Choosing the right technique for the right restriction – a domain-specific approach for enforcing search-space restrictions in evolutionary algorithms

Christina Plump[1,3], Bernhard J. Berger[2], and Rolf Drechsler[1,3]

[1] Faculty of Computer Science, University of Bremen, Bremen
{cplump,drechsler}@uni-bremen.de
[2] Institute of Embedded Systems, Hamburg University of Technology, Hamburg
bernhard.berger@tuhh.de
[3] DFKI GmbH - Cyber-Physical Systems, 28359 Bremen

Abstract. Evolutionary algorithms are a well-known tool for optimising problems that are hard to solve analytically. They mirror the evolutionary approach of recombination and mutation as well as a selection process according to the fitness of an individual. Individuals who violate set search space restrictions are either killed at birth or penalised in their fitness calculation. Which possibility is best to choose depends on the problem at hand and therefore subject to change. Furthermore, restrictions can be vague, for example, when stemming from experiments. We propose a noise-sensitive penalty for violating restrictions and develop a framework where an expert might choose which penalising technique to choose for what kind of restriction. We evaluate our configurable approach against configurations where one technique is used for every type of restriction and find that our approach achieves better results than a strict configuration. Additionally, the noise-sensitive penalising method allows individuals to survive, which may only violate the given restrictions due to a noised testing environment, leading to better results.

1 Introduction

Optimisation is a tedious but necessary task in many areas. May it be economic problems, climate predictions, machine learning programs, scheduling problems or just an optimisation *for the sake of it*, it accompanies researchers everywhere. The research community developed a whole bundle of techniques to do so over the last decades. They start at analytical approaches like computing gradients, continue with numerical solutions and certainly do not stop at heuristic approaches. The class of evolutionary algorithms has proven to be an effective tool when dealing with optimisation tasks that are hard to solve or do not have an analytically defined problem. They copy the basic ideas of evolution, such as recombination, mutation and a selection based on fitness. The population reshapes itself repeatedly, and when the operators are set fittingly, it converges to an optimum, thus solving the optimisation task.

Unfortunately, it is seldomly quite that easy, and the expression *when the operators are set fittingly* is a challenge in its own right. There are several issues to be considered for the configuration of an evolutionary algorithm: From encoding to choice of mutation and recombination and definition of fitness as well as a selection scheme. For this work, we focus on a problem lying at the intersection of selection, fitness and encoding: Restrictions. Most optimisation problems are subject to restrictions. In production, machines can not run longer than a defined amount of time, resources (material and human) are finite, and some minimal production may be necessary. In material sciences, it may not be allowed to combine different alloys for fear of health hazards, and when conducting experiments on structural materials, the results are usually not independent of one another. There are tons of examples where it is necessary to optimise under restrictions.

Logically, several techniques have been developed to deal with these restrictions. For example, individuals may be *killed-at-birth* if they violate these restrictions. Another technique is more subtle: The individuals may survive but receive a worse fitness value than correct individuals. This downgrading might allow the population to surpass infeasibility regions when the search space is not convex. However, when addressing real-world applications from a domain with experimental setups, restrictions might not be to handle *just as they are*. For example, if theory states that one value is greater than another, but in experiment reality, results may be subject to noise, samples may be imperfect, the conducting researcher may have collected them differently from another one. All in all, there is too much uncertainty to strictly prohibit a solution that violates such a restriction from staying in the population if it is otherwise fit. Therefore, we developed a noise-sensitive restriction handler that allows mistakes in a specified range, which the standard deviation for this value influences.

This adaption led us to another obstacle when dealing with restrictions: Even in experimental setups, there are certain restrictions that have to be upheld definitely. One example of this are physical laws, e.g. a length may not be smaller than zero. For this type of restriction, noise sensitivity might be disadvantageous. To solve this issue, we built on the domain-specific language from [9] and constructed selection operators that are capable of using different restriction handling techniques for different types of restrictions.

