Thesis Dissertation

DISCOVERING THE POSSIBILITIES OF A DECOUPLED GRID ARCHIVE IN QUALITY DIVERSITY AND MULTIMODAL OPTIMIZATION PROBLEMS

Konstantinos Christou

UNIVERSITY OF CYPRUS

COMPUTER SCIENCE DEPARTMENT

May 2022

UNIVERSITY OF CYPRUS

COMPUTER SCIENCE DEPARTMENT

Discovering the possibilities of a Decoupled Grid in Quality Diversity and Multimodal Optimization problems

Konstantinos Christou

Supervisors

Dr. Chris Christodoulou

Dr. Vassilis Vassiliades

Theses submitted in partial fulfilment of the requirements for the award of bachelor's degree in Computer Science at University of Cyprus.

May 2022

Acknowledgements

The completion of this study would not have been possible without my supervisors Dr. Chris Christodoulou and Dr. Vassilis Vassiliades. Their support and guidance were valuable. I would like to specifically thank Dr. Vassiliades for introducing me to the subject of Quality Diversity algorithms and generously providing me with his knowledge and experience through the undertaking of the project. I would also like to highlight my gratitude towards the academic journey I have followed at the University of Cyprus, all the courses I have taken and all the teachers I had the pleasure of being lectured by. Last, but not least, I want to thank my family for supporting my choices and helping me along the way. I appreciate their patience and their constant support.

Abstract

Quality-Diversity (QD) optimisation is a new family of evolutionary algorithms that contrasts with classic algorithms. Instead of searching for a single solution, QD algorithms are searching for a large collection of both diverse and high-performing solutions. The role of the collection is to cover the range of possible solution types as much as possible, and to contain the best solution for each type. MAP-Elites is a popular QD algorithm, which uses an N-Dimensional grid as a collection, demonstrating promising results in numerous applications. However, the N-Dimensional grid is not scalable in high dimensions, due to exponential memory requirements.

In this work, we introduce the Decoupled Grid archive, a new collection for MAP-Elites that directly extends the algorithm and improves its ability in collecting solutions in higher dimensions. The new collection leverages the diversity of a heterogenous set of One-Dimensional grids. Each grid aims at optimising a different dimension of the given problem. We evaluate the performance of the proposed archive on four tasks from the QD framework and three tasks from the Multimodal Optimisation (MMO) framework.

Our comparisons against MAP-Elites and CVT-MAP-Elites show that MAP-Elites with a Decoupled grid archive performs poorly in a QD framework and somewhat better in a MMO framework. More research needs to be conducted to further determine the applicability of the new collection in MMO.

Contents

Chapter 1: Introduction

1.1 Motivation

Quality-Diversity (QD) algorithms are a recently introduced class of evolutionary algorithms that aim at evolving repertoires of both diverse and high-performing solutions [1]. These repertoires provide simple ways to quickly adapt to new or unseen situations by switching from one solution to another [2].

A well-known algorithm of this family is MAP-Elites, introduced by Mouret and Clune (2015) [3].MAP-Elites uses an N-Dimensional Grid as a collection to store its solutions. The N-Dimensional grid is not scalable in high dimensions. That is, as the number of dimensions of a problem increases, the harder it is for the algorithm to fill the grid with diverse and high-performing solutions [4].

1.2 Objective

The primary goal of this work is to propose a new collection for MAP-Elites, namely the "Decoupled Grid" archive, that potentially behaves well in high-dimensions in the task of returning a behavioural repertoire for a given problem.

To determine that, the Decoupled Grid archive has been compared to other collections of the Quality Diversity framework, that is, the MAP-Elites' N-Dimensional Grid archive and the archive used by the CVT-MAP-Elites algorithm.

The secondary goal of this work is to examine the behaviour of the proposed archive with respect to the Multimodal Optimisation framework. For this reason, the Decoupled Grid archive has been compared with the CVT-MAP-Elites algorithm, in their ability to return the optima of a given problem.

Consequently, experiments have been conducted in both the Quality Diversity and Multimodal Optimization frameworks in order to investigate the general behaviour of the new collection.

1.3 Outline

The work is split in two sections (1) Quality-Diversity Optimization where the ability of the new collection in returning a behaviour repertoire is being examined and (2) Multimodal Optimization where the ability of the new collection in finding the optima is being examined. The Quality Diversity background and the Multimodal Optimization background are in Chapter 2. The details of the proposed collection are in Chapter 3. The experiments of the Quality Diversity Optimization, along with their respective results and analysis are in Chapter 4. The experiments of the Multimodal Optimization, along with their respective results and analysis are in Chapter 5. Finally, the general discussion regarding the evaluation of the new collection is in Chapter 6.

