection.

These datasets created by Vélez et al. are useful for a multitude of vineyard and grapevine analysis tasks, including the detection of individual vineyard rows. We pivoted to this task in particular, with a special focus on using conventional search tree and grouping algorithms in conjunction with normals and oriented bounding boxes. Basic DBSCAN and RANSAC were integrated into three distinct solutions that became increasingly complex. The underlying goal of said increasing complexity was to compare the effectiveness of individual solutions and the impact of specific add-ons, such as normals, density-based normal calculation, and oriented bounding boxes. The results of individual solutions were compared with a manually created ground truth, whose individual rows were extracted in CloudCompare [2]. The final evaluations of the detection accuracy and the viability of the individual solutions were evaluated in light of their speed, accuracy and applicability to different types of data.

This paper is organised as follows. Section 2 presents the methodology, while Section 3 describes the obtained results. Finally, Section 4 concludes the paper.

2 METHODOLOGY

Our overall architecture, as seen in Figure 1, utilises preprocessing and postprocessing steps, with intermediary use of one of three solutions. These are based on DBSCAN and RANSAC, whose clustering and line-fitting capabilities align with the problem of identifying grapevine rows from LiDAR data.

2.1 Data preprocessing

The data we used in our experiments were acquired from the publicly available drone-based LiDAR dataset of Vélez et al. [13] and the publicly available plane-based LiDAR dataset of the Slovenian

https://doi.org/10.51939/scores25.8

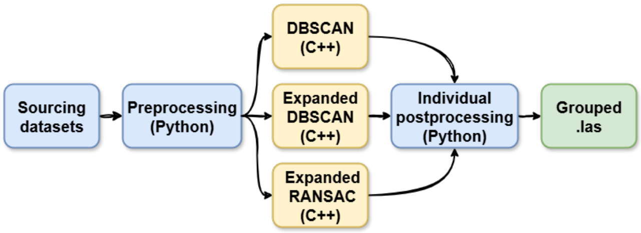


Figure 1: The architecture of the presented methodology.

Environment Agency [3]. We chose the B9 vineyard captured on 08.09.2022 at an altitude of 30 metres from among the study's ten datasets. We then chose a vineyard of similar structure from among the government's data, more specifically, a vineyard from square number 542_122 located within block b_22, which was captured between the 12th of March 2014 and the 20th of October 2014 as part of the D96TM projection.

The chosen vineyards data sets were first converted from a compressed format to a decompressed one using laszip64 [5], one of the LAStools [6] programmes. The decompressed files were processed with a 1 by 1 metre moving window, with each window placement contributing its lowest point to a draft of a digital terrain model. If a cell within the digital terrain model did not contain a point at the end of the window's full grid pass, it was assigned one based on the nearest neighbour value interpolation, effectively giving us a rough terrain model.

The rough terrain model was then used to calculate the height of individual points, which were processed further by a mask that removed all points below 0.35 metres and above 2.25 metres. An additional step was applied to the government's data, which, unlike the study's data, contained point classifications. The government's data were filtered further based on the 3rd and 4th classes, with the goal of retaining the associated low and medium vegetation. Whichever points remained at the end of the preprocessing were then extracted, for both the study's and government's data, to a file which contained only the X, Y, and Z coordinates of each point.

2.2 Ground truth data

A crucial initial step towards constructing our individual solutions was understanding the data that we had on hand. The study's LiDAR data were dense, with a total of 5,180,772 points after preprocessing. The inter-row spacing was vast, with gaps within individual rows being few and far between. However, the observed vineyard did contain a semicircular segment of grapevine. The government's data were sparse, with our chosen vineyard containing only 7,833 points after preprocessing. The inter-row spacing was narrow, with gaps within the individual rows being large and frequent. Moreover, the gaps within the rows themselves often exceeded inter-row spacing in terms of size, thus deviating completely from any uniform pattern. Creating the ground truth, therefore, relied on evaluating the RGB display manually, in the case of the study's data, or the classification display, in the case of the government's data. The manual extraction described of individual vineyard rows was performed using CloudCompare's segmentation tool [4].

2.3 Individual Solutions

In this subsection we will dive into the individual composition of our three solutions. The solutions will be presented with increasing complexity, with the expanded DBSCAN and expanded RANSAC containing many similar components. All three implementations were based on C++, with the code being built as executable files prior to use. The parameters of each solution were set individually for both the study and the government's data. We explore the parameter setting further in the Results section.

2.3.1 Basic DBSCAN

Our basic DBSCAN [9] implementation relied heavily on the use of a KD Tree [8], which was constructed based on its conventional architecture. Individual leaves of the said KD Tree contained points from the input file that was prepared in the preprocessing step. The distance between the points was calculated using the squared Euclidean distance in a 3D space. If a seed point, as per DBSCAN's conventional rules, contained at least a minimum number of points within a set radius, it was expanded into the surrounding area and grouped as part of one cluster. The repetition of the mentioned process resulted in data that were clustered into multiple independent groups, with data outside the bounds of DBSCAN's rules being clustered as noise.

2.3.2 Expanded DBSCAN

Our expanded DBSCAN approached the issue of clustering LiDAR data into individual rows a bit differently. It started with a density-based normal computation, which determined the normals of individual points in light of the point density of their surrounding area. Individual points, which were again read from an input file and organised into a KD Tree, first had their neighbours tallied based on a set radius. If the count exceeded a set density threshold, a high k was used, otherwise, a base k was used. Said k was crucial for the k-nearest neighbours algorithm (KNN) [12], with the neighbours being used to calculate the normals of individual points based on a covariance matrix, eigen decomposition and its principal direction. Each normal was then appended to its corresponding point, enriching the data prior to its evaluation in our DBSCAN.

In addition to evaluating points based on the number of neighbouring points within a set radius, our expanded DBSCAN, which calculated distances in a 2D space, included the angle between the normals in its clustering. The associated angle was limited to a preset value, with points that passed both the neighbours and normals evaluations being considered valid starting points. These were then explored, and had their neighbouring points evaluated as per the rules of DBSCAN, with a normals angle check being integrated into each neighbourhood evaluation. Points that failed to pass these criteria and were not clustered were marked as noise. The identified clusters were then organised in descending order, based on their size, with only the N largest clusters being retained and the rest being marked as noise.

2.3.3 Expanded RANSAC

Our expanded RANSAC took inspiration from our expanded DBSCAN, with density-based normal calculation again being the initial step right after the creation of a KD Tree based on input data in the 2D space. However, our approach ran into line orientation issues due to the nature of RANSAC [10], which finds lines regardless of their orientation. The result was a RANSAC output that did not match the layout of the grapevine rows and, in many cases, intersected multiple independent rows. We addressed this issue by calculating the primary orientation of our observed vineyard based on our calculated normals. Then, a perpendicular line was laid on the primary orientation of the vineyard. Any candidate generated by RANSAC had to intersect with said perpendicular line in order to be considered correctly oriented. Candidates were enlarged temporarily artificially for the purpose of checking their intersection with the perpendicular line, since some would otherwise fall short despite being orientated correctly. An additional requirement imposed at this stage was a maximum angle of 30°, which removed diagonally oriented candidates early on.

In addition to rejecting candidates early with the aforementioned intersection and angle checks, our solution also considered classic RANSAC requirements, such as distance and consensus thresholds. Moreover, candidates had their oriented bounding boxes calculated, with no two boxes being allowed to intersect. If an intersection occurred, the points of each box and their associated candidate lines were returned to the initial point pool. The latter was probed a set number of times, based on a maximum iteration count. If a candidate passed all the criteria, it and its associated bounding box points were marked with a group index, and, as such, considered part of the clustered output.

2.4 Postprocessing

The outputs of all three solutions were similar in structure, with the generated output file containing the X, Y, and Z coordinates of each point and its associated group index. Each group index was unique, with -1 being reserved for noise points.

These output files were read by a Python script, which created a new LiDAR file based on the structure of the associated original file. The scalar fields section of the generated LiDAR file was enriched with an expanded class that accepted more than 31 groups, which is the group count of the default classification class within the scalar fields. Additionally, we assigned each group its own randomly selected colour.

3 RESULTS

The preprocessing step that prepared the data for our three solutions reduced the scope of our problem greatly. The study's data were, for example, reduced by 75% from 20,699,135 points to 5,180,772 points. The government's data, for comparison, were reduced by 65.5% from 22,689 points to 7,833 points. In addition to saving further processing time, the preprocessing step removed ground points, most shrubbery points, and points belonging to trees and other tall vegetation, thus reducing our problem area to the growth dimensions of the grapevine.

The datasets resulting from our preprocessing step required different arrays of parameters. These were tested for individual solutions using PowerShell-based pipelines, which tested a series of parameters within a pre-explored promising range. The results of each pipeline test were evaluated based on the accuracy of the output grapevine row segmentation in comparison to the ground truth. Groups that were present in only one row and presented at least 10% of that row's total points were considered correctly classified. A breakdown of the percentage of correct clustering for each dataset and solution can be found in Table 1. The fastest solution for the study's data was expanded RANSAC, which was 55.7% faster than its DBSCAN-based counterparts, as seen in Table 1. The fastest solution for the government's data was expanded DBSCAN, with the previously fastest expanded RANSAC taking second place, as seen in Table 1. Furthermore, weighing the runtime of individual solutions against their segmentation effectiveness highlights expanded RANSAC as the most balanced solution, with some accuracy, in the case of the study's data, being sacrificed due to a semicircular row whose shape did not adhere to the solution's line-fitting.

The grapevine row segmentation of the study's data using our basic DBSCAN implementation, as represented by Figure 2a, consists of almost completely correctly clustered rows. Small segments of noise were detected alongside the rows and on the periphery. Additionally, some clusters were quite small, resulting in grouped islands among a larger row structure, as can be seen in the red semicircle on the left side of Figure 2a. The only incorrect clustering occurred in a row with a detached top, whose data were too sparse to form a singular row. Significantly more noise and detached row segments appeared in the case of our expanded DBSCAN, whose use of normals resulted in stricter clustering, as seen from Figure 2b. The tail ends of rows were, due to their sparse data, often categorised as noise or small independent clusters. The semicircle on the left of the vineyard was split in half, with plenty of noise appearing throughout its structure. A similar detachment effect was also observed as in the case of basic DBSCAN, marking this solution as highly sensitive to sparse and misoriented data. Our expanded RANSAC, as seen from Figure 2c, ran into significant issues with the semicircular row, whose points were split between four groups. An adjacent row was missed completely during the classification process. However, while the solution was highly sensitive to and found plenty of noise, it grouped rows quite accurately, which is a result of the oriented bounding boxes imposed on each RANSAC candidate.

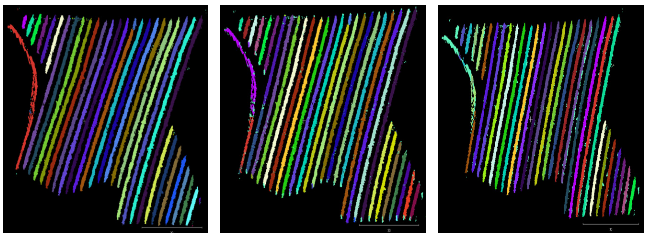

(a) (b) (c)

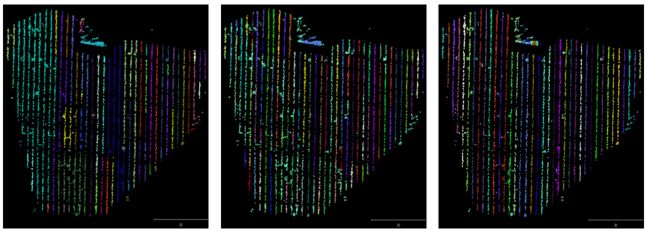
Figure 2: Segmented study data, where (a) DBSCAN, (b) expanded DBSCAN and (c) expanded RANSAC. Each segment has its own colour, with noise shown in light blue. Colours spanning multiple rows indicate incorrect segmentation. Optimal segmentation features one colour per row.

While the basic DBSCAN proved highly effective for the study's data, it proved highly ineffective for the government's data, whose segmentation can be seen in Figure 3a. Multiple groups spilled over into adjacent rows, with some rows being split into multiple groups. A few rows, whose structure was relatively dense and compact compared to most rows from the government's dataset, were identified in their entirety. A different result, as seen in Figure 3b, emerged from using our expanded DBSCAN, whose sensitivity to orientation, due to the inclusion of normals, prevented inter-row spillage effectively. Moreover, many of the inter-row points that previously

Table 1: A statistical overview of each solution's row segmentation success rate and processing time per dataset as an average of 10 runs.

Dataset	Solution	Mean [%]	Min. [%]	Max. [%]	Std.	Normals (s)	Clustering (s)	Total (s)
Study	DBSCAN	99.90	97.63	100	0.43	/	4,320.21	4,320.21
	Expanded DBSCAN	97.77	93.01	99.58	1.59	658.12	1,764.39	2,422.51
	Expanded RANSAC	94.74	0.00	99.95	18.64	658.94	413.96	1,072.9
Government	DBSCAN	48.89	0.00	100	43.26	/	0.089	0.089
	Expanded DBSCAN	80.05	0.00	100	20.02	0.032	0.023	0.055
	Expanded RANSAC	96.17	0.00	100	16.11	0.0042	0.075	0.0792

caused spillage were classified as noise. However, the implemented solution still faced issues when clustering the government's sparse and gap-filled data, with most lines being split into multiple groups. The expanded RANSAC proved the most effective, as seen from Figure 3c, when faced with scarce and gap-filled data. The normals proved their effectiveness in row delineation, with inter-row noise being classified as such. Moreover, the use of oriented bounding boxes resulted in uniform row segmentations. However, one row was classified as noise due to the angle associated with its candidate points.



(a) (b) (c)

Figure 3: Segmented government data, where (a) DBSCAN, (b) expanded DBSCAN (c) and expanded RANSAC. Each segment has its own colour, with noise shown in light blue. Colours spanning multiple rows indicate incorrect segmentation. Optimal segmentation features one colour per row.

4 CONCLUSIONS

Preprocessing reduced our problem significantly, and the KD Tree additionally lowered the processing time of each implemented solution. Another significant addition was the use of density-based normals, which proved effective in delineating rows, especially in areas prone to inter-row spillage due to noise. When the normals were combined with oriented bounding boxes, they proved effective, both for dense and sparse data. The dense drone data and the sparse plane data posed different challenges that highlighted the pros and cons of individual solutions. The simplest solution, the basic DBSCAN, while the slowest, proved effective in dense data with clear-cut and uniform rows, but missed the mark in the case of sparse data. The expanded DBSCAN proved slightly better, especially in terms of detecting noise, but fell short of the versatility that the expanded RANSAC showcased. We can, therefore, say that our expanded RANSAC, besides being relatively fast, proved the most universal solution that produced acceptable segmentation of dense study and quality segmentation of the sparse government data. If our RANSAC implementation had been adjusted to other geometric shapes, including semicircles, it would have been the most effective solution across the board.

Our future work will be to adjust our expanded RANSAC to include multiple geometric shapes. Moreover, the study's dense data could prove useful in identifying components of individual rows, such as the grapevines themselves, thus taking our identification of vineyards from LiDAR data deeper than just row segmentation, making it viable for annual automated vineyard census taking.

ACKNOWLEDGMENTS

This research was supported by the Slovenian Research and Innovation Agency through Research Project J2-4424 and Research Programme P2-0041.

REFERENCES

[1] 2025. *Bilanca po: PROIZVODNJA IN PORABA, VINO , LETO. PxWeb.* https://pxweb.stat.si/SiStatData/pxweb/sl/Data/Data/1563407S.px/table/tableViewLayout2/ Accessed: 3.9.2025.

[2] 2025. *CloudCompare - Open Source project.* https://www.cloudcompare.org/ Accessed: 3.9.2025.

[3] 2025. *gis.arso.gov.si/geoportal/.* https://gis.arso.gov.si/geoportal/ Accessed: 6.8.2025.

[4] 2025. *Interactive Segmentation Tool - CloudCompare wiki.* https://www.cloudcompare.org/doc/wiki/index.php/Interactive_Segmentation_Tool Accessed: 3.9.2025.

[5] 2025. *LASzip.* https://rapidlasso.de/laszip/ Accessed: 29.7.2025.

[6] 2025. *Overview of all LAStools.* https://rapidlasso.de/product-overview/ Accessed: 3.9.2025.

[7] 2025. *Skupine zemljiških kategorij in njivskih posevkov (ha) po: RABA KMETIJSKIH ZEMLJIŠČ V UPORABI, LETO. PxWeb.* https://pxweb.stat.si/SiStatData/pxweb/sl/Data/Data/1502401S.px/table/tableViewLayout2/#:~:text=16.602,14.102 Accessed: 11.5.2025.

[8] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (Sept. 1975), 509–517. https://doi.org/10.1145/361002.361007

[9] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise.* https://ui.adsabs.harvard.edu/abs/1996kddm.conf..226E Conference Name: Second International Conference on Knowledge Discovery and Data Mining (KDD'96). Proceedings of a conference held August 2-4 Pages: 226-331 ADS Bibcode: 1996kddm.conf..226E.

[10] 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

[11] Adam J. Mathews and Jennifer L. R. Jensen. 2012. An airborne LiDAR-based methodology for vineyard parcel detection and delineation. *International Journal of Remote Sensing* 33, 16 (Aug. 2012), 5251–5267. https://doi.org/10.1080/01431161.2012.663114 Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/01431161.2012.663114.

[12] Jingwen Sun, Weixing Du, and Niancai Shi. 2018. A Survey of kNN Algorithm. *Information Engineering and Applied Computing* 1 (May 2018). https://doi.org/10.18063/ieac.v1i1.770

[13] Sergio Vélez, Mar Ariza-Sentís, and João Valente. 2023. VineLiDAR: High-resolution UAV-LiDAR vineyard dataset acquired over two years in northern Spain. *Data in Brief* 51 (Dec. 2023), 109686. https://doi.org/10.1016/j.dib.2023.109686

[14] Sergio Vélez, Mar Ariza-Sentís, and João Valente. 2024. EscaYard: Precision viticulture multimodal dataset of vineyards affected by Esca disease consisting of geotagged smartphone images, phytosanitary status, UAV 3D point clouds and Orthomosaics. *Data in Brief* 54 (June 2024), 110497. https://doi.org/10.1016/j.dib.2024.110497

Comparing dimensionality reduction methods for predicting skin deformations in 3D models

Rok Sekirnik
rok.sekirnik@student.um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

Marko Bizjak
m.bizjak@um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

David Jesenko
david.jesenko@um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

Abstract

In this paper we present a neural network for predicting skin deformations of 3D characters. We focus on dimensionality reduction techniques, namely, Principal Component Analysis and autoencoders. We compare both approaches in terms of accuracy, usability and computational cost, and provide guidelines for their practical application in generating realistic animated characters for film, computer graphics and real-time environments. The results indicate that autoencoders offer higher prediction quality and better generalisation of complex deformations, whereas PCA excels in the speed and simplicity of implementation. Our findings confirm that the choice of method depends on specific problem requirements, enabling a flexible and efficient approach for various scenarios in computer graphics.