We evaluated our approach on four different benchmark functions with carefully chosen restrictions. We compare using one singular technique for every type of restriction to our adapted approach and find that we achieve better results than the singular approach.

2 Background

Several sources identify the same groups of constraint handling [1, 2, 8]: First, they name handling techniques that tackle the constraint compliance itself, e.g., repairing genotypes or deleting individuals that violate the given constraints. These techniques usually go by the name *restrictive*. Second, there are penalty

techniques. In principle, they work similarly: If an individual violates one or more constraints, penalty terms worsen its fitness accordingly. They fall under the term *tolerant*. Third, there are *decoding*-techniques, which choose a different representation. This change in representation allows for the creation of a valid individual. Fourth, the constraints become additional optimisation goals and the given problem a multi-objective optimisation problem.

We will shortly discuss the advantages and disadvantages of the first two types to illustrate the importance of customisation. Restrictive methods usually either delete all constraint-violating individuals during their creation, i.e. they do not become part of the population or repair all constraint-violating individuals such that they represent a valid individual [3, 12]. This fact leads to the advantage that only valid individuals are in the population and, therefore, the optimisation will converge to a valid individual. However, killing all invalid individuals can be time-costly for two reasons: One, new individuals have to be created to keep the population size stable; second, depending on the complexity of the constraint, an evaluation may require much computational time. Repairing individuals requires knowledge about the possibility to repair a given individual, which is not always present. Furthermore, it may produce a distorted distribution of individuals when the repairing process is skewed. Both techniques' disadvantage is that they decrease the possibility of finding an optimum in complex landscapes. For example, invalid regions may surround the optimum. Then, killing all invalid individuals would reduce the chance of slowly surpassing these regions to propagate to the optimum. In conclusion, it might be wise to choose restrictive methods for constraints that are easy to compute and easy to repair and are boundaries than constraints - for example, the condition that a length should not be negative.

Tolerant techniques allow the presence of invalid individuals, but they are disadvantaged. This disadvantage can happen in different ways. One, only valid parents can be chosen for reproduction, thus decreasing the reproducibility of invalid individuals [12]. Second, an expansion of the fitness function with one penalty term per constraint [10]. If an individual violates one constraint, it will worsen his fitness. Usually, the penalising terms are weighted and additionally, they may depend on the extent of the violation. Furthermore, the issue of normalisation has to be taken care of when defining such penalising. The main advantage has already been named: Invalid individuals may survive if their fitness is good enough and their violations are not too high. Therefore, it is possible to overcome infeasible areas that separate feasible areas from one another. The disadvantage, especially with penalty functions, is the distortion of the fitness function and the high degree of necessary fine-tuning of the penalty function's parameters to ensure just the right amount of adjustment.

3 Methodology

We identified two challenges when dealing with real-world applications, especially from experimental setups, in evolutionary algorithms: First, although there

are many techniques to deal with constraints, they do not explicitly consider experimental noise. However, constraint-handling techniques need to include noise when dealing with a situation where the search space consists of experimentally derived data. Second, usually, there is not just one type of constraint present. There may be strict constraints that need to be upheld; there may be constraints that hold but are subject to noise; there may be constraints where domain experts are rather vague about when defining them to designers of evolutionary algorithms. This section will present two expansion of constraint-handling techniques: First, we define noise-sensitive restrictions– an application is possible for tolerant and restrictive techniques. Second, we adapt the domain-specific language from Plump et al. to include a type specification for given constraints [9]. When defining the evolutionary algorithm, the designer then can decide which constraint-handling technique to use for which constraint type.

3.1 Noise-sensitive constraint handling

Commonly, the search space of an optimisation problem may be of noised nature. One example is the method by Ellendt et al., which deals with the systematic discovery and development of structural materials [6]. Due to the two-step nature of the optimisation, the search space of the latter one consists of experimental values (cf. Drechsler et al. [4, 5]). Weather forecasts with noised data from weather stations - that are necessary to predict the weather situation - are another example. If this is turned into an optimisation problem (what needs to be measured to obtain a given weather situation), the search space will be subject to noise again.

