Ealain: A Camera Simulation Tool to Generate Instances for Multiple Classes of Optimisation Problem

Quentin Renau
Edinburgh Napier University
Edinburgh, Scotland, UK
q.renau@napier.ac.uk

Johann Dreo
Institut Pasteur, CSB lab.†
Paris, France
johann.dreo@pasteur.fr

Emma Hart
Edinburgh Napier University
Edinburgh, Scotland, UK
e.hart@napier.ac.uk

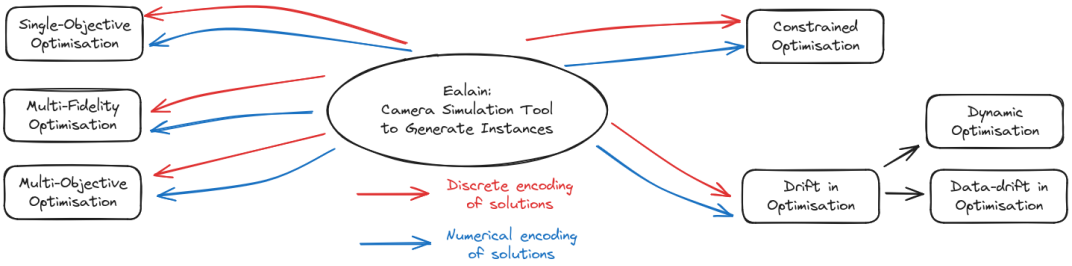

Figure 1: *Ealain*: camera simulation instance generator supporting two solution encodings and capable of generating instances for multiple classes of optimisation problems.

ABSTRACT

Artificial benchmark datasets are common in both numerical and discrete optimisation domains. Existing benchmarks cover a broad range of classes of optimisation, but as a general rule have limited value due to their poor resemblance to real-world problems, and generally lack the ability to generate arbitrary numbers of instances. In this paper, we introduce *Ealain*, an instance-generator that creates instances of optimisation problems which require placement of a number of cameras in a domain — this has many real-world analogies for example in environmental monitoring or providing security in a building. The software provides two types of camera-model and can be used to generate an infinite number of instances of black-box, real-world-like optimisation problems which can be single-objective, multi-objective, multi-fidelity, or constrained. The software is also flexible in that it also permits a range of different objective functions to be defined. Furthermore, generated instances can be solved using either a numerical or discrete encoding of solutions. The C++ library targets fast computation and can be easily plugged into a solver of choice. We summarise the key features of the *Ealain* software and provide some examples of the type of instances that can be generated for different classes of optimisation problems.

KEYWORDS

Instance-Generation, Black-Box Optimisation, Software Tools

ACM Reference Format:
Quentin Renau, Johann Dreo, and Emma Hart. 2024. Ealain: A Camera Simulation Tool to Generate Instances for Multiple Classes of Optimisation Problem. In *Genetic and Evolutionary Computation Conference (GECCO '24 Companion), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3638530.3654299

1 INTRODUCTION AND MOTIVATION

A broad range of benchmark instances have been proposed for different optimisation problems. They cover a large spectrum of applications in combinatorial optimisation [3, 11] and in numerical optimisation [6], as well as different categories of optimisation problems such as numerical black-box optimisation [6], discrete optimisation problems [13], etc. (see [2] for a complete list). While platforms such as COCO [6] and Nevergrad [10] have had a positive influence in the benchmarking field by gathering instances in one place, in other domains, instances are found at many different sources.

Despite the existence of many test suites and instance-generators, generally, both are customised to a particular domain and/or class of optimisation problem, resulting in a proliferation of different test suites. In contrast, we propose a new instance-generator that generates instances that accurately model characteristics of real-world problems and can be used to generate instances for five classes of optimisation problems: single-objective, multi-objective, multi-fidelity, constrained, and for generating drift: data-drift in optimisation streams where the drift at the instance level is correlated with an evolution of an algorithm performance and dynamic optimsation where the problem changes during the run of a solver. Furthermore, all generated instances can be solved using both a numerical or a discrete representation of solutions (Figure 1).

