CS4205 - Evolutionary Algorithms

Assignment 2

This document describes the final practical assignment of the TU Delft course Evolutionary Algorithms (CS4205). This assignment will count for 30% of the final grade and can be done on one of the different topics listed below, each with a respective supervisor:

- *Gray-Box Optimization for the MaxCut Problem* (Single-Objective Discrete Optimization)
  Supervisor: Anton Bouter
- *Evolutionary Van Gogh* (Single-Objective Discrete Optimization)
  Supervisor: Georgios Andreadis
- *Optimization for the TSP & QAP problems* (Single-Objective Permutation-based Optimization)
  Supervisor: Arthur Guijt
- *Circles in a Square* (Single-Objective Real-Valued Optimization)
  Supervisor: Renzo Scholman
- *Boots on the Moon* (Genetic Programming)
  Supervisor: Marco Virgolin

A more detailed description for each of these assignments is provided in the appendix.

The basic principles of the assignment are the same across the topics. This assignment is to be done in *groups of 4 students*. You are supposed to self-arrange into groups, there is a discussion forum for group finding on Brightspace to help you with this. Once you have a group, each of you needs to sign up on the same group in the Brightspace group interface. We will assign the topics to groups using a preference-maximization approach. For this, prior to the start of this assignment, each group of 4 students must denote their relative preference for each of the 5 possible exercises in the Brightspace survey. The deadline for signing up for a group *and* for indicating your topic preference is **Thursday, May 19, 2022, 23:59**.

Requirements

The following holds for all project topics. For topic-specific requirements, see the respective topic document or ask the topic supervisor.

Each group must hand in the following *deliverables* on Brightspace in a single zip file prior to the deadline of **Friday, June 24, 2022, 23:59**:

- All source code in a *zip* file used to produce experimental results described in the report.
- A report in a *pdf* file written using the ACM conference template (also used for GECCO, the premiere conference on evolutionary algorithms). LaTeX and Word templates are provided on Brightspace. The following additional conditions apply:
  - *Do not change* the *font size*, nor the *margins* in the template.
  - You can use *at most 8 pages* (including figures and tables).
  - We expect your report to include at least the following sections:
    1. Introduction (*describes the motivation and goal of this assignment*)
    2. Background (*describes the problem and the baseline algorithm used to tackle the problem*)

3. Proposed Improvements (*improvements should be clearly motivated by hypotheses of what possible limitations of the baseline are*)
4. Experiments (*starts with describing the setup, proceeds with showing results in the form of tables and/or figures*)
5. Discussion and Conclusions (*discusses the results & concludes*)

Grading

A **complete** submission of the deliverables is required and the source code should run (else, it is a fail). The final grade is based on the following criteria. For each of these criteria, examples are given of aspects we consider to be important. Note that the grading criteria are not limited to these examples.

- Weekly sub-deliverables [**10% of grade**]
  - Each sub-deliverable is graded as a pass or fail.
- Motivation and Background [**10% of grade**]
  - Clear description of the problem.
  - Clear description of the motivation and purpose.
- Soundness & Originality of Proposed Improvements [**20% of grade**]
  - Describe the improvement.
  - Argue for why the proposed improvement would improve performance.
  - Originality of the proposed improvement.
- Quality of Experiments [**30% of grade**]
  - Formulate hypotheses and connect them to your experiments.
  - Ensure reproducibility of results.
  - Ensure and motivate fairness of comparisons you make.
- Analysis and Discussion of Results [**30% of grade**]
  - Critically interpret and reflect on results.
  - Lay out potential future steps.

CS4205 - Evolutionary Algorithms

Assignment 2 - Gray-Box Optimization for the MaxCut problem

Supervisor: Anton Bouter (bouter@cwi.nl)

1 Problem Definition

The Weighted Maximum Cut (MaxCut) problem is a well-known NP-complete graph problem. The objective of this problem is coloring each vertex in the graph either blue or red, and to maximize the total weight of edges between vertices of different colors. In a sense, each edge going from a red to a blue vertex, or vice versa, is cut. This is shown in Figure 1, where the total objective value is equal to 13.

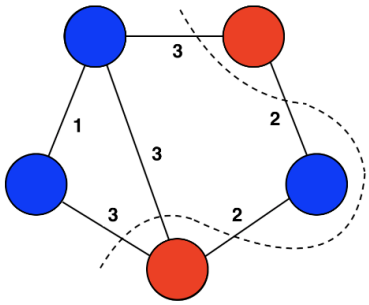

Figure 1: An example instance of the MaxCut problem.