For the following concepts, we consider real-valued search space of n dimensions, i.e.,. $\mathscr{S} \subset \mathbb{R}^n$. Please note, however, that other search spaces are possible, e.g. processes or states, and our concepts are extensible to those.

However, knowledge about constraints in the search space, e.g. $x_1^2 - 5 \leq x_2$ is based on the actual values in the search space, not the ones that are measured. That is, a measurement might actually violate the constraint, although the *true* value behind the noised measurement does not [11]. For example, the pair $(4, 12)$ fulfills the exemplary inequality mentioned above. However, if the *true* value 4 is measured as 4.2 due to noise, and 12 is measured as 11.9, $4.2^2 - 5 = 12.64 \leq 11.9$ is no longer true, although the underlying candidate $(4, 12)$ is. Let us therefore define a noised value as

$$\hat{x}_i = x_i + \epsilon_i, \epsilon_i \sim \mathscr{N}(0, \sigma_i) \tag{1}$$

where x_i denotes the *true* value and we assume σ_i to be a known (or estimated) standard deviation for measurement setup for x_i. Please note, that since ϵ_i can be negative as well, this definition is equivalent in meaning (but not in notation) to $x_i = \hat{x}_i + \epsilon_i$.

Let furthermore w.l.o.g. a constraint in an n-dimensional search space be given as $g_j(x_1, ..., x_n) \leq 0$, referred to by g_j. For example, the above constraint can be written as $x_1^2 - x_2 - 5 \leq 0$ and fulfills the denotation.

Definition 1. *A constraint-handling technique T on a given constraint g_j is called* noise-sensitive *of degree α when it satisfies the following condition:*
For a given search space candidate $(\hat{x}_1, ..., \hat{x}_n)$ T is only applied, if $\forall (b_1, ..., b_n), b_i \in \{-1, 0, 1\} : g_j(\hat{x}_1 + b_1\alpha\sigma_1, ..., \hat{x}_n + b_n\alpha\sigma_n) > 0$.

The presented definition also formulates as follows: A noise-sensitive constraint-handling technique first checks whether an individual might fulfil the constraint had it not been for noise, and only if there is no possible noise-combination, it considers the constraint violated and applies its technique. Considering the example from above, if $\sigma_1 = 0.1$ and $\sigma_2 = 0.01$, we observe the following: For $\alpha = 1$ and $b_1 = -1$ and $b_2 = 0$, we obtain $(4.2 - 1 \cdot 0.1)^2 - 11.9 - 5 = -0.09 \leq 0$. Therefore, there is a noise-combination such that the constraint is fulfilled for $\alpha = 1$. That is, every noise-sensitive constraint-handling technique with $\alpha \geq 1$ would not consider this constraint violated by the pair $(4.2, 11.9)$.

We would like to point the reader to several interesting observations: First, the choice of α gives the designer of the evolutionary algorithm the possibility to be rather strict on noise or somewhat tolerant. The higher the choice for α, the more solutions would not receive appropriate handling although they violate the constraints (one would obtain more *false corrects*). We advise (due to the 68-95-99 rule of the normal distribution) not to choose an $\alpha > 3$, for this would already include 99% of the possible *actual* values underlying a given measurement. We propose choosing an α between 0.5 and 1.5, but this is up to the designer.

Second, along with the first observation, α should be chosen depending on the constraint technique. A smaller α may suffice for a tolerant technique, while a higher α might be more appropriate for a restrictive technique.

Third, defining $b_i \in \{-1, 0, 1\}$ is a compromise as it has to be ensured that an algorithm employing this technique will terminate. More true to the cause would be $b_i \in [-1, 1]$. However, we achieve identical results when g_j is continuous in the relevant neighbourhood. With this technique mainly being developed for experimental setups, this is the case for most situations.

Fourth, domain experts must supply the evolutionary algorithm expert with the necessary information on standard deviations of the measurements. This necessity of configuration motivates our next contribution, which the following subsection explains.