† Full affiliation for J. Dreo: *Institut Pasteur, Université Paris Cité, Bioinformatics and Biostatistics Hub, Department of Computational Biology, Computational Systems Biomedicine laboratory, F-75015 Paris, France.*

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

ACM ISBN 979-8-4007-0495-6/24/07.
https://doi.org/10.1145/3638530.3654299

The software is written in C++ and called *Ealain*[1]. The basic premise is a model of camera software that detects objects. This enables a 'camera-placement' optimisation problem where there are one or more objectives. There are many real-world examples or analogies of such problems: as a security system in a gallery; environmental monitoring via optimising sensor placements [15] or optimising radar placements [12].

Given the nature of the problem, *Ealain* generates real-world-like problem instances. As the objective function(s) are computed via simulation, it can be considered a black-box problem as it is not possible to obtain a formula representing each instance. The difficulty of instances can be controlled in multiple ways, for example increasing the number or type of cameras, increasing the size of the instance, or modifying the terrain covered by the instance, e.g. adding walls or obstacles that the camera cannot see through. All of these modifications affect the size of the search space, the dimension of the problem, or add ruggedness to instances. In addition, constrained instances can be generated by fixing locations at which a camera must or must not be present. Furthermore, the ability to control the precision at which a camera detects an object enables the problem to be solved as either a discrete or continuous optimisation problem. Given the variety of parameters and camera models, *Ealain* can thus generate a very large set of instances for a chosen optimisation class.

In the next section, we define the problem. For simplicity, we refer to the generic problem as the 'Art Gallery Coverage Problem' throughout the rest of the paper but it should be clear that this generic formulation covers a large number of potential applications.

Code availability: Code of the project is available at https://github.com/qrenau/Ealain

2 ART GALLERY COVERAGE PROBLEM

2.1 Definition

The Art Gallery *Coverage* Problem (AGCP) is derived from the Art Gallery Problem introduced in 1973 by Victor Klee [4] as a geometry decision problem. Given a number of guards, the original formulation aims at finding the optimal position of each guard such that every point in the gallery can be seen by at least one guard for a given layout. The Minimal AGP is a related optimisation problem in which the task is to find the *minimal* set of guards needed for complete coverage. Over the years, this problem has been widely studied in computational geometry with recent work on the complexity of the decision problem [1], given that most of the optimisation problem variants are NP-hard [8].

Ealain permits the generation of both the original version of the optimisation problem and a variant of the Minimal AGP which aims at maximizing the coverage by a *fixed* number of cameras. Rather than working with exact computational geometry tools, we compute a fast approximation of the probability of detection and the related coverage using a simulator which discretises the problem on a regular grid of pixels to improve efficiency.

Instance Parameters. The simplest version of the art gallery that can be generated with *Ealain* is an empty instance with no walls. Nevertheless, this instance is defined by two parameters: (1) its size s in meters; (2) its discretisation D which determines the size in pixels as s/D. Varying the instance size or the discretisation will result in different instance complexity.

Solution Encoding. Two types of encoding are supported in *Ealain*. Given that instances are discretised, discrete encodings can be used by a solver, for instance using a bitstring (with length equal to the number of pixels) to indicate camera locations or a list of n integers indicating the pixel location of each camera. A numerical encoding of solutions is also supported. For example, the real-valued coordinates (x, y) of each of n cameras can be used. At runtime, the real-valued coordinates are projected in the discretised space by *Ealain* in order to return a fitness value. Hence, both discrete and numerical algorithms can be used to solve this problem.

2.2 Instance Scenarios

The simplest scenario that can be generated in *Ealain* is an empty instance. The complexity of this scenario can be changed by modifying the size and the discretisation of the instance. For example, a bigger instance size and discretisation will directly impact the discrete encoding as the dimension of the problem increases.

The second scenario is the addition of walls. Walls can be added by specifying their coordinates. They modify the instance in two ways:

- By blocking the line of sight of cameras;
- By creating areas where cameras do not sense anything, i.e., cameras on the top of a wall do not sense anything.

