PD.TVZ.HR
POLYTECHNIC & DESIGN Vol. 13, No. 2, 2025.
Mrežna inačica: ISSN 2459-6302
Tiskana inačica: ISSN 1849-1995
DOI: 10.19279/TVZ.PD.2025-13-2-17

GENETSKO PROGRAMIRANJE KAO ALAT ZA RAZVOJ INTERPRETABILNIH STRATEGIJA ODLUČIVANJA

GENETIC PROGRAMMING AS A TOOL FOR THE DEVELOPMENT OF INTERPRETABLE DECISION STRATEGIES

Željko Kovačević[1], Aleksandar Stojanović[1], Maja Dabčević[2]

[1]Zagreb University of Applied Sciences, Vrbik 8, 10000 Zagreb, Croatia
[2]Zagreb University of Applied Sciences, Vrbik 8, 10000 Zagreb, Croatia, Student

SAŽETAK

Genetsko programiranje predstavlja evolucijsku tehniku koja omogućuje automatsko stvaranje programa, pravila ili logičkih stabala pogodnih za donošenje odluka u različitim domenama. Za razliku od neuronskih mreža, koje često postižu visoku točnost, ali nisu dovoljno interpretabilne, genetsko programiranje daje rezultate u obliku eksplicitnih logičkih izraza i uvjetnih struktura koje je moguće razumjeti i analizirati. Glavni doprinos rada je u prikazu kako se ovom metodom mogu razviti razumljive strategije odlučivanja, čime se omogućuje transparentno tumačenje odluka u kontekstu složenih problema. Kao ilustrativni primjer razmatra se klasifikacija zahtjeva za kredit, gdje su ulazni atributi definirani kao zaposlenje, kreditna povijest, dugovanja i dob klijenta. Evolucijom logičkih stabala dobivaju se strategije koje ne samo da učinkovito razlikuju rizične od pouzdanih kandidata, već i zadržavaju jasnoću u objašnjenju kriterija odlučivanja. Time se potvrđuje praktična primjenjivost genetskog programiranja i naglašava njegov potencijal u područjima gdje je jednako važna prediktivna točnost i razumljivost same odluke.

Ključne riječi: *genetsko programiranje, strategije odlučivanja, klasifikacija, interpretabilnost, logička stabla*

ABSTRACT

Genetic programming is an evolutionary technique that enables the automatic creation of programs, rules or logic trees suitable for decision-making in different domains. Unlike neural networks, which often achieve high accuracy but lack interpretability, genetic programming provides results in the form of explicit logical expressions and conditional structures that can be understood and analyzed. The main contribution of the work is in the presentation of how this method can be used to develop understandable decision-making strategies, which enables a transparent interpretation of decisions in the context of complex problems. As an illustrative example, the classification of loan requests is considered, where the input attributes are defined as employment, credit history, debts and age of the client. By evolving logic trees, strategies are obtained that not only effectively distinguish risky from reliable candidates, but also maintain clarity in the explanation of decision criteria. This confirms the practical applicability of genetic programming and emphasizes its potential in areas where predictive accuracy and comprehensibility of the decision itself are equally important.

Keywords: *genetic programming, decision strategies, classification, interpretability, decision trees*

1. UVOD
1. INTRODUCTION

Data-driven decision-making is a central aspect of contemporary computer science and artificial intelligence. Developing methods that provide not only high predictive accuracy but also interpretability of decisions is increasingly important in areas such as finance, medicine, resource planning, and decision support systems. Traditional methods, such as statistical models and machine learning algorithms, have significantly improved classification and prediction capabilities [1, 2]. However, many of these methods suffer from interpretability issues, as they generate models that are difficult to explain to end users or domain experts [3].

Neural networks, particularly deep architectures, have become standard across many domains due to their high efficiency [4, 5]. However, their structure is often described as a "black box", as it is extremely difficult to decompose the network's decisions into simple, human-understandable rules [6]. This lack of interpretability can be problematic in domains where decisions have significant consequences, such as credit scoring, medical diagnostics, or legal systems [7].

Given these challenges, there is growing interest in methods that combine predictive power with interpretability. Genetic Programming (GP), a subfield of evolutionary computation, provides a natural framework for such an approach. Based on the idea of automatically evolving computer programs through biologically inspired operations, GP enables the creation of solutions in the form of logical expressions, decision trees, or other human-readable structures [8]. This makes it possible to develop models that are both effective and easily comprehensible.