Formally, the MAXCUT problem can be defined as follows: Given an undirected graph $G = (V, E)$, where each edge $e \in E$ has an assigned weight $w(e)$, we are interested in finding a partitioning of V into two subsets, $S \subseteq V$ and $\bar{S} = V \setminus S$, such that the total weight of edges between different partitions is maximized.

Many different graph structures are possible, e.g., sparse, dense, fully connected, planar, grid, or torus graphs. The structure of the graph can greatly influence how difficult it is to optimize, and the performance of different algorithms, or different crossover operators, is expected to vary between graphs with different structures.

EAs generally consider an optimization problem from a Black-Box Optimization (BBO) perspective, where nothing about the structure of the optimization problem is known. The MAXCUT problem can however be viewed from a Gray-Box Optimization (GBO) setting, where the structure of the graph is known, and problem knowledge can be used to improve the efficiency of the optimization. Despite this optimization setting, the MaxCut problem remains NP-complete and is therefore a non-trivial problem.

In the GBO setting that we consider in this assignment, it is possible to query the problem instance to retrieve the degree (i.e., number of connected edges) of any vertex, and a list of all its connected edges. Furthermore, this GBO setting allows for partial evaluations in order to relatively efficiently update the fitness of solutions that have been partially modified. This can be done by only considering the neighboring edges of vertices that are modified by a variation operation. For example, if in the graph displayed in Figure 1 a variation operation would change the rightmost blue vertex to a red vertex (and not change the neighboring red vertices), the two adjacent edges would no longer be cut, and the cut of the solution would be reduced by 4.

Furthermore, the performance of a standard Genetic Algorithm (sGA) greatly depends on the choice of the variation operator and how well this operator matches the structure of the problem

instance. Therefore, in a GBO setting, it may be possible to dynamically adapt the variation operator based on the available problem-specific knowledge.

2 Goal of the Assignment

The main objective of this assignment is to adapt the sGA to a GBO setting for the MaxCut problem, e.g., by designing a custom crossover operator, and analysing the performance of the GBO improvements compared to a BBO baseline. For this purpose, the following are the main research questions to be answered in this assignment:

- For which type(s) of MaxCut instances is it beneficial to use domain knowledge provided by the GBO setting?
- How can the performance of the sGA be improved in a GBO setting, through the use of problem-specific knowledge, for these type(s) of instances?
- How does the performance of the sGA for GBO compare to that of the sGA for BBO?

3 Materials

For this assignment, code of the sGA in Python, including traditional crossover operators and the MaxCut problem, will be provided to you. MaxCut problems instances are divided into sets A through D, with each set having problem instances of a similar structure with an increasing number of problem variables. The first line of each problem instance lists the number of vertices and the number of edges, respectively. Each of the remaining lines lists two vertices (1-based index) that are connected by an edge, and the weight of this edge, respectively. The optimum of each problem instance is given in the file with the .BKV extension. These values are automatically read in the code, and finding a solution with this fitness value will terminate the sGA. Hint: the structure of a graph can easily be visualized using an online graph visualization tool (https://csacademy.com/app/graph_editor/).

4 Deliverables

- **Week 1:**
  - Present initial ideas for selection of MaxCut instances to be considered.
  - Present initial ideas for GBO improvements.
  - Show some initial results for baseline sGA.
- **Week 2:**
  - Present some initial results for GBO improvements, including graphs and/or tables.
- **Week 3:**
  - Draft of Introduction and Background sections.
  - Outline of report.
  - Overview of intended experiments.
- **Week 4:**
  - No deliverables.
- **Final report:**
  - See guidelines in main document.

5 Related Work

- **Genetic Algorithms** (also part of mandatory reading material): [Whi94] up to (excluding) Chapter 4.1.
- **Gray-box Optimization:** [BAB21] Chapter 5.

References

[BAB21] Anton Bouter, Tanja Alderliesten, and Peter AN Bosman. Achieving highly scalable evolutionary real-valued optimization by exploiting partial evaluations. *Evolutionary Computation*, 29(1):129–155, 2021.