The two last scenarios available are based on polygonal constraints. These constraints can be generated randomly or manually for each instance. We labelled these constraints 'Constraint Out' and 'Constraint In'. The 'Constraint Out' represents areas where cameras cannot be placed if the solution is to be considered feasible, whereas 'Constraint In' defines the opposite situation, i.e., areas where at least one camera should be positioned for the solution to be feasible. Each 'Constraint Out' or 'Constraint In' can be defined either for all cameras or a subset of cameras, i.e., one constraint can be generated for one particular camera but not for the others.

The method by which to penalise infeasible solutions is left to the user.

3 *EALAIN* MODULES

3.1 Camera Models

The coverage of a pixel in *Ealain* is linked to a probability of detection. A pixel is said to be covered if the probability of detection of that pixel is above a predefined threshold. This threshold is a parameter that can be changed and that will directly impact the number of covered pixels.

In order to cover the space, *Ealain* provides two built-in two-dimensional camera models. These models can already be used to create a wide range of problem instances. Note that it is also possible to easily extend *Ealain* to add custom two-dimensional camera models or to create three-dimensional camera models.

Both camera models are defined by a range r of visibility and this range is applied at 360 degrees around the position of the camera, i.e., the camera can sense everything 360 degrees for a distance r.

[1] From Scottish Gaelic, meaning *art*.

The probability of detection of *omnibinary cameras* p of is simple: $p = 1$ for every pixel within range and 0 otherwise. The probability of detection of *omnidir cameras* differ as p in this model decreases linearly over the range.

3.2 Camera Groups

In *Ealain*, when more than one camera are simulated, they have to be aggregated into groups. Groups define the way the probability of detection of a pixel by each camera is aggregated which is important when the same pixel is sensed by more than one camera. Examples of implemented groups are the following:

Additive: probability of detection of each pixel by each camera is summed;

Min: if two or more cameras detect the same pixel, the minimum probability of the group of cameras detecting it is assigned;

AtLeastOne: probability of having a least one detection across the group assuming independence of the camera detections.

It is possible to easily create new custom camera groups that are not already available within *Ealain*.

3.3 Cost Computation

Once all cameras are created and gathered within groups, the cost computation module computes the value of the configuration in the layout of the instance at hand. *Ealain* comes with a built-in cost computation which is the coverage. Given a threshold τ, the cost computation will output the number of pixels having a probability of detection greater or equal to τ. The coverage metric can be used to design objective functions to optimise. The design of an objective function is left to the user but an example is: given a fixed number of cameras, find the locations that maximise the number of pixels covered. As for camera models and camera groups, other ways to compute the cost on a given instance can be added.

3.4 Constraints

Implemented constraints in *Ealain* are polygon-based. It is possible to define one or multiple polygons in the instance and check whether a particular camera is located within a polygon. By default, the computation returns a Boolean and the distance to the border of the polygon. The processing of this information and the constraint handling method is left to the user. As other modules in *Ealain*, it can be easily extended to integrate other types of constraints.

4 GENERATION OF CLASSES OF OPTIMISATION PROBLEMS

In this section, we present the five classes of optimisation problems that can be generated using *Ealain*. For each class, an example code is provided that permits to create an instance and evaluate solutions on that instance. Each code can be launched using the same format that can be divided into three parts:

```
$ ./example_code instance_description solution
```

(1) *./example_code* is related to the problem class, i.e., one of the five described below.
(2) *instance_description* refers to parameters used to create instances, i.e., its size, the number of cameras, and the number of constraints (if available).
(3) *solution* refers to camera locations in the art gallery to be evaluated.

The specific launching command for each code can be found in the documentation and in the code file.

4.1 Encoding Solutions

For all the classes of optimisation problems described below, *Ealain* offers the possibility to encode solutions in two different ways, i.e., it will accept solutions encoded in a numerical or discrete form. For example, a solution in an instance of size $s = 3$ with two cameras with no constraints can be of the form:

- $[x_0, y_0, x_1, y_1]$ in an numerical encoding.
- $[0, 0, 1, 0, 0, 0, 0, 1, 0]$ in a discrete encoding where 1 represent the location of a camera.
- $[1, 2]$ as alternative discrete encoding where each digit in the list represents the discrete identifier of a pixel where a camera should be placed.