In the financial sector, interpretability is critically important, as decisions directly affect individuals and institutions. For example, the classification of loan applications is a typical problem where a balance between accuracy and transparency must be found [9]. When decisions rely on opaque models, users may lose trust in the system, and regulators may demand explanations that "black box" models cannot provide [10]. Genetic Programming, due to its ability to evolve logical rules, is well suited to such tasks, as it produces solutions that can be translated into clear conditions and decision-making strategies.

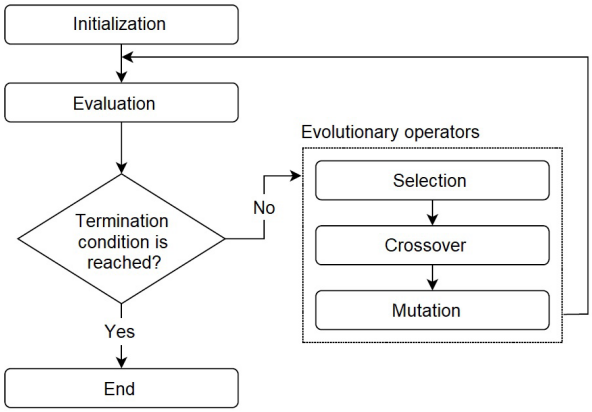
Beyond finance, Genetic Programming is also applied in medicine, where it enables the construction of diagnostic rules that are more comprehensible to physicians than complex deep learning models [11]. It is also used in robotics and planning, where interpretable behaviour policies facilitate the verification and control of complex systems [12]. This broad range of applications confirms that GP is not only theoretically appealing but also a practically valuable tool in domains where decision transparency is essential.

The objective of this paper is to demonstrate how Genetic Programming can be used to develop interpretable decision-making strategies and to illustrate its practical application through a case study of credit application classification. By evolving logical trees, it is possible to generate rules that combine various input attributes and produce decisions that are both accurate and easy to interpret.

In this way, GP stands out as a method that can enhance the acceptability of artificial intelligence systems in environments where decisions must be both accurate and explainable. In comparison with neural networks and other machine learning methods, Genetic Programming has distinct advantages and limitations. Neural networks perform exceptionally well when large numbers of input examples are available and when the primary goal is high predictive accuracy, particularly in nonlinear and high-dimensional problems. However, they require longer training times, are sensitive to the choice of hyperparameters, and generally do not provide explanations for their decisions. In contrast, Genetic Programming naturally constructs models in the form of logical expressions or decision trees, allowing complete interpretability and straightforward explanation of the criteria on which a decision is based. Therefore, deep learning methods are more suitable when high accuracy is the priority and large datasets are available, whereas Genetic Programming is more appropriate in domains where transparency and comprehensibility of decisions are as important as accuracy.

2. GENETSKO PROGRAMIRANJE
2. GENETIC PROGRAMMING

Genetic programming represents an evolutionary method that automatically generates program solutions in the form of expressions, logical, and syntactic structures. It is typically used for program optimization and evolution, in which a population of candidate programs undergoes iterative processes of selection, crossover, and mutation until the algorithm's termination condition is satisfied (Figure 1).



Slika 1 *Opći oblik algoritma genetskog programiranja [8].*

Figure 1 *General form of a genetic programming algorithm [8].*

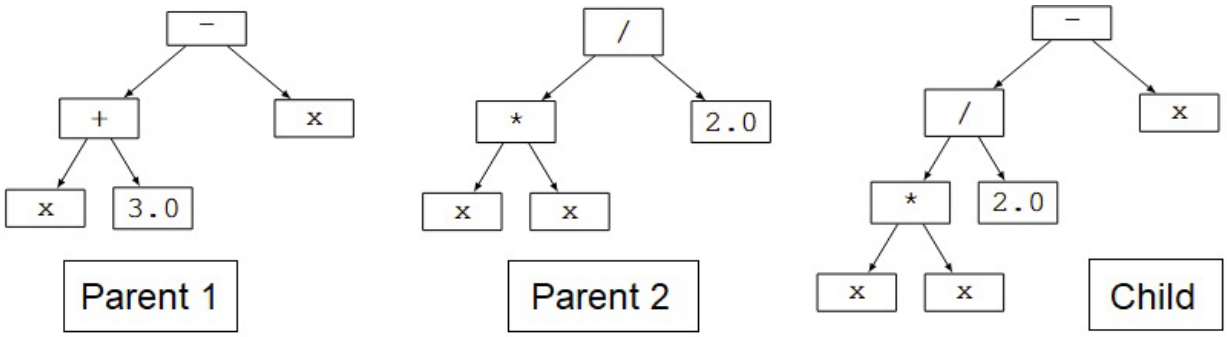
GP begins with initialisation, that is, the creation of a randomly generated initial population of solutions. These solutions are generally of low quality, so throughout the evolutionary process, over multiple generations, the operators of selection, crossover, and mutation are repeatedly applied with the aim of gradually improving the population and approaching the optimal solution.