[Whi94] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

CS4205 - Evolutionary Algorithms

Assignment 2 - Evolutionary Van Gogh
Supervisor: Georgios Andreadis (G.Andreadis@lumc.nl)

Figure 1: Vincent van Gogh's "Wheat Field with Cypresses" (1889). *Google Art Project.*

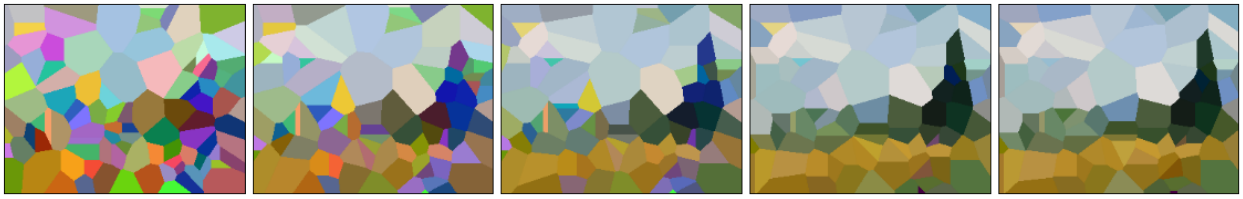

Figure 2: One run of a Genetic Algorithm imitating Figure 1 with Voronoi cells. Shown are the best imitations at various generations.

1 Problem Definition

Genetic Algorithms excel at solving many optimization problems, but in this assignment we will be considering a rather unusual one: imitating art. The optimization problem here is to find an approximation of an existing painting with a given (limited) set of primitives. In order to be able to "draw" shapes and therefore mimic an existing painting, we will be using Voronoi diagrams. To make a Voronoi diagram, we start by placing a number of points in an image space, each point having a color associated with them. We then color the space as follows: For each pixel in that image, we give it the color of the closest point (by Euclidean distance). This partitions the space into regions (also called *cells*) around each point having the associated color of that point.

With this drawing strategy, we let the GA evolve imitations of the painting, using the conventional mechanisms of a GA (mutation, recombination, selection). A particular Voronoi diagram instance is encoded in an individual's genotype as follows: Per point, a string of 5 integers (x, y, r, g, b) is stored describing the point's position ((x, y), ranging from 0 to the image's width or height -1) and color ((r, g, b), ranging from 0 to 255). These point information strings are concatenated to form a long string with all the points of the diagram, forming the genotype.

For the fitness of an individual, we need to establish how well it imitates the reference painting. We use `imgcompare`[1] for this, which subtracts the corresponding pixels in the imitation and the reference image from each other, and then assigns a score to this difference image (lower is better). Given this fitness, we use P+O tournament selection with size 4 to form the next generation.

2 Goal of the Assignment

In this assignment, we want you to experiment with Genetic Algorithms through a hands-on example. You will start out with an existing implementation of a GA imitating Van Gogh's "Wheat Field with Cypresses" oil painting (Figure 1) and extend it with your own features, to gain an understanding of how it works and to potentially even improve it. You will also conduct a set of experiments to test your hypotheses about why these features (do not) work. Note that we are not looking for a different way of drawing – we want to make improvements to the underlying components of the GA that lead to measurable improvement in the GA's performance. A good first step can be to consider here is the current Black-Box Optimization (BBO) approach, and whether elements of Gray-Box Optimization (GBO) can be used.

3 Materials

You are provided with a working version of the algorithm, using *one-point* crossover and a *random* initialization of points across image space. We want you to extend this implementation as described below. For this, you can use the Jupyter Notebook located in the provided ZIP file, which includes code to plot the progress of optimization over time and to display intermediate images.

Please execute all experiments with a point budget of either **50** or **100** points and a generation budget of **500**. A run can take 3-5 minutes, so take this into account when planning your experiments. You can experiment with available hyperparameters, such as different population sizes and mutation strength.

4 Deliverables

The sub-tasks for this assignment are divided into the weeks until submission. This is done to avoid overload towards the deadline and to help get the most out of the weekly supervision meetings in weeks 1-3. Not meeting the deliverables of a week will lead to point deduction. Feel free though to work ahead, the suggested weekly deliverables are the baseline. Extra work, such as more creative ideas/experiments, will be rewarded accordingly.

4.1 Weekly

Week 1

- Get the provided code installed and running on your machines.
- Prepare potential ideas for improvement to the GA.

Week 2

- Implement at least one new feature.
- Evaluate why it works or does not work, based on plots that you can show us at the meeting. We expect you to not only tell us what you observe but to also *explain* why you think you observe this.
- Prepare a full set of (at least) 3 hypotheses that you want to test, corresponding to different feature ideas and hyperparameter settings you want to try. Make sure at least 2 of these hypotheses consider features you have developed.

