EvoAl — Codeless Domain-Optimisation

Bernhard J. Berger*
University of Rostock
Rostock, Germany
bernhard.berger@uni-rostock.de

Christina Plump
DFKI GmbH—Cyber-Physical Systems
Bremen, Germany
christina.plump@dfki.de

Lauren Paul
University of Bremen
Bremen, Germany
lau_pau@uni-bremen.de

Rolf Drechsler†
University of Bremen
Bremen, Germany
drechsler@uni-bremen.de

ABSTRACT

Applying optimisation techniques such as evolutionary computation to real-world tasks often requires significant adaptation. However, specific application domains do not typically demand major changes to existing optimisation methods. The decisive aspect is the inclusion of domain knowledge and configuration of established techniques to suit the problem. Separating the optimisation technique from the domain knowledge offers several advantages: First, it allows updating domain knowledge without necessitating re-implementation. Second, it improves identification and comparison of the optimisation methods employed. We present EvoAl, an open-source data-science research tool suite that focuses on optimisation research for real-world problems. EvoAl implements the separation of domain-knowledge and detaches implementation from configuration, facilitating optimisation with little programming effort, allowing direct comparability with other approaches (using EvoAl), and ensuring reproducibility. EvoAl also includes options for surrogate models, data models for complex search spaces, data validation, and benchmarking options for optimisation researchers.

CCS CONCEPTS

• **Theory of computation → Evolutionary algorithms**; • **Computing methodologies → Genetic programming**; *Support vector machines.*

ACM Reference Format:
Bernhard J. Berger, Christina Plump, Lauren Paul, and Rolf Drechsler. 2024. EvoAl — Codeless Domain-Optimisation. In *Genetic and Evolutionary Computation Conference (GECCO '24 Companion), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3638530.3664154

1 INTRODUCTION

Solving optimisation problems is a challenge faced across a wide variety of different domains. While some problems can be solved using deterministic and/or linear methods, the last few decades have seen a huge rise in meta-heuristics, such as evolutionary computation and swarm intelligence methods. The versatility and adaptability of these methods allows their application to many different optimisation problems in many different domains. They are even capable of handling constraints (frequent in real-world applications) and black-box or expensive optimisation functions (using surrogate models).

Therefore, it is unsurprising that evolutionary computation methods are a common technique for solving domain-related optimisation tasks. Unfortunately, many research results are only presented in closed form, and their specific software is not disclosed. This not only hinders reproducibility, giving other researchers no choice but to believe the presented results, it also leads to sustainability issues: well-thought-out solutions that could be used for comparable cases have to be redeveloped and might also be republished without a sensible comparison of both tools.

We encountered this exact problem in several domain-applications of optimisation, ranging from material sciences [4] to medical science [8, 16] and applications in the hardware domain (e.g., optimising the usage of hardware accelerators [21]). We came across several issues in these projects that finally inspired the development of EvoAl[1], a publicly available open-source data-science research tool suite aimed at providing a platform for optimisation researchers who work with real-world domain optimisation problems.

One important aspect of real-world problems is communication with domain experts and the utilisation and implementation of their expert knowledge. The inclusion of this knowledge is highly important for developing an optimisation algorithm that yields results that can actually be used in the respective domain. However, domain experts frequently add or change information or variables over the course of a project, which makes hard-coding such information into optimisation algorithms impractical. EvoAl contains a data description language that offers domain experts the opportunity to document their knowledge in a formalised but simple manner. This information can then be used to modify and structure the optimisation algorithm. Separating domain knowledge from optimisation specifications is beneficial for several reasons: First, it increases efficiency during project runtime, and second, it allows the comparison of optimisation algorithms without being clouded by the domain specifics.

One further benefit is the ability of all involved experts to focus on their respective areas of expertise. Domain experts should focus

*Also with Hamburg University of Technology.
†Also with DFKI GmbH—Cyber-Physical Systems.

GECCO '24 Companion, July 14–18, 2024, Melbourne, VIC, Australia

[1]EvoAl is publicly available at https://www.evoal.de.

on solving their domain problem and supplying data-science experts with the necessary domain knowledge. Data-science experts should focus on choosing the right technique with the best configuration for the task at hand instead of implementation details. Therefore, EvoAl offers a collection of pre-implemented data-science techniques (optimisation as well as machine learning) that can be configured with the help of domain-specific languages. Hence, the data scientist can focus on his primary task: putting his expert knowledge to good use. Additionally, comparisons of different solutions can be traced back to different configurations, i.e., parameter settings or technique choices, and are less dependent on implementation issues.

EvoAl is designed in a way that fosters extensions in an open-source setting [2]. All tools and algorithms provided in EvoAl are extensible, and EvoAl offers configuration-based orchestration and execution of these components. Internally, EvoAl integrates model-driven software engineering practices, compiler-techniques for parsing configuration files, and static model validation. EvoAl builds upon an extensible component architecture that can be extended at runtime by using the provided plugin mechanism, thus allowing the addition of different optimisation or machine-learning techniques, benchmarks, and evaluation policies.

The remainder of the paper is structured as follows: Section 2 introduces the open-source software EvoAl. Next, Section 3 illustrates several use-cases of EvoAl based on a simple running example. After showing EvoAl's usage, Section 4 introduces EvoAl's current and planned features and shows how EvoAl can be extended to support project-specific features. Afterwards, Section 5 discusses the advantages and challenges of EvoAl and compares it to other existing optimisation tools. Finally, Section 6 concludes the paper.