Keywords

PCA, autoencoder, neural network, machine learning, physical simulation, mesh deformation, skin deformation, 3D models, Houdini

1 Introduction

Realistic animation of digital characters has long been a central challenge in computer graphics. In films, games and virtual production, audiences expect subtle details such as skin sliding, muscle bulging and soft tissue motion to appear natural, yet producing these effects remains expensive and time consuming. Physics-based simulations can deliver convincing results, but they are computationally heavy and rarely suitable for real-time use [4, 8].

Machine learning has emerged as a promising alternative. Instead of hand-crafting every corrective motion or relying entirely on slow simulations, data-driven methods can learn from examples and predict complex deformations directly from skeletal poses. One of the main challenges is the sheer amount of data involved. A single surface mesh can contain tens of thousands of vertices, making predicting deformations directly both inefficient and impractical. For this reason, compact representations of the deformation data are essential.

Classical methods such as Principal Component Analysis (PCA) have been used widely in computer graphics for dimensionality reduction [3, 7]. However, they are inherently linear, and struggle to capture the non-linear nature of soft tissue dynamics. Autoencoders, on the contrary, can learn compact non-linear representations that preserve complex deformation patterns better while reducing the computational load [2].

This paper investigates and compares these two strategies in the context of 3D character deformation. A comprehensive pipeline was developed, spanning from automated data generation to model training and integration in a production environment, to assess the trade-offs between accuracy, complexity and runtime efficiency.

This paper is organised as follows. Section 2 presents the problem description, while Section 3 reviews the related work. The implementation is described in Section 4, and our findings are discussed in Section 5. Finally, Section 6 concludes the paper.

2 PROBLEM DESCRIPTION

The main objective of this work is to predict realistic surface deformations of 3D characters based on skeletal configurations. Traditional skinning techniques cannot reproduce many secondary effects, such as soft tissue shifts or complex body interactions [5]. Full physical simulations, while accurate, are far too slow for real-time applications. The objective is, therefore, to find a practical compromise, for example, a data-driven method that captures the richness of simulation while remaining fast enough for interactive use.

A key step in achieving this goal is the reduction of dimensionality. The space of all possible vertex displacements is vast, often involving tens of thousands of dimensions. Compressing this information into a smaller latent representation makes learning feasible and reduces the memory footprint of the prediction model. Linear methods, such as PCA, provide a simple way to perform this compression, but may lose crucial nonlinear deformation details. Autoencoders, on the other hand, offer a nonlinear alternative that can potentially capture more complex behaviour [10].

The overall pipeline can be broken down into several steps. First, a data set of poses and the corresponding deformations is generated through physical simulation. Next, the deformation data are compressed using either PCA or an autoencoder. A regression network is then trained to map the skeletal pose parameters to the reduced latent space. Finally, the predicted latent representation is decoded back into full vertex displacements and applied to the mesh. This process enables the model to predict realistic deformations in real-time, while keeping computational demands manageable.

3 RELATED WORK

The deformation of 3D characters has long been tackled with a mix of simple, fast tricks and heavyweight physics. The most common baseline is skeletal animation with skinning and weight painting, which is cheap and easy to use, but produces artefacts like collapsing or sharp folds. Artists therefore often add corrective shapes, or run separate physical simulations for problematic regions [5]. At the opposite extreme, multilayer physical simulations model muscles,

https://doi.org/10.51939/scores25.9

fat and skin for biomechanically plausible motion, but they are costly and reserved mostly for high-end film work [4, 8].

More recent approaches fall somewhere between these two extremes. Learning-based techniques aim to approximate high-fidelity simulation results at reduced computational cost, for instance by predicting vertex offsets or surface displacements using neural networks. Mesh-aware deep models work directly on 3D surface meshes, which are inherently irregular, and exploit the local topology to generate more expressive and spatially consistent surface features [6, 9]. Other works explore physics-informed neural networks (PINNs) that blend mechanical knowledge with neural networks to achieve more realistic outcomes [1]. Our approach takes a different, pragmatic path: we use a fixed, canonical surface ordering, so that every training sample has the same set of vertices in the same order. This allows deformation fields to be treated as dense, per-vertex data, making it easy to compare linear (PCA) and nonlinear (autoencoder) compression methods and streamlining integration into a Houdini-based workflow.

4 IMPLEMENTATION

This chapter describes the practical implementation of the pipeline developed in this paper. The following development order follows: system architecture, physics-based simulation and deformation transfer, geometric data preprocessing, latent space compression, and the regression network used for pose-to-deformation prediction. Each of the mentioned steps is described in detail in the following subsections. The focus is on the concrete choices and engineering steps needed to reproduce the workflow and to integrate the trained models back into a Houdini-based production environment.

4.1 Architecture description

We organised the system as a compact, modular pipeline with two clear stages: a learning phase and an inference phase. During learning, the models are trained on a large set of simulated deformation examples, so they learn the mapping between a skeleton configuration and the resulting surface response. During usage the trained model receives a new skeletal pose, and returns the predicted deformations, which are then applied to the mesh.

The skeleton poses are generated procedurally, the input surface is converted to a tetrahedral volume for the physics simulation, and the resulting per-vertex displacements are collected as training data. The pipeline then reduces the high-dimensional deformations, either by PCA or an autoencoder, trains a regression network to map poses to the compact latent representation, and, finally, in the inference phase, decodes the code back to full vertex offsets. The trained networks are exported as ONNX models and re-plugged into Houdini for real-time inference. The modular design makes it straightforward to swap components, such as dimensionality reduction methods, or extend the system later.

4.2 Physical simulation and deformation of 3D mesh

This subsection describes the whole process used to produce high-quality deformation examples: preparing the geometry, generating skeleton poses, running soft-tissue simulations on a tetrahedral volume, and, finally, transferring the resulting displacements back to the surface mesh. The goal was to create consistent geometry-aligned deformation data suitable for training and validating neural models.

We start with a clean polygon surface of a capybara model, clean the topology, and adjust the mesh density so that areas that require detailed deformations have enough resolution. Then, we convert the polygon surface into a volumetric tetrahedral mesh ready for simulation. For the skeleton, we limited the setup to only the neck and head joints, both because of hardware constraints and the training complexity of neural networks, and also because these joints produce large, well-defined folds that the network can learn more quickly and effectively. Then, we set realistic joint rotation and translation limits, derived from motion analysis of various animations, and generated a wide set of poses procedurally using Houdini's pose generation tools. These sampled skeleton configurations formed the inputs to the simulation stage.

The physical simulations were performed inside Houdini on the prepared tetrahedral model. To isolate skin behaviour, we disabled gravity and ran a two-phase 20-second simulation. The first 10 seconds were used for the dynamic transition of the model to the target pose, and the last 10 seconds allowed the tissue to settle, producing the final deformed tetrahedral mesh. Then, those displacements are mapped onto the original polygon surface, to create the training targets used by the rest of the system.

4.3 Geometric data preprocessing

The goal of the pre-processing is to turn the raw simulation output into a compact, consistent description of the surface deformations that the learning stage can use. To do this, the pipeline first establishes a local 3×3 coordinate system for every vertex of the base polygon mesh. Expressing vertex displacements with the help of these local coordinate systems removes the influence of global rotations and indirect motions coming from neighbouring joints, so the remaining vectors reflect only the true local skin response.

The polygon mesh deformations were then converted into a point cloud to facilitate learning. For each surface point, we computed the raw offset by subtracting (see Equation 1) the deformed points from their original positions, so the data contain only per-vertex displacement vectors:

$$\Delta x_i = x_i^{\text{deform}} - x_i^{\text{orig}} \tag{1}$$

Each offset is then expressed (see Equation 2) in the local coordinate system of the vertex, ensuring that the data set is invariant to the global pose of the character:

$$\Delta x_i^{\text{local}} = T_i^{-1} \cdot \Delta x_i \tag{2}$$

Finally, we apply a region mask to remove points outside the target area of the head and neck. Because the simulation ran on the entire character, distant regions sometimes sagged or shifted, introducing offsets unrelated to the head and neck motion. The mask therefore excludes those irrelevant displacements. The masked vertices are set to (0,0,0), which prevents meaningless data from degrading training, and produces a cleaned point cloud of local relative displacements ready for dimensionality reduction and network training.

4.4 Dimensionality reduction

To make the per-vertex displacement clouds usable for regression, we compress them into a compact latent representation. This paper evaluates two complementary approaches: a non-linear autoencoder and the classical linear PCA. Both methods were restricted to the same latent dimensionality (128), allowing their results to be compared directly in later stages.

The autoencoder implementation was built with simplicity and robustness in mind. An encoder-decoder baseline architecture, input normalisation, and an explicit mask to eliminate noisy or irrelevant points were all part of the design. The mask and normalisation help the network ignore uninformative vertices and stabilise training. The model was upgraded progressively, with the depth and capacity increased, and, finally, trained on an extended data set of 51,200 samples until the learned latent codes produced faithful reconstructions of the target deformations. This route yields a flexible non-linear representation that captures the complex non-linear behaviour of the skin better.

PCA was implemented as a practical reference method. After collecting the deformation samples into a single accumulated point cloud and standardising the data, a PCA analysis was carried out to extract the top 128 principal components. The approach is quick and straightforward to deploy, making it a solid baseline.

4.5 Neural network implementation for deformation prediction

The regression model is a compact, fully connected neural network that maps the skeletal pose information to the latent deformation code produced by the dimensionality reduction methods. As input, it takes the 3×3 transformation matrices of the head and neck joints, which encode the rotation and scale of each joint. Each matrix has 9 values, and flattening the two matrices results in 18 features in total. The architecture itself consists of a small number of dense layers with non-linear activations, designed to keep the inference lightweight while still capturing the relationship between pose and deformation faithfully.

Training is performed on a simulated dataset as described above. The loss is a straightforward mean squared error between the predicted latent vector and the ground truth deformation. A held-out validation set is used to stop training and choose the best model. The training procedure is the same, regardless of which latent representation is used, and the setup allows models to scale naturally with different dataset sizes.

For deployment, the trained networks are exported as ONNX models and embedded back into the Houdini pipeline. At run time, an incoming pose is converted to the two 3×3 joint matrices, normalised, passed through the network to obtain a latent code and then decoded to full vertex offsets that are applied to the mesh, resulting in the finished deformed surface. The components are coupled loosely making it easy to swap the latent representation or adjust the regression model without changing the rest of the pipeline.

5 RESULTS

This chapter presents the results of our approach. It includes quantitative measures, visual examples, training and inference times, and a discussion of the trade-offs between accuracy and computational cost. All the experiments were performed on a computer with an AMD Ryzen 9 3900X 12-core CPU, 32 GB of RAM, and an NVIDIA GeForce RTX 2080 GPU running Windows 11.

5.1 Quantitative comparison

We evaluated both pipelines (PCA and autoencoder) on a set of 10,240 validation poses, and compared the predictions to the reference deformations produced by the physics simulation. Quantitative assessment uses two complementary metrics: the Root Mean Square Error (RMSE) measured in metres, and the Chamfer distance between the predicted and simulated point clouds. Tables 1 and 2 summarise the key statistics.

Table 1: RMSE comparison between PCA and the Autoencoder

Method	Mean (m)	Std (m)	Min (m)	Max (m)
PCA	2.79×10^{-3}	1.18×10^{-3}	2.55×10^{-3}	8.52×10^{-3}
Autoencoder	9.63×10^{-5}	5.88×10^{-5}	8.07×10^{-5}	1.20×10^{-3}

The RMSE results in Table 1 reveal a large gap between the two reduction strategies: PCA yielded a mean RMSE of 2.79×10^{-3} m, while the autoencoder achieved a mean RMSE of 9.63×10^{-5} m. These numbers indicate that the autoencoder reconstructions are substantially closer to the simulation ground truth across the validation set.

Table 2: Chamfer distance comparison between PCA and the Autoencoder

Method	Mean (m)	Std (m)	Min (m)	Max (m)
PCA	2.78×10^{-6}	3.75×10^{-6}	1.40×10^{-6}	4.64×10^{-5}
Autoencoder	4.10×10^{-8}	5.29×10^{-8}	2.66×10^{-8}	1.77×10^{-6}

The Chamfer distance statistics in Table 2 confirm the RMSE findings. The Chamfer mean of PCA is 2.78×10^{-6}, while the mean of the autoencoder is 4.10×10^{-8}. The lower Chamfer values for the autoencoder show that its predictions are not only closer on average, but also more geometrically consistent with the simulated shapes, even when the vertices do not correspond one-to-one.

5.2 Qualitative comparison

We also inspected the qualitative results. The examples in Figure 1 compare the original simulated mesh (shown as white mesh) with the reconstructions of the PCA and the autoencoder (shown as a grey 3D model). PCA reconstructions often appear overly smooth and fail to capture sharper localised features, while the autoencoder outputs preserve more detail and follow complex head and neck motions better. These visual checks align with the quantitative metrics and help explain where PCA loses expressive power.

5.3 Training time comparison

Importantly, the pre-computation and training costs differ dramatically between the two approaches. The PCA analysis is very cheap, less than five minutes for the complete set, while training the autoencoder is much more expensive: the training took about 9 hours on 1,024 samples and 516.12 hours (≈ 21.5 days) on 51,200 samples. However, the final regression networks had similar training times for both pipelines (≈ 4.6 h for the PCA-based pipeline vs. ≈ 4.87 h for the AE-based one), so the main time penalty comes from

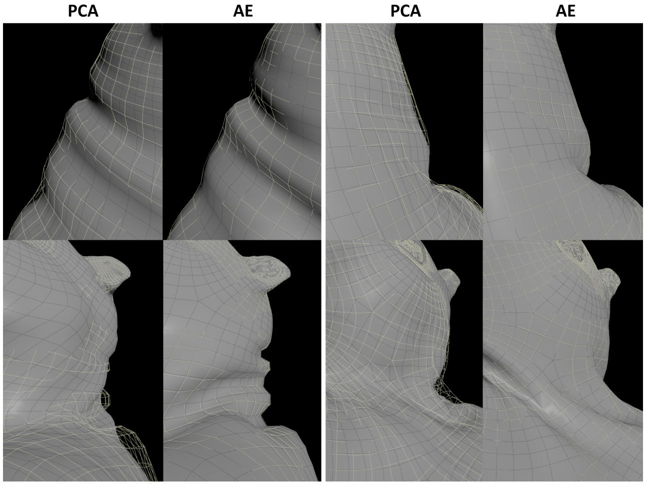


Figure 1: Comparison of simulation quality between the PCA and autoencoder process. Original physical simulation represented by transparent mesh.

learning the non-linear encoder rather than from the regression step itself.

5.4 Comparison with the physical simulation

Finally, we measured the timing to assess practical utility. A complete physical soft tissue simulation for a 5-second animation at 24 FPS (120 frames) took roughly 600 seconds. In contrast, the ONNX inference pipeline produced a prediction in about 24 seconds, which is approximately 5 frames per second, or about 25 times faster than the physical simulation. Although this does not yet reach 24 FPS interactivity on the machine used, it represents a substantial speedup that makes near-real-time and preview workflows feasible.

5.5 Discussion

The results show a clear trade-off between representational power and practical cost. The autoencoder-based latent space produces reconstructions that are markedly closer to the simulation ground truth, while PCA offers a much cheaper deterministic baseline. These differences are evident in the quantitative Tables and Figures.

That precision comes at a price. Computing the non-linear encoder is the dominant offline cost (hours to days, depending on the dataset size), whereas PCA and the regression training are much faster to prepare. Practically, this means that teams must trade longer precomputation times for higher fidelity.

At runtime, the learned pipelines provide substantial savings compared to full soft tissue simulation, producing usable results in a fraction of the time. Although the current implementation does not yet achieve real-time playback on the test hardware, it is already fast enough for interactive previews and efficient rendering of animations. This balance between speed and accuracy makes this approach a practical tool for production workflows where rapid iteration is more valuable than perfect physical accuracy.

6 CONCLUSION

We present a practical pipeline for approximating soft tissue deformations of a 3D character from skeletal poses. By generating a large set of simulation-driven pose–deformation pairs, preprocessing the data into local per vertex deformations, compressing the resulting displacement fields, and training a compact pose-to-latent-space regression network, the system produces plausible surface deformations at almost real-time speeds, and it integrates directly into a Houdini-based production workflow.

The core empirical finding is a clear trade-off, as the latent space of a nonlinear autoencoder captures simulation details much better than the linear PCA method, but learning that nonlinear mapping costs substantially more training time. At runtime, however, the models produce vertex offsets in a fraction of the time required for a full soft-tissue simulation, making them useful for rapid previews and animation pipelines, even when they do not yet deliver film-rate playback on the test hardware. These results quantify the practical compromise between fidelity and time cost that the studios must weigh.

There are several straightforward avenues for improvement, such as expanding pose coverage, increasing mesh fidelity in critical regions, adding temporal models, and exploring physics-informed neural networks that incorporate mechanical laws into the learning process. Overall, the work demonstrates that learned approximations can bring simulation-level detail into faster content-creation workflows, provided that teams are willing to accept the initial training investment.

Acknowledgments

This research was supported by the Slovenian Research and Innovation Agency through Research Project J2-4424 and Research Programme P2-0041.