[1] https://pypi.org/project/imgcompare/

Week 3

- Hand in a completed first section of the report, as well as the layout of the rest of the report, with a plan of what experiments will be performed and intended main results and how these will be plotted.
- Proceed with implementation and experiments.

Week 4 No deliverables this week, to allow you to focus on the exam :)

Week 5 Submit the final deliverables (code and report).

4.2 Final

We expect you to hand in *all code* made in the course of the assignment, as well as a detailed report on your findings. Please see the general document for detailed requirements on what the report should contain.

5 Related Work

1. Darrell Whitley. A genetic algorithm tutorial (up to Chapter 4). *Statistics and computing*, 4(2):65–85, 1994. *(Also part of the mandatory reading material.)*
2. Sebastian Proost. Minimalist Art Using a Genetic Algorithm. 2020. `http://blog.4dcu.be/programming/2020/02/10/Genetic-Art-Algorithm-2.html`

CS4205 - Evolutionary Algorithms

Assignment 2 – Optimization for the TSP & QAP problems

Supervisor: Arthur Guijt

1 Introduction

When using Evolutionary Algorithms, one particularly important aspect of applying such an EA is the choice of **encoding**. In many cases the encoding used is trivial, for example a mapping from positions on a discrete string to the variables used within evaluation, or similarly for continuous variables. Such encodings are commonplace, and you have already (potentially unknowingly) encountered them.

The right encoding can however provide significant benefits to the performance of an EA! Specific encodings can be used to avoid infeasible solutions by construction, bias the search space towards regions of interest, or allow for better & more effective recombination. In this assignment we utilize various encodings (and crossover operators) to investigate their applicability to a particularly common (yet constrained) search space: Permutations.

Permutations are used to define sequences, orderings of preference, or unique one-to-one mappings. If you ever had a problem that takes such a input, we generally use a permutation to describe them. The trivial way to 'encode' permutations is a discrete search space, assigning a value 1 through the string length for each position. This approach does however not deal with the uniqueness constraint. The search space therefore also includes invalid solutions.

In this assignment we instead investigate different approaches, each with their own benefits and downsides. One of these approaches is to require a solution to always be a valid permutation, requiring our operators to be modified to preserve this property. Alternatively, we use the Random Key encoding which uses a string of continuous variables as encoding, with the encoded permutation being the order that sorts them.

2 Problems

Within this assignment you will be using the following problems to analyze how the performance of an operator/configuration relates to a problem.

2.1 Travelling Salesperson (TSP)

Imagine being a delivery driver. Given a set of places you want to deliver a package to, you would likely want to minimize the distance travelled in order to deliver these packages. Every package needs to be delivered once, and we need to return where we started. Given a distance matrix D, and an ordering of deliveries s of length ℓ, the total distance travelled (to be minimized) is as follows:

$$d(s) = D_{s_\ell, s_1} + \sum_{i=1}^{\ell-1} D_{s[i], s[i+1]}$$

For this problem we recommend taking a look at TSPLIB [2] for instances.

2.2 Quadratic Assignment Problem (QAP)

A different kind of problem, which can also be represented by a permutation is the Quadratic Assignment Problem (QAP). Rather than an ordering the permutation in the QAP, as the name implies, is a one-to-one assignment. Specifically, given two matrices A and B, find an assignment s of length ℓ such that the cost (to be minimized) is as follows:

$$c(s) = \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} A_{i,j} B_{s[i],s[j]}$$

To provide an idea how this problem can be used in practice, take a look at an example from literature is the layout of a keyboard[4]: you only have as many keys as letters, and need to assign each letter to a key (or: each key to a letter). This problem, and corresponding instances, can be found within QAPLIB [1].

3 Assignment Outline

For this assignment the primary goal is to investigate the design of an Evolutionary Algorithm for permutation based problems. How much does the structure of the problem impact the EA, and what adjustments can you make to make the EA work better on a problem? And how much does this change impact the behavior on another?

4 Materials

You are provided with source code for both a permutation-preserving GA (with OX [5], PMX [6], CX [8] already implemented) and Random Keys [3] DE [9]. Instances from TSPLIB [2] and QAPLIB [1] are recommended and can be downloaded automatically using a provided notebook (which contains instructions similar to those found in this document).

Other reading material of interest, that may inspire you (but is not required reading):

- Want to use Random Keys in a different way, [7] discusses some changes one can make to the Random Key encoding.
- Prefer working with Permutations directly? In [10] other ways to perform crossover for permutations are discussed.

