Explanation in Bio-inspired Computing:
Towards Understanding of AI Systems

Rolf Drechsler*†
0000-0002-9872-1740
*Group of Computer Architecture
University of Bremen
Bremen, Germany

Christina Plump*†
0000-0003-0392-6397
†DFKI GmbH—Cyber-Physical Systems
Bremen, Germany

Bernhard J. Berger‡
0000-0001-6093-9229
‡Computer Engineering Group
Hamburg University of Technology
Hamburg, Germany

***Abstract*—Artificial intelligence methods and applications have recently seen a massive surge, partially caused by the success of neural networks in areas like image classification and LLMs for generating near-perfect natural-language texts. Unnoticed by the public, but highly important for many AI methods to function, bio-inspired optimisation techniques have also seen a rising usage. However, the more techniques are used, the more complex the explainability decreases. Even developers of Neural Networks can seldom state why the Neural Network's results are what it is. The explainability of AI methods, as well as systems in general, is, however, essential for safety, security reasons, and to gain and maintain trust with system users. While research in explainability has therefore gained significant traction with prominent AI methods, such as neural networks, bio-inspired optimisation techniques have seen less research in this regard. The complexity of explainability with these algorithms lies in the use of populations and randomness. We present an approach to track individuals in bio-inspired optimisation techniques, aiming to improve our understanding of the quality of results from such optimisation algorithms. To that end, we introduce a data model, include this model in the standard implementations of these approaches, and provide a visualisation that allows for understanding the relational information of these individuals, yielding more insight into these optimisation techniques and providing a first step toward improved explainability.**

***Index Terms*—explainability, artiftical intelligence, randomised optimisation, individual tracking, inheritance.**

I. Introduction

Artificial Intelligence (AI) has evolved to influence our everyday lives. [1] Many people across all educational levels now utilise GenAI to support them with repetitive tasks or help them gain a deeper understanding of unfamiliar topics. Doctors use AI methods to detect tumours [2] or predict Alzheimer's [3]. Tesla, Mercedes, and other car manufacturers push the boundaries of self-driving cars [4]. Due to widespread use, the term AI no longer has a strict definition. Usually, it refers to a wide variety of data-based, computation-intensive techniques that involve stochastic aspects. Machine-learning techniques, such as neural networks, are one key ingredient, while LLMs for GenAI are another, and bio-inspired optimisation techniques are widely used to tackle complex optimisation tasks. However, while these techniques steadily increase their presence in our lives, it is barely impossible for a *normal human being* to retrace or understand the results produced by these techniques. It may be possible to verify that the image processed by a neural network does indeed contain a tumour, for example. Still, it is unclear why the method classified the image as it did. More importantly, if a human expert holds a different opinion, a chain of causation would help emphasise the AI's decision. LLMs can sometimes change their answer entirely when switching a simple expression in the request prompt—however, why they obtained the answer they provided and which training data was responsible—these aspects remain hidden and unclear to the user.

In bio-inspired optimisation, a user can confirm the result, but usually not its optimality, and especially not why the process yielded that result or which parts of the population were relevant to that solution. For quite some time now, researchers have been developing methods for explaining the results of AI methods. Explainable Artificial Intelligence (XAI) has become a coined term for this research direction. However, its definition varies greatly. Another research direction in that regard is the *step-back* from a dedicated AI-explanation to an explanation of systems, following the question: *Why does the system do what it does?* [5]. An area that has received less research attention in this regard is bio-inspired optimisation. Especially an analysis of stochastic population-based optimisation algorithms, such as Particle Swarm Optimisation (PSO), Evolutionary Algorithms (EA), or Ant Colony Optimisation (ACO), as well as stochastic optimisation algorithms in general, like Simulated Annealing (SA), is very complex and theoretical work is limited to very small populations so far. The challenge with stochastic (population-based) optimisation algorithms is two-fold. First, the stochastic nature across many iterations makes theoretical analysis challenging, or even impossible. So, it is usually unclear whether a result *was luck* or could be easily reproduced. This can be tested statistically, but these results cannot explain why the returned value was considered optimal. Second, the advantage that populations offer to optimisation algorithms is a hurdle

Parts of this work are funded by Deutsche Forschungsgemeinschaft (DFG) GRK 2972 – Project Number: 513623283.