References

[1] Federica Caforio, Francesco Regazzoni, Stefano Pagani, Elias Karabelas, Christoph Augustin, Gundolf Haase, Gernot Plank, and Alfio Quarteroni. 2025. Physics-informed Neural Network Estimation of Material Properties in Soft Tissue Nonlinear Biomechanical Models. *Computational Mechanics* 75, 2 (2025), 487–513. https://doi.org/10.1007/s00466-024-02516-x

[2] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. 2006. Reducing the Dimensionality of Data with Neural Networks. *Science* 313, 5786 (2006), 504–507. https://doi.org/10.1126/science.1127647

[3] H. Hotelling. 1933. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology* 24 (1933), 417–441. https://doi.org/10.1037/h0071325

[4] Naoya Iwamoto, Hubert P. H. Shum, Longzhi Yang, and Shigeo Morishima. 2015. Multi-layer Lattice Model for Real-Time Dynamic Character Deformation. *Computer Graphics Forum* 34, 7 (2015), 99–109. https://doi.org/10.1111/cgf.12749

[5] Ladislav Kavan, Steven Collins, Jan Žára, and Christopher O'Sullivan. 2007. Skinning with Dual Quaternions. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*. ACM, 39–46. https://doi.org/10.1145/1230100.1230107

[6] Yudi Li, Min Tang, Yun Yang, Zi Huang, Ruofeng Tong, Shuangcai Yang, Yao Li, and Dinesh Manocha. 2022. N-Cloth: Predicting 3D Cloth Deformation with Mesh-Based Networks. 41, 2 (2022), 547–558. https://doi.org/10.48550/arXiv.2112.06397

[7] K. Pearson. 1901. LIII. On lines and planes of closest fit to systems of points in space. *Philos. Mag.* 2, 11 (1901), 559–572. https://doi.org/10.1080/14786440109462720

[8] Shunsuke Saito, Zi-Ye Zhou, and Ladislav Kavan. 2015. Computational Bodybuilding: Anatomically-based Modeling of Human Bodies. In *Proceedings of ACM SIGGRAPH 2015 / ACM Transactions on Graphics (TOG)*, Vol. 34. 1–12. https://doi.org/10.1145/2766957

[9] Edgar Tretschk, Ayush Tewari, Michael Zollhöfer, Vladislav Golyanik, and Christian Theobalt. 2020. DEMEA: Deep Mesh Autoencoders for Non-Rigidly Deforming Objects. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, Proceedings, Part IV*. Springer, 601–617. https://doi.org/10.1007/978-3-030-58548-8_35

[10] Laurens van der Maaten, Eric Postma, and Jaap van den Herik. 2009. *Dimensionality Reduction: A Comparative Review*. Technical Report. Maastricht University, MICC. https://lvdmaaten.github.io/publications/papers/TR_Dimensionality_Reduction_Review_2009.pdf

Prompts or Policies: Which Plays Yamb Better?

Anton Žabkar
anton.zabkar@gmail.com
Faculty of Mechanical Engineering,
University of Ljubljana
Aškerčeva cesta 6
SI-1000 Ljubljana, Slovenia

Vid Kališnik
vidkalisnik@gmail.com
Faculty of Computer and Information Science,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

Jure Žabkar
jure.zabkar@fri.uni-lj.si
Faculty of Computer and Information Science,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

ABSTRACT

We compare two large language models, two reinforcement learning algorithms and human players in playing Yamb. Yamb is a social dice game that requires a combination of tactical dice-rolling decisions and long-term strategic planning. Our aim is to study the differences in their strategies and their final scores. In our experiments, reinforcement learning method PPO achieves the score of a beginner human player after 1M steps and advances to an average human level after 10M learning steps. AlphaZero achieves this level considerably faster. LLMs can play without pretraining but take longer time in making decisions and achieve beginner human level performance. The most advanced LLM model in our experiments was *ChatGPT-5-Thinking* which provided reasonable explanations aligned with experienced human player.

KEYWORDS

Yamb, Large language models, AI in games, Reinforcement learning

1 INTRODUCTION

Modern game AI is strongest in deterministic, perfect-information games such as Chess and Go [3, 4]. While playing, humans mostly rely on intuition and recognizing the patterns, computers outplay them in performing deep search (neural nets + Monte-Carlo tree search) and learned value policies (using Reinforcement Learning [3]). In Chess and Go, the same position always leads to the same outcomes under the same play - there is no randomness in the game. In this paper, we investigate a different type of game, called Yamb that is a perfect-information game, but also stochastic. We focus on the capabilities of LLMs in playing the game and compare the scores to humans' and reinforcement learning (RL), which was investigated in [1].

Yamb [5] is a dice game in which the objective of the player is to score as many points as possible using strategy and luck. The name varies across regions - it is also called Jamb, Jima or Džima. It originates from the Balkans. It is closely related to Yahtzee/Yatzy but using a more elaborate scorecard with multiple columns and categories. It is typically played with five dice. The player's strategy is in reading the odds while managing limited slots across columns. Luck comes from the dice: good choices can still fail, and risky ones can sometimes succeed. Over many games, smart play pays off, but the randomness keeps the game unpredictable.

1.1 The rules of Yamb

A player may roll up to three times per turn, choosing after each roll which dice to keep and which to re-roll. After the final roll the player must record a score (or zero) in an available box. The game ends when all boxes are filled. The scorecard (see Fig. 1) is divided into an "upper" section (ones through sixes) and a "lower" section of combinations. Lower-section categories include Max and Min (sum of all dice; scores as $(\text{Max}-\text{Min})\times 1s$ in the same column), straights, full house, poker (four-of-a-kind), and yamb (five-of-a-kind). The four columns constrain when rows may be filled: ascending (fill in order from 1s to yamb), descending (filled from yamb to 1s), free (anytime), and announced (choose the target row immediately after the first roll).

Combination	↓	↑	↑↓	**A**	Σ
1s	4	5	3	4	
2s	0	6	4	4	
3s	9	9	3	12	
4s	12	8	4	12	
5s	15	5	15	20	
6s	12	12	18	18	
Σ	**52**	**45**	**47**	**100**	**244**
Max	26	24	24	25	
Min	10	6	11	8	
$(\text{Max}-\text{Min})\times 1s$	**64**	**90**	**39**	**68**	**261**
Tris	19	25	25	19	
Full House	32	0	35	33	
Straight	0	0	50	45	
Poker	48	64	48	60	
Yamb	0	60	0	0	
Σ	**99**	**149**	**158**	**157**	**563**
SCORE					**1068**

Table 1: Example of a filled-in Yamb score sheet.

1.2 Playing strategies

Strategy comes into play in two different forms. In short term you need to decide which dice you will put aside and which to roll again. On the other hand long term strategy decides which columns are filled first, which boxes get sacrificed for really bad rolls and which are left for the very end of the game when you are forced to fill them with zero if you do not get the correct numbers.

Typical results for an average human player are between 800 and 900 and can get up to between 1000 and 1300 for experienced players.

Designing a strategy for this game differs from finding the best moves in other board games like Chess and Go because we do not know how the game will progress in the future due to random

https://doi.org/10.51939/scores25.10

rolls of dice. This also makes finding the best strategy using reinforcement learning a bit harder. A great final score using a specific strategy can be a result of luck and does not mean that this strategy is always the best option. You must adapt your long-term plans at every turn.

2 METHODOLOGY

We implemented the game in Python and used it to compare two RL algorithms (*PPO*, *AlphaZero*), two LLMs (*ChatGPT-5-mini* and *ChatGPT-5*), and 3 experienced human players. Our aim is two-fold: (i) to determine which method scores most points, and (ii) what strategy each model would use. The experiments with reinforcement learning algorithms are described in detail in [1].

2.1 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [2] is a popular reinforcement learning method that trains an agent to choose actions by improving a policy step by step, but without changing it too much at once. It's an on-policy, actor–critic algorithm: the actor proposes actions, the critic estimates how good they were, and PPO uses that feedback to update the policy. PPO is robust and easy to tune. It works with both discrete and continuous actions, and is often used in AI games, control problems, and robotics.

Figure 1 shows the learning process of the PPO algorithm with 3 different reward functions [1]:

end The agent receives the reward only at the end of the game. After 10M steps, PPO scores approximately 500–550 points, which shows that this type of reward does not provide enough information for effective learning.

sparse The rewards are assigned after each turn based on the current score when filling the score-sheet. The agent learns faster and manages to achieve around 750 points.

shaped The best results are achieved by shaping the rewards. In this case, the agent progresses quickly in the initial stages and eventually scores 850–900 points.

2.2 AlphaZero

AlphaZero is a self-play reinforcement learning system that learns board games from scratch using only the game rules. It trains a single deep neural network that, given the current state, predicts which actions are promising. A single game with the *AlphaZero* algorithm is more time-consuming compared to PPO, since for each move selection it is necessary to expand the Monte Carlo tree, in our case to 300 nodes. Because of this, it was possible to play much fewer games than with PPO, which affects the results especially for our game, where the variance is very large. In our case, we played and saved 20 games in each iteration, and then trained the model for 10 epochs. A total of 20 such iterations were performed, which means that the number of training examples was much smaller compared to the other algorithms.

The *AlphaZero* algorithm was originally designed for games where the outcome is known only at the end, the reward is determined according to victory, draw or defeat, and has the values -1, 0 and 1, respectively. Since Yamb has a different reward structure, it was necessary to normalize the rewards accordingly. For this purpose, we determined the approximate maximum of the game (1500 points) and the minimum (300 points) and used the following equation $r = 2 \cdot \frac{\text{score}-\text{min}}{\text{max}-\text{min}} - 1$, which maps the results to the interval $[-1, 1]$. Despite the smaller number of games played, the model learned a strategy that averages 800 points.

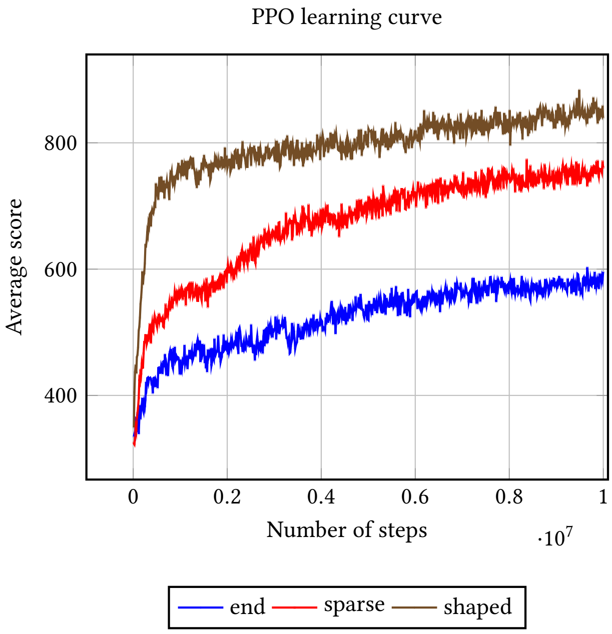


Figure 1: Comparison of learning PPO model with different rewards.

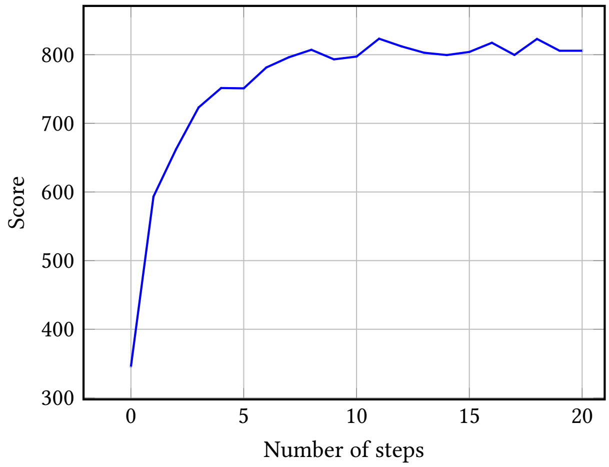


Figure 2: Learning curve of AlphaZero.

2.3 ChatGPT-5

ChatGPT-5-mini was used via OpenAI Responses API. Each prompt provided the full information of the current state of the game and the tools were used to enforce the structured response of the LLM agent. The agent also receives a list of dice rolls and gets the opportunity to make an announcement. After each roll, the agent chooses the dice to hold, the target category and the column, along with a short explanation of its reasoning. The following is an example from the first turn in a game:

```
===== TURN 1/52 =====
Roll 1 -> [3, 6, 5, 6, 3]
Model decision: {
'action': 'hold',
'hold_indices': [1, 2, 4, 5],
'category': 'FH', 'column': 'f',
'rationale': 'Hold both 3s and both 6s (indices
    1,2,4,5) and reroll only the 5. With 2 rerolls
     left this best converts to Full House (need
    one more 3 or 6) while preserving pair options
    .'
}
Roll 2 -> [3, 6, 1, 6, 3]
Model decision: {
'action': 'hold',
'hold_indices': [2, 4],
'category': '6', 'column': 'f',
'rationale': 'Hold both 6s (indices 2 & 4) to
    reroll the other three dice aiming to get more
     6s-improves chances for 6/Max/P/FH/Yamb in
    the final roll.'
}
Roll 3 -> [3, 6, 6, 6, 2]
Model decision: {
'action': 'score',
'hold_indices': [],
'category': 'T', 'column': 'f',
'rationale': 'Roll 3: must score. Dice show three
    6s (sum = 3+6+6+6+2 = 23). Score T (three-of-a
    -kind) in free column to take 28 points,
    better than scoring only 6s (18).'
}

Scored T in 'f' for 28 with dice [3, 6, 6, 6, 2]

Current scoresheet:
      d     u     f     a
1    None  None  None  None
2    None  None  None  None
3    None  None  None  None
4    None  None  None  None
5    None  None  None  None
6    None  None  None  None
Max  None  None  None  None
Min  None  None  None  None
T    None  None    28  None
FH   None  None  None  None
S    None  None  None  None
P    None  None  None  None
Y    None  None  None  None
```

2.4 Comparison of LLM, PPO and AlphaZero

Due to high time-complexity and pricing, we ran *ChatGPT-5-mini* to play only 10 games. It scored an average of 740 points with the minimal score of 567 points and maximal score of 888 points. AlphaZero and PPO were more effective and averaged 865 and 807 points. Regarding the strategy, PPO often chooses the first two columns at the beginning of the game, filling in the less valuable or riskier squares with lower scores, in order to have more chances to score higher points in other categories. A distinctive feature of its strategy is the early completion of the yamb field in the second column, regardless of whether it manages to roll it or not (it fills in 0). This increases its chances of using its rolls more efficiently in other, more likely categories. The strategy of AlphaZero resembles the approach of a beginner human player. The model initially fills the squares in the third and last columns, and partly also in the first, while in the second column it waits for a yamb. *ChatGPT-5-mini* has a very similar strategy and first fills free and announcement columns. It also prioritizes the bottom section (see Table 2).

Table 2: Comparison of average results in different sections on score sheet

Section	Upper sum	Max-Min	Bottom sum	Total
PPO	204	116	545	865
AlphaZero	245	99	463	807
ChatGPT	186	50	504	740

Table 3: Comparison in average scoring, minimum and maximum of different players.

Player	ChatGPT-5-mini	Human	PPO	AlphaZero
Average	740	1045	865	807
Maximum	888	1271	1195	1134
Minimum	567	733	584	512

2.5 Comparison with human players

In this comparison, an experienced human player – who typically scores 900–1100 points – played the identical game with the same initial rolls as *ChatGPT-5-mini*. The human had no knowledge of future rolls and thus no informational advantage over the LLM model. The only difference between the human and the model was in choosing which dice to hold and the category to fill in the score sheet. Under these conditions, the human scored 958 points, and *ChatGPT-5-mini* scored 808 points.

The largest difference appeared in the middle section (Min/Max), where the human outscored the model by 133 points. The human also performed better in the upper section (269 vs. 149 points), whereas the model outperformed the human in the lower section by 103 points. This pattern suggests that the model emphasized the lower section and pursued category bonuses available there. More surprising was the neglect of the Min/Max categories: in one column the recorded minimum exceeded the maximum (min > max), incurring a penalty that a human would almost certainly avoid.

We applied the same protocol to the *ChatGPT-5-Thinking* model. This run took approximately 4 h and yielded 755 points; with the same initial rolls and the same re-roll algorithm, the human scored 829 points.

Given that only one game was played with each model, the comparison is rather weak but still interesting. Their scores differ by 53 points, which lies within the normal variation observed across games by the same player. It is therefore plausible to conclude that the models are similarly strong, or that the *ChatGPT-5-Thinking* is at least as strong but experienced less favorable rolls. In its explanations after each decision, *ChatGPT-5-Thinking* appeared more human-like and persuasive, demonstrating awareness of both short- and long-term strategy. The following excerpts illustrate typical responses given specific dice values; the first example traces the model's reasoning across all re-rolls.

```
Roll 1 -> [1, 5, 5, 4, 2]
Model decision: {'hold_indices': [1, 2, 4, 5],
'category': 'S', 'column': 'f',
'rationale': 'Hold 1,2,4,5 to chase a straight in
    free column; two rolls to hit a 3 (~31%).
    Better EV than forcing 5s/Max now; skip
    announce.'}

Roll 2 -> [1, 5, 1, 4, 2]
Model decision: {'hold_indices': [1, 2, 4, 5],
'category': 'S', 'column': 'f',
'rationale': 'Chase small straight: keep 1,2,4,5
    and reroll the extra 1 for a 3 (~16.7%). On
    miss we can still score 1s (down/free) or
    another safe option in free.'}

Roll 3 -> [1, 5, 6, 4, 2]
Model decision: {'hold_indices': [],
'category': 'Y', 'column': 'u',
'rationale': 'No strong combo; avoid poor score in
     free or low 1 in down. Taking 0 in Y up
    unlocks the up column and preserves better
    scoring chances elsewhere.'}
```

In the example above, we see that the model tries to score *straight* for which it has reasonable hope of scoring. When *ChatGPT-5-Thinking* does not get the right number, it decides to sacrifice yamb in the second column, which is a common strategy of an experienced human player. It is also aware that putting just one 1 in the first column would significantly affect the points when ones are multiplied with (Max-Min).

The following example presents a turn towards the end of the game, with only a few options to play left. The model calculates that chasing yamb is the best option and re-rolls the die with number four:

```
Roll 1 -> [5, 4, 5, 5, 5]
Model decision: {'hold_indices': [1, 3, 4, 5],
'category': 'Y', 'column': 'f',
'rationale': 'Keep four 5s and reroll the 4 to
    chase Y in free (~30% in two rolls). Don't
    announce--too risky to lock Y now. Scoring S(d
    ) or 1(u) would waste value compared to a
    potential Y.'}
```

3 CONCLUSION

The goal of our research was achieved by comparing the results of different models. ChatGPT reaches the level of a novice human player in playing Yamb. It successfully finishes most games, correctly announces rolls, and uses a certain degree of strategy when filling in the score sheet. Most difficulties occur in the final rolls, especially when only the announcement column is left, and it still fails to announce after the first roll. Such mistakes occurred even though the program, once the first three columns were completed, displayed a message stating that from that point onward every roll must be announced. This issue could be resolved by providing, along with the current score sheet and dice values, all the restrictions and rules of the game at each step, although that would make one game significantly longer and more expensive due to higher number of tokens used.