In the selection process, a chosen mechanism (for example, tournament selection [13]) is used to select higher-quality solutions from the existing population. These then proceed to subsequent stages of evolution, thereby increasing the likelihood that their traits (genes) will be passed on to the next generation. After selection comes crossover (recombination), in which two parent solutions are combined by exchanging genes (e.g., subtrees) to create potentially better offspring.

An example of crossover (recombination) in GP is shown in Figure 2. By randomly replacing subtrees of two parents `(x + 3.0) - x` and `(x * x) / 2.0`, a new solution (child) with the expression `x * x / 2.0 - x is created`. In this case, the new solution was generated by replacing the "+" node of the first parent with the root node "/" of the second parent.

Finally, the mutation operator randomly alters a selected part of the solution (in this case, a subtree). Mutation introduces additional variability into the population and helps the algorithm avoid local optima, thus preventing stagnation in regions of the solution space that do not lead to a global optimum. The probability of mutation is typically set to a relatively low value, as an excessive mutation rate causes overly frequent random changes and slows convergence, turning the evolutionary process into an almost random search.

The evolution of expressions can yield simple functions for data approximation or rules for classification. However, in practice, GP is not limited to optimising mathematical expressions and logical conditions; it may also be used for generating source code. This approach is demonstrated in [14, 15], where GP is employed



Slika 2 *Primjer križanja (rekombinacije) rješenja u genetskom programiranju.*

Figure 2 *An example of crossover (recombination) of solutions in genetic programming.*

as a mechanism for the evolutionary generation of source code for compilers and interpreters of domain-specific languages.

Genetic programming can be further accelerated by combining global evolutionary search with local optimisation techniques. One approach is memetic algorithms, in which, after each crossover or mutation, candidates are further improved through local search within the solution space, reducing the number of generations required for convergence. Additionally, heuristic adaptations of operators may be used, such as selectively focusing mutations on promising substructures or restricting tree size to expedite evaluation. The use of adaptive local search, where the intensity of optimisation is adjusted according to candidate quality, and the reuse of successful subtrees, further reduces the time needed to identify high-quality solutions. These approaches collectively enable more efficient evolution with fewer evaluations and faster improvement of the population.

3. RAZVOJ STRATEGIJE ODLUČIVANJA ZA KREDITNE ZAHTJEVE

3. DEVELOPMENT OF DECISION-MAKING STRATEGY FOR CREDIT APPLICATIONS

Below, we illustrate how genetic programming can be applied to a simplified version of the credit application classification problem. The aim is to evolve a logical tree that, based on a given set of input data, determines whether an application should be accepted or rejected.

Tablica 1 *Parametri za odobrenje kreditnog zahtjeva.*

Table 1 *Parameters for credit application approval.*

Parameter	Description	Possible values
A	Permanent employment	1 – Yes, 0 – No
B	Positive credit history	1 – Yes, 0 – No
C	Current debt	1 – Yes, 0 – No
D	Under 25 years of age	1 – Yes, 0 – No

When approving a credit application, the values of the binary parameters described in Table 1 are taken into consideration. Additionally, to ensure that the developed strategy for approving credit applications aligns with the bank's established business practices, Table 2 contains records of some previously processed credit applications.

For example, client 11 has an ideal profile. He is employed (A=1), has a positive credit history (B=1), no debts (C=0), and is not under 25 years of age (D=0). His credit application is approved (1). The same applies to client 12, who is under 25 years of age, but all other characteristics are positive, so his application is also approved.