979-8-3315-5309-8/25/$31.00 © 2025 IEEE

when explaining results. Which aspect made the swarm turn to its new position? Was this aspect sensible in relation to the parameter settings? How much influence did the original population have on the returned individual, and how significant was the impact of chance? The population approach loses the information about individuals.

In our work, we take the first step toward understanding stochastic and population-based optimisation by introducing a data model for individuals in these algorithms. This data model not only represents individuals but also models their relationships over generations. Introducing such a data model has several advantages: First, it allows us to analyse an optimisation run not only based on statistical features of its population (the best fitness in the population, average fitness in the population, the worst fitness in the population) but also based on the individuals and their trajectory with respect to fitness, search space and the relationship between one another. We are now capable of analysing the *heritage* of individuals and thus potentially explaining its existence. Second, this individual-based information allows the development of metrics for analysing stochastic influence. Third, having a precise data model enables us to *debug* the algorithm, in the sense that we can analyse the population step-by-step and step through each iteration and adjustment of the individuals. These aspects represent the first contribution to understanding these optimisation algorithms, making them, and therefore, some parts of AI methods, explainable.

II. Background

This section's purpose is to keep this paper self-contained and allow readers to familiarise themselves with our terminology for optimisation algorithms, as various slightly differing notations and terminology are present in the literature. We will first introduce the general concept of an optimisation problem, followed by the general introduction of stochastic optimisation algorithms.

A. Optimisation Problem

In general, optimisation problems are defined as finding the input to a function that minimises or maximises the function value. More formally:

Definition 1: Let $\mathcal{S}$ be a search space and $\mathcal{O}$ be an optimisation space (each of arbitrary dimensions). Further, let $f : \mathcal{S} \to \mathcal{O}$ be a function mapping from the search space to the optimisation space, called optimisation function. Given this triple $(\mathcal{S}, \mathcal{O}, f)$, the optimisation problem is defined as finding $x \in \mathcal{S} : x = \arg\min_{s \in \mathcal{S}} f(s)$.
Please note that maximisation problems can easily be transferred to the upper definition by minimising $-f$. Additionally, search problems (given a value $o \in \mathcal{O}$, find $x \in \mathcal{S} : f(x) = o$) can be transferred by adapting f as follows $f_{search} = |f - o|$ yielding a minimisation problem once again.

The complexity of an optimisation algorithm depends on the properties of the search and optimisation space, as well as the properties of the optimisation function. These properties also determine the choice of algorithm. For example, differentiable optimisation functions can often be solved efficiently using Newton's method or gradient descent. Linear programming methods can solve linear optimisation functions. Non-linear functions that are non-differentiable, however, require algorithms that only evaluate the function (rather than its gradient) and typically utilise either randomness or population strength. Space properties can include the number of dimensions, the types of operations possible on the data (nominal, ordinal, or cardinal data), whether it is discrete or continuous, and the smoothness. Not every optimisation algorithm can handle every type of space property. Often, stochastic optimisation algorithms based solely on function evaluations exhibit greater flexibility, which is why we focus on them in the following subsection.

B. Stochastic Optimisation Algorithms

Stochastic optimisation algorithms differ from other optimisation algorithms in that they include randomness beyond initialisation. The primary reason for including randomness is to escape local optima. The standard deterministic function-evaluation-based optimisation algorithm is the Hill-Climbing algorithm, which continually moves to an improved neighbour until no such neighbour exists. This algorithm quickly finds the next local optimum but can never leave it. In a multi-modal problem (f has several local optima), this is an undesirable behaviour. Therefore, almost all other algorithms in that category introduce randomness. We divide them into algorithms that use only one candidate per iteration (local search algorithms) and those that use populations per iteration (population-based algorithms).

```
function LocalSearch() begin
 individual   = random();
 bestFitness = fitness(individual);

 while not terminate() loop
  for neighbour in neighbours(individual) loop
   if acceptingCondition(fitness(neighbour),
                         bestFitness, randomness) then
    individual   = neighbour;
    best-fitness = fitness(neighbour);
   end if;
   adjustParameters();
 end loop;

 return (individual, fitness);
end LocalSearch;
```