2 EVOAL

This section introduces EvoAl—an optimisation research tool suite. First, Subsection 2.1 describes some of EvoAl's use cases. Subsection 2.2 describes the design goals that were considered during EvoAl's development. Finally, Subsection 2.3 gives an overview of additional design decisions made during development.

2.1 Use Cases

In real-world projects, there are—typically—three groups of stakeholders that collaborate while solving an optimisation problem. First, there is the *Domain Expert*, who knows details of the particular problem to solve and the corresponding domain data. Second, in projects where the optimisation function is a) unknown, or b) too expensive to compute, a *Machine-Learning Expert* may be tasked with creating a surrogate model. Lastly, the *Optimisation Expert* maps the optimisation problem to an appropriate optimisation algorithm using the information provided by the domain expert and the machine-learning expert.

Figure 1 shows a UML use-case diagram including the stakeholders and their related use-cases, which are described below.

define optimisation problem The domain expert defines the actual domain problem. Therefore, she first describes the data (see next use case) relevant to the problem and which properties should be maximised or minimised. This information is the foundation of the following use cases.

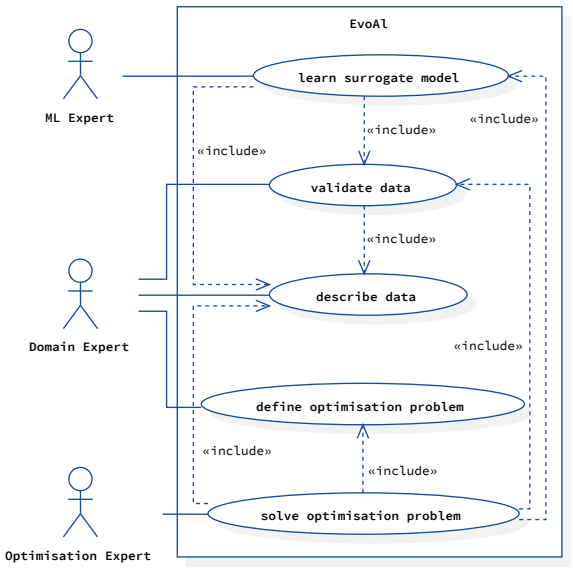

Figure 1: Overview of EvoAl's default use cases

describe data The domain expert describes the domain data in terms of which data exists, the measurement scales which apply to it, and the data constraints of which she is aware. This information significantly influences the later use cases, as they all work with the domain data.

validate data In this use case, the domain expert makes sure that the domain data is specified correctly. This aids in verifying the validity of potential solutions from the surrogate model or the optimisation process. This use case applies to the domain expert when the data collection process is unstable or the data constraints are not yet clear. In such situations, the domain expert may want to validate the domain data according to the data description she made to make sure that her understanding of the domain data is correct. Data validation helps the machine-learning expert validate predicted values from the machine-learning model as an additional quality measure. Furthermore, it supports the optimisation expert in checking candidate solutions for validity and determining whether a suggested solution is—for some reason—infeasible.

train surrogate model In cases where the optimisation function is a) unknown, or b) too expensive to compute, a surrogate model is necessary to solve the optimisation problem. A machine-learning expert constructs a surrogate model based on the data the domain expert provides. The data description helps her map the domain data to a proper algorithm that is appropriate for the data scales and constraints. The training process usually happens offline, which means the model is fixed for optimisation. However, there are several situations where additional data points could be collected, prompting an update to the model (this is the case with, for example, online-learning algorithms). At other times, collecting new data points is impossible or too expensive, in which case the optimisation algorithm needs to account for imprecision in predictions.

[2] EvoAl is published under the Apache 2.0 license and its source code is available at https://gitlab.informatik.uni-bremen.de/evoal/source/evoal-core.

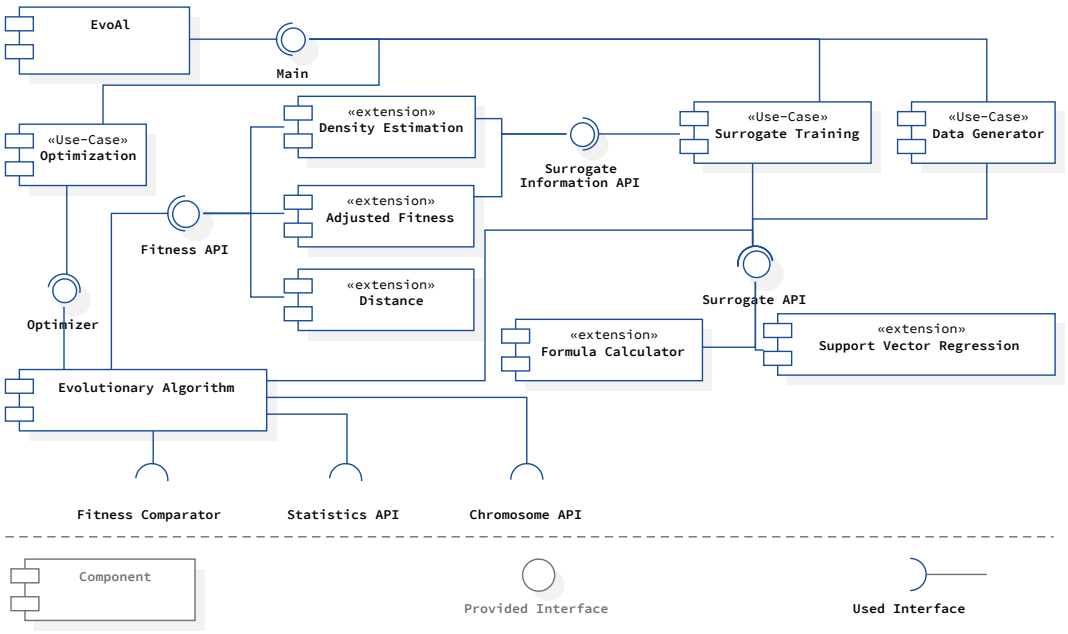

Figure 2: Excerpt of EvoAl's component diagram [2]

solve optimisation problem In the *solve optimisation problem* use case, the optimisation expert decides on an optimisation algorithm and chooses a way to map the domain data to the algorithm. The optional surrogate model helps her to deal with unknown optimisation functions or expensive to compute problems.