This solution could also be useful for achieving higher scores in the ones row and in the min and max rows. Namely, the model neglected these three rows quite a bit and sometimes even achieved negative results, which never happens when humans play. This is also where the biggest score difference was created compared to the other algorithms and humans. Perhaps the model did not pay such attention to these rows due to the demanding scoring method and could have fulfilled them better if it had been reminded of their value every time through the entire game rules. In further research, it would be interesting to observe how high scores the ChatGPT-5 thinking model can achieve and whether much better results can be achieved with some additional help or hints to the model.

REFERENCES

[1] Vid Kališnik. 2025. *Spodbujevano učenje igre Jamb.* Bachelor's thesis. University of Ljubljana, Faculty of Computer and Information Science, Ljubljana, Slovenia. https://repozitorij.uni-lj.si/IzpisGradiva.php?lang=slv&id=172654 In Slovene.

[2] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint* (2017). https://arxiv.org/abs/1707.06347

[3] David Silver et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* (2018). https://www.science.org/doi/10.1126/science.aar6404

[4] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2017. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *arXiv preprint* (2017). https://arxiv.org/abs/1712.01815

[5] Wikipedia contributors. n.d.. *Draft: Yamb (game).* https://en.wikipedia.org/wiki/Draft:Yamb_(game) Accessed: 2025-09-08.

Proc. of the 11th Student Computing Research Symposium (SCORES'25), Ljubljana, Slovenia, September 30, 2025

Numeric Computations of Landau's constant

Matija Derganc
matija.derganc@gmail.com
Faculty of Mathematics and Physics
University of Ljubljana
Jadranska cesta 21
1000 Ljubljana, Slovenia

Luka Urbanč
urbancluka3@gmail.com
Faculty of Mathematics and Physics
University of Ljubljana
Jadranska cesta 21
1000 Ljubljana, Slovenia

Abstract

Landau's constant, denoted by λ, arises naturally in complex analysis from the fact that entire nonconstant holomorphic functions are open maps. It is associated with the largest disc that can be contained in the image of any holomorphic function on the unit disc. Despite its theoretical significance, the exact value is unknown. The best known bounds are

$$\frac{1}{2} < \lambda \leq \frac{\Gamma\left(\frac{1}{3}\right)\Gamma\left(\frac{5}{6}\right)}{\Gamma\left(\frac{1}{6}\right)} = 0.54325\ldots$$

Unlike most mathematical constants such as π or e, for which we have efficient computation algorithms that yield millions of digits, no such algorithms have been implemented for Landau's constant until now. We present an implementation of the algorithm proposed by Rettinger in Rust, which allows us to quickly compute Landau's constant. It is organized around modular components that handle rational arithmetic, covering grids, and evaluation methods. We have also implemented a minimal dashboard that allows users to experiment with complex functions and provides a quick overview of the project's theme. Rust was chosen for its strong type system, memory safety, and performance, making it well-suited for numerical computation. The modular approach ensures clarity of flow, ease of debugging, and adaptability for future refinements.

Keywords

Landau's Constant, Rust, Computation, Complex analysis, Algorithms, Euclidean distance transform

1 Introduction

Complex functions that are differentiable are called *holomorphic*. They possess many useful mathematical properties; for example, each holomorphic function is equal to its Taylor series. Another important theorem states that nonconstant holomorphic functions are *open*, which informally means that they map sets with area, such as the unit disc, to other sets with area (they cannot, for example, map a disc into a line, which has no area). A natural question that arises is how large a disc can fit inside the image over a given domain. Does there exist a lower bound on the size of the disc?

The answer is yes, and Landau's constant determines this lower bound. More precisely, Landau's constant λ is defined so that for any radius $r > 0$ and any holomorphic function defined on the disc

$$K_r(z_0) := \{ z \in \mathbb{C} : |z - z_0| < r \}$$

with $f'(z_0) \neq 0$, the image $f(K_r(z_0))$ contains a disc of radius $|f'(z_0)| \cdot r \cdot \lambda$, where the constant $\lambda > 0$ is independent of f [1]. If we restrict ourselves to the unit-disc domain (radius 1) and to *normalised* functions with $f'(0) = 1$, then Landau's constant is the largest number such that we can fit a disc of radius λ into the image of the unit disc, regardless of the function. We also denote by λ_f the radius of the largest disc that can fit inside the image of a specific function f, making Landau's constant the infimum of λ_f as f ranges over all possible normalised functions. This is also how the implemented algorithm works: by checking λ_f for a sufficient number of carefully chosen functions, we can approximate the constant effectively.

2 Methodology

We present the algorithm we used to compute Landau's constant. The project is structured into distinct modules. The first modules cover the tools used to implement the algorithm: dyadic complex numbers suitable for fast and accurate computation, holomorphic functions and their generation for accurate representation of functions, and covering grids used to measure sizes of complex sets. We then present the Euclidean distance transform (EDT) algorithm and the steps taken to improve performance, such as multithreading, as well as a brief summary of the implemented dashboard. Finally, we use these modules to implement the main algorithm and compute the constant effectively.

2.1 Dyadic types

The first step in the implementation was to ensure greater control over numerical accuracy during computation. To achieve this, we implemented dyadic numbers: numbers of the form

$$n \cdot 2^m$$

for integers n and m. By defining our own types, we gained precise control over the program's numerical precision and ensured that we did not lose too much information. The `dyadic.rs` module defines operations such as addition, multiplication, and scaling, ensuring that numerical results are representable without hidden rounding error. This is because types like `f64` have a limited number of bits, which are shared between the exponent, mantissa, and sign. Our implementation addresses this by giving the exponent and mantissa their own allocations. This design choice was crucial for maintaining rigorous bounds. We then represented complex numbers as pairs of dyadics.

2.2 Holomorphic function representation and generation

We implemented the type `ComplexFunction`, which represents a function's Taylor coefficients as well as its bounding sequence, which must be a convergent series. `ComplexFunction` objects were

---

https://doi.org/10.51939/scores25.11

obtained using the Ψ function proposed in [3], which generates a sequence of expansion coefficients from words.

Because the constant is defined as a lower bound taken over all possible functions, we require a systematic way to generate a sufficient number of them. For this purpose, we use words that encode the function's expansion coefficients. The alphabet consists of the digits $1, 2, 3, 4$. We first split a word into multiple shorter words, which represent the different expansion coefficients. For the n-th coefficient, we select the letters in the word that occur at the positions of n in the t-sequence $0, 0, 1, 0, 1, 2, 0, 1, 2, 3, \ldots$. For the word `123412`, this splits into the words `124`, `31`, `2`, matching the positions of the numbers 0, 1, and 2 in the t-sequence.

We then use each of the shorter words to determine a specific coefficient for a function. In our case, this yields three expansion coefficients. For each one, we start with a complex square of a given size. Then, for each letter, we split the square into quarters and recursively move into the sub-square represented by the letter: 1 means NE, 2 NW, 3 SW, 4 SE (the Cartesian quadrants). When we run out of letters, we take the midpoint of the remaining square as the expansion coefficient.

For example, if we start with a 2×2 square spanning $\mathrm{Re}(z) \in [-1, 1]$ and $\mathrm{Im} z \in [-1, 1]$ about the origin, and apply the first shorter word `124`, the first letter moves us to the quadrant $[0, 1] \times [0, i]$. The letter 2 refines this to $[0, \frac{1}{2}] \times [\frac{i}{2}, i]$. The letter 4 then selects $[\frac{1}{4}, \frac{1}{2}] \times [\frac{i}{2}, \frac{3i}{4}]$. Having exhausted the letters, we take the midpoint of this final square as the first coefficient, in this case $\frac{3}{8} + \frac{7}{8}i$.

In this way, we can obtain a large number of different functions by generating words of given lengths, with longer words resulting in more functions [3].

2.3 Covering Grids

After using a word to create a unique holomorphic function, we evaluate the function on the unit disc. The disc is represented as a grid with spacing 2^{-n} for a given natural number n. We then compute the value of the function at each grid point of the disc and store the results as an image represented by a vector of complex dyadics in Rust.

The next step is to turn this image—currently a vector of scattered complex numbers—into a compact grid in which each point is classified as inside or outside the image. These grids are called ϵ-covering grids. The points in the grid are spaced $\delta = \frac{\epsilon}{4}$ apart. A point x in the grid is inside the image if there exists a point z in the image such that the Euclidean distance $|z - x|$ is smaller than ϵ. By choosing ϵ sufficiently small, we obtain an accurate grid representation of the image. For the grid dimensions, we locate the extreme inside points and then add an extra outer layer so that the inside is clearly surrounded by the complement. The grid is stored in a `GridBitmap` structure, which records its height, width, and a bitmap of inside points saved as a vector of `0`s and `1`s indicating whether a point is inside or outside. This structure provides the fastest computation speed and uses less memory than storing dyadic numbers directly.

2.4 Approximation on Grids using EDT

After we have created an ϵ-covering grid G of an image, we can approximate the radius of the largest disc that can fit inside:

$$\max_{z \in G} \min_{x \in G^C} |z - x|.$$

In words, we compute, for each interior point, its distance to the boundary and then take the maximum of those distances.

This yields accurate results, but it must be done efficiently: a naïve loop that, for each interior point, scans all exterior points runs in quadratic time in the grid size. The grid itself scales quadratically with $1/\epsilon$, so this would imply a complexity of $O(n^4)$, which is not suitable for this project. Therefore, we implement the Euclidean distance transform (EDT) [2]. As input, we take a `GridBitmap` and operate directly on its data. As output, the algorithm assigns to each grid point its squared distance to the nearest point in the complement.

The algorithm first processes the rows. For each point, we consider the parabola

$$\phi_i(x) = (x - i)^2 + g(i),$$

where $g(i)$ is a distance prior: it is set to 0 for points in the complement and to ∞ for interior points. For each x in a row we then assign the squared distance to the nearest exterior point,

$$D(x) = \min_{0 \le i \le n-1} \big((x - i)^2 + g(i)\big), \qquad x \in \{0, 1, \ldots, n-1\},$$

which corresponds to the lower envelope of $\{\phi_i\}$.

For example, if a row is $[0, 1, 1, 1, 0]$ (with 1 indicating inside and 0 outside), the EDT result is $[0, 1, 4, 1, 0]$, since we record squared distances. A row with no exterior points, such as $[1, 1, 1, 1, 1]$, is transformed to $[\infty, \infty, \infty, \infty, \infty]$ because $g(i) = \infty$ for all positions in that row.

After processing each row, we run the algorithm on each column. The previously computed values mean interior points no longer have value ∞ but differ according to their horizontal distances to the exterior, which allows the column pass to recover the true minimum distance to the outside for each interior point. Finally, we take the maximum over interior points.

The running time of EDT is linear per row and per column, so it is linear in the total grid size: two sweeps suffice for the whole grid. As a result, we can approximate the largest inscribed disc even on very large and dense grids containing millions of points.

2.5 Multithreading

One of Rust's standout features is its excellent support for parallel computing, which further accelerates our final computation. At the coarse level, independent words are evaluated in separate tasks, requiring no synchronization beyond aggregating the running minimum of λ_f. At the fine level, loops over grid rows and over image samples are safe to parallelize with data-parallel iterators because all shared state is read-only. The current binaries rely on asynchronous orchestration, and the code structure allows a drop-in replacement with a data-parallel library if desired. For this, we use Rust's `tokio` library, which supports asynchronous programming and provides excellent concurrency.

2.6 User Interface and Dashboard

Alongside the core algorithm, we implemented a minimal but functional dashboard that allows users to interactively explore holomorphic functions and visualize results. Through the dashboard, users can enter words that encode functions via the Ψ construction, after which the system automatically generates the corresponding holomorphic function and its coefficients. The image of the unit disc under the function is displayed using domain coloring, with the ϵ-covering grid overlaid to illustrate the approximation. The largest inscribed disc found by the EDT algorithm is highlighted, providing immediate insight into λ_f for the chosen function. Users can also adjust computation settings such as grid resolution, ϵ values, and word lengths to balance accuracy and performance. In addition, the dashboard presents summary metrics, including the computed lower bound l for each function, along with runtime statistics and memory consumption.

The dashboard also includes a short introduction to the theory behind Landau's constant and the implementation proposed in [3], as well as a practical guide to reading domain-coloring plots. The main algorithm can also be run from the dashboard, providing users with information regarding the best possible approximation for a given maximum word length and accuracy. We have also added the option to evaluate a number of random words and display the one with λ_f closest to the constant's precise value, which also allows us to evaluate the program's performance on words of lengths 1000 or more—cases that are not computationally feasible when approximating the constant directly.

3 Results

We now have all of the required implementations for the final algorithm. We start by selecting a natural number n, which determines the accuracy of a quantity l satisfying

$$(1 - 2^{-n})\,\lambda \le l \le \lambda.$$

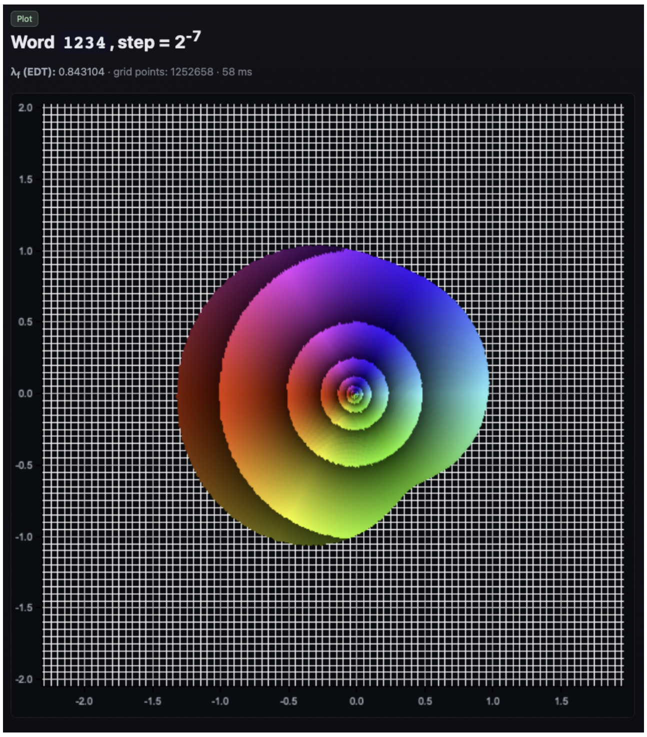
This is achieved by approximating the constant from below, using the methods described above on discs of smaller radius. We generate all words of length t, for a length t sufficient for the algorithm [3]. For each word, we construct the corresponding function as described in the function-generation section. We then evaluate the function on a disc of radius $1 - 2^{-n}$, create an ϵ-covering grid for a theoretically suitable ϵ (as explained in [3]), and approximate the radius of the largest inscribed disc using EDT. The parameter ϵ must be small enough to ensure that the image of the function is accurately represented by the grid. For the domain spacing, we choose $\frac{\epsilon}{4}$ so that grid generation leaves no holes. This yields an approximation $l(\omega)$ for each word ω, and we take

$$l = \min_{\omega}\{\, l(\omega) \,\}.$$

Theoretical guarantees ensure that this value is correct and that the algorithm terminates in finite time.

The experiments were conducted on a MacBook Air M2, and computations were parallelized across available cores using the `tokio` runtime. In each case, we measured:

- the estimated value of λ obtained for a given depth n and word length t,
- the execution time for grid creation, EDT, and function evaluation.



Word `1234`**, step =** 2^{-7}

λ_f **(EDT):** 0.843104 · grid points: 1252658 · 58 ms

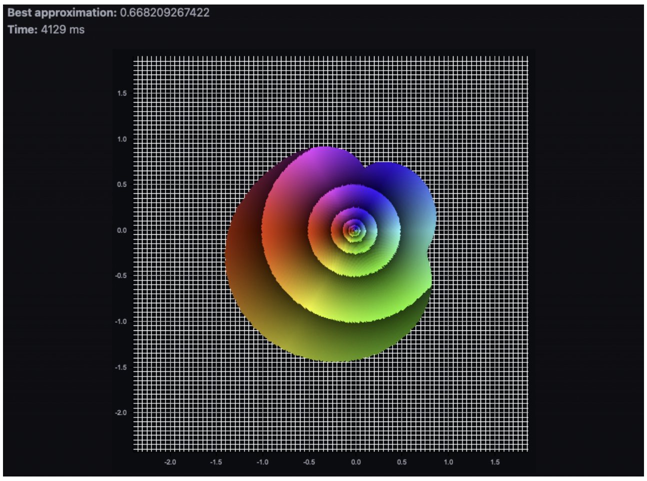
Figure 1: Image of the function generated from the word `1234`**.**

The results of the approximation are presented in Table 1. Note that these results were obtained with step = 1, as runs with larger steps could not be fully executed on our computer (see the discussion in the next section). Also, with increasing word length, we generate more functions and consequently a lower minimum, although all of these are valid approximations lying between $\frac{\lambda}{2}$ and λ.

Word Length	Computed λ
4	0.4981760
5	0.4832330
6	0.4354750

Table 1: Approximations of λ obtained from words of increasing length using step = 1.

Another approach we tried was random word generation, which approximates the constant from above. When evaluating a random batch of words, we specify the word length, the number of words, and the disc accuracy n (so $\epsilon = 2^{-n}$ and the domain spacing is $\frac{\epsilon}{4}$). We then evaluate the corresponding functions on the whole unit disc to obtain an upper bound for λ. For words of length 1000 and accuracy $n = 7$, evaluating 1000 different words yielded a best λ_f of about 0.6682, which is fairly close to the actual value. The elapsed



Best approximation: 0.668209267422

Time: 4129 ms

Figure 2: Best λ_f from 1000 random words of length 1000 with $n = 7$.

time for this computation was 4129 ms, leaving room to test larger random samples.

3.1 Performance

The performance benchmarks show that the EDT-based approximation scales linearly with grid size, enabling efficient computation on grids containing millions of points. For a selected function, the measured times were 100 ms for a grid with 4.8 million points, 471 ms for 20 million points, and 2119 ms for 80 million points. Horner's algorithm, used for evaluating Taylor expansions of holomorphic functions, significantly reduced polynomial-evaluation overhead compared to naïve summation. Parallelization provided valuable speedups when distributing function evaluations across CPU cores.