5 Deliverables

In order to ensure the entire workload is distributed somewhat nicely over the available weeks, the following schedule should be adhered to, and doing so will provide a small amount of points for the grade of this assignment, as seen in the grading section of the general document. The weekly meetings will serve as a check of whether the deliverable was completed (besides asking questions, discussing ideas, etc.). ***Important!*** *Be prepared to show the weekly deliverable during the weekly meeting, e.g. bring your laptop, show running code, show graphs, bring written notes of ideas! Points will only be awarded in cases where the deliverables can be shown.*

Week 1 Pick the instances that will be used for the remainder of the assignment. Don't forget to argue your choice! (Hint: Keep in mind that instances with large ℓ may be significantly more difficult to solve.). Pick a baseline configuration and get it running. Prepare *potential* ideas for improvement (what actions could you take to improve the algorithm?)

Week 2 Evaluate how the baseline performs on the chosen problem & instances. Some results should be ready-to-show and discuss. Furthermore, propose how to improve the algorithm & have something new implemented (e.g. the proposed improvement, or: something that was tried out).

Week 3 Introduction section of report has been written (incl. motivation, background), general layout (sections) as well as a general idea of what experiments will be performed & intended main results (and corresponding plots).

Week 4 Exam Week - No deliverables for the assignment this week. Prepare for the exam instead!

Week 5 Submission: final (non-weekly) deliverable! Don't forget: for this assignment you will all have to write a report together discussing what you have done, and how, why, etc. Read the general document for the minimal sections that it needs to contain, and how things will be graded.

References

[1] QAPLIB Home Page. http://www.mgi.polymtl.ca/anjos/qaplib/.

[2] TSPLIB. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

[3] James C. Bean. Genetic Algorithms and Random Keys for Sequencing and Optimization. *ORSA Journal on Computing*, 6(2):154–160, May 1994.

[4] R. E. Burkard and J. Offermann. Entwurf von Schreibmaschinentastaturen mittels quadratischer Zuordnungsprobleme. *Zeitschrift für Operations Research*, 21(4):B121–B132, August 1977.

[5] Lawrence Davis. Applying adaptive algorithms to epistatic domains. In *IJCAI*, volume 85, pages 162–164, 1985.

[6] David E Goldberg, Robert Lingle, et al. Alleles, loci, and the traveling salesman problem. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, volume 154, pages 154–159. Lawrence Erlbaum, Hillsdale, NJ, 1985.

[7] Pavel Krömer, Vojtěch Uher, and Václav Snášel. Novel Random Key Encoding Schemes for the Differential Evolution of Permutation Problems. *IEEE Transactions on Evolutionary Computation*, 26(1):43–57, February 2022.

[8] IM Oliver, DJd Smith, and John RC Holland. Study of permutation crossover operators on the traveling salesman problem. In *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA*. Hillsdale, NJ: L. Erlhaum Associates, 1987., 1987.

[9] Rainer Storn and Kenneth Price. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11(4):341–359, December 1997.

[10] L Darrell Whitley, Timothy Starkweather, and D'Ann Fuquay. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In *ICGA*, volume 89, pages 133–40, 1989.

Solving Circles in a Square

CS4205: Evolutionary Algorithms
Supervisor: Renzo Scholman
Topic: Real-Valued Single-Objective Optimization

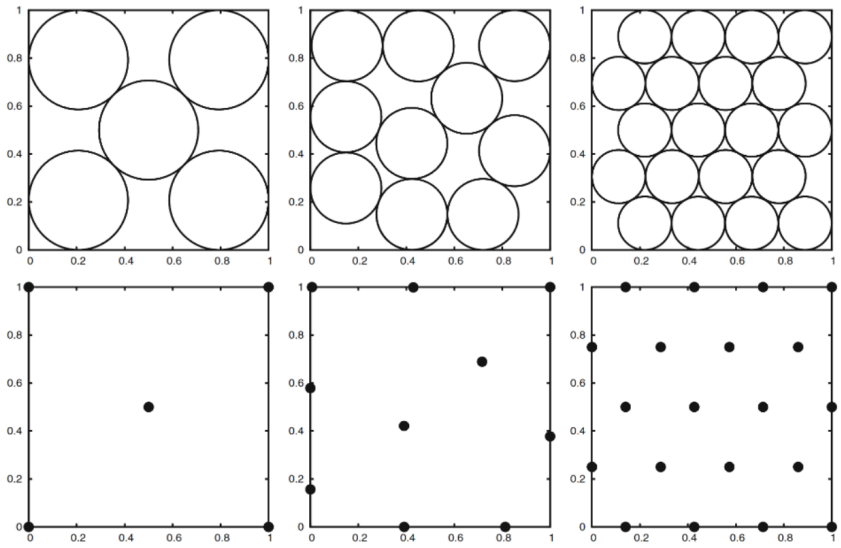

Figure 1: Top row: optimal circle packings for 5, 10, and 20 circles. Bottom row: corresponding point scatterings [1].