Listing 1: Individual-based Local Search Algorithms

Local Search Algorithms: Listings 1 shows the pseudocode of local search-based optimisation algorithms. First, the algorithm randomly selects a starting individual from the search space and calculates its fitness. The fitness is a value that is based on the optimisation value $f(x)$ for a given individual x. It expresses how well the individual already performs on the respective optimisation problem. Mostly, the fitness is identical to the optimisation value. Sometimes, however, some adjustments are made to the original function to improve the optimisation process. The following code block is executed as long as a stopping criterion for termination does not hold.

There are different strategies for the stopping criterion, such as that the best fitness does not improve or the algorithm has iterated for a predefined number of times. During each iteration, the algorithm explores the local neighborhood of the current solution. If a neighboring solution satisfies the accepting condition the respective algorithm employs, it replaces the current solution. This step can include randomness. The comparator considers the fitness of both individuals but can also take into account other relevant information. Before proceeding to the next iteration, parameters may be adjusted if necessary. Finally, the algorithm returns the best-found solution along with its fitness value.

Basically, all local search algorithms work as the given algorithm. They differ only in the condition that must be satisfied and in the parameters to adjust. The aforementioned HillClimbing algorithm, for example, uses a strict *isBetter* as the accepting condition, uses no randomness, and adjusts no parameters. A standard local search optimisation algorithm that uses randomness is Simulated Annealing, which allows worse individuals to be accepted with a decreasing probability. This includes a randomness aspect, a more sophisticated satisfying condition, and additional parameters. Another algorithm in this category is the threshold accepting (TA) algorithm.

```
function PopulationSearch() begin
 for i in 1 .. size loop
  population[i] = random();
  fitness[i] = fitness(population[i]);
 end loop;

 while not terminate() loop
  next  = adapt(select(population, fitness),
                                  randomness)
  next += select(population, fitness)

  for i in 1 .. size loop
   population[i] = next[i]
   fitness[i]    = fitness(population[i]);
  end loop;
 end loop;

 return (population, fitness);
end PopulationSearch;
```

Listing 2: Population-based Search Algorithms

Population-based Algorithms: Listings 2 shows the pseudocode of population-based optimisation algorithms. As in Listings 1, the initial population consists of random starting individuals. The algorithm then iteratively refines the population until a termination criterion is met. In each iteration, a new set of candidates is created by adapting the existing population and copying a subset of it. The pseudocode uses two operators that need explanation. The `select` operator selects a subset of the population, and the literature specifies different selection operators. In the first usage, the result is passed to the `adapt` operator. Depending on the concrete algorithm, the operator combines multiple individuals, mutates existing individuals, or adapts them based on information from other individuals. The population is then updated with this new set, and the fitness values of its members are evaluated. The algorithm returns the final population along with their associated fitness scores.

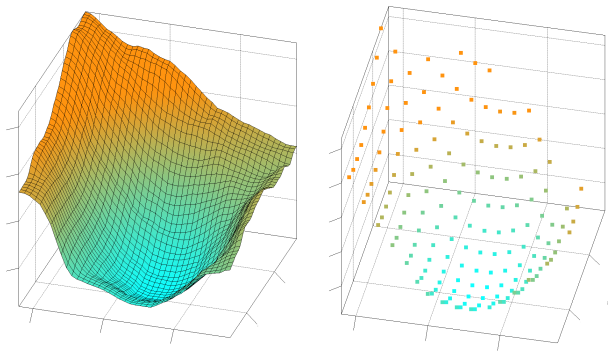

Fig. 1. Classical visualisation techniques for analysing the progression of individuals across the search space or the optimisation space

A popular example of a population-based algorithm is Particle Swarm Optimisation. The initial swarm is randomly generated in the beginning. After fitness evaluation, each individual's position is updated relative to both the common and personal optima. The distribution of influence between the common and the personal optimum is influenced by a random factor, introducing randomness. The select operator, in this case, chooses all individuals. Another very prominent group of algorithms in this category is the group of evolutionary algorithms. This group includes evolutionary strategies, genetic algorithms, genetic programming, and evolutionary programming. In these algorithms, the adaptation step includes recombination operators that combine two parents to produce two children and mutation operators that introduce a random component into individuals. The select operator has many variants and is chosen based on the properties of the optimisation function.