3.2 Configurable constraint handling

Especially in real-world applications, domain knowledge is significant for designing the evolutionary algorithm. Designers make some decisions initially, e.g., the encoding or the choice of mutation and recombination operators. Others, however, may be subject to change when new variables are introduced or information about the included variables changes. When faced with a dynamic environment, a changing infrastructure or a developing research system, these changes are likely to happen more often than less often. Plump et al. propose a data description language for providing information on data to the evolutionary algorithm without having to change the principal design of this algorithm [9].

```
experiment example ("x") {
  measuring method measurement ("x") {

    quotient descriptor "value 0" ("x.0") in ["µm"]
      § constraint("x.0" >= 0.0, "strict")
      § standardDeviation(1.5);

    quotient descriptor "value 1" ("x.1") in ["µm"]
      § constraint("x.1" >= 0.0, "strict")
      § standardDeviation(0.8);

    § constraint("x.1" / "x.2" >= "x.0", "vague")
    § constraint("x.1" >= "x.0", "noise")
  }
}
```

Listing 1.1. Exemplary data description file

They use this to include dependency information on the search space dimensions to modify mutation and recombination operators. We enhance this data description language to include information about constraints on all search space dimensions. Listing 1.1 shows an example of this, where one can see the included boundary and constraint information.

However, as mentioned above, it might make sense to treat different constraints with different techniques depending on the domain. Which technique to choose for which constraint requires the expertise of both experts (domain and evolutionary algorithm). The domain expert needs to assess the strictness and assurance of a given constraint—whether, for example, it is defined by a physical law or only a known relationship that may be somewhat vague. On the other hand, the evolutionary algorithm expert needs to define the constraint-handling technique to use. We introduce a separation of concern in this case: In the data description file, the domain expert can label each constraint with a type. Additionally, he can supply the standard deviation necessary for noise-sensitive constraint handling. The evolutionary algorithm expert can define the constraint-handling technique for every type defined by the domain expert in a configuration file for the evolutionary algorithm. Listing 1.2 shows one possible corresponding configuration file for a given data description file.

4 Implementation

We implemented the proposed constraint handling with Jenetics [13] to evaluate our proposed approach. Jenetics is a genetic algorithm, evolutionary algorithm, genetic programming, and multi-objective optimisation framework. The implementation uses Xtext [7] for parsing data description language files. We generate the configured constraint handling mechanisms based on the domain information and the evolutionary algorithm's configuration. Since the constraint handling mechanisms differ, we used two of Jenetic's extension points to implement the strategies. For eliminating individuals, we use Jenetic's constraint mechanism. This mechanism allows a client application to check all individuals

```
{ "algorithm": {
    "constraint-handling": {
      "strict": {
        "calculation": { "name": "normal" },
        "handling": { "name": "killAtBirth" }
      },
      "vague": {
        "calculation": { "name": "normal" },
        "handling": { "name": "malusForFitness", "smoothing": 1.0 }
      },
      "noise": {
        "calculation": { "name": "standard-deviation", "alpha": 1.0 },
        "handling": { "name": "killAtBirth", "smoothing": 1.0 }
      }
    },
  ... }
... }
```

Listing 1.2. Excerpt of the evolutionary algorithm configuration

of a generation. If the constraint implementation marks an individual as invalid, Jenetis eliminates the individual and produces a new one. For the malus-based constraint handling, we decorated the fitness function. The decorator pattern allows us to use the default fitness calculation and modify the result based on the constraint evaluation and the configuration.

5 Evaluation

To gain insight into the effects of our presented approach, we carried out a thorough evaluation. With this evaluation, we plan to investigate the following research questions:

Research Question 1 *Which influence does the adjustability of constraint handling techniques have on the conformance of individuals to the given constraints?*

Research Question 2 *Is the above mentioned influence, if present, dependent on the choice of encoding: real-valued or bit encoding?*

5.1 Setup of evaluation

We use four standard benchmark functions for our evaluation as fitness functions: the Ackley-Function, the Rastrigin Function, the Rosenbrock Function, and the Weighted Sphere Function. These four functions have been chosen according to the classification attributes of functions vital for evolutionary algorithms: Separability and Modality. Each function represents one combination of these two attributes. All four functions were evaluated with ten dimensions as well as two dimensions. Additionally, to achieve some shift of the functions, we did not simply minimise towards zero but minimised the distance to the given target and varied that.