This permits the use of different classes of solvers.

4.2 Single-Objective Optimisation

Single-objective optimisation instances are the simplest class of instances that can be generated with *Ealain*. The simplest instances only need to generate a defined number of cameras (that can be from different types) and a cost computation that will output the fitness of a given configuration.

4.3 Multi-Objective Optimisation

Since *Ealain* can generate cameras with different ranges and different detection profiles, one possibility can be to add a *cost* to these cameras. Thus, an example multi-objective optimisation problem can be defined as minimising the cost of the cameras while maximising the coverage of the art gallery. Other formulations are clearly possible, for example, adding an objective that minimises unsatisfied constraints.

4.4 Constrained Optimisation

Constrained optimisation can be merged with other type of optimisation problems. For example, we can add constraints to a single-objective optimisation problem. As mentioned above (Section 2.2), constraints may be of two types: forbidden areas (no cameras should be positioned there) or mandatory areas (a camera should be positioned there). To enforce the constraints, the constraint handling technique is left to the user.

Generating constraints in a problem instance is very similar to generating that same instance without constraints. The only addition in the C++ code is the location of extreme points forming the edges of the polygon representing a constraint.

4.5 Multi-Fidelity

Every instance is generated specifying its discretisation, i.e., defining the number of pixels that will compose the instance. The impact of the number of pixels is two-fold: a larger number of pixels will

increase the computation time of the function but will also improve the precision of the estimation of the coverage. Hence, the same instance can be evaluated twice with different level of precision of the computation and different computation times.

4.6 Drift

Ealain can be used to generate a stream of instances that exhibit data-drift or have a dynamically changing fitness landscape (i.e. dynamic optimisation [9]).

Data-drift description: In many practical applications of optimisation, instances arrive in a continual stream. Algorithm selector models trained on historical data might fail to generalise over new data distributions: this is formally described as *data-drift*, defined as a change in the input data distribution of a stream of instances. As the direction of future data-drift is unknown and it is unrealistic to train general models in instances that cover the entire instance-space, it is therefore important to detect drift and react appropriately. This is a well-studied topic in machine-learning (ML) (see comprehensive surveys such as [5]) but has received little attention in optimisation. As a result, unlike in ML, there are no datasets to evaluate drift-detection in optimisation domains. An important characteristic of a dataset in this case is that there is a correlation between drift in the data distribution describing instances and the performance of a solver (or algorithm-selector). Although instances from existing optimisation benchmarks such as BBOB can be sorted in terms of performance to simulate *performance* drift, there is no guarantee this is a result of *data* drift.

Ealain can be used to generate controlled data-drift and therefore new benchmarks for evaluating ML methods for detecting it. This is achieved by adding 'Constraint In' locations to the definition of an instance. As the number of constraints increases, the performance of numerical solvers decreases relatively to the optimal solution.

Hence *Ealain* can be used to design new data-drift test suites for optimisation. Unlike existing benchmarks where the purpose is to benchmark against performance metrics, the goal here is to benchmark ML method for detecting and reacting to drift.

Dynamic Optimisation description: The term dynamic optimisation refers to an optimisation scenario in which there are time-varying changes in objective functions, constraints, and/or environmental parameters [9]. Unlike the data-drift scenario described above in which drift occurs *between* instances in a stream, in dynamic optimisation the drift occurs *while an instance is being solved.*

Although there are existing benchmark suites in this field [14], *Ealain* can be used to mimic dynamic optimisation in a semi-realistic setting, where for example the number of constraints increases during a run, thereby changing the fitness landscape. As the change in the number of constraints can be easily controlled, the extent of the change in the fitness landscape can also be controlled.