3.2 Encountered Problems

There were multiple issues encountered in the algorithm. After mapping the disc-domain grid for each function corresponding to specific words, we had to cover the image with an ϵ-covering grid. The parameter ϵ must satisfy several requirements, as described in [3]. These bounds ensure that the grid gives an accurate representation of the image and are tied to bounds on second derivatives of the functions, which can grow large near the boundary. Increasing n (corresponding to a disc radius of $1 - 2^{-n}$) increases these bounds by at least a factor of 8 over the previous step. For $n = 1$, ϵ is approximately 2^{-12}; consequently, the grid spacing $\delta = \frac{\epsilon}{4}$ is about $6 \cdot 10^{-5}$. This implies a grid with roughly 1.1 billion points, already posing a significant computational and storage challenge.

For further steps, grid sizes increase to the point where they cannot be stored in system RAM, exceeding 30 GB. At present, this appears to be a practical boundary we cannot cross with our current equipment, as our personal computers lack the memory and compute capacity required for grid generation and the EDT on such large data structures.

```
Data: Word length $t$, result-accuracy parameter $n$, tolerance $\epsilon$
Result: Approximation $l$ of Landau's constant $\lambda$
initialization;
generate all words $\omega$ of length $t$;
foreach word $\omega$ do
    construct holomorphic function $f_\omega$ using $\Psi(\omega)$;
    evaluate $f_\omega$ on a disc of radius $1 - 2^{-n}$;
    create an $\epsilon$-covering grid $G$ of the image $f_\omega(D)$;
    apply the Euclidean Distance Transform to $G$ to
      compute the largest inscribed disc $l(\omega)$;
end
set $l = \min_\omega\{\, l(\omega) \,\}$;
/* Theoretical guarantee: $(1 - 2^{-n})\,\lambda \le l \le \lambda$ */
```

Algorithm 1: General pipeline for approximating Landau's constant

4 Discussion

The results are promising: we were able to lay the foundation for an accurate and efficient algorithm, given the theoretical guarantees of the framework [3]. Several key implementation choices contributed significantly to performance. First, using EDT to reduce the compute time for finding the largest inscribed radius greatly improved the algorithm's performance. Manual implementations of number types ensured clarity and computational accuracy that would otherwise be difficult to guarantee and could lead to numerical errors, while algorithmic choices such as fast powering reduced time complexity. Although the work presented here is promising, the subject would benefit from additional research aimed at further reducing the time complexity of the proposed algorithm, especially regarding the bounds on ϵ required by the final algorithm.

5 Conclusions

We are the first to have successfully implemented and improved the algorithm presented in [3], as far as possible given our current computational constraints. We achieved a best approximation of 0.4981760, which was obtained relatively quickly, leading us to believe it is possible to improve our work and develop an algorithm that would efficiently compute Landau's constant to arbitrary precision.

Acknowledgments

We would like to express our gratitude to Prof. Andrej Bauer for recommending this theme to us, as well as to Prof. Robert Rettinger for his extensive theoretical work, which formed the basis of our research. We would also like to thank Prof. Matija Pretnar, Gašper Žajdela, and Filip Koprivec for their helpful insight, advice, and support during our work.

References

[1] Edmund Landau. 1929. Über die Blochsche Konstante und zwei verwandte Weltkonstanten. *Mathematische Zeitschrift* 30, 1 (1929), 608–634.

[2] Daniel Huttenlocher Pedro Felzenszwalb. 2004. Distance transforms of sampled functions. *Theory of computing* 8 (2004).

[3] Robert Rettinger. 2012. On computable approximations of Landau's constant. *Logical Methods in Computer Science* 8 (2012).

Proc. of the 11th Student Computing Research Symposium (SCORES'25), Ljubljana, Slovenia, September 30, 2025

Quantifying Player Death Impact in League of Legends

Ruben Ferreira
ruben@ferreira.si
Faculty of Computer and
Information Science,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

Jana Faganeli Pucer
jana.faganelipucer@fri.uni-lj.si
Faculty of Computer and
Information Science,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

ABSTRACT

League of Legends is one of the world's most played and watched e-sports. We study how the death of an individual player affects team win probability. From Diamond-tier matches, we construct minute-level snapshots to train a calibrated win-probability model. We formalize a *death window* based on the state just before death until the first state after the respawn. We attribute the team-level win-probability change to players via a role/time standardized Performance Score weighted by SHAP feature scores. Results across 150,472 death windows show, the team of the deceased player loses on average 2.03% of win probability; probability decreases in 58% of death windows and increases in 42%. The approach enables fast identification of impactful deaths and targeted learning opportunities.

KEYWORDS

esports analytics, win probability, player performance attribution, League of Legends

1 INTRODUCTION

Since its release in 2009, *League of Legends* (LoL) has grown from a niche computer game into one of the world's most played and watched esports. Despite the maturity of its competitive scene and the availability of large amounts of public data, academic research in this domain remains scarce. Compared to traditional sports with long-established analytic infrastructures, esports are relatively young, and their player base—predominantly younger—has only recently become more active in research. As a result, analytic tools that could help players learn and improve more efficiently are still underdeveloped.

Research on LoL has followed several directions. Early work by Jiang et al. [8] used the NICE model, based on tensor factorization of player–champion–game version embeddings for win/loss prediction. Similarly, Costa et al. [6] applied random forests and gradient boosting on professional match data, using only pre-game meta-information. While effective, these approaches rely on static snapshots, which limit generalization. More recent studies focus on dynamic modeling of in-game events. Jalovaara [7] developed a real-time win probability model and introduced the Win Probability Added (WPA) metric to measure how specific actions shift a team's chance of winning. The study also examined the role of individual items. Maymin [12] proposed metrics to distinguish between "smart kills" and "useless deaths,".

Among the many events in LoL, the death of a champion represents one of the most important discrete events. A single death can prevent the player from gaining resources, reward the opponent with gold, and create a temporary numerical advantage (often referred to as a *power play*) that enables the opposing team to secure major objectives. However, not all deaths are equally impactful. An early death in a low-stakes context may be nearly irrelevant, whereas a late-game death can instantly shift a 90% win probability into a likely defeat. To make post-game analysis effective, players need a metric that isolates and quantifies the situational impact of each death on their team's win probability.

The primary goal of this work is therefore to model and measure the effect of each recorded player death on the evolution of their team's win probability. Our aim is to determine whether a death had a major (sometimes even positive) impact on the game outcome or whether it was negligible. Such insights allow players to quickly review their most consequential mistakes, accelerating learning and improving decision making. We emphasize that our method quantifies associations between deaths and subsequent changes in win probability, rather than establishing strict causal effects.

1.1 Description of League of Legends

LoL is a multiplayer online battle arena (MOBA) where two teams of five compete to destroy the opponent's Nexus on the *Summoner's Rift* map. The map features three lanes and a jungle with neutral monsters that provide strategic advantages.

At the start, players select one of over 170 champions with unique abilities. Champions earn gold and experience to level up and buy items, with matches typically moving from a laning phase to team-based play focused on major objectives such as dragons, Baron Nashor, and turrets. Gameplay combines *micro* skills (mechanical execution and ability use) with *macro* strategy (positioning, coordination, and objective control). While the ultimate aim is to destroy the enemy Nexus, victory depends on a sequence of interdependent events—kills, objectives, and map control—that make LoL a rich subject for analytics and predictive modeling.

2 METHODOLOGY

Our methodology consists of three main components. First, we describe the collection and preparation of a large dataset of competitive *LoL* matches suitable for modeling win probability. Second, we outline the design, training and calibration of the predictive model. Finally, we present the procedure for quantifying the impact of individual player deaths by attributing changes in win probability to specific players.

---

https://doi.org/10.51939/scores25.12

2.1 Data

We built a pipeline that combines the official Riot API [15] with publicly available player identifiers from OP.GG [1]. From the European Diamond tier, we collected 37,965 Riot IDs, retrieved their match histories from Jan 1–31, 2025, and obtained 2,944,841 records; after deduplication, 811,625 unique matches remained.

From the API we obtained match metadata (version, duration, outcome) and timelines: cumulative stats at one-minute resolution and key events (kills, deaths, objectives) at one-second resolution. While the timeline exposes champion positions, they are sampled only at one-minute frames. This granularity is too coarse for within-minute pathing or fight geometry, so we omit spatial features and focus on event-level and cumulative signals (e.g., gold, objectives, items). To train the win-probability model (see Secion 2.2), we sample exactly one random minute per match to avoid within-game correlation. Each snapshot includes team aggregates, individual player stats, and role-mirrored differences—sufficient to represent game state for prediction.

We define a *death window* as the interval from the last state before a player dies to the first after respawn, record the team's probability change, and attribute it as in Section 2.3. The corpus spans three consecutive patches; results are specific to this window, though the pipeline generalizes with retraining. In total we analyze over 800,000 matches and around 150,000 death windows, providing robust statistical power.

2.2 Win Probability Model

2.2.1 *Evaluation metrics.*

In *LoL*, matches normally end in either victory or defeat, making win probability estimation a binary classification problem.[1] As the game is zero-sum, it is sufficient to model the probability of victory for one team. True win probabilities cannot be computed exactly due to the multidimensional space of possible game trajectories [5, 17], so we approximate them with data-driven models.

To evaluate performance, we report three metrics. First, *accuracy*, the proportion of correct predictions, reflects classification quality but does not assess probabilistic calibration [14]. Second, the *Brier score* [4, 13] measures the mean squared error between predicted probabilities $\hat{p}_i$ and actual outcomes $y_i \in \{0, 1\}$:

$$\mathrm{BS} = \tfrac{1}{N} \sum_{i=1}^{N} (\hat{p}_i - y_i)^2 .$$

Lower values indicate better discrimination and calibration. Finally, we assess calibration explicitly with the *Expected Calibration Error* (ECE) [2]. Predictions are partitioned into M confidence bins B_m, and ECE is defined as

$$\mathrm{ECE} = \sum_{m=1}^{M} \tfrac{|B_m|}{N} \left|\mathrm{acc}(B_m) - \mathrm{conf}(B_m)\right|,$$

where $|B_m|$ is the number of samples in bin m, N is the total number of samples, and $\mathrm{acc}(B_m)$ and $\mathrm{conf}(B_m)$ are the empirical accuracy and average confidence in bin m.

2.2.2 *Models.*

We benchmarked several families of models, including logistic regression, ensemble methods (Random Forest, Extra Trees, gradient boosting), and neural networks. While specialized deep networks have been proposed for MOBA games [9], prior work shows that for tabular data they rarely outperform gradient boosting methods [3].

Results across methods were remarkably similar, suggesting that predictive performance is driven primarily by the quality of the constructed features rather than by the specific model. In other words, the feature set appears to capture much of the relevant game dynamics, leaving relatively little additional signal to be exploited by more complex models.

Among the tested models, gradient boosting methods performed best, with XGBoost slightly outperforming LightGBM and CatBoost. Despite the modest margins, we selected XGBoost as our primary model, balancing predictive accuracy with robustness and interpretability. Future work could broaden the comparison by including models such as support vector machines, to further test the generality of our findings.

2.2.3 *Calibration.*

Gradient boosting models often produce poorly calibrated probabilities—overconfident at extremes and underconfident in the mid-range [14]. To correct this, we applied *isotonic regression* [18] as a post-hoc calibration step.

Isotonic regression is a non-parametric method that learns a piecewise constant, monotonically increasing function that maps the model's raw predicted probabilities to calibrated ones. This monotonicity constraint ensures that if one instance is assigned a higher score than another, its calibrated probability will not be lower, thereby preserving the ranking of predictions. Isotonic regression makes no assumptions about the shape of the calibration curve, which often yields good results for models like XGBoost that can exhibit complex miscalibration patterns [14].

The calibration function was trained on a held-out validation set to prevent bias and overfitting. All subsequent analyses, including death impact attribution, use calibrated probabilities $\hat{p} = g(\mathrm{XGB}(x))$, ensuring that a predicted probability of, for example, 0.7 corresponds to an empirical win rate of approximately 70%.

2.2.4 *Feature importance.*

Reliable feature attribution is essential both for model transparency and for downstream use in player impact estimation. XGBoost provides built-in feature importance metrics (weight, gain, cover), but these capture only global trends and are not guaranteed to be consistent across model versions [10]. We therefore adopt SHAP (Shapley Additive Explanations) [11] as our primary method. SHAP provides both global and local explanations, and its theoretical consistency ensures that features contributing more to predictions receive higher importance. In our framework, we compute SHAP values from the previously trained and calibrated XGBoost model, and use the resulting global weights in the *Performance Score* metric (see Section 2.3.2) for allocating changes in win probability.

2.3 Modeling the Impact of Player Deaths

Deaths in *LoL* represent moments when a team temporarily loses a player's contribution. Since the absence of one player can shift control of objectives, vision, and map presence, we aim to quantify

---

[1] An exception is early-game *remakes*, which occur only when a player fails to connect to the game. These cases are rare and excluded from this study.

the impact of deaths on a team's win probability and distribute this effect fairly among individual players.

2.3.1 Problem Definition. Let $w(x)$ denote the calibrated win probability model for a game state $x \in \mathcal{X}$. For any ordered pair of states (x_0, x_1) we define the team-level change in win probability as

$$\Delta W_{\text{team}} = w(x_1) - w(x_0). \tag{1}$$

For a given player $p^\dagger$, we define the *death window* as

$$\Omega(p^\dagger) = [x_0, x_1],$$

where x_0 is the state immediately before the player's death, and x_1 is the first state after they return to active play. Our goal is to distribute the observed team-level change ΔW_{team} among the five teammates such that

$$\sum_{i=1}^{5} \Delta W_i = \Delta W_{\text{team}},$$

where ΔW_i is the attributed impact of player i.

2.3.2 Performance Score. We summarize each player's in-game contribution by standardizing their features relative to role and game time. Let $\mathcal{M}$ be the set of features. Using a role/time assignment function $r(i,x)$ and $\tau(x)$ (the game minute for state x), we compute centered and normalized feature values

$$z_{i,m}(x) = \frac{s_{i,m}(x) - \bar{s}_{r(i,x),m}(\tau(x))}{\sigma_{r(i,x),m}(\tau(x))},$$

where $s_{i,m}(x)$ is the value of feature m attributable to player i, and $(\bar{s}, \sigma)$ are historical role- and time-specific statistics.

The *Performance Score* of player i at state x is then

$$\text{PerfScore}_i(x) = \sum_{m \in \mathcal{M}} \omega_m \, z_{i,m}(x), \tag{2}$$

where ω_m are global SHAP-derived feature weights capturing the predictive importance of each feature for win probability. Positive weights correspond to features correlated with higher win probability, and negative weights to those correlated with losses.

2.3.3 Allocating Win-Probability Changes. Given a death window $\Omega(p^\dagger) = [x_0, x_1]$, we first compute the change in performance score for each player:

$$\Delta\text{PerfScore}_i = \text{PerfScore}_i(x_1) - \text{PerfScore}_i(x_0). \tag{3}$$

Let $\delta = \Delta W_{\text{team}} = w(x_1) - w(x_0)$ be the observed change in win probability, and define its sign as $a = \text{sgn}(\delta)$. For each player we decompose their change into parts aligned with (p_i) or opposed to (o_i) the team-level outcome:

$$p_i = \max\{0,\ a \cdot \Delta\text{PerfScore}_i\}, \qquad o_i = \max\{0,\ -a \cdot \Delta\text{PerfScore}_i\}.$$

Aggregate aligned and opposed contributions are

$$M = \sum_{i=1}^{5} p_i, \qquad O = \sum_{i=1}^{5} o_i.$$

To account for cases where some players' performance is inconsistent with the overall outcome (e.g., one player improves while the team's win probability goes down), we allocate a small fraction η of the team-level change to opposed contributions:

$$\eta = \min\left\{\frac{O}{M+O},\ 0.30\right\}.$$

This cap ensures that at most 30% of the observed team-level shift is attributed against the overall trend, preventing over-penalization or over-crediting.

The effective pools for allocation are then

$$N_{\text{tot}} = \eta|\delta|, \qquad P_{\text{tot}} = |\delta| + N_{\text{tot}}.$$

Finally, the attribution for player i is

$$\mathcal{A}(i, x_0, x_1, w) = a \cdot P_{\text{tot}} \cdot \frac{p_i}{M} - a \cdot N_{\text{tot}} \cdot \frac{o_i}{O}, \tag{4}$$

with the second term omitted if $O = 0$.

This rule guarantees two properties: (i) $\sum_{i=1}^{5} \mathcal{A}(i, x_0, x_1, w) = \Delta W_{\text{team}}$; (ii) players whose performance improves in line with the outcome are rewarded, while those whose performance moves against it are penalized.

2.3.4 Limitations. Our attribution relies on (i) additivity of standardized features, (ii) approximate equal sharing of some team-level features (e.g., objectives), and (iii) minute-level granularity of API data. Despite these limitations, the method provides a consistent and interpretable way to link player deaths to changes in win probability, suitable for large-scale quantitative analysis.

3 RESULTS

In this section we present the main empirical findings of our study. We first evaluate the performance and calibration of the win-probability model, and then analyze the impact of player deaths using the attribution framework described earlier.

3.1 Win Probability Model

We benchmarked several families of models and selected XGBoost as the primary predictor (see Section 2.2). After hyperparameter tuning with Bayesian optimization and nested cross-validation, the calibrated XGBoost model achieved an accuracy of 77.8% and an Expected Calibration Error (ECE) of 0.56% (Table 1). Calibration reduced ECE by more than 60% relative to the uncalibrated model, while preserving accuracy and Brier score. The model therefore provides both strong predictive power and reliable probability estimates, which are crucial for downstream attribution of death impacts.

Table 1: Final win-probability results. Results from related work were not obtained on the same dataset.

Model	Accuracy [%]	ECE [%]	Brier
RNN [16]	63.9–83.5	–	–
Calibrated DNN [9]	73.8	0.57	–
XGBoost (ours)	77.7	1.48	0.148
Calibrated XGBoost (ours)	77.8	0.56	0.148

3.2 Impact of Player Deaths

We extracted 150,472 *death windows* (see Section 2.1), restricting to at most one death per player per game to avoid within-game dependence. Results show that on average, a player's death is mildly detrimental: the team's win probability changes by −2.03 percentage points on average, with substantial variance across contexts.