Furthermore, to effectively answer Research Question 2 we used both encodings: A 64-bit floating-point real-valued encoding and a bit encoding with 14 decimal places. We chose an elite selector combined with a tournament selector for parent and offspring selection. For the real-valued encoding, we chose a Gaussian Mutator and a Line Recombinator and for the bit encoding a Swap Mutator and a single crossover as recombination. These operator combinations had proven the most effective for our given situation in previous work.

We carefully chose data description files for the 2D and 10D case with standard deviations and constraints. We varied the assigned type of constraint between `strict`, `vague`, `noise` to symbolise the domain expert's preferred type of constraint-handling.

As constraint-handling techniques, we implemented the kill-at-birth approach as representative for the restrictive methods and the penalty approach as representative for tolerant methods. For both the kill-at-birth and the penalty approach, we added the option of noise-sensitivity with a configurable degree α.

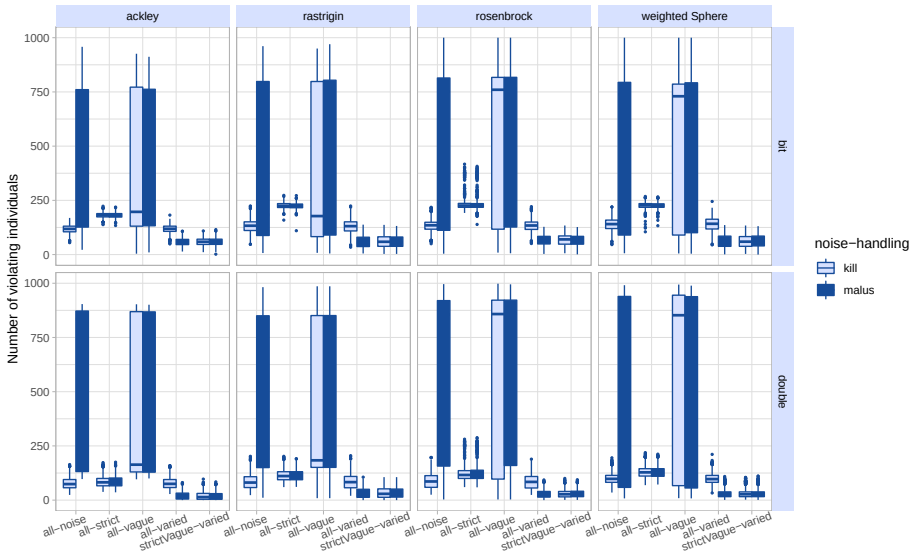

Fig. 1. Number of constraint-violating individuals for constraint $x_0 \geq 0.0$ in a two-dimensional search space with $x_0$representing the first dimension. The columns show the results for the different benchmark functions, the rows differentiate the chosen encoding. For each configuration, there is one boxplot for the bit-encoding and one for the double encoding.

We defined five different configurations: all-strict (all constraints are to be handled with a strict technique), all-vague (all constraints are to be handled with

a vague technique), all-noise (all constraints are to be handled noise-sensitive), and all-varied (all constraint types are used) and strict-vague-varied (only strict and vague types occur).

All in all, we had 8000 different configurations. Each was run 20 times for statistical reasons and with 100 generations and 1000 individuals.