Chapter 2: Background

2.1 Evolutionary Algorithms (EAs)

Evolutionary algorithms (EAs) are inspired by the same mechanisms that drive biological evolution. EAs maintain a group of possible solutions, called a *population*. Every solution in a population is called an *individual*. Each individual has its gene representation, called *genetic code*. Individuals also have a value which determines their quality, called *fitness value* (also known as *objective value*). The fitness value is determined by a *fitness function* (also known as *objective function*) that is associated with a certain problem. [5]

Evolution of the population is the repeated application of certain biological operators (reproduction, mutation, recombination, and selection), until a specified terminating condition is satisfied. [6]

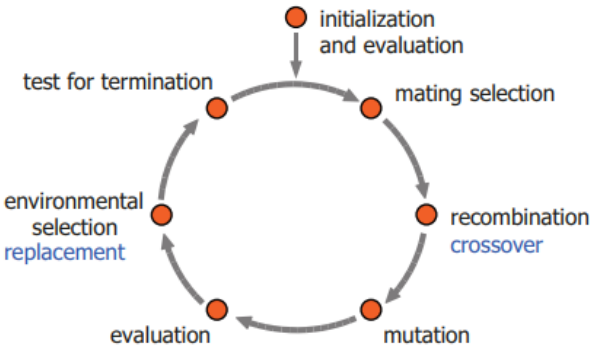


Figure 1 is from [6]

The evolution is performed in hope of locating, by the end of the algorithm, the individuals with the best fitness value. In other words, the purpose of a traditional EAs is to locate the single global value that optimises (maximises or minimises) the objective function of a problem.

EAs have not been envisioned as local search methods. Their goal was merely to locate the global optima of the given objective function. However, since about the ’80s, many EA variants have been suggested, especially for copying with highly multimodal problems. [6]

2.2 Multimodal Optimisation (MMO)

Multimodal Optimisation (MMO) is defined as the simultaneous detection of several optimisers (local or global), in a *multimodal problem*. A global optimiser is the location (or

locations) in the search space for which the objective function returns the global optimum. A local optimiser is the location in the search space that corresponds to a local optimum.
A problem is considered *multimodal* if its objective function has at least two global optimisers. Some objective functions possess one optimiser. These are generally considered easier to optimise and are called *unimodal* problems. [6]

2.3 Quality-Diversity Optimisation

Two algorithms, the Novelty Search (NS) and Novelty-Search with Local Competition (NSLC) were the building blocks of QD Optimisation.
J. Lehman and K. Stanley in 2008 introduced the NS algorithm [7]. NS is a divergent evolutionary technique inspired by natural evolution's drive to novelty. In essence, NS directly rewards novel behaviours instead of progress towards the optima of an objective function.
Three years later, they introduced the NSLC algorithm, which extended upon the ideas of NS [8]. NSLC managed to balance the diversity, with the high-performance of the individuals and as such aid in the notion of Quality Diversity.

Quality-Diversity (QD) is the concept of generating a collection of diverse and high-performing solutions [9] a process which is also known as *returning a behavioural repertoire* or *illuminating the behavioural space.*

Given a problem, Quality-Diversity (QD) Optimization can be roughly described as the process of searching for a set of optima solutions in the landscape of the objective function [1]. The solutions found by the algorithm are collected in a *collection* or an *archive* with the optimization algorithm aiming to expand and improve the collection. Each entity in the collection represents a different *solution type*, *species* or *elite* as it is called. Ideally, one wants to have a diverse and high-performing collection of solutions [10], to potentially be queried upon distinct aspects of the given problem.

Regarding the applications of QD optimisation. QD optimisation has been used in several fields including deep learning, robotics and gaming. As an example, concerning robotics, QD algorithms have been used to determine the parameters that allow a 6-legged robot to walk at any direction in its vicinity – not solely forward – even if it is damaged in several ways [11]. As an example, concerning gaming, QD algorithms have been used to generate new levels to games [12].

2.3.1 QD: Problem Formulation

In QD the objective function returns the fitness value and a behavioural descriptor (or feature vector).

$$f_\theta, \hat{b}_\theta \leftarrow f(\hat{\theta})$$

The behavioural descriptor (BD) typically describes how the solution solves the problem, while the fitness value quantifies the performance of the solution. For example, the BD can be the trajectory of a robot, while the fitness values would be the energy consumption, or the distance to the target state.