2.2 Design Goals

Besides implementing the use cases introduced in the last section, which correspond to the functional requirements implemented in EvoAl, we had additional design goals (non-functional requirements) that we respected during development.

Adaptability Designing and implementing optimisation software that can be configured to different domain problems requires great adaptability. Thus, EvoAl has a modular and configurable design, which is necessary to allow easy integration of changes to the domain data, optimisation strategy, or surrogate model.

Usability The different stakeholders are not necessarily programming experts, and some may not have any programming experience at all. Therefore, we made supporting easy configuration and code-free usage of EvoAl one of our central design goals during development. The configuration process of all problem-solution components is constantly checked for internal correctness, such as correctness of used data scales.

Extensibility Because most research projects are aimed at investigating new techniques, one of EvoAl's core design goals is extensibility. We implemented two levels of extensibility in EvoAl: functional and configuration extensibility.

Functional extensibility requires a well-defined architecture for all configurable elements in, for instance, an evolutionary algorithm. The use of interfaces allows for integration of new functionality without having to rewrite the application. At the same time, this compartmentalisation supports reconfigurability, since the orchestration of an algorithm must be dynamic to compose a domain solution without implementation work.

The second level, configuration extensibility, makes configuration accessible even to non-programmers. To this end, EvoAl supports new functional features via extensible configurations and also offers automatic validation to help users apply these features correctly.

Reproducibility Reproducibility is an important factor for domain experts and researchers for many reasons.

When the solved problem occurs frequently, domain experts are interested in using the problem solution (the actual optimisation solution) over and over again to find an optimal solution every time. They might even adapt their knowledge on the domain data or want to retrain the surrogate model as more training data becomes available.

Optimisation researchers have also identified the need to improve the reproducibility of experiments [1]. Additionally, FAIR principles, including data reusability, are becoming more and more important [20].

2.3 Design Decisions

EvoAl's goal is not to provide new optimisation algorithms but to make them easier to access for non-programming experts through a configuration mechanism.

We chose to use Java over Python, although the latter is more widely used in data science applications. Unlike Python, Java supports a static type system and compile-time checks that allow errors to be identified at compile time rather than at run time. This is useful in the application scenario of EvoAl, because the number of possible configuration combinations is very high. In a dynamic programming paradigm such as what Python offers, this would

```
1  import "definitions" from de.evoal.core.constraint;
2
3  module weather {
4    types:
5      /** Specification of 'degree Fahrenheit'. */
6      cardinal type Fahrenheit with constraints:
7        value > -459.67;;
8
9    data:
10     /** Daily maximum temperature. */
11     real data 'day temperature' of type Fahrenheit
12       with constraints: value > -40; value < 150;;
13
14     /** Nightly minimum temperature. */
15     real data 'night temperature' of type Fahrenheit
16       with constraints: value > -40; value < 150;;
17
18     /** Air humidity during the day */
19     quotient real data 'humidity'
20       with constraints: value >= 0; value <= 100;;
21
22     /** The perceived quality of sleep. */
23     quotient real data 'sleep quality'
24       with constraints: value >= 0; value <= 100;;
25
26   constraints:
27     /**
28      * We expect the day temperature to be higher than
29      *  the night temperature.
30      */
31     constraint(
32       data 'day temperature' > data 'night temperature',
33       "vague"
34     );
35 }
```

Listing 1: Data description of the running example

require a very high number of integration tests to ensure smooth functionality.

To support extensibility and usability, we decided to use model-driven software engineering techniques and domain-specific languages. Internally, EvoAl has a model of all algorithms, extension points, and existing extensions that is based on the Eclipse Modeling Framework[3]. Figure 2 shows a small excerpt of EvoAl's component view taken from a previous publication [2]. The domain-specific languages—which are realised using Xtext [5]—are used to describe all aspects of the problem solution, including the domain data, and a component orchestration plan. At configuration time, the orchestration plan is statically checked for correctness using Xtext's validation mechanism. At runtime, the orchestration plan is used to instantiate, configure, and connect the components accordingly. Therefore, EvoAl uses a blackboard architecture and CDI [9] for the orchestration process.

3 APPLYING EVOAL

In this section, we describe the application of EvoAl using a simple exampleThe example and its assumptions are first described in Subsection 3.1. Then, Subsection 3.2 shows the DSL-based domain-data description. Subsection 3.3 describes how a surrogate model can be trained for later usage. Finally, Subsection 3.4 shows the final optimisation solution.

3.1 Running Example

Please note that this example is for demonstration purposes only and is neither complete nor completely representative of reality.