In contrast, client 4 has no permanent employment

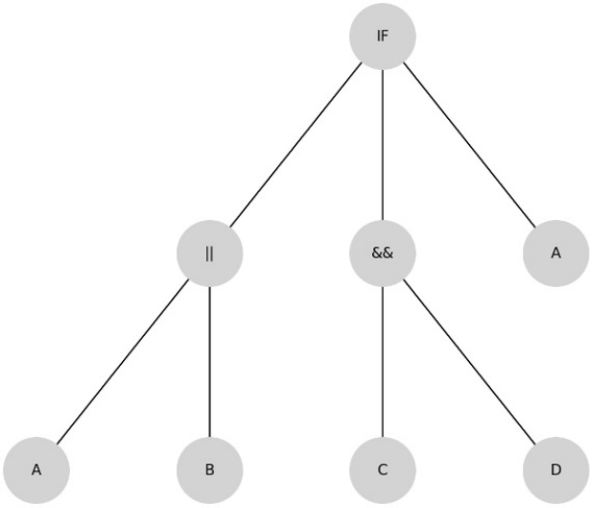
Client	A (Permanent employment)	B (Credit history)	C (Debt)	D (Young)	Approved
1	0	0	0	0	**0**
2	0	0	0	1	**0**
3	0	0	1	0	**0**
4	0	0	1	1	**0**
5	0	1	0	0	**1**
6	0	1	0	1	**1**
7	0	1	1	1	**0**
8	1	0	0	0	**1**
9	1	0	0	1	**1**
10	1	0	1	1	**0**
11	1	1	0	0	**1**
12	1	1	0	1	**1**
13	1	1	1	0	**1**

Tablica 2 *Dosadašnja praksa banke pri odobravanju kreditnog zahtjeva.*

Table 2 *The previous practice of the bank when approving a credit request.*

(A=0), no positive credit history (B=0), has debts (C=1), and is under 25 years of age (D=1). Due to this combination of factors, his credit application is rejected (0). To develop the optimal strategy, each potential solution must be tested against each of the 13 cases listed, with the most effective solution (strategy) being the one that aligns with the bank's established practice most frequently.

To arrive at a high-quality solution (strategy) using GP, it is first necessary to define an appropriate representation for the candidates. In genetic programming, this is most commonly a tree structure (Figure 3). At the top of the tree is the conditional statement IF, whose branches represent the expressions in the THEN and ELSE sections. Within the tree, logical operators ==, !=, && and || are used, while the leaves of the tree are terminals, which may be problem variables (A, B, C and D) or constants (0 and 1).



Slika 3 *Vizualizacija stabla za izraz IF (A || B) THEN (C && D) ELSE A.*

Figure 3 *Visualization of the expression tree for the statement IF (A || B) THEN (C && D) ELSE A.*

During the algorithm's initialization, a random population of 50 individuals (strategies) was created. Each individual is represented by a tree structure similar to that shown in Figure 3. Each strategy is then evaluated separately using examples from Table 2, with accuracy in classifying individual input data examined. The number of correctly classified examples constitutes the quality measure (fitness) of each strategy.

Selection is then performed based on this measure. In this case, tournament selection of size 2 was used: two strategies are randomly chosen from the population, and the tournament winner (the strategy with higher fitness) enters the parent pool for the next generation. This approach increases the likelihood of reproduction for superior solutions while maintaining diversity.

The next step is crossover (recombination). Two selected parents are combined so that, in the first tree, a randomly chosen subtree is replaced by a subtree from the second parent (Figure 2). This process produces offspring inheriting different parts of the structures from both parents.

After crossover, mutation may occur with a probability of p = 0.2. Mutation is performed by replacing a randomly selected subtree with a newly generated random tree.

Elitism was applied in each generation: the best individual from the previous generation is directly transferred to the new population, ensuring the preservation of the highest-quality solution. The process of selection, crossover, mutation, and elitism is repeated until the new population of 50 individuals is filled. This evolutionary cycle continues for a predetermined number of generations (50 in this case) or until a strategy that correctly classifies all examples from the data set is found.

4. REZULTATI
4. RESULTS

In the experimental section, it was established that there are many optimal solutions (strategies) for the given problem. Thus, Example 1 presents one possible strategy obtained in the 21st generation.