Without loss of generality, we assume hereafter that the fitness function is maximised. Let B be the feature space. The goal in QD optimisation is to find for each point in the feature space $\hat{b} \in$ B, the parameters $\hat{\theta}$ that maximise the fitness value.

$$\forall \hat{b} \in B, \qquad \widehat{\theta^*} = \arg\max_{\theta} f_\theta \ \ s.t. \hat{b} = \widehat{b_\theta}$$

When the BD is two-dimensional, this result is usually displayed as a coloured image or heatmap. [1]

2.1 Quality-Diversity (Illumination) Algorithms

For the purposes of this work, we have concentrated on two famous illumination (QD) algorithms. The MAP-Elites [3] and the CVT-MAP-Elites [13].

2.4.1 Multi-Dimensional Archive of Phenotypic Elites (MAP-Elites)

The Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) [3] is an EA inspired by NS and NSLC, capable of producing a large archive of diverse, high-performing solutions in a single run. MAP-Elites is considered an *illumination algorithm*, a term introduced at that time. [3]

> *illumination algorithms (QD algorithms)*: Algorithms that try to find the highest-performing solutions at each point in a user-defined feature space.

The idea behind MAP-Elites is quite simple. The EA algorithm uses an N-Dimensional archive to store the individuals and by the end of the evolution returns the grid itself.

In detail, the MAP-Elite algorithm takes a set of discretisations for each dimension of a pre-determined continuous feature space as parameters. MAP-Elites then discretises that continuous feature space into unique regions corresponding to the given discretization set.

At each evolution of the algorithm, MAP-Elites alters copies of solutions that are in the grid to form new solutions. These alterations are done using the same biological operators as in traditional EAs. The new solutions are evaluated and then potentially added to the cell

corresponding to their BD. If the cell is empty, the solution is added to the grid. Otherwise, only the best solution is kept in the cell. [1]

The strength of MAP-Elites lies in its simplicity to be understood and implemented. The pseudocode of the algorithm is shown below in Figure 2.

Algorithm 1 MAP-Elites Algorithm

```
procedure MAP-ELITES([n_1, ..., n_d])
    (X, P) ← create_empty_archive([n_1, ..., n_d])
    for i = 1 → G do
        x ← random_solution()
        ADD_TO_ARCHIVE(x, X, P)
    end for
    for i = 1 → I do
        x ← selection(X)
        x' ← variation(x)
        ADD_TO_ARCHIVE(x', X, P)
    end for
    return archive (X, P)
end procedure

procedure ADD_TO_ARCHIVE(x, X, P)
    (p, b) ← evaluate(x)
    c ← get_cell_index(b)
    if P(c) = null or P(c) < p then
        P(c) ← p, X(c) ← x
    end if
end procedure
```

Figure 2 is from [4]

One main drawback of MAP-Elites and specifically the N-Dimensional grid collection it uses, is that it cannot scale to high-dimensional feature spaces. The reason is that the number of regions increases exponentially with the number of dimensions of the feature space. This exponential increase in regions makes it harder for the algorithm to locate diverse individuals that are high performing as well, even when memory is not a problem. [4]

This justifies the restricted use of MAP-Elites in settings with low-dimensional feature spaces. Subsequently, scaling to high dimensions is a desirable property as it would potentially allow MAP-Elites to be used with more expressive descriptors and create archives of better quality and diversity.

2.4.2 Centroidal Voronoi Tessellation (CVT) MAP-Elites

The Centroidal Voronoi Tessellation (CVT) MAP-Elites [4] is an extension of MAP-Elites that addresses the limitation of scalability. Contrary to MAP-Elites, CVT-MAP-Elites has a constant predefined number of regions irrespective of the dimensionality of the feature space. The main idea of CVT-MAP-Elites is that the high-dimensional space is partitioned into well-spread geometric regions.

In essence, the algorithm uses a CVT [13] to divide the feature space into a desired number of regions. Then, it simply places every generated individual in its closest region. Following the same pattern as the MAP-Elites algorithm: if a generated individual is about to be placed in a region with an already occupied individual, then the best of the two is kept in the region.

The pseudocode of the algorithm is shown below in figure 3.