We assume a situation where a domain expert on sleep quality (maybe from the medical domain) works on identifying the best conditions for a good night's sleep. To that end, he considers parameters like day temperature, night temperature, humidity, and potentially others. Sleep quality is, for example, measured by the proportion of REM-sleep relative to a person's total sleeping hours. The domain expert may have some data from experimental setups, including day temperature, humidity, and sleep quality of test subjects. He wants to determine which combination of humidity and day temperature leads to optimal sleep quality. In the following subsections, we will describe the standard process of configuring this problem in EvoAl, starting with the data description.

3.2 Data Description

First of all, it is important to specify the relevant data and to describe its properties and features. In this small example, we consider four different variables: day temperature, night temperature, humidity and sleep quality. In general, data is introduced with its scale (`nominal`, `ordinal`, `cardinal`, `quotient`), its storage type (`real`, `integer`, `string`, etc) and its name[4]. Additionally, constraints can be added. Listing 1 shows the data description. Humidity and sleep quality demonstrate the above-mentioned structure exactly. Both are measured as percentages, i.e., have `quotient` scale, storage type `real`, and the respective constraints to enforce the percentage property (not smaller than zero, and not higher than 100). The temperature-related variables show an additional feature of our data description language, namely types. Types can be used for data that have the same general properties (i.e., the same type). In this example, both variables stand for temperatures measured in degrees Fahrenheit. Their storage type and scale are identical, as well as the general constraint (no temperatures below $-459.67°F$ are possible). The specific data description then refers to this type, and inherits its properties but can, however, be extended with further constraints. In our case, we restrict the possible day and night temperature to more realistic values on the earth's surface.

Finally, besides univariate constraints (containing only one variable) that are defined within the data definition itself, bivariate or multivariate constraints (containing more than one variable), can be added at the end of the data description file (see lines 31–33).

3.3 Surrogate Learning

In our example, there is no direct relationship between the *temperatures*, *humidity*, and *sleep quality* known to the domain expert. He is, however, equipped with experimental data mapping these variables to one another. To solve his optimisation problem, he needs to develop a model which relates *temperatures*, *humidity*, and *sleep quality*. Listing 2 shows the corresponding configuration in our machine learning language. The first important step is to include the information from the data description (cf. line 5). This is necessary

[3] https://eclipse.dev/modeling/emf/

[4] Not all combinations of storage type and scale are probable. However, storing nominal or ordinal data as integer is quite common, and should not lead to mistaking it as cardinal.

```
1  import "definitions" from de.evoal.surrogate.ml;
2  import "definitions" from de.evoal.surrogate.ml;
3  import "definitions" from de.evoal.surrogate.smile.ml;
4
5  import "data" from weather;
6
7  module weather {
8    prediction svr
9      maps 'day temperature', 'humidity'
10       to 'sleep quality'
11       using layer transfer
12         with function 'gaussian-svr'
13           mapping 'day temperature', 'humidity'
14           to 'sleep quality'
15           with parameters
16             'ε' := 1.4;
17             'σ' := 3.0;
18             'soft-margin' := 0.15;
19             tolerance := 0.1;
20
21   predict svr from "../data.csv"
22   and measure
23     'cross-validation'(10);
24     'R²'();
25   end and store to "surrogate.pson"
26 }
```

Listing 2: Surrogate learning of the running example

to ensure that the chosen ML-techniques can indeed be used for the given data properties. Lines 8–20 define the actual ML model. They specify input and output parameters (using keywords `maps` and `to`) and the ML-technique to be used (keyword `function`) as well as the respective hyperparameters. Finally, we specify which training data to use and how to assess the quality of the final model. In our case, we decide on a 10-fold cross-validation measure and the determination coefficient. In order to be able to use the trained model later on, we store the respective information in a special file.

3.4 Optimisation

After specifying the data and training an ML model for predicting *sleep quality* from *day temperature* and *humidity*, the domain expert or the optimisation expert (depending on the project structure) can finally focus on the improvement of sleep quality. To that end, he needs to specify two major aspects: First, the actual optimisation problem, and second, the algorithm used for its solution.

An optimisation problem (lines 7–12) as such can be specified by defining search and optimisation spaces, an optimisation function and optimisation direction, i.e., whether it is a maximisation or a minimisation problem (they can, of course, be transferred to one another, this just makes it easier). Both the search and optimisation spaces refer to data defined in the data description. Similar to the ML case, this makes it possible to check whether a given algorithm can actually be used on this data. The optimsation function can be explicitly stated (if known), or specified as an unknown function which will lead to a required definition later on.

The optimisation algorithm (lines 13–61) describes the solution approach to the problem. In our case, the experts decide to use a standard genetic algorithm. They specify standard parameters such as the population size, maximum number of generations, and maximum age of an individual (lines 15–19). Then, search space data is mapped to an encoding in the genotype definition. It is possible to combine different chromosome types in one genotype. Linking these encoding definitions to the search and, therefore, the data description allows semantic crosschecks and enables repair mechanisms for constraint handling or the computation of penalty functions, as the data properties are always known. The following lines define standard operators of genetic algorithms: Which crossover and mutation operator to use, which selection operators to use, and how to treat constraints. Line 35 ff. show an additional feature that was introduced in [14]: The severity of constraints: Some constraints may have to be strictly upheld (physical laws, e.g., the basic constraint for temperatures). Others, however, could be closer to guidelines than strict constraints, i.e., that day temperature always exceeds the night temperature. There may be some weather hiccup (thunderstorm during the day, for example) for which this constraint wouldn't hold. They can then be marked with a constraint category (in this case: vague) and their handling can be defined accordingly inside the constraint-handling.

Finally, since we are using a surrogate model as a fitness function, we specify this at the end of our optimisation language file. Here, it would also be possible to add other modifications to the fitness functions. Documentation for statistics can also be included to specify which information to document for optimisation runs.