1 Problem Definition

This final assignment considers a scalable and real-valued optimization problem called "Packing n Circles in a Square (CiaS, see, e.g., [1])." This problem has many applications in real-world optimization problems like loading containers, communication networks, facility dispersion/layout, etc. Even though CiaS can be compactly formulated and efficiently evaluated, it is not trivial to solve and has interesting characteristics that differ from many other commonly used benchmark problems. For at least 2 circles, the CiaS problem is equivalent to finding an optimal scattering of points in the unit square. To obtain the optimal scattering of points, the smallest distance between any pair of points has to be maximized. Examples of optimal circle packings and their corresponding point scatterings are shown in Figure 1. The fitness function for the point scattering problem is as follows:

$$f_{scatter}(\boldsymbol{x}) = \min_{0 \leq i < j < \ell/2} ||(x_{2i}, x_{2i+1}) - (x_{2j}, x_{2j+1})||$$

2 Goal of assignment

To obtain state-of-the-art performance on CiaS is not trivial, as specific properties of the CiaS problem must be considered and exploited. It would be interesting to see to what extent EAs can be tuned to CiaS to also obtain state-of-the-art results with an EA. You will be doing so by improving one of the provided EAs and perform a performance comparison before and after it has been tuned for CiaS.

An especially useful resource is the Packomania website [2]. On this website a large list of the optimal, or best known, packings is maintained. Currently, the list goes up to 9996 circles, with some intermittent omissions. You can also find other references and resources for more information on the website.

3 Provided algorithms

In order to give you a head start with some readily available code, the implementations for two algorithms are provided. Both have been implemented in a different language in order to give you a choice that suits your preference. The two algorithms are:

- AMaLGaM [3] (C code)
- Evolution Strategies [4] (Python code)

4 Weekly Deliverables

The deliverables are divided over several weeks. This allows you to progressively work on the assignment and get guidance in the meantime. Furthermore, it will prevent you from being overloaded with work and unable to ask any questions in the last week. A deliverable is specified for each of the first three weeks, with a final deliverable in the form of code and report in the last week. You are to show the weekly deliverables to your supervisor, which can then assess if it has been met and help with any questions or discuss ideas. Note that you have to show either working code, graphs/tables or written out ideas for the deliverables. Just mentioning that you have done something without bringing anything to the weekly meetings will not be accepted and will count as a fail for that particular weekly deliverable.

Week 1. Your assignment is to improve the baseline performance of an EA and report the findings. To measure the baseline, you are to benchmark the EA as provided and produce graphs or tables with the results. You should also make yourself familiar with the code and think of ideas for the improvement that you are going to make (see week 2). This will be also your deliverable for the first week, i.e., propose the changes your are going to make to the algorithm and show the baseline performance.

Week 2. In the second week you should start with implementing the proposed changes in order to improve the performance of the EA on the CiaS problem. To give you an idea of what you should work on, you should change or work on at least the following three points for the EA:

1. Change the constraint handling technique by implementing either boundary repair [1] or the more advanced constraint domination [5].
2. Create a problem specific initialization scheme
3. Optimize the hyperparameters (population size, etc)

The deliverable for the second week is an implementation for the first two points mentioned above. Some preliminary results on whether they are working are also required. Hyperparameter optimization is something you should start with, but you have time to finish it in the other weeks as well.

Weeks 3. Starting in week three, you should begin writing the final report. The altered EA has to be benchmarked in order to evaluate if the performance has improved. If one of the changes did not work as expected, also try to explain why they did not. By doing so, you might be able to fix or improve the changes in the last two weeks after the final meeting with your supervisor. More specifically, the third deliverable for the last meeting in week 3 will be to have the first section of the report finished, i.e., the introduction, and show some preliminary results of the performance comparison. The introduction should introduce the general idea of the EA you use and how it can be used to solve the CiaS problem.