Algorithm 2 CVT-MAP-Elites Algorithm

```
procedure CVT-MAP-ELITES(k)
    C ← get_centroids_CVT(k)
    (X, P) ← create_empty_archive(k)
    for i = 1 → G do
        x ← random_solution()
        ADD_TO_ARCHIVE(x, X, P)
    end for
    for i = 1 → I do
        x ← selection(X)
        x' ← variation(x)
        ADD_TO_ARCHIVE(x', X, P)
    end for
    return archive (X, P)
end procedure

procedure ADD_TO_ARCHIVE(x, X, P)
    (p, b) ← evaluate(x)
    c ← CVT(b, C)
    if P(c) = null or P(c) < p then
        P(c) ← p, X(c) ← x
    end if
end procedure

procedure CVT(k, K)
    D ← sample_points(K)                    ▷ K random samples
    C ← kmeans(D, k)                        ▷ Cluster dataset D using k centroids
    return centroids C
end procedure
```

Figure 3 is from [4]

Chapter 3: Decoupled Grid Archive

3.1 Description

The proposed collection follows a similar approach as the N-Dimensional grid of MAP-Elites, yet it differs in some key respects. The collection stores the solutions in a discretized feature space, where each dimension is divided into a user-defined number of ranges – just like the N-Dimensional grid. Assuming that the feature space of a problem has N-dimensions, and one would like to divide each dimension into a specified number of ranges $(b_1, \dots, b_N)$, then the decoupled version of the MAP-Elites algorithm would try to fill each of the $B = \sum_{i=1}^{N} b_i$ bins, through a variation-selection loop, with the corresponding genotype and its fitness, replacing the lesser fit solutions with a fitter solution if it exists.
The problem solved by the decoupled version of MAP-Elites is:

> "Find b_i solutions that are as different and as high-performing as possible solely in the ith dimension, for $i \in [1, N]$"

The MAP-Elites algorithm with a Decoupled Grid archive aims in finding N solutions that their projections in these dimensions are as different and as high-performing as possible. Thus, the proposed collection can be seen as a list consisting of N One-Dimensional Grid Archives, where the ith grid focuses on the ith dimension of the behaviour.

The decoupled approach of MAP-Elites mitigates the curse of dimensionality, since the number of regions increases linearly with the number of feature dimensions.

3.1 Implementation

A pseudocode of the MAP-Elites algorithm with a Decoupled Grid archive is shown below in figure 4. The grey highlights indicate the points in which the collection differs from the classic N-Dimensional Grid.

The Decoupled Grid naturally returns N elites (at max) when asked for an elite with a similar behaviour to a given behavioural descriptor, yet a single elite needs to be returned – for the purposes of QD optimisation. The method "return_elite_with_behaviour" was implemented in

such a way to return the elite with the maximum fitness value, among the N returned by the collection.

Algorithm 3 MAP-Elites Algorithm with Decoupled Grid Archive

```
procedure MAP-ELITES([n_1, ..., n_d])
    (X, P) ← create_empty_archive([n_1, ..., n_d])
    for i = 1 → G do
        x ← random_solution()
        ADD_TO_ARCHIVE(x, X, P)
    end for
    for i = 1 → I do
        x ← selection(X)
        x' ← variation(x)
        ADD_TO_ARCHIVE(x', X, P)
    end for
    return archive (X, P)
end procedure

procedure ADD_TO_ARCHIVE(x, X, P)
    (p, b) ← evaluate(x)
    cs ← get_cell_index_of_each_dimension(b)
    for each c in cs do
        if P(c) = null or P(c) < p then
            P(c) ← p, X(c) ← x
        end if
    end for
end procedure
```

Figure 4

The full implementation of the algorithm, the problems, and the figures are all publicly available at https://github.com/KonstantinosChristou/de-grid.

3.2 Experimental Scenarios

The Decoupled Grid has been examined in two different frameworks (1) Quality Diversity framework and (2) Multimodal Optimisation framework.

In Quality-Diversity framework the proposed collection has been compared with the N-Dimensional Grid archive of MAP-Elites and the archive used by the CVT-MAP-Elites. These algorithms have been compared in their ability to return a behavioural repertoire on four problems of different complexity.

In Multimodal Optimisation framework, MAP-Elites with the proposed collection has been compared with the CVT-MAP-Elites algorithm. These two algorithms have been compared in their ability to return the maxima of three problems of different complexity.

Chapter 4: Testing within Quality-Diversity framework

4.1 Quality-Diversity Optimization

Testing a collection with regards to QD Optimization means testing the collection in its ability to return a behaviour repertoire for a given simulated problem. A problem should be able to return a tuple of two vertices at any given point. A vector of objective values for each given individual (resulting from the “fitness value”) and a vector representing the behaviour characteristics of the given individual in relation with its genotype.