5.2 Results

Figure 1 presents an overview of the acquired data. The boxplots show the distribution of the number of individuals in the last generation, which violates this configuration's first constraint. The description *noise-handling* refers to the chosen constraint-handling technique for constraints of type noise (kill for the restrictive one, and malus for the tolerant one). The first observation is the high variance for configurations with only type vague (all-vague) constraints and, of course, all-noise with noise-handling malus. Second, both configurations with variations and the all-strict configuration have much lower variance and, in most cases, a smaller median, i.e., they have fewer constraint-violating individuals in more cases. Furthermore, the smallest results are achieved either by all-varied, when employing *malus* as noise-handling and strictVague-varied. Figure 2 shows

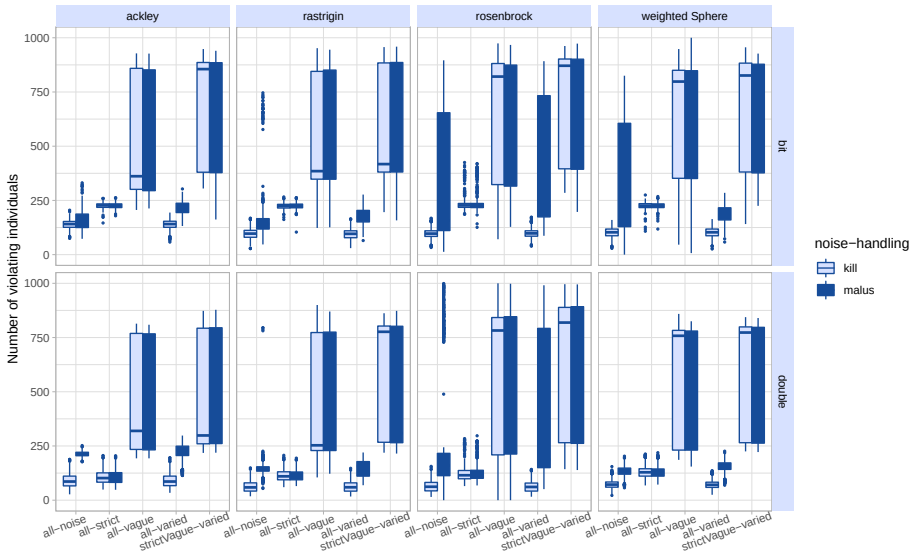

Fig. 2. Number of constraint-violating individuals for constraint $x_1 \geq x_0$ in a two-dimensional search space, x_0, x_1 representing the first and second dimension, resp. The columns show the results for the different benchmark functions, the rows differentiate the chosen encoding. For each configuration, there is one boxplot for the bit-encoding and one for the double encoding.

the same overview for the second constraint. Please note, that for *varied* configurations the first constraint was of type `strict` and the second of type `vague` or `noise`, resp.. We see again a higher variance for the *all-vague* case and occasionally also in the *all-varied* configuration with *malus* noise-handling. However, with the exception of these outliers, *all-varied* outperforms the other configurations. For both constraints, *all-strict* seems to perform better when employed with a double (real-valued)encoding instead of a bit-encoding. The difference in the results for both constraints stems partially from the number of variables in the constraints and additionally, because the second constraint receives the `vague` or `noise` technique in the *varied* configurations.

5.3 Discussion

With the above-made observations, we can discuss our research questions from above. Research Question 2 has already been answered by the last sentence above: It seems to have an influence, especially with constraints of type `strict`. This may be caused by the high variability of the bit encoding, as a single bit flip can have a huge impact on the offspring, whereas the Gaussian Mutator (employed for double encoding) instead produces offsprings in the neighbourhood. Therefore, double encoding is more respective of restrictive techniques. Research Question 1 asked for the influence of the configuration-possibility: For both situations, the *all-varied* case outperform or equals the others. Even—and this is interesting—when the constraint handling for the specific constraint is equal. However, the difference in the other constraint seems to have an effect through the recombination. This is particularly interesting, and we would like to further investigate this in the future.