Primjer 1 *Prva optimalna strategija za odobrenje zahtjeva za kredit.*

Example 1 *The first optimal strategy for approving a loan application.*

```
(IF ((IF (((1 && B) || (C != D)))
  THEN
    ((1 == D) && (A != 0))
  ELSE
    IF ((1 || 1))
      THEN
        (1 && 1)
      ELSE
```

```
        (1 == D) != C))
  THEN
    (1 == A)
  ELSE
    (B && (0 || B)) || (IF (1)
  THEN
    0
  ELSE
    B || (B != B)))
```

If the logical expression from Example 1 is simplified, its C++ equivalent appears as follows:

```
bool strategija(bool A, bool B, bool C, bool D)
{
    return B || (A && (D || !C));
}
```

The displayed strategy provides accurate results for all clients in Table 2. For independent verification, the strategy can be tested on three randomly selected examples, clients 3, 6, and 11:

- Client 3: A=0, B=0, C=1, D=0 → B || (A && (D || !C)) = 0 || (0 && (0 || 0)) = 0. Correct.
- Client 6: A=0, B=1, C=0, D=1 → 1 || ... = 1. Correct.
- Client 11: A=1, B=1, C=0, D=0 → 1 || ... = 1. Correct.

Experiments also showed that parameter D (under 25 years of age) is not necessarily significant in the decision strategy. For example, the strategy in Example 2 also yields correct results for all 13 clients in Table 2, without considering the client's age.

Primjer 2 *Druga optimalna strategija za odobrenje zahtjeva za kredit.*

Example 2 *The second optimal strategy for approving a loan application.*

```
IF (((0 == C) == (0 == A)))
  THEN
    IF ((A || 0))
      THEN
        (B && C)
      ELSE
        B
  ELSE
    ((0 == 1) || ((IF (A)
  THEN
    D
  ELSE
    (A || 0) == (1 || 1)) || ((1 && A) ||
(C == 0))))
```

The simplified C++ equivalent of Example 2 is as follows:

```
bool strategija(bool A, bool B, bool C) {
    return (A == C) ? B : (A || !C);
}
```

If this strategy is also tested on the same clients 3, 6, and 11, the following results are obtained:

- Client 3: A=0, B=0, C=1, D=0 → (A==C) ? B : (A||!C) = (0==1) ? **0 : (0||0) = 0. Correct.**
- Client 6: A=0, B=1, C=0, D=1 → (A==C) ? B : (A||!C) = (0==0) ? **1 : ... = 1. Correct.**
- Client 11: A=1, B=1, C=0, D=0 → (A==C) ? B : (A||!C) = (1==0) ? **1 : (1||1) = 1. Correct.**

Although both strategies yield accurate results for all 13 clients in Table 2, the question arises whether they will always produce the same outcome in new, previously unevaluated cases. For example, for new client 14 (A=0, B=1, C=1, D=0), the two strategies produce different results:

- S1 → B || (A && (D || !C)) = **1 || … = 1** (Approved credit application)
- S2 → (A==C) ? B : (A||!C) = (0==1) ? 1 : **(0||0) = 0** (Rejected credit application)

The simpler strategy (S2), although more transparent, may overlook certain parameters and thus incorrectly classify unseen examples. Therefore, it is advisable in practice to test on additional data to assess the true robustness and general applicability of the developed strategy for the given problem.

5. ZAKLJUČAK
5. CONCLUSION

In this paper, we have demonstrated how GP can be used to develop decision-making strategies that, in their initial form, often appear highly complex and difficult to interpret. However, GP-generated expressions can be simplified into logical rules that are easily interpretable. Using the example of credit approval, we observed that simplified patterns can be extracted from evolved trees, clearly linking variables such as employment, credit history, and indebtedness with the final decision.

The results indicated that different strategies may describe the given data set equally well, yet their reliability varies when applied to new cases. This highlights the need for a larger amount of input (test) data to ensure that the final strategy achieves maximal precision.

The primary contribution of this work is to show that GP enables the development of models which, although initially formally complex, can be reduced to comprehensible and explainable decision-making rules. Future research would benefit from investigating the behaviour of such approaches on larger and more realistic data sets, developing procedures for automated tree simplification, and applying GP in domains beyond finance, such as medicine or education.