Especially in PSO and EA, it is difficult to analyse an individual's path from beginning to end and the influences it was subjected to. In general, population-based algorithms are evaluated based on statistics for each generation's population rather than the development of individuals. Another potential analysis uses the visualisation of individuals' positions throughout the search space or on the fitness landscape (see, for example, Figure 1). To finally allow the tracking of individuals throughout the history of an optimisation algorithm, we introduce our data model in the next section.

III. Methodology

As already stated in the last section, evaluation and analysis of population-based algorithms are usually based on statistical information about the entire population of each generation. For example, the evolution of the population's best (minimal) value is investigated. Additionally, the diversity of the population attracts much interest, i.e., whether the individuals in the population are spread out over the search space or concentrated in one area. However, the heritance of a single individual is not analysed, at least not in populations of arbitrary size. To work towards our goal of explaining the results of stochastic optimisation-based algorithms, we therefore want to track the

trajectory of each individual: How it influences other individuals, how it results from other individuals, and how many development steps include stochastic elements. To that end, we developed a data model for individuals that captures this information and can be included in standard implementations of the stochastic algorithms mentioned in Section II.

Figure 2 shows an excerpt of the Ecore-based data model for stochastic optimisation approaches that focuses on describing the execution of and tracking the history of individuals in population-based stochastic optimisation problems. Ecore is a UML-like modelling language and can be used to create metamodels and models, and the graphical representation is similar to UML class diagrams. The model stores the individuals of every iteration. An individual consists of the search-space representation (the input space) and an optimisation-space representation (the output space). We assume that the domain problem is also modelled using Ecore. Thus, the spaces are of type `EClass`, and instances are of type `EObject`. While the individuals store information on the search- and optimisation-space values, the relationships store information on the evolution and the reasons for the evolution. The identity relation, for instance, shows that an individual is directly copied from one of the previous versions, while the mutation and recombination relation store information about the adaptation introduced in Listing 2. The relationship holds additional information, not shown in the depiction, such as mutation rates or the recombination form.

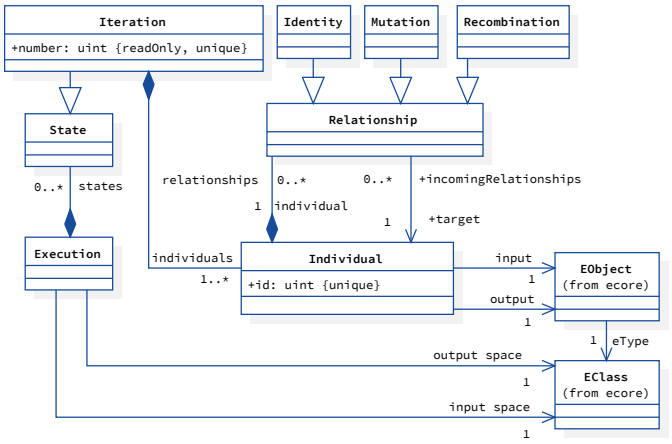

Fig. 2. Unified data model for tracking the history of individuals in population-based stochastic optimisation algorithms.

A. Relationships between individuals

A great advantage of most bio-inspired algorithms is their usage of populations - following the predicate of the sentiment: *There's strength in masses*. They explicitly differ from executing individual-based algorithms, like Simulated Annealing, repeatedly or with several starting points in parallel, in one key factor: They let their individuals interact. They can therefore exchange their knowledge of the search space with one another. However, when only analysing populations over generations as a whole, this particular information on the exchange is lost. By expanding our data model beyond search space and optimisation space information to include knowledge about the relations between individuals, we enable a more detailed analysis and therefore understanding of the optimisation process. For example, in a PSO algorithm, where a global optimum influences every individual, a respective relationship type is stored between the individual that reached this global optimum and the current individual. In an evolutionary algorithm, a form of heritance can be tracked by stating parental (recombination) relationships between individuals. In the end, for each individual in the final population, we are capable to state which individuals influenced it along the way and to what degree (which type of relationship). This will yield essential insights into why the population progressed to which part of the search space and why it developed the way it did.