Weeks 4 and 5. In the fourth week you are advised to study for the exam, but in the fifth week you are to hand in the final deliverables (code and report). Thus, you should finish the benchmarks for the algorithm and complete the report. The report has to contain a clear comparison of the performance before and after the changes in order to show if the improvements had any success. The choice of the proposed changes must be motivated and the outcome of the comparison clearly conveyed. Make sure that the comparison is done in a fair and reproducible manner and that statistical significance testing has been performed to determine if the changes resulted in a significant improvement.

Related work (also to get you started)

[1] Peter A. Bosman and Marcus Gallagher. The importance of implementation details and parameter settings in black-box optimization: A case study on gaussian estimation-of-distribution algorithms and circles-in-a-square packing problems. *Soft Comput.*, 22(4):1209–1223, feb 2018.

[2] Eckard Specht. Packomania website, Feb 2022. `www.packomania.com` (accessed: 30.03.2022).

[3] Peter A. N. Bosman, Jörn Grahl, and Dirk Thierens. Benchmarking parameter-free amalgam on functions with and without noise. *Evolutionary Computation*, 21:445–469, 9 2013.

[4] Nikolaus Hansen, Dirk V. Arnold, and Anne Auger. Evolution strategies. In Janusz Kacprzyk and Witold Pedrycz, editors, *Springer Handbook of Computational Intelligence*, pages 871–898. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[5] Kalyanmoy Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2):311–338, 2000.

Boots on the Moon with Genetic Programming

CS4205: Evolutionary Algorithms
Topic: Genetic programming, Supervisor: Marco Virgolin (`marco.virgolin@cwi.nl`)

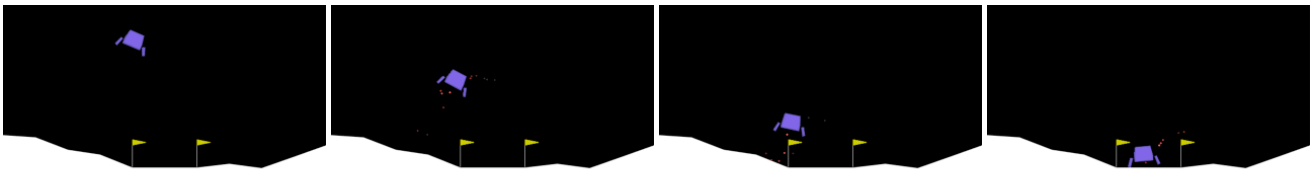

Figure 1: Example of behavior of an agent evolved with genetic programming that solves the *Lunar Lander* environment of OpenAI's `gym`. From left to right, the agent fires the side engines and the main engine of the lander (red dots show propulsion) at the right time to minimize fuel consumption and gently land between the two yellow flags.

1 Problem Definition

Reinforcement learning (RL) is the task of training an *agent* to perform a certain task in an *environment*. The RL problem in consideration here is the *Lunar Lander* (*LL*) environment of OpenAI's `gym`. The specification of *LL* is as follows (taken from `https://gym.openai.com/envs/LunarLanderContinuous-v2/`):

> *Landing pad is always at coordinates (0,0). Coordinates are the first two numbers in state vector. Reward for moving from the top of the screen to landing pad and zero speed is about 100..140 points. If lander moves away from landing pad it loses reward back. Episode finishes if the lander crashes or comes to rest, receiving additional -100 or +100 points. Each leg ground contact is +10. Firing main engine is -0.3 points each frame. Solved is 200 points. Landing outside landing pad is possible. Fuel is infinite, so an agent can learn to fly and then land on its first attempt. Action is two real values vector from -1 to +1. First controls main engine,* -1—0 *off,* 0—$+1$ *throttle from 50% to 100% power. Engine can't work with less than 50% power. Second value* -1.0—0.5 *fire left engine,* $+0.5$—$+1.0$ *fire right engine,* -0.5—0.5 *off.*

Further info: The state vector, or *observation*, should be used as **input** to the agent. This vector takes values in $\mathbb{R}^6 \times \{0,1\}^2$, with the elements representing: (1) horizontal coordinate, (2) vertical coordinate, (3) horizontal velocity, (4) vertical velocity, (5) tilt angle, (6) angular velocity, (7) Boolean flag left leg touches ground, (8) Boolean flag right leg touches ground. The action is **output** of the agent. The action must be a 2-dimensional vector, taking values in $[-1,+1] \times [-1,+1]$ (respectively, for main engine and side engines).