Using the attribution rule from Section 2.3, we ask whether the dying player's *individual* contribution around the time of death (via ΔPerfScore) aligns with the team-level outcome ΔW. As shown in Table 2, in 59.1% of examined windows the player receives a negative attribution (performance worsens relative to pre-death), while in 40.9% the attribution is positive. Conditional on the team outcome, alignment is strong: when the team's win probability *increases*, the player receives positive attribution in 56.6% of cases; when it *decreases*, the player receives negative attribution in 70.4% of cases. This indicates that deaths are not uniformly detrimental—context matters—and our attribution captures that polarity.

Table 2: Attr. sign vs. team outcome in the death window.

	All	**Positive attr.**	**Negative attr.**
All windows	150,472	61,597 (40.9%)	88,875 (59.1%)
$\Delta W > 0$ (42.0%)	63,161	35,739 (56.6%)	27,422 (43.4%)
$\Delta W < 0$ (58.0%)	87,311	25,858 (29.6%)	61,453 (70.4%)

What drives the polarity? Table 3 shows that windows with *positive* attribution coincide with superior objective control (especially towers and late-game epic monsters), while *negative* attribution windows show the opposite pattern. Complementarily, Table 4 summarizes head-to-head lane differentials: deaths with positive attribution tend to preserve or slightly improve gold/CS vs. the mirrored opponent, whereas negative attribution windows deepen deficits.

Table 3: Objective control by attribution sign (averages per death window).

Objective	**Positive contrib.**	**Negative contrib.**
Dragons (won / lost)	0.105 / 0.106	0.096 / 0.122
Barons (won / lost)	0.028 / 0.018	0.009 / 0.029
Heralds (won / lost)	0.181 / 0.126	0.144 / 0.171
Atakhans (won / lost)	0.018 / 0.019	0.015 / 0.026
Towers (won / lost)	0.512 / 0.379	0.244 / 0.577
Net objectives	+0.196	−0.417

Table 4: Head-to-head role differentials (player vs. lane opponent) by attribution sign.

Metric	**Positive contrib.**	**Negative contrib.**
Gold difference	+8.68	−192.59
Kill difference	−0.024	−0.268
CS difference	+0.185	−1.015

Deaths tend to harm teams when the opponents successfully convert the temporary numbers advantage into objectives and lane leads. In contrast, a death can sometimes help if it coincides with, or enables, favorable trades and objective captures. The large variance and consistent polarity alignment motivate individual-level attribution rather than treating all deaths uniformly.

4 CONCLUSION

We presented a method to quantify the impact of player deaths in *League of Legends* by combining calibrated win-probability models with role- and time-standardized performance attribution. Defining *death windows* allowed us to measure team-level probability shifts and distribute them consistently among players. Results show that deaths are not uniformly harmful: while the average effect is negative, many deaths coincide with favorable trades or objective gains, highlighting the strong role of context. The approach offers players an interpretable tool to identify which deaths were most consequential and why. Future work could extend the framework with richer data sources and apply it to other key in-game events beyond deaths.

ACKNOWLEDGMENTS

I would like to thank my family for their support, understanding, and encouragement throughout all the years of my studies. Special thanks goes to my supervisor doc. dr. Jana Faganeli Pucer, who enabled me to work on a topic that interests and motivates me.

REFERENCES

[1] 2024. OP.GG - League of Legends Statistics and Match History.

[2] Jarosław Błasiok and Preetum Nakkiran. 2023. Smooth ECE: Principled Reliability Diagrams via Kernel Smoothing. *arXiv preprint arXiv:2309.12236* (2023).

[3] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. 2024. Deep Neural Networks and Tabular Data: A Survey. *IEEE Transactions on Neural Networks and Learning Systems* 35, 6 (2024), 7499–7519.

[4] Glenn W. Brier. 1950. Verification of forecasts expressed in terms of probability. *Monthly Weather Review* 78, 1 (1950), 1–3.

[5] Euihyeon Choi, Jooyoung Kim, and Wonkyung Lee. 2023. Rethinking Evaluation Metric for Probability Estimation Models Using Esports Data. *arXiv preprint arXiv:2309.06248* (2023).

[6] Lincoln Magalhães Costa, Rafael Gomes Mantovani, Francisco Carlos Monteiro Souza, and Geraldo Xexéo. 2021. Feature Analysis to League of Legends Victory Prediction on the Picks and Bans Phase. In *Proceedings of the IEEE Conference on Games (CoG).*

[7] Perttu Jalovaara. 2024. *Win Probability Estimation for Strategic Decision-Making in Esports.* Master's thesis. Aalto University.

[8] Julie Jiang, Kristina Lerman, and Emilio Ferrara. 2020. Individualized Context-Aware Tensor Factorization for Online Games Predictions. In *Proceedings of the IEEE International Conference on Data Mining Workshops (ICDMW).* 1120–1129.

[9] Dong-Hee Kim, Changwoo Lee, and Ki-Seok Chung. 2020. A Confidence-Calibrated MOBA Game Winner Predictor. *arXiv preprint arXiv:2006.15521* (2020).

[10] Scott Lundberg. 2018. Interpretable Machine Learning with XGBoost.

[11] Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems.* 4765–4774.

[12] Philip Z. Maymin. 2020. Smart kills and worthless deaths: eSports analytics for League of Legends. *Journal of Quantitative Analysis in Sports* 16, 4 (2020), 277–291.

[13] Allan H. Murphy. 1973. A new vector partition of the probability score. *Journal of Applied Meteorology* 12, 4 (1973), 595–600.

[14] Alexandru Niculescu-Mizil and Rich Caruana. 2005. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd International Conference on Machine Learning.* 625–632.

[15] Riot Games. 2025. Riot Games Developer Portal — APIs.

[16] Michael Schubert, Anders Drachen, and Tobias Mahlmann. 2016. Esports analytics through encounter detection. In *Proceedings of the 10th Annual MIT Sloan Sports Analytics Conference.*

[17] David Silver, Aja Huang, Chris J. Maddison, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[18] Bianca Zadrozny and Charles Elkan. 2002. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* 694–699.

Temperature Imputation Using Graph Neural Networks

Jan Jure Stojanovič
js25381@student.uni-lj.si
University of Ljubljana,
Faculty of Computer and Information Science,
Ljubljana, Slovenia

Jana Faganeli Pucer
jana.faganeli@fri.uni-lj.si
University of Ljubljana,
Faculty of Computer and Information Science,
Ljubljana, Slovenia

ABSTRACT

Weather data often contains missing measurements due to sensor failures, communication errors, or maintenance interventions, which complicates forecasting and monitoring. The purpose of this paper is to replace missing temperature measurements by exploiting the spatio-temporal dependencies between meteorological stations. The goal of our paper is to develop a machine learning model using graph neural networks to impute temperature measurements in incomplete data sets. We structured the data into graphs and developed two main approaches. The GNN-GRU model combines graph attention and recurrent GRU units to predict temperature for the next time step and is used when historical measurements at the target station are available. Meanwhile the GAT-Spatial model uses multiple consecutive GAT layers to replace current temperature values without historical data at the target station are not used. Our best results were achieved with the GNN-GRU model with an average absolute error of 0.371 °C, while the GAT-Spatial model achieved an average absolute error of 1.220 °C.

KEYWORDS

Graph neural networks, temperature imputation, machine learning, weather station measurements

1 INTRODUCTION

Ensuring data integrity is increasingly important, yet meteorological datasets are often incomplete due to sensor failures, communication issues, or maintenance, which complicates climate modeling, forecasting, and operational monitoring. High-quality measurements matter directly for planning and operations: solar radiation informs siting of photovoltaic assets, while wind direction and speed trends influence wind farms and transmission lines [12, 17, 19]. Imputation principled replacement of missing values restores dataset completeness and enables reliable analysis and deployment [18, 23].

In this paper we focus on temperature time series and represent historical observations as a spatio–temporal graph: stations are nodes, edges capture geographic proximity with attributes (e.g., distance, elevation difference), and node features combine temperature with cyclic encodings of time and static metadata. This representation lets models aggregate information from relevant neighbors while accounting for daily and yearly periodicity.

1.1 Problem description and objectives

Weather data is frequently incomplete due to previously mentioned reasons, which hampers downstream tasks such as climate modeling, forecasting, and monitoring. High measurement quality is critical in energy and infrastructure planning: solar radiation guides siting of PV installations, while wind direction/speed trends shape wind farms and transmission lines [12, 17, 19].

We focus on temperature time series and represent historical observations as a spatio–temporal graph, aiming to predict the current temperature at a location from nearby stations. We address two practical scenarios with complementary models: (i) when target-station history exists but the current reading is missing, we perform next-step imputation using local history, geographic attributes, and neighbors (GNN–GRU); and (ii) when no history exists at the target (e.g., a new or virtual location), we estimate the current temperature from neighboring stations and geographic context using a purely spatial GAT model. Broadly, our goal is to advance meteorological imputation methods and deliver a practical tool for temperature estimation across Slovenia.

2 RELATED WORK

Imputation of missing values spans from simple statistical methods to advanced machine learning. In meteorological time series, studies compare means/regression with EM–MCMC, which is robust for filling temperatures and precipitation [24], while satellite imagery can further improve tree-based models [9]. Multidimensional ML frameworks such as MMDIF also address randomly missing values [13]. Recently, graph neural networks (GNNs) have gained traction for spatio–temporal data; see [21] for an overview, with examples like GRAST-Frost in meteorology [3] and transferable models from traffic such as DSTGCN and GCN-M [14, 15]. Beyond GNNs, matrix/tensor completion [22], deep learning models (BRITS, SAITS, NHITS) [1, 2, 6], and generative approaches (PriSTI, MiWAE) enable probabilistic imputation with uncertainty [5, 16]. At UL FRI, related work explored CNNs and Random forests.

3 BASELINES FOR TEMPERATURE IMPUTATION

To compare the effectiveness of our graph–neural approach, we include simple baseline methods that are easy to implement and make improvements measurable. We distinguish two baseline families: (i) methods that use historical data from the same station (history available), and (ii) purely spatial methods that use only current measurements from neighboring stations (no history at target).

Baselines with history at the target station

- **Last measurement (persistence):** use the value measured 30 minutes ago (or the last available value).
- **Mean of last three:** arithmetic mean of the three most recent measurements at the station.
- **24-hour lag:** use the value measured exactly 24 hours earlier.

https://doi.org/10.51939/scores25.13

- **Same time on last three days:** mean of the values measured 24, 48, and 72 hours earlier (same time).

Spatial baselines without history at the target

- **Mean of 10 nearest neighbors:** average of current measurements at the 10 geographically closest stations.
- **Mean of 15 nearest neighbors:** average of current measurements at the 15 closest stations.
- **Mean across stations:** average of current measurements across all stations in the dataset.

The first set enables comparison for stations with available historical temperatures, while the second enables comparison at locations without past measurements (purely spatial estimation).

4 ADVANCED METHODS FOR TEMPERATURE IMPUTATION

Graphs are ubiquitous for modeling entities and relations—from molecules to social networks—and are central in modern ML tasks such as protein function classification, link prediction, and recommendation [8]. Building on this, GNNs achieve strong results across domains (social, recommendation, fraud, environmental) and are well suited for spatio–temporal temperature imputation, where the graph encodes spatial dependencies between stations for effective information propagation [25]. Unlike CNNs/RNNs that target Euclidean data, geometric deep learning extends neural operations to non-Euclidean structures via GNNs, enabling learning on irregular, relational graphs [11].

4.1 Graph neural networks (GNN)

An undirected graph is defined as $G = (V, E)$, where V is the set of vertices ($|V| = N$) and $E \subseteq V \times V$ the set of edges. Each vertex $v \in V$ has a feature vector $x_v \in \mathbb{R}^F$ (feature dimension F). The graph structure is represented by an adjacency matrix $A \in \{0,1\}^{N\times N}$, where $A_{uv} = 1$ iff $(u, v) \in E$, otherwise $A_{uv} = 0$.

In a GNN, every node v obtains an embedding $h_v^{(l)} \in \mathbb{R}^d$ at layer l (with $h_v^{(0)} = x_v$). Embeddings are updated iteratively from the previous layer using the node's neighborhood $\mathcal{N}(v) = \{u \in V : (u, v) \in E\}$:

$$h_v^{(l)} = \sigma\Big(W^{(l)} \cdot \mathrm{AGG}\big(\{h_v^{(l-1)}\} \cup \{\, h_u^{(l-1)} : u \in \mathcal{N}(v)\}\big)\Big),$$

where $W^{(l)}$ is a learnable linear map, $\mathrm{AGG}(\cdot)$ is a permutation-invariant aggregator (e.g., mean, sum, max), and σ a nonlinear activation. After one layer, nodes incorporate information from 1-hop neighbors; after two layers, from 2-hop neighbors, and so on.

4.2 Graph Attention Networks (GAT)

Graph Attention Networks (GAT) [20] extend GNNs by learning context-dependent neighbor importances via attention during message passing. For a node v and its neighbor u, the (unnormalized) attention score is

$$e_{vu} = \mathrm{LeakyReLU}\Big(a^\top \big[\, Wh_v \,\|\, Wh_u \,\|\, e_{vu}^{\mathrm{edge}} \,\big]\Big),$$

where W is a learnable linear map, $\|$ denotes concatenation, and e_{vu}^{edge} are optional edge attributes (such as distance, similarity). Scores are normalized with softmax over the neighborhood:

$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k\in\mathcal{N}(v)} \exp(e_{vk})}.$$

The updated node representation is a weighted sum of neighbor messages:

$$h'_v = \sigma\left(\sum_{u\in\mathcal{N}(v)} \alpha_{vu}\, Wh_u\right).$$

Multi-head attention improves expressiveness and stability, especially when edge attributes carry semantic meaning.

4.3 Recurrent Neural Networks (GRU)

To capture temporal dynamics that plain GNNs do not model, we use sequential models, specifically Gated Recurrent Units (GRUs) [4], a lightweight RNN variant with fewer parameters than LSTMs [10]. GRUs update hidden states via

$$z_t = \sigma(W_z x_t + U_z h_{t-1}), \quad r_t = \sigma(W_r x_t + U_r h_{t-1}),$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1})), \quad h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t,$$

where z_t and r_t are update and reset gates, $\odot$ is the Hadamard product, and σ is sigmoid; x_t is the input, h_t the output (hidden state), and W, U are learnable matrices. GRUs effectively model long-term dependencies and periodic patterns (e.g., daily/seasonal cycles), which is crucial for forecasting and imputing missing values in time series.

4.4 GNN-GRU Model - available historical data

Our first model performs temporal imputation of weather data by jointly modeling spatial relations among stations and temporal patterns in their histories. The architecture couples a Graph Attention Network (GAT) for spatial aggregation with a GRU for time–series modeling. Using a sliding window of length $W = 24$ (12 hours at 30-minute cadence; historical steps $t-23, \ldots, t$), each time step is first processed by a GAT layer that ingests dynamic node features (temperature, cyclic time encodings) and static geographic features (latitude, longitude, elevation), i.e., input dimension $(F + S)$. Edges carry attributes (inter-station distance, elevation similarity, geographic proximity), which are incorporated in the attention mechanism. The GAT layer yields a sequence of spatially aggregated embeddings of size H, which a GRU with hidden size G processes per station to capture temporal dependencies; the last hidden state is mapped by a linear layer to a scalar temperature for the next step $(t+1)$ for all stations.

4.5 GAT-Spatial Model - unavailable historical data

Our second model performs *spatial* imputation when no historical data exists at the target station: it predicts the temperature at time t using only simultaneous observations from neighboring stations and geographic context. The architecture stacks three GAT layers with gradual multi-head attention over a 3-hop neighborhood: the first layer uses $h = 8$ heads with `concat=True` (output $H \times h$), the second uses $h/2 = 4$ heads with `concat=True` (output $H \times (h/2)$), and the final uses a single head with `concat=False` (output H). Node inputs combine dynamic features (current temperature and time encodings) with static geographic features (latitude, longitude, elevation), while edge attributes (inter-station distance, elevation difference, geographic similarity) are included in all attention computations. A three-layer MLP then maps $H \to H \to H/2 \to 1$ with ReLU and dropout to produce the scalar temperature at t.

Stacking GAT layers enables information flow across multiple stations—important in sparser graphs—while reducing the number of heads in the middle layer provides a practical trade-off between expressiveness and compute.

5 SUMMARY OF THE DATA SET

The dataset includes measurements from 97 meteorological stations in Slovenia for the period 2016–2024, obtained from the Slovenian Environment Agency. Due to two measurement intervals (10 and 30 minutes), we harmonized all data to a 30-minute time step; from the 10-minute measurements, we took into account those that coincide with the 30-minute grid. Each node (station) is described by static characteristics (latitude, longitude, altitude) and dynamic characteristics (temperature and cyclic time coding for the hour of the day and day of the year using sine-cosine mapping), which enables the model to detect daily and annual periodicity.

We present the spatial relationships between stations using graphs of three topologies: k-NN(10), k-NN(15) (neighborhoods by spatial distance), and a fully connected graph (for analyzing the influence of connection density) as shown in Figure 1. We assign attributes to the connections that capture the distance between stations, the difference in altitude, and geographical similarity. The entire data structure is stored in the form of `TemporalData` (PyTorch Geometric) [7]. This representation enables the training of models that simultaneously exploit the spatial structure of the graph and the temporal evolution of measurements.

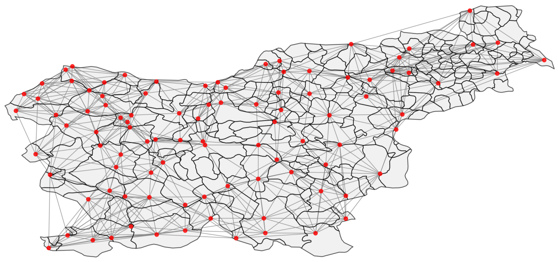


Figure 1: *k-NN* graph linking each node to its 10 nearest neighbors; shows 97 stations across Slovenia and 581 unique edges.

6 RESULTS

The results of the baseline methods show that persistence (last measurement) is relatively solid due to the short, 30-minute step, while the "24 hours back" and "three days at the same time" approaches are significantly weaker (e.g., MAE ≈ 2.406 °C and ≈ 2.677 °C, respectively). For spatial averages without history at the target (average of 10/15 neighbors or all stations), the differences between topologies are small and the errors remain relatively high; using all stations is only slightly better, but still does not achieve the wanted accuracy.