4 FEATURES OF EVOAL

In this section, we describe the currently available features of EvoAl (as of April 2024) and give an outlook on features planned for the upcoming months. Additionally, we describe how EvoAl can be extended to showcase its usability for the research community.

4.1 Current Features

4.1.1 *Optimisation.*

Genetic algorithm and evolutionary strategies. EvoAl supports general genetic algorithms and evolutionary strategies. To that end, it is based on the Java-based library Jenetics [19]. It supports standard operators for different genotype encodings, like gaussian mutators, and bit flip mutators (as representants for mutation operators), line crossovers, single point crossover, and uniform crossover (as representants for recombination operators). Additionally, it offers a wide variety of selection operators, such as strictly elitist selectors, tournament selectors, roulette-wheel selectors, and others. EvoAl has wrapped them to introduce the configurability necessary for separating configuration from implementation.

Over the course of several publications, EvoAl has also been extended with correlation-aware recombination and mutation operators [11, 13]. Furthermore, EvoAl supports standard constraint-handling techniques such as penalty-functions, kill-at-birth, or repair mechanisms [14]. For handling constraints, we also introduced the ability to define constraints of varying severity and to define a unique constraint-handling technique for each severity level. Besides penalty functions inside the fitness function, EvoAl also supports surrogate models and related adaptions as fitness functions. It is, e.g., possible to include the precision of a surrogate model's estimation in the computation of the fitness function (see, e.g., [12, 15]).

EvoAl also supports different encoding variants. Standard encodings like double or bit encoding are supported by Jenetics and have simply been included. More specialised encodings such as gray

```
import "definitions" from de.evoal.core.optimisation;
import "definitions" from de.evoal.core.ea.optimisation;
import "definitions" from de.evoal.surrogate.optimisation;
import "data" from weather;

module weather {
  specify problem 'optimal-sleeping' {
    'maximise' := true;
    'search-space' := [data 'day temperature', data 'humidity'];
    'optimisation-space' := [data 'sleep quality'];
    'optimisation-function' := 'unknown-function' {};
  }

  configure 'evolutionary-algorithm' for 'optimal-sleeping' {
    'number-of-generations' := 100;
    'size-of-population' := 50;
    'maximum-age' := 100;
    initialisation := 'random-population' {};
    'comparator' := 'numeric-comparator' {};

    genotype := 'vector-genotype' {
      'chromosomes' := [
        'bit-chromosome' {
          scale := 12;
          genes := [gene {content:= data 'day temperature';}];
        },
        'double-chromosome' {
          genes := [gene {content:= data 'humidity';}];
        }
      ];
    };

    handlers := [
      'constraint-handler' {
        'category' := "vague";
        'calculation' := 'normal-calculation' {};
        'constraint-handling' := 'malus-for-fitness'
          {smoothing := -1.5;};
      }
    ];

    selectors := selectors {
      offspring := 'roulette-wheel-selector' {};
        survivor := 'elite-selector' {
          'size-factor' := 0.3;
          'non-elite-selector' := 'tournament-selector' {
            'size-factor' := 0.1;
          };
        };
      };

      alterers := alterers {
        crossover := [
          'single-point-crossover' { probability := 0.5; }];
       mutator := [
         'probability-mutator' { probability := 0.5; }];
     };

    'optimisation-function' := surrogate {};
    documenting := ['best-candidate-per-generation' {}];
  }
}
```

Figure 3: Problem solution of the running example

encoding or genotypes with chromosomes of different encodings have been added over time. Additionally, an encoding technique for repetitive processes has been developed to allow a dynamic growth of genotype length.

Genetic Programming. Besides the standard genetic algorithms and evolutionary strategies, EvoAl also supports the basics of genetic programming. It is possible to define expression operators (e.g., for mathematical functions, such as addition, subtraction, square root, and the like), the depth of the tree, as well as the number and constraints for ephemeral variables. EvoAl also has mutation and crossover operators that work on trees and can, therefore, be used in this structure.

Swarm Intelligence Algorithms. EvoAl supports two types of swarm intelligence algorithms at the moment: Particle Swarm Optimisation (PSO), and Ant Colony Optimisation (ACO). Both are supported in their standard variants with configurable parameters.

The distinction between optimisation problem and algorithmic solution allows a sensible comparison between swarm intelligence algorithms and evolutionary computation methods. The difference in results can be traced back to the choice of algorithm and configuration, not the problem specification, as this is equal for all solution approaches.

4.1.2 Machine Learning. EvoAl supports different machine-learning algorithms for training surrogate models. Currently, there is support for various learning techniques based on the Java SMILE library [10]. SMILE supports Support Vector Regression with different kernels, random forest, gradient boosting, and Gaussian processes. We also support standard goodness-of-fit measures for these techniques, e.g., the determination coefficient R^2, a k-fold cross-validation with *MSE*, *MAPE*, and *RMSE*, which can be configured as well.

4.1.3 Benchmarking. For benchmarking purposes, EvoAl integrates the COCO benchmark suites [7] and several standard functions, such as the Sphere function, the Rastrigin function, the Ackley function, and the Rosenbrock function, just to name some standard examples. Usually, these functions come with parameters which can be configured when defining their choice. These benchmarks are not necessary for every usage of EvoAl or every evaluation of evolutionary algorithms, but come into play when testing new methods on synthetic data.