6. REFERENCE

6. REFERENCES

[1.] Hastie, T.; Tibshirani, R.; Friedman, J. The Elements of Statistical Learning, 2nd edition, Springer, 2009. DOI: 10.1007/978-0-387-84858-7. ISBN: 978-0387848570, 978-0387848587, ISSN: 0172-7397 / 2197-568X.

[2.] Mitchell, T. M. Machine Learning, McGraw-Hill Education, 1997. ISBN: 978-0070428072.

[3.] Doshi-Velez, F.; Kim, B. "Towards a Rigorous Science of Interpretable Machine Learning", 2017, https://doi.org/10.48550/arXiv.1702.08608.

[4.] LeCun, Y.; Bengio, Y.; Hinton, G. "Deep learning," Nature 521(7553):436–444, 2015. DOI: 10.1038/nature14539. ISSN: 0028-0836, 1476-4687.

[5.] Goodfellow, I.; Bengio, Y.; Courville, A. Deep Learning, MIT Press, 2016. ISBN: 978-0262035613; eISBN: 978-0262337373.

[6.] Ribeiro, M. T.; Singh, S.; Guestrin, C. "Why Should I Trust You? Explaining the Predictions of Any Classifier," KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, str. 1135–1144. DOI: 10.1145/2939672.2939778.

[7.] Guidotti, R. i dr. "A Survey of Methods for Explaining Black Box Models," ACM Computing Surveys 51(5):93, 2018. DOI: 10.1145/3236009. ISSN: 0360-0300 (print), 1557-7341 (online).

[8.] Koza, J. R. Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, 1992. ISBN: 978-0262111706.

[9.] Oreski, S.; Oreski, G. "Genetic algorithm-based heuristic for feature selection in credit risk assessment," Expert Systems with Applications 41(4):2052–2064, 2014. DOI: 10.1016/j.eswa.2013.09.004.

[10.] European Banking Authority. Guidelines on loan origination and monitoring (EBA/GL/2020/06), 2020. URL: https://www.eba.europa.eu.

[11.] Vladislavleva, E. J.; Smits, G. F.; den Hertog, D. "Order of Nonlinearity as a Complexity Measure for Models Generated by Symbolic Regression via Pareto Genetic Programming," IEEE Transactions on Evolutionary Computation 13(2):333–349, 2009. DOI: 10.1109/TEVC.2008.926486.

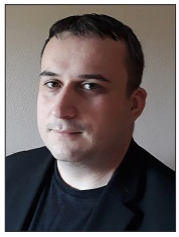
[12.] Poli, R.; Langdon, W. B.; McPhee, N. F. A Field Guide to Genetic Programming, Lulu, 2008. ISBN: 978-1-4092-0073-4.

[13.] Fang, Yongsheng & li, Jun. (2010). A Review of Tournament Selection in Genetic Programming. 5th International Symposium, ISICA 2010, Lecture Notes in Computer Science (LNCS 6382), pp. 181–192. DOI: 10.1007/978-3-642-16493-4_19

[14.] Mernik, M., Ravber, M., Kovačević, Ž., & Črepinšek, M. (2020). From Grammar Inference to Semantic Inference—An Evolutionary Approach. Mathematics, 8(5), 816. MDPI.DOI: 10.3390/math8050816

[15.] Ravber, M., Kovačević, Ž., Črepinšek, M., & Mernik, M. (2021). Inferring Absolutely Non-Circular Attribute Grammars with a Memetic Algorithm. Applied Soft Computing, 100, 106956. Elsevier. DOI: 10.1016/j.asoc.2020.106956

AUTORI · *AUTHORS*



• **Željko Kovačević** - Biografija autora nalazi se na web stranicama https://nastava.tvz.hr/zkovacevic/.

Korespondencija · *Correspondence*
zeljko.kovacevic@tvz.hr

• **Aleksandar Stojanović** - Nepromijenjena biografija nalazi u časopisu P&D, Svezak. 11, Br. 4 iz 2023. godine.

Korespondencija · *Correspondence*
aleksandar.stojanovic@tvz.hr

• **Maja Dabčević** - Studentica je pete godine Informacijske sigurnosti i digitalne forenzike na Tehničkom veleučilištu u Zagrebu. Trenutno je zaposlena na radnom mjestu razvojnog inženjera u poduzeću Innovez d.o.o. U slobodno vrijeme razvija vlastite projekte te sudjeluje na raznim aktivnostima i natjecanjima u sklopu fakulteta.