4.2 QD Optimization Problems

These were the problems that have been used to test the capability of the archive in returning a behaviour repertoire: (1) 10-Dimensional Robotic Arm Repertoire, (2) 2-Dimensional Inverted Sphere, (3) 2-Dimensional Inverted Rastrigin, and (4) 2-Dimesnionnal Inverted Vincent.

4.2.1 10-Dimensional Robotic Arm Repertoire

The first problem is the control of a simulated N-Degree-of-Freedom (N-DoF) robotic arm [10]. The aim of this task is to find how to access all the points reachable by the arm, while minimising the variance between the different angles applied in each of its DoF. The best joint angles are considered to be as close to each other as possible. This makes the arm look like a smooth curve in its final position.

For the purposes of this experiment N was set to be 10.

Genotype: A solution is defined by a set of real-valued angles, one for each of the 10-DoF of the arm: $\hat{\theta} = (\theta_1, \dots, \theta_N)$ where each $\theta_i \in [-\pi, \pi]$.

Descriptor: A controller is described by the (x, y) position of the end-effector of the arm, after applying the control values in the joints. To calculate these coordinates the forward kinematics equations have been used [14]:

$$x = l_1 \cos(\theta_1) + \cdots + l_N \cos(\theta_1 + \cdots + \theta_N)$$
$$y = l_1 \sin(\theta_1) + \cdots + l_N \sin(\theta_1 + \cdots + \theta_N)$$

In which, $l_i$is the length of the i[th] link, in our case that is always one.

Fitness: The performance of a solution is given by the negative variance of the angles $(\theta_1, \dots, \theta_N)$.

$$f(\hat{\theta}) = -\frac{1}{N}\sum_{i=1}^{N}(\theta_i - \bar{\theta})^2$$

4.2.2 2-Dimensional Inverted Sphere

The second problem is an N-Dimensional Sphere domain.
For the purposes of this experiment N was set to be 2.

Genotype: A solution is given by the N real-valued variables used to compute the fitness function $\hat{x} = (x_i, \dots, x_N)$ such that $\forall x_i \in [0, 1]$

Descriptor: Each solution is described by the values of the first two of the N variables: (x_1, x_2) being mapped such that $\forall x_i \in [-5.12, 5.12]$

Fitness: The fitness is given by the Inverted N-Dimensional Sphere function:

$$f(\hat{x}) = -\sum_{i=1}^{N} x_i^2$$

4.2.3 2-Dimensional Inverted Rastrigin

The third problem is an N-Dimensional Rastrigin domain.
For the purposes of this experiment N was set to be 2.

Genotype: A solution is given by the N real-valued variables used to compute the fitness function $\hat{x} = (x_1, \dots, x_N)$ such that $\forall x_i \in [0, 1]$

Descriptor: Each solution is described by the values of the first two of the N variables: (x_1, x_2) being mapped such that $\forall x_i \in [-5.12, 5.12]$

Fitness: The fitness is given by the Inverted N-Dimensional Rastrigin function:

$$f(\hat{x}) = -\sum_{i=1}^{N}[x_i^2 - 10\cos(2\pi x_i) + 10]$$

4.2.4 2-Dimensional Inverted Vincent

The fourth problem is an N-Dimensional Vincent domain.

For the purposes of this experiment N was set to be 2.

Genotype: A solution is given by the N real-valued variables used to compute the fitness function $\hat{x} = (x_1, \dots, x_N)$ such that $\forall x_i \in [0, 1]$

Descriptor: Each solution is described by the values of the first two of the N variables: (x_1, x_2) being mapped such that $\forall x_i \in [0.25, 10]$

Fitness: The fitness is given by the Inverted N-Dimensional Vincent function:

$$f(\hat{x}) = \frac{1}{N} \sum_{i=1}^{N} \sin(10 \ln(x_i))$$

4.3 Metrics used for Returning the Behavioural Repertoire

The performance of the archives has been measured along the following set of metrics.

QD-Score: The sum of objective values of all the elites in an archive. This metric requires the objective values to be all positive.

Coverage: The percentage of cells that are occupied compared to the total number of cells (Archive Size) that are initially allocated to an archive.

Mean-Objective: The mean of the objective values of all the elites in an archive.