4.1.4 Data Generation. For evaluation purposes, EvoAl implements a configurable training-data generator, which facilitates the construction of surrogate models for benchmark functions. Similar to other tools from the EvoAl suite, the generator can be configured using a domain-specific language. Typically, normally or multivariate normally distributed input data are generated. The input data can then be fed into a benchmark function to generate the correct benchmark distribution given the input data. Afterward, some or all of the data can be noised to mimic measurement noise that can be found in real-world projects. If required, the generated data can be validated using constraints specified in the data description. Finally, the resulting data can be used as input for ML methods to train surrogate models for an optimisation algorithm. Besides this, the data generator can be used to generate initial populations for the evolutionary algorithm.

4.2 Planned Features

We have a long list of planned features in our backlog that will find their way into upcoming EvoAl releases. Currently, we are working on

- supporting additional learning techniques for surrogate-model learning. A first demonstrator plugin for neural nets exists that uses Eclipse Deeplearning4j [5] to configure and train neural nets.

[5] Eclipse Deeplearning4j is available online https://deeplearning4j.konduit.ai.

- a hyper-parameter optimisation mechanism that uses the model information on available machine-learning, or optimisation parameters and their value ranges to automatically map the parameters to an optimisation algorithm that can be optimised.
- more detailed semantic checks at the configuration level to support users in selecting the correct configuration for their problem.
- a visualisation framework for teaching optimisation algorithms. First—partly interactive—visualisations for fitness evolution, individual inheritance, and dynamic environment changes already exist but need some improvement.

A major change for end-user usability and accessibility of EvoAl is the current switch to Eclipse Theia [6]. Theia, a Cloud-enabled editor, offers simpler deployment options compared to the current Eclipse-based implementation. Combining configuration and visualisation ideas in a Cloud environment makes EvoAl more accessible to students and end users.

4.3 Adding custom features

This subsection will show a simple custom optimisation-function plugin for the sleeping quality example introduced in Section 3. For this purpose, we assume that the actual function that maps *night temperature* and *humidity* to *sleeping quality* is now known to the domain expert. Thus, the machine-learning expert only has to train a model that predicts *night temperature* based on *day temperature* and *humidity*. For this example, we assume that the mapping between *night temperature*, *humidity*, and *sleep quality* that is now known to the domain expert is given as follows:

$$quality = \max\left\{-0.01(night - 63)^2 - 0.04(humidity - 50)^2 + 100; 0\right\}$$

This requires us to extend EvoAl such that it can use a custom fitness function, using the ML model for predicting the *night temperature* and the given formula for computing the *sleep quality*.

To implement an EvoAl plugin, the best starting point is to clone the EvoAl template project [7]. Usually, it is necessary to take care of two aspects: First, the actual implementation of the extension and second, the extension (if necessary) of the configuration. We will describe both in the following paragraphs.

Listing 3 shows the exemplary implementation of a custom optimisation function. The `Named` annotation states the name that can be used in the configuration to access the component. Since this component should be an optimisation function, it has to implement the `OptimisationFunction` interface. During creation, EvoAl will inject the loaded surrogate model into the annotated attribute `function` for later usage, and then call the `init` function. The initialisation function receives the instance configuration parsed from the configuration file as a parameter and thus allows the component to configure itself according to the user's choices. The `evaluate` function, which receives a possible candidate solution `properties`[8], calculates the fitness of the passed candidate. In the example case, the candidate will contain the *day temperature* and the *humidity*, which it passes to the surrogate model. The surrogate model predicts the *night temperature*, and the predicted

```
17   */
18  @Dependent
19  @Named("com.example.fitness.gecco.custom-fitness")
20  @Slf4j
21  public class CustomFitness implements OptimisationFunction {
22      @Inject
23      private SurrogateFunction function;
24
25      private final static double idealTemp = 63;
26      private final static double idealHumidity = 50;
27
28      @Override
29      public double[] evaluate(final Properties properties) {
30          int hIndex = properties.getSpecification().indexOf("humidity");
31          final Properties oFeatures = function.apply(properties);
32
33          double oFeature = Math.max(0.0,
34                  -0.01 * Math.pow(oFeatures.getAsDouble(0) - idealTemp, 2.0)
35                  -0.04 * Math.pow(properties.getAsDouble(hIndex) -
                    ↪  idealHumidity,2.0)
36                  +100.0
37                  );
38
39          return new double[] {oFeature};
40      }
41
42      @Override
43      public OptimisationFunction init(final Instance configuration) throws
        ↪  InitializationException {
44          log.info("Creating a custom optimisation function");
45
46          return OptimisationFunction.super.init(configuration);
47      }
48  }
```

Listing 3: Implementation of the custom optimisation function

value is then used to calculate the actual *sleeping quality* based on the given formula. The result is returned to the function's caller.

After implementing the fitness function, it is necessary to adapt the configuration to use the newly created fitness function. Listing 4 shows the unified diff between the two optimisation solution configurations. First, an import statement is added to the configuration file that imports the newly created module containing our fitness function. The name corresponds to the name in the aforementioned annotation without the name part after the last dot (cf. Listing 3). This import allows the usage of the custom optimisation function. Second, we substitute the definition of the fitness function as `surrogate` through the custom fitness function, which refers to our newly implemented fitness function.

5 DISCUSSION

Section 5.1 discusses the advantages and Section 5.2 the disadvantages of EvoAl. Section 5.3 gives an overview on related research.

