NaCo 21/22 assignment report, group number

First and last name of each student in the group

Leiden Institute of Advanced Computer Science, The Netherlands

Abstract. This document contains the instructions and the format for the report required for submission of the practical assignment for the course Natural Computing.

1 Introduction

This document serves as a *description of the practical assignment* for the course Natural Computing in academic year 2021/2022. The second part of the assignment has now been added, which you should combine with your first part.

For part 1 of this assignment, we would like you to write a detailed description of the Genetic Algorithm (GA), and provide an implementation in Python. For this assignment you are asked to provide:

- implementation of a GA in plain Python,
- report, written and formatted as a *scientific paper* containing:
  - description of the GA,
  - description of your implementation,
  - literature review of the various search operators used in the GA,
  - literature review of an application of a Genetic Algorithm.

For part 2 of this assignment, you need to apply your GA to a problem based on a Cellular Automata (CA), and investigate how your GA performs when changing internal settings, such as the modules you discussed before. The use-cases you need to work on are described in Section 5 of this document.

To help you structure your report, we provide you with a *brief report outline* in this document. Please complete the following sections with your own results, explanations and conclusions. This includes the abstract and this introduction! For this section: introduce what the paper is about and provide a background to any relevant literature, such as the Genetic Algorithm [2].

2 Algorithm Description

Give a general overview of the working principles of a Genetic Algorithm (GA). For assignment part 1, you will implement your own version of a GA, and provide a description of your implementation in this report. Try to use clear mathematical formulations wherever possible. Make sure this description is *complete*: it should explain all core concepts, such that one should not need to refer back to the original papers to get the global picture of the working principals. For your

implementation, we would like you to focus on a *modular* design of the search operators (see Section 3 on what is meant by this).

In the report, we would like you to describe at least two variants found in the literature for each of the following operators; *recombination*, *mutation*, and *selection*. We also would like you to discuss two different ways a candidate solution can be represented in the GA in the section on *representation*. You do not need to implement all these operators in your code[1], but you are encouraged to try them out to gain a better understanding. In your implementation, it should be possible to replace the operators with any of the variants from the literature with minimal changes to the code.

For each of these three operators, in subsequent subsections, provide an overview on the general methodology of the operator and explain how and why it can be used in the GA. For each of the two operator variants you found in the literature, describe it and how it works. In addition, provide a description about the differences between each operator variant, and discuss when it might be beneficial to use one or the other.

2.1 Recombination

(see above)

2.2 Mutation

(see above)

2.3 Selection

(see above)

2.4 Representation

Give a description on how search points are represented in a GA, i.e. describe the genotype of the canonical GA. Additionally, describe alternative forms of representation, and discuss how you would implement this in your algorithm [2].

2.5 Notation

Make sure to follow the following notation conventions:

- n: The dimensionality of the search space
- M: Number of individuals in set/array
- $\mathbf{x} = (x_1, x_2, \ldots, x_n)$: A solution candidate.

[1] You do need at least one version of each operator implemented to be able to run your algorithm.

[2] i.e. the representation will be binary, but how would changing this, to for example a discrete representation impacts the algorithm

- $\mathbf{x}_i$: Solution candidate i (for $i \in \{1 \dots M\}$) in the set/array
- $f(\mathbf{x}_i)$: Objective function value of $\mathbf{x}_i$ ($f : \mathbb{Z}^n \to \mathbb{R}$)
- $\leftarrow$: Assignment operator
- $\mathcal{U}(\mathbf{x}^{\min}, \mathbf{x}^{\max})$: Vector sampled uniformly at random. Here it is 'U' for uniform. For other distributions, use for example $\mathcal{N}(0, 1)$ for a single number sampled according to the *normal* distribution with mean 0 and variance 1.

If you need to use any other notation, please be consistent and clearly define your added notation. In case of doubt, feel free to contact the TAs.

3 Implementation

You should implement your algorithm in Python, using the IOHexperimenter [1] package. A skeleton implementation is provided on Brightspace as an attachment to this project description in the file `implementation.py`. You can follow the steps in the provided README to install the required dependencies and run the test functions. The skeleton implementation provides an example of running the random search algorithm on a OneMax objective function. You can use this for testing your GA as well. In case of issues, please do not hesitate to contact the TAs for assistance.

When coding your algorithm, make sure to not use any hard-coded values. Instead, all of the parameters you use should be modifyable by the user. Include a table in your report here, describing all parameters of your implementation, their default values and their range of accepted values.

As mentioned earlier, we want you to focus on a *modular* design of the search operators. What we mean by this, is that we would like you to be able to switch out a given operator variant for another, without breaking parts of the program and with minimal changes to your code. With this we do not mean replacing a *mutation* operator by a *recombination* operator, but replacing a *mutation* operator for another *mutation* operator, which performs mutation in a different fashion. In addition, your implementation should consider alternative forms of *representation* for the search candidates.

To verify whether your algorithm works correctly, call the function `test_algorithm` provided in the notebook. **Passing this test is required for submission**, so if your algorithm fails this test, please contact the TAs.

When doing experiments with your GA during part 2, you should make sure to explicitly mention the used parameters (such as population size) in your report, for example by creating a table in this section.