Also, it is worth mentioning that the Max-Objective, 25-Percentile, and the 75-Percentile of the objective values were at some point used. However, they were all opt out, due to the way the experiments were conducted. Ideally, the Max-Objective would be used as an upper bound for each Objective with the 25-Percentile and the 75-Percentile would be used to create a faded upper and lower bound for each algorithm. However, the experiments were only executed once with the same seed, thus these values were very close to one another and would be difficult to distinguish among them.

4.4 Implementation

The new archive has been implemented using the pyribs library[1] framework, which is specialized for optimizing fixed-dimensional continuous domains. The code responsible for extending the library can be seen in Appendix A.

The metrics of the problems were all inherently collected by the pyribs library. Since, the QD-Score required positive objective values, the results from the fitness function were all

[1] https://pyribs.org/ is “a bare-bones Python library for quality diversity optimization”.

normalised based on the global minimum and global maximum values of the function. Essentially, resulting in values in the range of zero to one.

The problems have initially been implemented by creating independent functions for each problem (Functional Programming). Each problem would return two vertices, given a set of individuals. Afterwards, this programming pattern was changed to Object Oriented Programming, introducing the class "Problem", for compatibility reasons with the Multimodal Optimisation problems.

4.5 Experimental Scenarios

The Decoupled Grid archive has been tested with the MAP-Elite's Grid Archive and the CVT-MAP-Elite's Grid Archive in returning a Behavioural Repertoire (BR). The experiments in this section are generated from a single run with a predefined seed equal to one. Normally, one would run multiple executions of the same problem to test it under the uncertainty of the randomised nature of the genetic algorithm. However, certain functions required significant time to execute and thus running them multiple times would exceed the deadline of this paper. Thus, for consistency all executed problems followed this pattern.

The executed experiments were using the following parameters:

Problem	**Genetic Dimensions**	**Iterations**	**Batch size**	**Emitters**	**Evaluations**
Robotic Arm	10	10^3	10^2	10	10^6
Inverted Sphere	2	10^3	10^2	10	10^6
Inverted Rastrigin	2	10^3	10^2	10	10^6
Inverted Vincent	2	10^3	10^2	10	10^6

The archive size is 10.000. This means that a single dimension of the MAP-Elites' grid archive consists of $10.000^{1/N}$ niches and a single dimension of the decoupled grid consists of 10.000/N niches, where N is the number of dimensions of the behavioural characteristics.

4.6 Experimental Results and Analysis

The results from the 10-D Robotic Arm Repertoire suggested the inability of the Decoupled Grid archive in returning the complete behavioural repertoire of the problem. In detail, both the N-Dimensional Grid archive and the CVT-MAP-Elites' archive could successfully return the complete repertoire of the robotic arm, however the Decoupled Grid archive seemed to return an incomplete repertoire. This observation was obvious when looking at the 2D plots returned by each algorithm.

Interestingly, the metrics did not point to the same direction. In fact, all the metrics seemed to favour the Decoupled Grid archive over the other archives. That is, the Decoupled Grid had increased QD-Score and Mean-Objective and was able to fill its niches faster than any other archive, as seen in the figures below.