B. Tracking stochastic influence through metrics

Stochastic optimisation algorithms have the obvious drawback that their results are not deterministic. Hence, when the task is not about finding a single solution but providing an optimisation algorithm that generally yields reliably reasonable solutions, algorithm candidates are run several times to describe the algorithm's quality using statistical metrics. After a single run, it is entirely unclear whether the resulting optimum is highly influenced by chance or whether the result is influenced only by small portions of chance. However, this degree of dependence could be an indicator of reliability. It is, therefore, of great interest to understand when, where, and to what extent randomness influences the algorithm.

After developing the above-introduced data model, it is now possible to develop metrics to track the influence of randomness. For each relationship, the influence of randomness varies. For example, if an individual has a relationship of type identity, then this iteration step (generation) has no stochastic influence. A recombination relationship introduces a small amount of randomness, as the selector randomly chooses a subset of individuals for recombination. Another aspect is the choice of distribution in the recombination. For example, a line crossover (a recombination operator from evolutionary algorithms) computes two weighted sums from the two given parents: $c_1 = u \cdot p_1 + (1-u) \cdot p_2$ and $c_2 = u \cdot p_2 + (1-u) \cdot p_1$. As long as u is randomly chosen between 0 and 1, the resulting values lie somewhere in between the parents. Randomness determines the exact location, but the possible locations can be computed very efficiently. A mutation relationship, on the other hand, introduces a high amount of randomness: Again, the chance of being picked up and the randomness in the actual mutation process. The mutation operation potentially introduces completely new information. For example, a single-bit flip mutator in evolutionary algorithms randomly flips an arbitrary bit, potentially moving the individual to an entirely different area of the search space. Having an understanding of the degree of introduced randomness by the operators, we are now capable of tracking the degree of introduced randomness on individuals throughout the entire population

and every generation, enabling statements like: The optimal individual has a degree of 50% randomness in its inheritance graph. This information can now be used for analysing the algorithm's behaviour but also to enable or disable trust in the result.

C. Debugging Optimisation Algorithms

The introduced data model, in combination with a unified meta-algorithm for optimisation algorithms — similar to the shown pseudo-algorithms — allows further steps towards understandable and explainable optimisation algorithms. The previous advantages enable the preservation of existing information or the derivation of new information. The unified data model and a unified pseudo-algorithm format, which illustrates how specific algorithms apply operators (selection and adaptation) to the data model, enable the implementation of a generic debugging interface on the abstract algorithmic level using standard techniques. In the presented case, we are using an implementation of Microsoft's Debug Adapter Protocol for EMF-based models [6].

This integration allows antemortem inspection of the decisions and workings of stochastic population-based optimisation algorithms. These debug capabilities are especially useful when explaining and teaching algorithms, as the decisions and progress of the algorithm can be directly inspected. Furthermore, in combination with the aforementioned metrics and visualisations, it is helpful to directly see and understand the influence of different operator implementations on the algorithm's behaviour.

IV. Proof of Concept

This section describes a proof-of-concept implementation based on the open-source optimisation framework EvoAl [7]–[9]. It presents screenshots of visualisations generated from the data model.

A. Adapted optimisation algorithm operators

The necessary changes only affect certain areas and, in many cases, can be made at the framework level, allowing them to be carried out without adapting the specific operators. First of all, the main loops of optimisation algorithms must utilise the concept of individuals. Listings 3 shows an adapted version of the population-based search algorithm.[1] Instead of using separate vectors for search and optimisation space values, the adapted version directly constructs the model containing several iterations, each containing a list of individuals, which store search and optimisation space information (cf. Figure 2). The key differences are the use of a modified `adapt'` and a `copy` operator.