4 Application of GA in practice

Search in the literature for one paper with an application of a GA outside of the field of computer science. Minor students are encouraged to take the lead in writing this section. Summarize the paper and describe the application, and add any relevant literature. In this section, be sure to answer at least the following questions about the paper:

- How was the GA applied? Describe the field and context.
- Why did the researchers choose a GA for their application, and where there any alternatives?
- Was their approach successful? Interpret their results.
- Give your opinion on their approach. Would you have you have used a GA in their situation?
- How would you improve on their setup?

5 Part 2: Cellular Automata

For the second part of the assignment, you will need to implement a basic 1-dimensional Cellular Automata (CA) of size N (non-connected boundaries, fixed to 0), which can follow a transition rule Θ, with a neighbourhood size $r = 1$. You should create a function to run your CA and provide an output state after T time steps for a given input state. The number of possible values k at each site for your CA should be flexible. We will use both $k = 2$ (the binary case shown in the lecture, with $a_t^t = \{0, 1\}$) and $k = 3$ ($a_t^t = \{0, 1, 2\}$) in this assignment. Note that you should implement a CA from scratch, following the explanations of CA in the slides.

With this CA, you will try to solve a version of the *Inverting Problem*: find a state C^0 such that when you use it as the initial state of your CA, after T time steps you get the given state C^T as output. You will do this for a number of different CA configurations, which will be given in the supplementary material. In there, you are given the final state C^T, number of time steps T, and transition rule Θ to use for each configuration. Note that both $k = 2$ and $k = 3$ are considered in these configurations.

Your implementation should be in Python, since you will also use your implemented GA in this assignment. In a similar fashion as with the GA in part 1, you should provide a description in your implementation of the CA in your report.

6 Part 2: Solving the Inverting Problem

In order to find this input state C^0 for a given output state C^T, you will use the GA you described in part 1.

The individuals in your GA will correspond to variations of input states $C^{0'}$ for your CA. Note that for a 2-state CA you can use the default binary operators, but for a 3-state CA you should make sure to modify your GA such that it can handle this different representation! Note: for this second part of the assignment, you will also have to *implement* the two versions of each of the GA operators you described in the first part of the assignment (mutation, recombination, selection).

To run your GA, you need to define an *objective function* for the inverting problem given a configuration of your CA. This should be function which calculates a similarity between a given input state $C^{0'}$ as suggested by the GA, and the actual input state C^0 of the CA. Since many different implementations are possible, you are asked to describe and implement *two different objective functions* here, and compare the performance of your GA on these two versions of the problems. In the file `objective_function.py` in supplementary material, an example is given of

creating your own objective function using the `ioh` framework, in addition to running a simple experiment.

In the file `ca_input.csv`, your input is given. The file contains 10 states and rules for your CA. Note that in addition, the number of time steps T to run a given rule and state combination is also given. You should process all these 10 CA states in your experiments, for both of the two aforementioned objective functions, giving you effectively 20 experiments to conduct. Moreover, you should collect data for every set of operator combination, in order to properly analyze which set of operators works best on which problem[3].

To analyze the performance of your GA, you should analyze the data you collected with `ioh` in the `IOHAnalyzer`. In this report, include the performance curves of your GA. Make sure to clearly compare the behaviour of your GA to a random search method, and look at the effect of changing the objective function and switching out different operators. Describe the differences in operator performance, clearly explain which operator and objective function combination should be considered as best, and try to reason about why a certain operator combination would be beneficial for a given situation.

7 Part 2: Application of CA in practice

Search in the literature for one paper which makes use of a CA outside of the field of computer science. Minor students are encouraged to take the lead in writing this section. Summarize the paper and describe the application, and add any relevant literature. In this section, be sure to answer at least the following questions about the paper:

- What kind of CA was used (dimensionality, states, transition rules,...)?
- Describe the field and context.
- Why did the researchers choose a CA for their application, and where there any alternatives?
- Was their approach successful? Interpret their results.
- Give your opinion on their approach. Would you have you have used a CA in their situation?
- How would you improve on their setup?

8 Conclusion

Write a short conclusion summarizing the most important findings of this assignment. Be sure to update this with the findings from part 2.

A Appendix

A.1 Submission, review and grading

Submission of the final version of the assignment should be done on Brightspace, and should include your report in *PDF* format and your code in *a single zipfile*. Make sure that the code is

[3] For these experiments, you should use a budget of 10000 evaluations, and do 5 independent runs on each function.

readable: clear variable names, comments in English, etc. Make sure to incorporate this feedback you have received for part 1 in the submission for part 2, as this is only aimed to help you improve your final report.

In addition to submitting via Brightspace, you should also submit your pdf via *EasyChair*, which is the platform we will use for peer reviews. Instructions on submitting via Easychair and on the peer-review process will be provided in November 2021.

References

1. Doerr, C., Wang, H., Ye, F., van Rijn, S., Bäck, T.: Iohprofiler: A benchmarking and profiling tool for iterative optimization heuristics. CoRR **abs/1810.05281** (2018), http://arxiv.org/abs/1810.05281
2. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI (1975), second edition, 1992