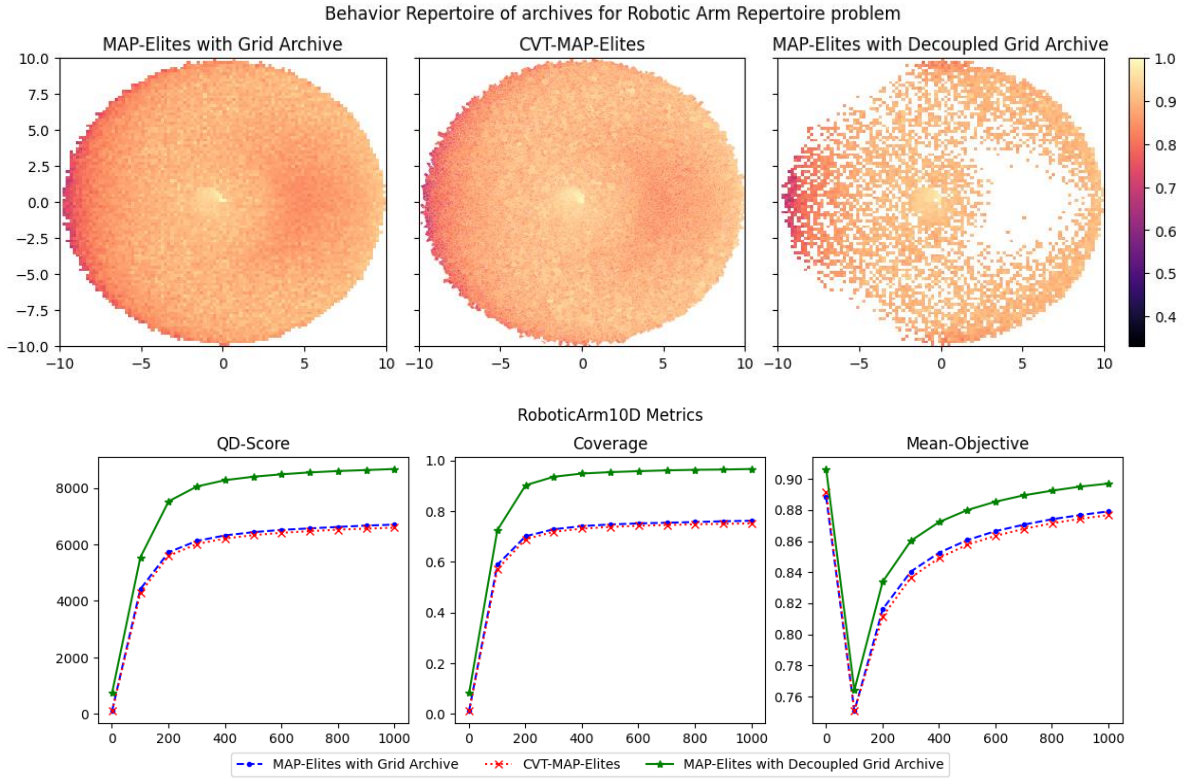
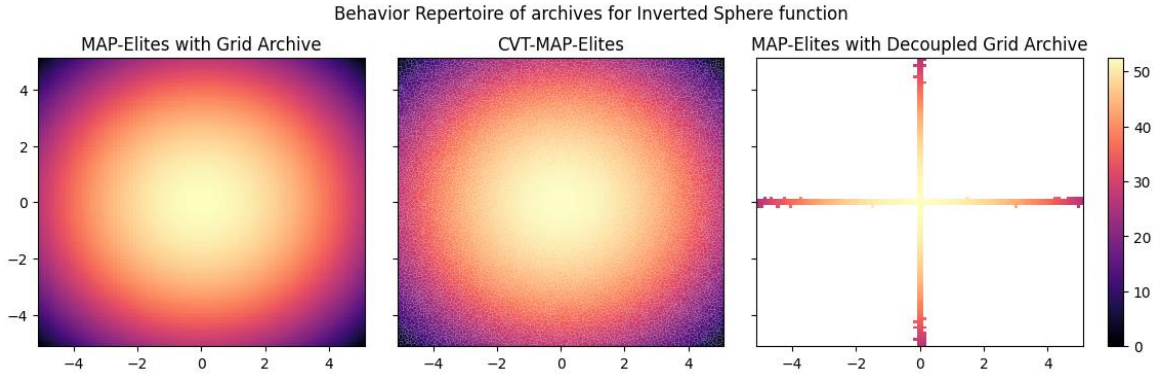


Figure 5

Following the observations seen in the 10-D Robotic Arm repertoire, a simpler (unimodal) problem was tested to examine the metrics collected by the algorithms.

The collections were tested with the 2-D Inverted Sphere and, as it was apparent by the 2D plots returned by each algorithm, the same observations were noticed. In detail, the Decoupled Grid returned an incomplete behavioural repertoire and the metrics seemed to be favouring the Decoupled Grid in terms of QD-Score and Mean-Objective.

Interestingly, at that point it was observed that the proposed grid could successfully locate the single maxima of the problem, as seen in the figures below.