Among the advanced methods, Model 1 (GNN–GRU), which uses a time window and combines graph attention (spatial context) with GRU (temporal patterns), proved to be the most successful on a fully connected graph, it achieves RMSE = 0.510, MAE = 0.371 and MSE = 0.265, with denser connectivity generally benefiting from a broader spatial context as visible in Table 1. Model 2 (GAT–Spatial) operates without target history and estimates temperature from neighboring stations and connection attributes; on k-NN(10) it achieves RMSE = 1.631, MAE = 1.220 and MSE = 3.389. However, its accuracy deteriorates in very dense graphs due to excessive smoothing. Overall, advanced approaches convincingly outperform all baseline methods; when target history is available, GNN–GRU is the clear choice, otherwise GAT–Spatial offers useful estimates from spatial context, albeit with higher error.

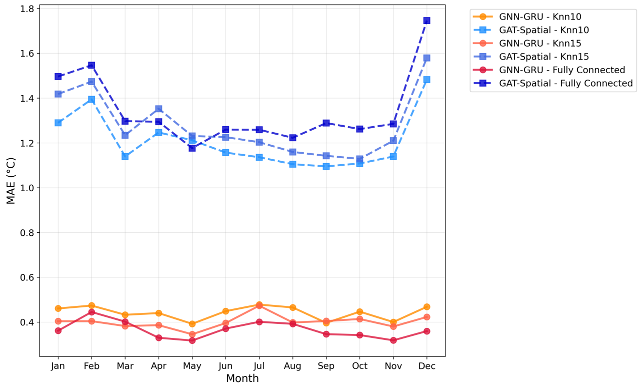


Figure 2: Average monthly MAE for all six models on the test set. Models are color-coded as in the legend (top-right).

Further analysis of graph connectivity showed only a negligible increase in error with larger average neighbor distance for GAT-Spatial, indicating that even remote stations can effectively leverage neighbors and adjust predictions using location, elevation, and temporal context. Because this signal was weak, we compared the strongest variants of both models (GNN-GRU on a fully connected graph and GAT-Spatial on k-NN(10)) and examined geographic patterns of performance; as expected, stations near the country border tended to be less accurate due to limited surrounding context which is shown in Figure 3. A monthly breakdown showed that GNN-GRU maintains high accuracy throughout the year regardless of topology, whereas GAT-Spatial exhibits higher errors in winter, consistent with more volatile conditions displayed in Figure 2. Qualitative inspection over short windows confirmed that GNN-GRU closely tracks temperature dynamics, while GAT-Spatial shows larger deviations, especially around rapid changes.

7 CONCLUSION AND KEY FINDINGS

We studied two graph-based approaches for imputing missing temperatures: (i) *GAT–Spatial*, which estimates the *current* temperature at a target from neighboring stations when no local history exists, and (ii) *GNN–GRU*, which uses spatial context and *target history* to predict the next 30-minute step. Leveraging the graph structure

Model	Graph type	RMSE	σ_{RMSE}	MAE	σ_{MAE}
GNN-GRU	k-NN (10)	0.615	0.204	0.444	0.162
GNN-GRU	k-NN (15)	0.551	0.174	0.401	0.142
GNN-GRU	Fully connected	0.510	0.068	0.371	0.051
GAT-Spatial	k-NN (10)	1.631	0.858	1.220	0.672
GAT-Spatial	k-NN (15)	1.711	0.816	1.300	0.626
GAT-Spatial	Fully connected	1.791	0.773	1.365	0.618

Table 1: Performance of Model 1 (GNN–GRU) and Model 2 (GAT–Spatial) for imputing missing values on the test set. Columns report per-method averages across all stations and their standard deviations.

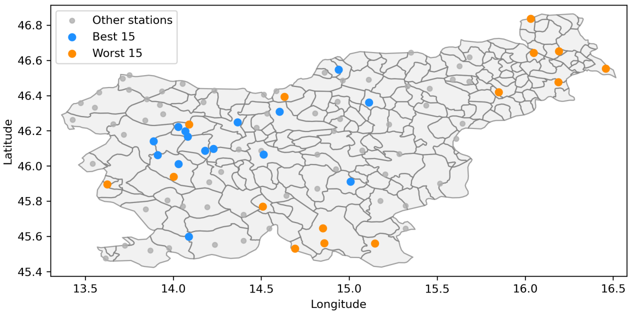


Figure 3: Top 15 (blue) and bottom 15 (orange) stations by per-station average RMSE for Model 1 (GNN–GRU) on the fully connected graph.

is well suited to meteorological data; target history substantially improves accuracy, and both models outperform simple baselines.

Key limitations are the narrow input set (no pressure/humidity/wind), fixed (non-dynamic) graph topologies, a limited set of GNN architectures, and training choices (linear interpolation; per-year normalization with mild leakage risk, albeit likely small due to stable station statistics).

Future directions include richer meteorological features and physical constraints, dynamic/learned connectivity, improved neighborhood selection (including cross-border stations), and hybrid GNN+sequence models (e.g., LSTM/Transformers) for longer temporal dependencies.

Overall, GNNs effectively exploit spatio-temporal structure for meteorological imputation; *GNN–GRU* is particularly strong when target history is available, while *GAT–Spatial* remains useful without it.

REFERENCES

[1] Wenjie Cao et al. 2018. BRITS: Bidirectional Recurrent Imputation for Time Series. In *Advances in Neural Information Processing Systems (NeurIPS)*. https://arxiv.org/abs/1805.10572
[2] César Challu et al. 2023. NHITS: Neural Hierarchical Interpolation for Time Series Forecasting. *Proceedings of AAAI* (2023). https://arxiv.org/abs/2201.12886
[3] Dionysios Chatzopoulos et al. 2022. GRAST-Frost: A Graph Neural Network with Spatio-Temporal Attention for Frost Forecasting. *Sensors* 22, 4 (2022), 1486. https://doi.org/10.3390/s22041486
[4] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).
[5] Shuyuan Deng et al. 2023. PriSTI: A Probabilistic Diffusion Model for Spatio-Temporal Imputation. *arXiv preprint arXiv:2306.00988* (2023). http
[6] Yuqing Du et al. 2023. SAITS: Self-Attention-based Imputation for Time Series. *Proceedings of AAAI* (2023). https://arxiv.org/abs/2206.00864
[7] Matthias Fey, Jinu Sunil, Akihiro Nitta, Rishi Puri, Manan Shah, Blaž Stojanovič, Ramona Bendias, Alexandria Barghi, Vid Kocijan, Zecheng Zhang, Xinwei He, Jan Eric Lenssen, and Jure Leskovec. 2025. PyG 2.0: Scalable Learning on Real World Graphs. arXiv:2507.16991 [cs.LG] https://arxiv.org/abs/2507.16991
[8] William L. Hamilton, Rex Ying, and Jure Leskovec. 2018. Representation Learning on Graphs: Methods and Applications. arXiv:1709.05584 [cs.SI] https://arxiv.org/abs/1709.05584
[9] Kadhim Hayawi, Sakib Shahriar, and Hakim Hacid. 2024. Climate Data Imputation and Quality Improvement Using Satellite Data. *Journal of Data Science and Intelligent Systems* 3, 2 (Jun. 2024), 87–97. https://doi.org/10.47852/bonviewJDSIS42022857
[10] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.
[11] Shengding Hu Zhengyan Zhang Cheng Yang Zhiyuan Liu Lifeng Wang Changcheng Li Maosong Sun Jie Zhou, Ganqu Cui. 2020. Graph neural networks: A review of methods and applications. *AI Open* 1 (2020), 57–81. https://doi.org/10.1016/j.aiopen.2021.01.001
[12] Mirian Jiménez-Torres, Catalina Rus-Casas, Lenin Guillermo Lemus-Zúiga, and Leocadio Hontoria. 2017. The Importance of Accurate Solar Data for Designing Solar Photovoltaic Systems—Case Studies in Spain. *Sustainability* 9, 2 (2017). https://doi.org/10.3390/su9020247
[13] Cong Li, Xupeng Ren, and Guohui Zhao. 2023. Machine-Learning-Based Imputation Method for Filling Missing Values in Ground Meteorological Observation Data. *Algorithms* 16, 9 (2023). https://doi.org/10.3390/a16090422
[14] Zhe Li et al. 2021. Dynamic Spatio-Temporal Graph Convolutional Networks for Traffic Forecasting with Missing Data. *arXiv preprint arXiv:2109.08357* (2021). https://arxiv.org/abs/2109.08357
[15] Han Liu et al. 2022. GCN-M: Graph Convolutional Networks for Complex Missing Traffic Data Imputation and Prediction. *arXiv preprint arXiv:2212.06419* (2022). https://arxiv.org/abs/2212.06419
[16] Pierre-Alexandre Mattei and Jes Frellsen. 2019. MIWAE: Deep Generative Modelling and Imputation of Incomplete Data. arXiv:1812.02633 [stat.ML] https://arxiv.org/abs/1812.02633
[17] K.S.R. Murthy and O.P. Rahi. 2017. A comprehensive review of wind resource assessment. *Renewable and Sustainable Energy Reviews* 72 (2017), 1320–1342. https://doi.org/10.1016/j.rser.2016.10.038
[18] TechRadar Pro. 2023. AI and machine learning projects will fail without good data. https://www.techradar.com/pro/ai-and-machine-learning-projects-will-fail-without-good-data.
[19] Stanley Risch, Rachel Maier, Junsong Du, Noah Pflugradt, Peter Stenzel, Leander Kotzur, and Detlef Stolten. 2022. Impact of Data Quality on Renewable Energy Potential Estimations. arXiv:2204.02334 [math.OC] https://arxiv.org/abs/2204.02334
[20] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations (ICLR)*.
[21] Hao Wang, Jie Wu, Yuxuan Li, et al. 2023. Graph Neural Networks for Time Series Imputation: A Survey. *arXiv preprint arXiv:2307.03759* (2023). https://arxiv.org/abs/2307.03759
[22] Wenlong Yin et al. 2024. Low-rank matrix and tensor completion methods for spatio-temporal missing data imputation. *arXiv preprint arXiv:2405.10995* (2024). https://arxiv.org/abs/2405.10995
[23] John Younger et al. 2019. The use of auxiliary variables to improve the efficiency of complete case analysis with missing patient-reported outcome data. *BMC Medical Research Methodology* 19, 1 (2019), 111. https://pmc.ncbi.nlm.nih.gov/articles/PMC6585083/
[24] Ceylan Yozgatlıgil, Cem İyigün, and İnci Batmaz. 2013. *Comparison of missing value imputation methods in time series: the case of Turkish meteorological data.* Master's thesis. Middle East Technical University. https://hdl.handle.net/11511/42652 Unpublished thesis, available via METU Institutional Repository.
[25] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph neural networks: A review of methods and applications. *AI Open* 1 (2020), 57–81.

Napovedovanje optičnih lastnosti spojin za detekcijo fibril amiloida β s strojnim učenjem

Luka Stražišar
ls43860@student.uni-lj.si
Faculty of Computer and Information Science
Faculty of Mathematics and Physics,
University of Ljubljana
Večna pot 113
Jadranska ulica 19
SI-1000 Ljubljana, Slovenia

asist. razisk. Matic Rogan
Matic.Rogan@fkkt.uni-lj.si
Faculty of Chemistry and Chemical Technology,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

prof. dr. Janez Košmrlj
Janez.Kosmrlj@fkkt.uni-lj.si
Faculty of Chemistry and Chemical Technology,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

asist. razisk. dr. Jerneja Kladnik
Jerneja.Kladnik@fkkt.uni-lj.si
Faculty of Chemistry and Chemical Technology,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

izr. prof. dr. Matevž Pesek
matevz.pesek@fri.uni-lj.si
Faculty of Computer and Information Science,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

POVZETEK

Fluorescenčne molekulske sonde z emisijo v bližnjem infrardečem področju predstavljajo obetavno diagnostično metodo za detekcijo agregatov amiloida β, biomarkerjev, značilnih za Alzheimerjevo bolezen. Ker so priprava kot tudi meritve optičnih lastnosti takih molekul zamudne in drage, bi bilo zato učinkovito, če bi na podlagi strukture molekule znali napovedati, kakšne bi bile optične lastnosti teh spojin v stiku z agregati amiloida β, s čimer bi zmanjšali število nepotrebnih sintez in karakterizacij molekul. Tako smo se osredotočili na napovedovanje treh optičnih lastnosti, in sicer: i) absorpcijskega maksimuma, ii) emisijskega maksimuma ter iii) amplifikacije fluorescence ob vezavi izbrane fluorescenčne molekule na fibrile amiloida β. Pri tem smo uporabili različne numerične predstavitve molekul in metode strojnega učenja. Ugotovili smo, da valovne dolžine absorpcijskih in emisijskih maksimumov negativno korelirajo s kemijsko trdoto. Z analizo pomembnosti značilk pri napovedovanju amplifikacije z naključnim gozdom smo identificirali podstrukturo s povprečno visoko aktivnostjo.

KLJUČNE BESEDE

strojno učenje, amiloid β, fluorescenčne molekulske sonde, optične lastnosti, linearna regresija, naključni gozd

1 UVOD

Za zgodnje odkrivanje Alzheimerjeve bolezni (AB) je v razvoju nova metoda fluorescenčnega slikanja v bližnjem infrardečem področju (angl. near-infrared fluorescence imaging, NIRF). Na tak način želimo s fluorescenčnimi molekulskimi sondami zaznati proteinske agregate amiloida β (Aβ), ki se nahajajo tako v možganih kot v različnih telesnih tekočinah, in veljajo za primarne biomarkerje AB.

Glavni izziv pri tej metodi je sinteza primernih fluorescenčnih molekulskih sond, ki se bodo učinkovito vezale na agregate Aβ in pri tem ustrezno spremenile svoje optične lastnosti ter s tem omogočile detekcijo biomarkerjev in posledično bolezni. Proces sinteze takšnih molekul in testiranja njihovih lastnosti je zamuden in drag, zato bi bilo izjemno koristno, če bi lahko s pomočjo strojnega učenja predhodno napovedali, katere spojine bodo imele želene optične lastnosti, oziroma ugotovili, katere značilke, ki jih lahko razberemo iz same strukture, vplivajo nanje. Tako bi raziskovalcem, ki delujejo na področju sintezne kemije, omogočili, da se osredotočijo na ožji izbor molekul, ki bi jih pripravili v laboratoriju. Osredotočili smo se na tri lastnosti, ki jih želimo napovedovati:

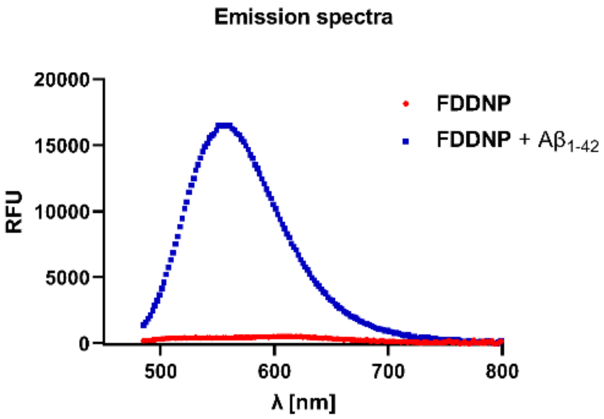
- **valovna dolžina emisijskega maksimuma v vezani obliki ($\lambda_{\mathbf{em}}$)**: valovna dolžina, pri kateri je intenziteta emisije največja – vrh modrega spektra na sliki 1.
- **valovna dolžina absorbcijskega maksimuma v vezani obliki ($\lambda_{\mathbf{ab}}$)**: valovna dolžina, pri kateri je intenziteta absorbcije največja.
- **amplifikacija fluorescence**: razmerje med intenziteto emisije v vezani obliki in intenziteto emisije v prosti obliki; pove, za kolikokrat se intenziteta fluoresecnce sond poveča ob prisotnosti fibril Aβ – razmerje med višino modrega in rdečega spektra na sliki 1.

2 PREGLED PODROČJA

V zadnjem času so bila na področju raziskovanja fluorescenčnih lastnosti molekul s pomočjo strojnega učenja dosežena številna odkritja [2, 3, 5, 8, 9]. V delu [2] avtorji poročajo o napovedovanju valovnih dolžin absorbcijskih in emisijskih maksimumov spojin v različnih topilih z uporabo Dragon 7 in kvantnih deskriptorjev. Kvantne deskriptorje so izračunali z metodo DFT ter pokazali, da obstaja močna korelacija med vrzeljo HOMO–LUMO in valovno dolžino absorbcijskega maksimuma. Podobno so v študiji [8] ugotovili, da je kemijska trdota, definirana kot

$$\eta = \frac{1}{2}(E_{\mathrm{LUMO}} - E_{\mathrm{HOMO}}), \tag{1}$$

https://doi.org/10.51939/scores25.14



Slika 1: Emisijska spektra za molekulo FDDNP. Modre točke ustrezajo meritvam fluorescence sonde FDDNP ob prisotnosti fibril Aβ, rdeče pa spojini v prosti/nevezani obliki sonde FDDNP. RFU (relative fluorescence unit) je enota intenzitete.

močno povezana tako z absorbcijskim kot emisijskim maksimumom spojin. V raziskavi [5] so testirali različice metode podpornih vektorjev (angl. support vector machine, SVM) na naboru 3066 spojin. Razvili so model za napovedovanje absorbcijskega in emisijskega maksimumuma s korenom srednje kvadratne napake (angl. root mean squared error, RMSE) 16,6 za λ_{ab} in 23,6 za λ_{em}.

Poleg tega so v delu [8] preizkusili napovedovanje intenzitete fluorescence spojin, pri čemer so ciljno spremenljivko logaritmirali in uporabili genetski algoritem za izbor značilk. Z uporabo analize pomembnosti značilk v modelu naključnega gozda so izbrali najpomembnejše deskriptorje, ki so jih nato uporabili pri napovedovanju z metodo SVM in metodo delnih najmanjših kvadratov (PLS).

Študija [3] pa se osredotoča na napovedovanje amplifikacije fluorescence v prisotnosti monomernega ali fibrilarnega α-sinukleina, biomarkerja za Parkinsonovo bolezen. Na podatkovni zbirki 320 spojin so primerjali metodo PLS in nevronske mreže. S pomočjo napovedi na še nesintetiziranih spojinah so pripravili ožji izbor potencialno aktivnih molekul, jih nato sintetizirali ter med njimi odkrili pet spojin, primernih za detekcijo α-sinukleina.