6 Conclusion and further research

Evolutionary algorithms are beneficial as a tool for optimisation when the optimisation task has many local optima or the function is very complex such that direct approaches like Hillclimbing fail. This increased complexity often occurs when trying to "invert" machine-learned predictions. These situations are often precisely those that need a considerable amount of domain knowledge. We proposed using this domain knowledge to choose the proper constraint-handling technique for a given restriction, allowing different techniques to occur for different constraints. Furthermore, to cope with the fact that experimental setups are usually subject to noise, we defined noise-sensitive constrained-handling techniques based on the standard deviation of the present noise. We expanded our framework for evolutionary algorithms to check for noise sensitivity and extended a domain-driven data description language to adopt these changes. We evaluated our approach by comparing evolutionary algorithms where every constraint received the same treatment to those adapted based on the type of constraint. We found our approach to have the desired effect and intend to include more constraint-handling techniques into our framework for future work.

References

1. Coello Coello, C.A.: Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Computer Methods in Applied Mechanics and Engineering **191**(11), 1245–1287 (2002). DOI https://doi.org/10.1016/S0045-7825(01)00323-1. URL https://www.sciencedirect.com/science/article/pii/S0045782501003231
2. Deb, K.: Multiobjective Optimization Using Evolutionary Algorithms. Wiley, New York (2001)
3. Deb, K., Goel, T.: A hybrid multi-objective evolutionary approach to engineering shape design. In: Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization, EMO '01, p. 385–399. Springer-Verlag, Berlin, Heidelberg (2001)
4. Drechsler, R., Eggersglüß, S., Ellendt, N., Huhn, S., Mädler, L.: Exploring superior structural materials using multi-objective optimization and formal techniques. In: International Symposium on Embedded Computing and System Design (ISED), pp. 13–17 (2016)
5. Drechsler, R., Huhn, S., Plump, C.: Combining machine learning and formal techniques for small data applications - A framework to explore new structural materials. In: 23rd Euromicro Conference on Digital System Design, DSD 2020, Kranj, Slovenia, August 26-28, 2020, pp. 518–525. IEEE (2020). DOI 10.1109/DSD51259.2020.00087
6. Ellendt, N., Mädler, L.: High-throughput exploration of evolutionary structural materials. HTM Journal of Heat Treatment and Materials **73**, 3–12 (2018). DOI 10.3139/105.110345
7. Eysholdt, M., Behrens, H.: 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, OOPSLA '10, pp. 307–309. Association for Computing Machinery, New York, NY, USA (2010). DOI 10.1145/1869542.1869625
8. Michalewicz, Z., Schoenauer, M.: Evolutionary algorithms for constrained parameter optimization problems. Evol. Comput. **4**(1), 1–32 (1996). DOI 10.1162/evco.1996.4.1.1. URL https://doi.org/10.1162/evco.1996.4.1.1
9. Plump, C., Berger, B.J., Drechsler, R.: Domain-driven correlation-aware recombination and mutation operators for complex real-world applications. In: IEEE Congress on Evolutionary Computation, CEC 2021, Kraków, Poland, June 28 - July 1, 2021, pp. 540–548. IEEE (2021). DOI 10.1109/CEC45853.2021.9504931. URL https://doi.org/10.1109/CEC45853.2021.9504931
10. Riche, R.L., Knopf-Lenoir, C., Haftka, R.T.: A segregated genetic algorithm for constrained structural optimization. In: Proceedings of the 6th International Conference on Genetic Algorithms, pp. 558–565. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1995)
11. Steinbacher, M., Alexe, G., Baune, M., Bobrov, I., Bösing, I., Clausen, B., Czotscher, T., Epp, J., Fischer, A., Langstädtler, L., Meyer, D., Raj Menon, S., Riemer, O., Sonnenberg, H., Thomann, A., Toenjes, A., Vollertsen, F., Wielki, N., Ellendt, N.: Descriptors for high throughput in structural materials development. High-Throughput **8**(4) (2019). DOI 10.3390/ht8040022
12. Weicker, K.: Evolutionäre Algorithmen. Vieweg+Teubner Verlag (2015). DOI 10.1007/978-3-658-09958-9
13. Wilhelmstötter, F.: Jenetics. https://jenetics.io (2021)