Important: *Note that the output of the agent must be a 2-dimensional vector but tree-based GP returns a single output (from the root)!*

2 Goal of the Assignment

The goal of this assignment is for you to obtain a better understanding about how genetic programming (GP) works, why it works that way, and how it can be used in another application than the one seen in the lecture (symbolic regression), namely, RL. Examples of aspects to consider for improving GP are: using a different representation, crafting new atomic functions, exploit symmetries of the environment, etc.

3 Materials

Documentation on the way *LL* is implemented is available at: `https://github.com/openai/gym/blob/master/gym/envs/box2d/lunar_lander.py` (refer to the *continuous* case). You are provided with a working GP library written in `python` available that is at `https://github.com/marcovirgolin/genepro`. You are further provided with a `requirements.txt` file that lists the libraries needed to implement your solution, as well as a Python notebook `solution.ipynb` that contains some utility functions. **All** the code you produce to tackle this assignment should be written in that notebook (unless a strong motivation for doing otherwise is brought forward and the supervisor agrees).

4 Deliverables

Deliverables are divided into weeks. This is done in your best interest, to ensure that you do not end up overloaded at the end. The weekly meetings with the supervisor will serve to assess whether the respective weekly deliverable has been met (besides asking questions, discussing ideas, etc.).

Important: *Bring evidence that you completed the weekly deliverable (e.g., bring the laptop and show running code, print-outs, written notes with ideas, etc.)! Statements like "We promise we completed this task but forgot to bring something to prove it" will* ***not*** *be accepted (and thus lead to a mark reduction).*

Week 1 deliverable:

1. Install the GP library `genepro` from `https://github.com/marcovirgolin/genepro`.
2. Install the `requirements` for this assignment.
3. Extend the provided `solution.ipynb` notebook to run GP to evolve an agent for *LL*. This does not need to evolve anything meaningful yet, the goal is just to have code where GP runs correctly.
4. Bring some initial ideas about how the provided GP library could be extended so that an *improved* agent could be achieved.

Week 2 deliverable:

1. Realize a *baseline* solution to the problem using the provided GP library, by running an evolution using a fitness function such that:
   (a) Maximization is sought
   (b) It evaluates a tree only if its root has at least two children (else, it returns $-\infty$).
   (c) It uses the output of the first child of the root to control the main engine.
   (d) It uses the output of the second child of the root to control the side engines.

(e) It returns the average reward obtained by running 5 episodes with maximum duration of 300 time steps.

2. Experiment with GP's setting (function and terminal sets, subtree crossover and mutation probability, etc.). The goal is to evolve an agent that performs decently, i.e., often lands on the landing pad and never crashes.

3. Bring more elaborated ideas (also taking inspiration from the scientific literature) about how the provided GP library could be extended so that an *improved* agent could be achieved. If possible, start implementing some of these ideas.
Important: *At this stage, each idea should be formalized into a motivating hypothesis (e.g., "Assuming that ... happens, then we expect an improvement by doing ...".*

Week 3 deliverable:

1. The section *Introduction* should be completed. This section should introduce what RL is, introduce what GP is, describe how GP can be set up to tackle RL, and introduce the *LL* environment.

2. Proceed with development obtain substantial first results on improving GP for *LL*. In connection to this, clearly identify what ideas did not work or seemed not to work, and which ideas show promise.

Week 4 deliverable: Nothing specific, proceed with the assignment and prepare yourself for the exam.

Week 5 deliverable: The completed assignment, uploaded on Brightspace (see the general guidelines).
Important: *For this assignement, the following applies to the final deliverable:*

- As mentioned before, all code should be contained in the notebook `solution.ipynb`.
- The notebook should also contain two (or more) GIFs: one showing the behavior of the baseline agent (i.e., the agent obtained by the baseline GP) and one (or more) showing the behavior of the improved agent(s) (depending on how many improvements you realize). See, e.g., `https://stackoverflow.com/questions/51527868/how-do-i-embed-a-gif-in-jupyter-notebook` for how to include GIFs in a notebook.

5 Related Work

1. Poli R., Langdon W.B., McPhee N.F., 2009: A field guide to genetic programming — Part II: Advanced genetic programming. (Available as reading material of lecture 3.)

2. `https://en.wikipedia.org/wiki/Reinforcement_learning`

3. Hein, D., Udluft, S. and Runkler, T.A., 2018. Interpretable policies for reinforcement learning by genetic programming. Engineering Applications of Artificial Intelligence, 76, pp.158-169.

4. Zhang, H., Zhou, A. and Lin, X., 2020. Interpretable policy derivation for reinforcement learning based on evolutionary feature synthesis. Complex & Intelligent Systems, 6(3), pp.741-753.