5.1 Advantages

One major advantage of EvoAl is the built-in documentation and reproducibility of conducted experiments. The configuration files associated with a given experiment state which algorithms and parameter settings were used. All configurable information is captured and documented. This allows anyone to a) use EvoAl to repeat the algorithm on different data[9] or b) to implement the algorithm themselves[10]. The high-level algorithm description provided in EvoAl's configuration files is better than a source-code-based

[6] Online available at https://theia-ide.org.
[7] The project can be accessed at https://gitlab.informatik.uni-bremen.de/evoal/source/evoal-template.
[8] Please note that the *Properties* is a EvoAl implementation that does automatic type checks and conversions according to the configuration.
[9] Compare ACM's definition of reproduction: Different Team, Same Setup
[10] Compare ACM's definition of reproduction: Different Team, Same Setup

```
--- weather1.ol
+++ weather2.ol
@@ -1,6 +1,7 @@
 import "definitions" from de.evoal.core.optimisation;
 import "definitions" from de.evoal.core.ea.optimisation;
 import "definitions" from de.evoal.surrogate.optimisation;
+import "definitions" from com.example.fitness.gecco;
 import "data" from weather;

 module weather {
@@ -56,7 +57,7 @@
                 'probability-mutator' { probability := 0.5; }];
         };

-    'optimisation-function' := surrogate {};
+    'optimisation-function' := 'custom-fitness' {};
     documenting := ['best-candidate-per-generation' {}];
   }
 }
```

Listing 4: Difference between the configuration shown in Figure 3 and new configuration that uses the custom optimisation function

solution, where it is necessary to understand the entire implementation to extract the algorithm configuration. Thus, the domain language acts as a documentation of optimisation solutions.

As part of an experiment, we even implemented a Python-based model interpreter for our configuration language that can execute the described model using Python libraries, showing that the configuration language is not only bound to EvoAl and the Java language. Another evident and prominent advantage is the required time-to-solution for new domain problems that can be solved with off-the-shelf algorithms and parameter adaptations. Even fundamental changes, such as switching data encoding or changing constraint-violation handling, can be done by experienced users in a matter of minutes, dwarfing the time required to make changes to a manually created solution. This allows the optimisation expert to focus on identifying the best algorithm and settings without spending too much time on implementation efforts. It also fosters the longevity of project solutions. When domain information slightly changes after a project officially ends, domain experts are capable of addressing these changes on their own. EvoAl's data description language allows them to handle changes without any relevant coding experience.

5.2 Disadvantages

One downside of our approach is the modeling and engineering effort required to design algorithms and adaptations in the most generic way possible. Even for trained software engineers and architects, finding algorithms and implementing programs that can be applied to all problem instances is challenging. Since EvoAl can be understood as a highly flexible frontend to different data science libraries, a large amount of integration testing is required. Currently, we are working on improving the automated integration-test coverage of EvoAl. However, this is a standard and well-known problem for configurable software systems and not specific to EvoAl.

5.3 Related Tools

A plethora of open-source tools and libraries for optimisation, and especially machine learning, exists. First, there are different libraries, such as Jenetics [19], `motipy` [18], `scipy.optimize`[11], or the MOEAFramework[12], that offer variants of optimisation algorithms. The MOEA Framework also focuses on visualising the optimisation process by plotting fitness values. Similarly, `HeuristicLAB` offers a wide range of optimisation algorithms and well-known problem instances and visualises the results of optimisation runs [17]. These tools offer great options for using or benchmarking optimisation and/or machine learning algorithms. Nevertheless, these libraries require programming skills to create customised domain-specific tools. EvoAl offers non-programmers the chance to use these techniques and focus on the actual problem instead of typical implementation and integration issues.

The modelling community focuses on model optimisation (searching for an optimal model, which is similar to genetic programming) tools that are available as open-source. *MOMoT* combines model-driven engineering and search-based optimisation to search for optimal model instances [6]. The *MDEoptimiser*, for instance, uses modelling techniques and domain-specific languages [3]. Similar to EvoAl, these approaches use modelling techniques, and in the case of *MDEoptimiser* even a domain-specific language. Nevertheless, the approaches are limited to genetic programming-like problems.

6 CONCLUSIONS

In this paper, we presented the open-source data-science tool suite EvoAl with a strong emphasis on optimisation. We showed a subset of the use cases EvoAl supports and how these are mapped to its component architecture, which is designed for extensibility. Using a running example, we showed how EvoAl can be configured to solve optimisation tasks that employ a learned surrogate function without additional programming effort. Furthermore, the paper gives a brief overview of EvoAl's current and planned features and shows with a simple example how EvoAl can be extended.

While EvoAl showed first good results in terms of usability for domain experts, we plan to do more experiments on the usability aspects for the other roles. As EvoAl configures at runtime, it is important to compare its performance with a hand-written algorithm implementation.

ACKNOWLEDGEMENT

The work was partially funded by the AI Center for Health Care of the U Bremen Research Alliance, financially supported by the Federal State of Bremen in Germany.