5 CONCLUSION

We describe a new toolbox *Ealain* that is capable of generating an arbitrary numbers of instances of a real-world inspired problem. It supports discrete and continuous encoding of a solution and provides a black-box interface to a function that returns a fitness value for a solution (a coverage metric denoting how many pixels are covered for the environment defined by the instance and a given camera placement). The software supports the generation of instances for multiple types of black-box problems which can be configured to be single-objective, multi-objective, multi-fidelity, or constrained and also permits a range of different objective functions to be defined as extensions to the code.

The goal was to provide a generator and fitness evaluator that supports the generation of datasets with specific characteristics of interest to a user. For example, instances can be generated to benchmark solvers — as it supports both discrete and continuous encodings, this enables a diverse range of solvers to be evaluated. It also supports the generation of streams of instances that can be used to define new datasets that fill gaps in current benchmarking data, for example, to define datasets which exhibit data-drift across a stream. This opens up new lines of research in optimisation that are currently neglected in contrast to the machine-learning field [7].

ACKNOWLEDGEMENTS

Quentin Renau and Emma Hart are funded by the EPSRC 'KeepLearning' project: EP/V026534/1

REFERENCES

[1] M. Abrahamsen, A. Adamaszek, and T. Miltzow. 2022. The Art Gallery Problem is ∃ℝ-complete. *J. ACM* 69, 1 (2022), 4:1–4:70. https://doi.org/10.1145/3486220
[2] T. Bartz-Beielstein, C. Doerr, J. Bossek, S. Chandrasekaran, T. Eftimov, A. Fischbach, P. Kerschke, M. López-Ibáñez, K. M. Malan, J. H. Moore, B. Naujoks, P. Orzechowski, V. Volz, M. Wagner, and T. Weise. 2020. Benchmarking in Optimization: Best Practice and Open Issues. *CoRR* abs/2007.03488 (2020). arXiv:2007.03488 https://arxiv.org/abs/2007.03488
[3] J.E. Beasley. 1990. OR-Library: distributing test problems by electronic mail. *Journal of the operational research society* 41, 11 (1990), 1069–1072.
[4] V. Chvátal. 1975. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory, Series B* 18, 1 (1975), 39–41. https://doi.org/10.1016/0095-8956(75)90061-1
[5] R.N. Gemaque, A.F.J. Costa, R. Giusti, and E.M. Dos Santos. 2020. An overview of unsupervised drift detection methods. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 10, 6 (2020), e1381.
[6] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tusar, and D. Brockhoff. 2021. COCO: a platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software* 36, 1 (2021), 114–144. https://doi.org/10.1080/10556788.2020.1808977
[7] E. Hart, I. Miguel, C. Stone, and Q. Renau. 2023. Towards optimisers tha tKeep Learning. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. 1636–1638.
[8] D. Lee and A. Lin. 1986. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory* 32, 2 (1986), 276–282.
[9] T.T. Nguyen, S. Yang, and J. Branke. 2012. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation* 6 (2012), 1–24.
[10] J. Rapin and O. Teytaud. 2018. Nevergrad - A gradient-free optimization platform. https://GitHub.com/FacebookResearch/Nevergrad.
[11] G. Reinelt. 1991. TSPLIB—A traveling salesman problem library. *ORSA journal on computing* 3, 4 (1991), 376–384.
[12] E.Y. Tema, S. Sahmoud, and B. Kiraz. 2024. Radar placement optimization based on adaptive multi-objective meta-heuristics. *Expert Systems with Applications* 239 (2024), 122568.
[13] T. Weise and Z. Wu. 2018. Difficult features of combinatorial optimization problems and the tunable *w-model* benchmark problem for simulating them. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2018, Kyoto, Japan, July 15-19, 2018*. ACM, 1769–1776. https://doi.org/10.1145/3205651.3208240
[14] X. Xiang, Y. Tian, R. Cheng, X. Zhang, S. Yang, and Y. Jin. 2022. A benchmark generator for online dynamic single-objective and multi-objective optimization problems. *Information sciences* 613 (2022), 591–608.
[15] TH. Yi, HN. Li, and M. Gu. 2011. Optimal sensor placement for structural health monitoring based on multiple optimization strategies. *The Structural Design of Tall and Special Buildings* 20, 7 (2011), 881–900.