Listing 4 shows the copy operator, which copies an individual from the previous version and creates the corresponding Identity relationship.

The modified adapt operator, shown in Listing 5, uses the original adapt operator. It first creates a copy of the individual,

[1]We thank Maximilian Piesbergen for implementing and providing the EvoAl visualisation additions.

```
function AdaptedPopulationSearch() begin
 model = Execution();

 initial = model.states[0] = Iteration();
 for counter in 1 .. size loop
  ind = initial.indivduals[counter] = Individual();
  ind.input = random();
  ind.output = fitness(ind.input);
 end loop;

 while not terminate() loop
  previous = model.states.last
  model.states += iteration = Iteration();
  iteration.indivduals =
      adapt'(select(previous.indivduals),
             randomness)
  iteration.indivduals +=
      copy(select(previous.indivduals))

  for ind in iteration.indivduals loop
   ind.output = fitness(ind);
  end loop;
 end loop;

 return model;
end AdaptedPopulationSearch;
```

Listing 3: Adapted population-based Search Algorithms

```
function CopyOperator(ind) begin
 copy = Individual();
 copy.input = ind.input;

 copy.relationships += Identity(copy, ind);
end CopyOperator;
```

Listing 4: Used copy operator

then executes the adaptation of the copy. Lastly, it stores the relationship by creating a `Mutation`.

```
function AdaptOperator(ind, randomness) begin
 copy = Individual();
 copy.input = ind.input;
 adapt(copy.input, randomness);

 copy.relationships += Mutation(copy, ind);
end AdaptOperator;
```

Listing 5: Modified adapt operator

The necessary changes for an adapted recombination operator are similarly lightweight. Traditionally, two search space representations are selected during recombination and passed to the recombination operator, which task is to calculate two new search space representations by recombining the input values. The adaptation creates two new individuals based on this output and stores the `Recombination` relationship.

B. Improving visualisation through individual tracking

The aforementioned inclusion of our proposed data model into function evaluation-based optimisation algorithms now enables highly improved visualisations compared to the one in Figure 1. For example, a simulated-annealing algorithm can now be visualised as shown in Figure 3. It is now visible how the algorithm checks all neighbours (see the circle around the current individual), and chooses either one with an improved fitness, or if all are worse, a worse one with a decreasing

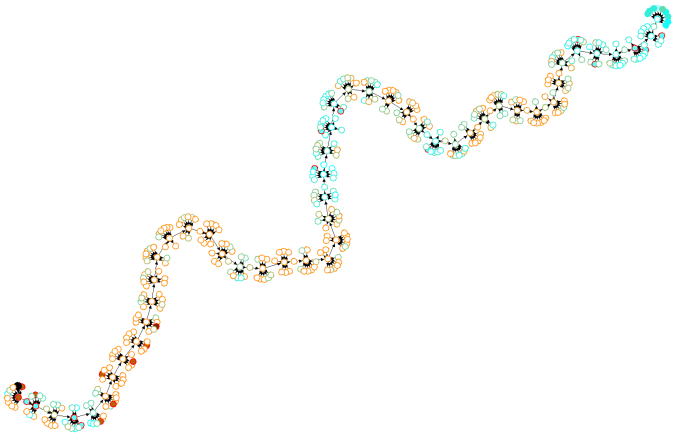

Fig. 3. A visualisation for a simulated annealing algorithm run with individual tracking.

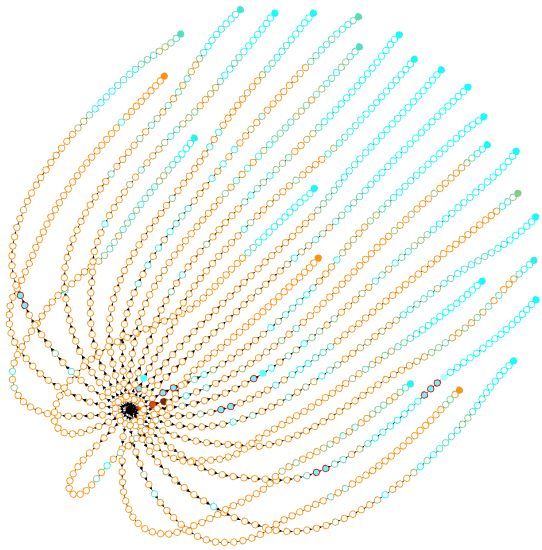

Fig. 4. A visualisation tracking individuals for a PSO run without global optimum usage.