REFERENCES

[1] Thomas Bartz-Beielstein, Carola Doerr, Jakob Bossek, Sowmya Chandrasekaran, Tome Eftimov, Andreas Fischbach, Pascal Kerschke, Manuel López-Ibáñez, Katherine M. Malan, Jason H. Moore, Boris Naujoks, Patryk Orzechowski, Vanessa Volz, Markus Wagner, and Thomas Weise. 2020. Benchmarking in Optimization: Best Practice and Open Issues. *CoRR* abs/2007.03488 (2020). arXiv:2007.03488 https://arxiv.org/abs/2007.03488

[2] Bernhard J. Berger, Christina Plump, and Rolf Drechsler. 2023. EVOAL: A Domain-Specific Language-Based Approach to Optimisation. In *2023 IEEE Congress on*

[11] Available online at https://docs.scipy.org/doc/scipy/tutorial/optimize.html
[12] Available online at http://moeaframework.org

Evolutionary Computation (CEC). 1–10. https://doi.org/10.1109/CEC53210.2023.10253985

[3] Alexandru Burdusel, Steffen Zschaler, and Daniel Strüber. 2018. MDEoptimiser: a search based model engineering tool. In *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings* (Copenhagen, Denmark) *(MODELS '18)*. Association for Computing Machinery, New York, NY, USA, 12–16. https://doi.org/10.1145/3270112.3270130

[4] N. Ellendt and L. Mädler. 2018. High-Throughput Exploration of Evolutionary Structural Materials. *HTM Journal of Heat Treatment and Materials* 73, 1 (2018), 3–12. https://doi.org/doi:10.3139/105.110345

[5] Moritz Eysholdt and Heiko Behrens. 2010. Xtext: Implement Your Language Faster than the Quick and Dirty Way. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion* (Reno/Tahoe, Nevada, USA) *(OOPSLA '10)*. Association for Computing Machinery, New York, NY, USA, 307–309. https://doi.org/10.1145/1869542.1869625

[6] Martin Fleck, Javier Troya, and Manuel Wimmer. 2016. *Search-Based Model Transformations with MOMoT*. Springer International Publishing, 79–87. https://doi.org/10.1007/978-3-319-42064-6_6

[7] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff. 2021. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. *Optimization Methods and Software* 36 (2021), 114–144. Issue 1. https://doi.org/10.1080/10556788.2020.1808977

[8] Daniel Christopher Hoinkiss, Jörn Huber, Christina Plump, Christoph Lüth, Rolf Drechsler, and Matthias Günther. 2023. AI-driven and automated MRI sequence optimization in scanner-independent MRI sequences formulated by a domain-specific language. *Frontiers in Neuroimaging* 2 (May 2023). https://doi.org/10.3389/fnimg.2023.1090054

[9] Jakarta Contexts and Dependency Injection Spec Project. 2019. *Specification: Jakarta Contexts and Dependency Injection 2.0*. Technical Report. Eclipse Foundation.

[10] Haifeng Li. 2024. Smile. https://haifengl.github.io.

[11] Christina Plump, Bernhard J. Berger, and Rolf Drechsler. 2021. Domain-driven Correlation-aware Recombination and Mutation Operators for Complex Real-world Applications. In *2021 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. https://doi.org/10.1109/cec45853.2021.9504931

[12] Christina Plump, Bernhard J. Berger, and Rolf Drechsler. 2021. Improving Evolutionary Algorithms by Enhancing an Approximative Fitness Function through Prediction Intervals. In *2021 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. https://doi.org/10.1109/cec45853.2021.9504722

[13] Christina Plump, Bernhard J. Berger, and Rolf Drechsler. 2022. Adapting mutation and recombination operators to range-aware relations in real-world application data. In *GECCO '22: Genetic and Evolutionary Computation Conference, Companion Volume, Boston, Massachusetts, USA, July 9 - 13, 2022*, Jonathan E. Fieldsend and Markus Wagner (Eds.). ACM, 755–758. https://doi.org/10.1145/3520304.3529066

[14] Christina Plump, Bernhard J. Berger, and Rolf Drechsler. 2022. *Choosing the Right Technique for the Right Restriction – A Domain-Specific Approach for Enforcing Search-Space Restrictions in Evolutionary Algorithms*. Springer International Publishing, 349–359. https://doi.org/10.1007/978-3-031-05359-7_28

[15] Christina Plump, Bernhard J. Berger, and Rolf Drechsler. 2022. Using density of training data to improve evolutionary algorithms with approximative fitness functions. In *IEEE Congress on Evolutionary Computation, CEC 2022, Padua, Italy, July 18-23, 2022*. IEEE, 1–10. https://doi.org/10.1109/CEC55065.2022.9870352

[16] Christina Plump, Bernhard J. Berger, Rolf Drechsler, Daniel C. Hoinkiss, Christoph Lueth, Matthias Günther, and Jörn Huber. 2024. Finding the perfect MRI sequence for your patient — Towards an optimisation workflow for MRI-sequences. In *2024 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. Accepted for publication.

[17] S. Wagner, G. Kronberger, A. Beham, M. Kommenda, A. Scheibenpflug, E. Pitzer, S. Vonolfen, M. Kofler, S. Winkler, V. Dorfer, and M. Affenzeller. 2014. *Architecture and Design of the HeuristicLab Optimization Environment*. Springer International Publishing, 197–261. https://doi.org/10.1007/978-3-319-01436-4_10

[18] Thomas Weise. 2023. Software - motipy: the Metaheuristic Optimization in Python Library. *SIGEVOlution* 16, 4, Article 3 (dec 2023), 2 pages. https://doi.org/10.1145/3638461.3638464

[19] Franz Wilhelmstötter. 2024. Jenetics — Genetic Algorithm, Genetic Programming, Evolutionary Algorithm, and Multi-Objective Optimization. https://jenetics.io/. (accessed on 26 March 2024).

[20] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, et al. 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data* 3, 1 (March 2016). https://doi.org/10.1038/sdata.2016.18

[21] Jan Zielasko and Rolf Drechsler. 2023. Virtual Prototype Driven Application Specific Hardware Optimization. (2023), 1–8. https://doi.org/10.1109/FDL59689.2023.10272131