Ta dognanja so bila navdih tudi za našo raziskavo, v kateri bomo na lastnih podatkih poskusili napovedati fluorescenčne lastnosti spojin, še posebej ob njihovi vezavi na fibrile Aβ.

3 METODE

Nabor podatkov

V sklopu raziskav Fakultete za kemijo in kemijsko tehnologijo Univerze v Ljubljani (raziskovalna skupina prof. dr. Janeza Košmrlja) je bilo sintetiziranih 69 spojin, namenjenih za dektekcijo fibirl amiloida β. V sodelovanju s Fakulteto za farmacijo Univerze v Ljubljani (izr. prof. dr. Damijan Knez) so bile izmerjene optične lastnosti teh molekul v prosti obliki v pufru HEPES in ob vezavi na fibrile Aβ v pufru HEPES (primer za FDDNP na sliki 1) [6, 7].

Ker so te lastnosti odvisne od eksperimentalnih pogojev kot so koncentracija raztopine in topil, se bomo omejili na podatke, pridobljene samo iz laboratorija prof. dr. Košmrlja. Zaradi tega je podatkov za namene strojnega učenja razmeroma malo, vendar so meritve izvedene v enakih pogojih.

Numerična predstavitev molekul

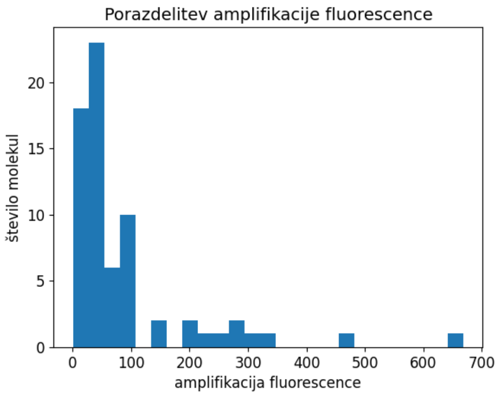
Za izbrane molekule smo z uporabo knjižnice RDKit izračunali 217 različnih molekulskih deskriptorjev (npr. LogP, topološko polarno površino, število vrtljivih vezi). Poleg tega smo izračunali molekulske prstne odtise MACCS, ki z binarnim vektorjem fiksne dolžine opisujejo prisotnost različnih funkcionalnih skupin, in njihovo nadgradnjo ECFP, ki vsaki molekuli pripiše vektor fiksne dolžine (običajno 1024). Za vsak njen atom glede na njegovo okolico s pomočjo zgoščevalne funkcije izračuna mesto bita, ki ga nastavi na 1.

Izračunali smo tudi nekatere kvantne deskriptorje, za katere je bila potrebna predhodna geometrijska optimizacija molekul. Zanjo smo uporabili semiempirično metodo GFN2-xTB [1], ki je hitrejša, vendar nekoliko manj natančna od *Ab initio* metod. Rezultati optimizacije so zajemali podatke o skupni energiji in dipolnem momentu, kemijski trdoti, nabojih, dipolnih momentih posameznih atomov ter energij molekulskih orbital.

Preprečevanje pretiranega prileganja

Različne metode strojnega učenja so primerne za različne vrste in količine podatkov. Ker je v naši nalogi podatkov razmeroma malo, je glavni cilj naloge, da preprečimo pretirano prileganje modelov. Pri napovedovanju λ_{em} in λ_{ab} smo iz virov [8] izvedeli, da sta pogosto korelirana s kemijsko trdoto. Tako smo lahko naredili ožji izbor značilk in uporabili linearno regresijo. Poleg tega smo poskusili še s pristopom z naključnim gozdom, ki se ne zanaša na domensko znanje. Tukaj smo uporabili različne numerične predstavitve molekul, na koncu pa smo iz razlage modela izluščili pomembnejše značilke.

Transformacija ciljne spremenljivke



Slika 2: Porazdelitev amplifikacije fluorescence ob vezavi različnih sond na fibrile Aβ.

Za izboljšanje napovedne zmogljivosti modela ter stabilizacijo variance ciljne spremenljivke smo pri napovedovanju amplifikacije fluorescence uporabili metodo transformacije ciljne spremenljivke. V primeru, ko porazdelitev ciljne spremenljivke odstopa od normalnosti oziroma je močno asimetrična (slika 2), so transformacije koristne za zmanjšanje vpliva ekstremnih vrednosti ter boljšo prilagoditev modela.

V našem primeru smo uporabili dve različni transformaciji ciljne spremenljivke y:

- **kvadratni koren:** $y' = \sqrt{y}$,
- **naravni logaritem:** $y' = \log(y)$.

Po transformaciji je bil model naučen z uporabo metode naključnega gozda. Ker je naključni gozd nelinearni model, transformacije niso nujno potrebne za delovanje modela, vendar lahko pomagajo izboljšati stabilnost in natančnost ocen.

Validacija

Za oceno napovedne sposobnosti modelov smo uporabili metodo *prečnega preverjanja* (angl. cross-validation). Pri tem smo uporabili strategijo **Leave-One-Out** (LOO), kjer model v vsakem koraku učenja izpusti en podatek, ga uporabi za testiranje, in postopek ponovi za vsak vzorec v množici. Tako dobimo vrednosti napovedi za vse spojine, na katerih potem uporabimo metrike za oceno natančnosti napovedi. Ta pristop omogoča maksimalno uporabo razpoložljivih podatkov za učenje in je še posebej primeren pri manjših naborih.

Uporabili smo standardne metrike za oceno kvalitete napovedi kot so MAE, MSE, RMSE in R^2 in korelacijski koeficient r. Poleg tega pa smo izračunali še

$$R^2_{\text{MAE}} = 1 - \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{\sum_{i=1}^{n} |y_i - \tilde{y}|}.$$

4 REZULTATI IN DISKUSIJA

Napovedovanje λ_{em} in λ_{ab}

V tabeli 1 lahko vidimo rezultate napovedovanja λ_{em} in λ_{ab} z linearno regresijo v odvisnosti od kemijske trdote. V tabeli 2 pa so zbrani rezultati napovedovanja z naključnim gozdom in molekulskimi deskriptorji.

Tabela 1: Rezultati napovedovanja emisijskih in absorbcijskih maksimumov z linearno regresijo za parameter kemijska trdota.

Ciljna spremenljivka	MAE	RMSE	R^2	R^2_{MAE}	r
λ_{em}	28,00	39,87	0,77	0,60	−0,88
λ_{ab}	25,57	37,38	0,26	0,28	−0,56

Tabela 2: Rezultati napovedovanja emisijskih in absorbcijskih maksimumov z naključnim gozdom za RDKit deskriptorje.

Ciljna spremenljivka	MAE	RMSE	R^2	R^2_{MAE}
λ_{em}	29,75	41,84	0,74	0,57
λ_{ab}	23,85	33,19	0,41	0,32

Pri napovedovanju z linearno regresijo smo izračunali tudi 95% intervale zaupanja (IZ) za vrednosti trdote v območju, ki ga zajemajo spojine iz baze. Zbrani so v tabeli 3.

Ciljna spremenljivka	95% IZ za min vrednost trdote	95% IZ za povprečno vrednost trdote	95% IZ za max vrednost trdote
λ_{em}	(274, 439)	(434, 593)	(590, 752)
λ_{ab}	(258, 413)	(312, 461)	(362, 514)

Tabela 3: 95% intervali zaupanja za različne vrednosti trdote.

Opazimo lahko, da je kvaliteta napovedi metod zelo podobna. Model linearne regresije pa je lažje razložljiv, saj lahko opazimo linearno odvisnost med kemijsko trdoto in λ_{em} ter med kemijsko trdoto in λ_{ab}. Smerni koeficient regresijske premice je negativen, kar nakazuje na to, da manjše vrednosti trdote ustrezajo višji vrednosti λ_{em}, ki jih želimo doseči.

Rezultati so primerljivi z rezultati v delu [2], kjer poročajo o RMSE 40,7 nm pri napovedovanju λ_{em} v različnih topilih. Še vedno pa so rezultati nekoliko slabši od rezultatov v delu [5], kjer poročajo o RMSE 10,6 nm in 16,6 nm za λ_{ab} in λ_{em}. Vseeno pa je tukaj treba upoštevati, da so λ_{ab} in λ_{em} merili v topilu brez prisotnega Aβ in so zato posledično imeli na voljo veliko večjo bazo spojin.

Z napovedovanjem emisijskih maksimumov omogočimo kemikom, da za novo spojino ocenijo verjetnost, da bo njen emisijski maksimum ležal v želenem infrardečem področju (nad 650 nm). To jim olajša načrtovanje in izbiro spojin, ki so primerne za nadaljnjo sintezo in testiranje.

Napovedovanje amplifikacije fluorescence

Pri napovedovanju amplifikacije fluorescence za razliko od emisije in absorbcije nismo vnaprej vedeli, katere značilke bi bile obetavne za napovedovanje. Zato smo testirali samo metodo z naključnim gozdom, pri kateri smo poskusili različne vrste numeričnih predstavitev. Zaradi asimetrične porazdelitve amplifikacije smo poskusili tudi z različnimi transformacijami ciljne spremenljivke. Rezultati so zbrani v tabeli 4. So nekoliko slabši kot v viru [3], kjer poročajo o R^2 0,63 pri napovedovanju amplifikacije fluorescence ob prisotnosti α-sinukleina, vendar pa je treba upoštevati, da so imeli na voljo več spojin in da je bila porazdelitev amplifikacije drugačna.

Tabela 4: Rezultati napovedovanja amplifikacije fluorescence.

Deskriptorji	Transformacija	MAE	RMSE	R^2	R^2_{MAE}
RDKit + ECFP	Kvadratni koren	48,32	88,47	0,40	0.22
RDKit	Kvadratni koren	49,40	93,61	0,32	0,21
ECFP	Kvadratni koren	51,92	90,31	0,37	0,17
RDKit	Logaritem	54,78	108,35	0,09	0,12
RDKit	Brez	58,81	95,38	0,30	0,06
MACCS	Kvadratni koren	66,42	113,53	0,00	−0,07

Za vse modele smo izračunali tudi pomembnost posameznih značilk. Razlaga najboljšega modela je za najpomembnejšo značilko določila 250. bit ECFP. Ko smo iz tega bita rekonstruirali, kateri podstrukturi pripada, smo dobili podstrukturo z radijem 2 (okolica dveh sosednjih atomov in njihovih vezi). V bazi je spojin s to podstrukturo 10, njihova povprečna amplifikacija pa znaša 267,39 (skupno povprečje vseh 69 spojin znaša 82,04).

5 ZAKLJUČEK

Pri napovedovanju optičnih lastnosti smo uporabili različne numerične predstavitve molekul in dva pristopa, s katerima smo poskusili karseda zmanjšati pretirano prileganje modelov. Pri prvem smo uporabili domensko znanje iz preteklih raziskav in uporabili samo značilko kemijska trdota, pri drugem pa smo iz velikega nabora značilk z razlago naključnega gozda prišli do ožjega nabora uporabnih značilk.

Odkrili smo, da imata valovna dolžina absorbcijskega in emisijskega maksimuma ob vezavi na fibrile Aβ močno negativno stopnjo korelacije s kemijsko trdoto. S prepoznavo te ključne lastnosti smo sodelavcem z UL FKKT omogočili lažje načrtovanje novih molekul, saj lahko ciljajo na molekule z nizkimi vrednostmi trdote. Podobno natančnost napovedi omogoča tudi pristop z naključnim gozdom in RDKit deskriptorji.

Rezultati napovedovanja amplifikacije fluorescence kažejo, da v splošnem te lastnosti še ne znamo natančno napovedati iz strukturnih značilnosti. Kljub temu smo z uporabo razlage naključnega gozda uspeli razpoznati podstrukturo, ki se pojavlja v molekulah s povprečno visokimi vrednostmi amplifikacije. S tem smo sodelavcem z UL FKKT omogočili, da se še bolj poglobijo v spojine s to podstrukturo.

Z uporabo transformacije asimetrično porazdeljene ciljne spremenljivke smo uspeli izboljšati napovedi modelov. Prav tako smo iz razlage modelov, ki so dosegli boljše rezultate od naivnih modelov, izluščili nekaj značilk, ki niso nujno linearno povezane z amplifikacijo, vendar so mogoče zanimive za nadaljno raziskovanje.

V želji po boljših rezultatih napovedovanja amplifikacije fluorescence bomo pri nadaljnih raziskavah poskusili tudi pri tej lastnosti uporabiti kvantne deskriptorje. Prav tako bomo poskusili izboljšati tudi rezultate napovedovanja absorbcijskih in emisijskih maksimumov s prenosom znanja iz napovedovanja lastnosti v prosti obliki. Z najboljšim modelom bomo generirali napovedi lastnosti za spojine iz baze [4] in tako naredili ožji izbor potencialno učinkovitih spojin.

LITERATURA

[1] Christoph Bannwarth, Sebastian Ehlert, and Stefan Grimme. 2019. GFN2-xTB—An accurate and broadly parametrized self-consistent tight-binding quantum chemical method with multipole electrostatics and density-dependent dispersion contributions. *Journal of chemical theory and computation* 15, 3 (2019), 1652–1671.

[2] Chia-Hsiu Chen, Kenichi Tanaka, and Kimito Funatsu. 2018. Random forest approach to QSPR study of fluorescence properties combining quantum chemical descriptors and solvent conditions. *Journal of fluorescence* 28 (2018), 695–706.

[3] DI Inshyn, VB Kovalska, MY Losytskyy, YL Slominskii, OI Tolmachev, and SM Yarmoluk. 2014. Development of a quantitative structure activity relations (QSAR) model to guide the design of fluorescent dyes for detecting amyloid fibrils. *Biotechnic & Histochemistry* 89, 1 (2014), 1–7.

[4] Joonyoung Francis Joung, Minhi Han, Minseok Jeong, and Sungnam Park. 2024. DB for chromophore.

[5] Kapil Dev Mahato and Uday Kumar. 2024. Optimized Machine learning techniques Enable prediction of organic dyes photophysical Properties: Absorption Wavelengths, emission Wavelengths, and quantum yields. *Spectrochimica Acta Part A: Molecular and Biomolecular Spectroscopy* 308 (2024), 123768.

[6] Luka Rejc, Damijan Knez, Gabriela Molina-Aguirre, Alba Espargaró, Jerneja Kladnik, Anže Meden, Lana Blinc, Matic Lozinšek, Ross D Jansen-van Vuuren, Matic Rogan, et al. 2024. Probing Alzheimer's pathology: Exploring the next generation of FDDNP analogues for amyloid β detection. *Biomedicine & Pharmacotherapy* 175 (2024), 116616.

[7] Luka Rejc, Lojze Smid, Vladimir Kepe, Crtomir Podlipnik, Amalija Golobic, Mara Bresjanac, Jorge R Barrio, Andrej Petric, and Janez Kosmrlj. 2017. Design, syntheses, and in vitro evaluation of new fluorine-18 radiolabeled tau-labeling molecular probes. *Journal of Medicinal Chemistry* 60, 21 (2017), 8741–8757.

[8] Siavash Riahi, Abolghasem Beheshti, Mohammad Reza Ganjali, and Parviz Norouzi. 2009. Quantum chemical calculations to reveal the relationship between the chemical structure and the fluorescence characteristics of phenylquinolinylethynes and phenylisoquinolinylethynes derivatives, and to predict their relative fluorescence intensity. *Spectrochimica Acta Part A: Molecular and Biomolecular Spectroscopy* 74, 5 (2009), 1077–1083.

[9] Jie Xu, Qi Xiong, Biao Chen, Luoxin Wang, Li Liu, and Weilin Xu. 2009. Modeling the relative fluorescence intensity ratio of eu (III) complex in different solvents based on QSPR method. *Journal of fluorescence* 19 (2009), 203–209.

Index of Authors

——/ A /——

Anžlovar, Andrej 1

——/ B /——

Beber, Jakob 13
Bizjak, Marko 35, 39
Brodnik, Andrej 13

——/ D /——

Derganc, Matija 47

——/ F /——

Faganeli Pucer, Jana 51, 55
Ferreira, Ruben 51
Fürst, Luka 5

——/ J /——

Jesenko, David 35, 39
Jug, Eva 17

——/ K /——

Kališnik, Vid 43
Kladnik, Jerneja 59
Košmrlj, Janez 59
Kržan, Neža 21

——/ L /——

Loknar, Maksim 27

——/ M /——

Mlakar, Uroš 27

——/ N /——

Novak Domijan, Jan 17

——/ P /——

Pangeršič, Bor 9
Pesek, Matevž 59
Požar, Rok 13

——/ R /——

Rogan, Matic 59

——/ S /——

Sekirnik, Rok 39
Stojanovič, Jan Jure 55
Stojanović, Vanja 9
Stražišar, Luka 59

——/ Š /——

Škrbec, Nejc 5
Šprajc, Žan Tomaž 35

——/ U /——

Urbanč, Luka 47

——/ Ž /——

Žabkar, Anton 43
Žabkar, Jure 21, 43

Proceedings of the
11th Student Computing Research Symposium (SCORES'25)

Uroš Čibej (ed.)
uros.cibej@fri.uni-lj.si
Faculty of Computer and
Information Science,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

Luka Fürst (ed.)
luka.fuerst@fri.uni-lj.si
Faculty of Computer and
Information Science,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

Lovro Šubelj (ed.)
lovro.subelj@fri.uni-lj.si
Faculty of Computer and
Information Science,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

Jure Žabkar (ed.)
jure.zabkar@fri.uni-lj.si
Faculty of Computer and
Information Science,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

ABSTRACT

The 2025 Student Computing Research Symposium (SCORES 2025), organized by the University of Ljubljana, Faculty of Computer and Information Science, in collaboration with the University of Maribor and the University of Primorska, showcases innovative student research in computer science. This year's symposium highlights advancements in fields such as artificial intelligence, machine learning, data science, and algorithmics. The primary goal of SCORES 2025 is to provide a platform for students to present their research, fostering early engagement in academic inquiry. Beyond research presentations, the symposium seeks to create an environment where students from different institutions can meet, exchange ideas, and build lasting connections. It aims to cultivate friendships and future research collaborations among emerging scholars. Additionally, the conference offers an opportunity for students to interact with senior researchers from institutions beyond their own, promoting mentorship and broader academic networking.

KEYWORDS

student conference, computer and information science, artificial intelligence, machine learning, data science, algorithmics

UNIVERSITY
OF LJUBLJANA

FRI
Faculty of Computer
and Information Science