probability for the next iteration. The colour indicates how fitness decreases between iterations before it finally increases again, and it also shows how the neighbours are behaving before the next best candidate is chosen. Another option is the embedding of this chain in a search space representation (which can only be done in up to two dimensions) to visualise the progressing optimisation embedded in the search space. It is a well-known challenge in optimisation to visualise high-dimensional search spaces. It can usually only be done via correlation plots. The technique shown in Figure 3, however, can be used with a semantic meaning to the position of the circles (embedded in a search space) and without, which enables visualising the behaviour of individuals even in a high-dimensional setting and additionally showing their various relationships to one another.

In Figure 4, we show the visualisation for a PSO (without the influence of a global optimum). Again, we can now see the original swarm and the progression of each individual. We can also see how the improved fitness of single individuals slowly improves the fitness of the others until the entire swarm has an improved fitness.

V. Conclusion and Outlook

The growth in the use of AI methods and digital systems has increased the need to understand and therefore explain their behaviour. For users, it is inherently important to have a good understanding of the system's behaviour to enable trust in these systems. While many AI methods have seen increased research on explainability, stochastic optimisation techniques have received less attention in this regard. We therefore propose an approach that allows tracking the behaviour of individuals throughout evolving populations. More specifically, it allows tracking the interactions between individuals and the influence of randomness, yielding more insight into the black box that is stochastic optimisation. When applying our approach, we can determine the degree to which a result is influenced by chance, providing an approximation of its reliability. Additionally, we can identify essential interactions between individuals by observing the respective fitness spikes. Our approach also allows us to visualise the behaviour of individuals in higher-dimensional settings, because it is possible to focus on the relationship (which is independent of the dimension) and abstract from the search-space position (which is unplotable from $n = 3$ onward).

References

[1] O. I. Iglesias R. and C. G. Quintero M., "Generative ai: The key for everyday problems. a comparison proposal for new users," in *2023 IEEE Colombian Caribbean Conference (C3)*. IEEE, Nov. 2023, pp. 1–6.

[2] S. Kaushal, Sonali, H. Gaur, G. Bhati, and A. K. Rai, "Explainable artificial intelligence for brain tumor detection," in *2024 2nd International Conference on Advances in Computation, Communication and Information Technology (ICAICCIT)*, vol. 1, 2024, pp. 965–969.

[3] G. Sharma, D. Gupta, P. Bhardwaj, Ramneet, and P. Verma, "A comparative analysis of alzheimer's disease detection using deep learning," in *2024 International Conference on Communication, Control, and Intelligent Systems (CCIS)*, 2024, pp. 1–5.

[4] R. Hussain and S. Zeadally, "Autonomous cars: Research results, issues, and future challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1275–1313, 2019.

[5] G. Fey, M. Fränzle, and R. Drechsler, "Self-explanation in systems of systems," in *2022 IEEE 30th International Requirements Engineering Conference Workshops (REW)*, 2022, pp. 85–91.

[6] J. Enet, E. Bousse, M. Tisi, and G. Sunyé, "dpdebugger: a domain-parametric debugger for dsls using dap and language protocols," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS Companion '24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 71–75.

[7] B. J. Berger, C. Plump, and R. Drechsler, "Evoal: A domain-specific language-based approach to optimisation," in *2023 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, Jul. 2023, pp. 1–10.

[8] B. J. Berger, C. Plump, L. Paul, and R. Drechsler, "Evoal - codeless domain-optimisation," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO'24 Companion. ACM, Jul. 2024, pp. 1640–1648.

[9] B. J. Berger, C. Plump, and R. Drechsler, "Why less is sometimes more: Using boolean literals to solve 2048," *it - Information Technology*, vol. 66, no. 4–5, pp. 174–186, Aug. 2024.