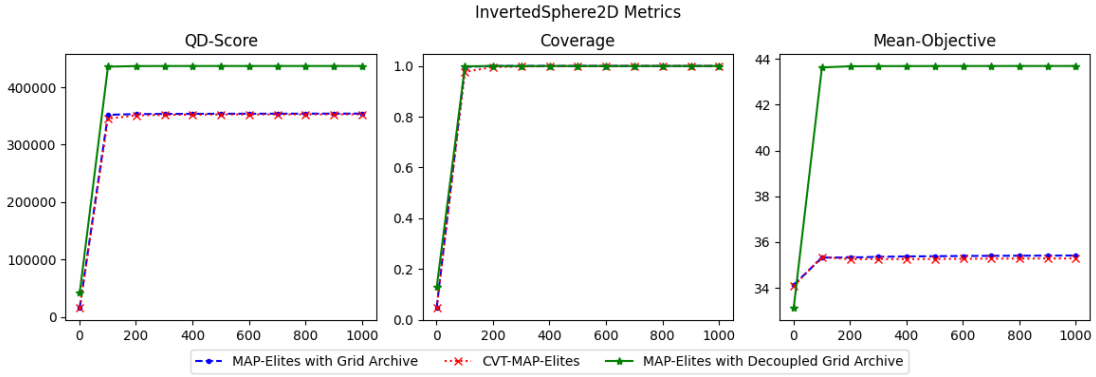


Figure 6

The previous observation was put to the test with the multimodal 2-D Inverted Rastrigin. The results showed that the Decoupled Grid behaved in similar manner with the previous experiments, returning an incomplete behavioural repertoire and metrics that pointed otherwise. However, it was observed that the Decoupled Grid had once again managed to locate the single global maxima of the problem.

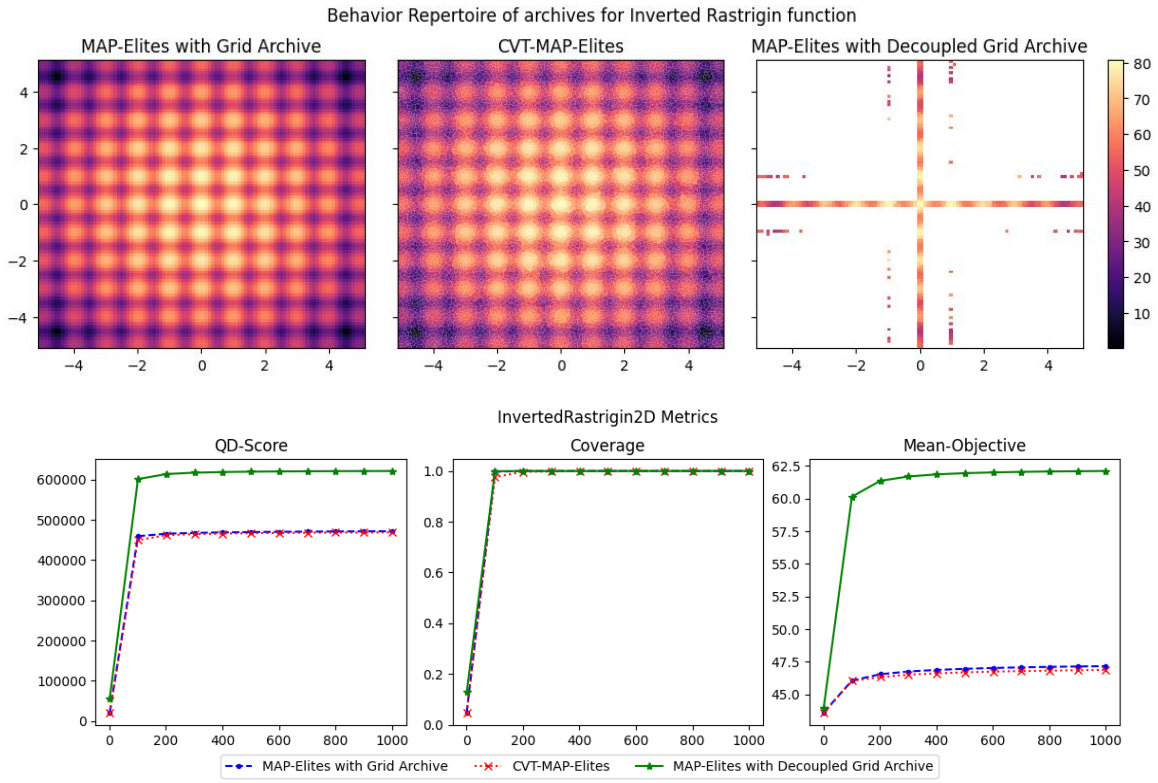


Figure 7

This previous observation was once again put to the test with the multimodal 2-D Inverted Vincent. It is worth noting that the 2-D Inverted Vincent consists of only global maxima.

The results from this experiment have once again shown that the Decoupled Grid was unsuccessful in returning a complete behavioural repertoire and the metrics were pointing otherwise.

Interestingly, it was observed that the proposed collection managed to find most, if not all, the global optima of the problem. In fact, it was noticed that all solutions were in a grid-like layout with the global maxima at each intersection, as it can be seen in the figure below.

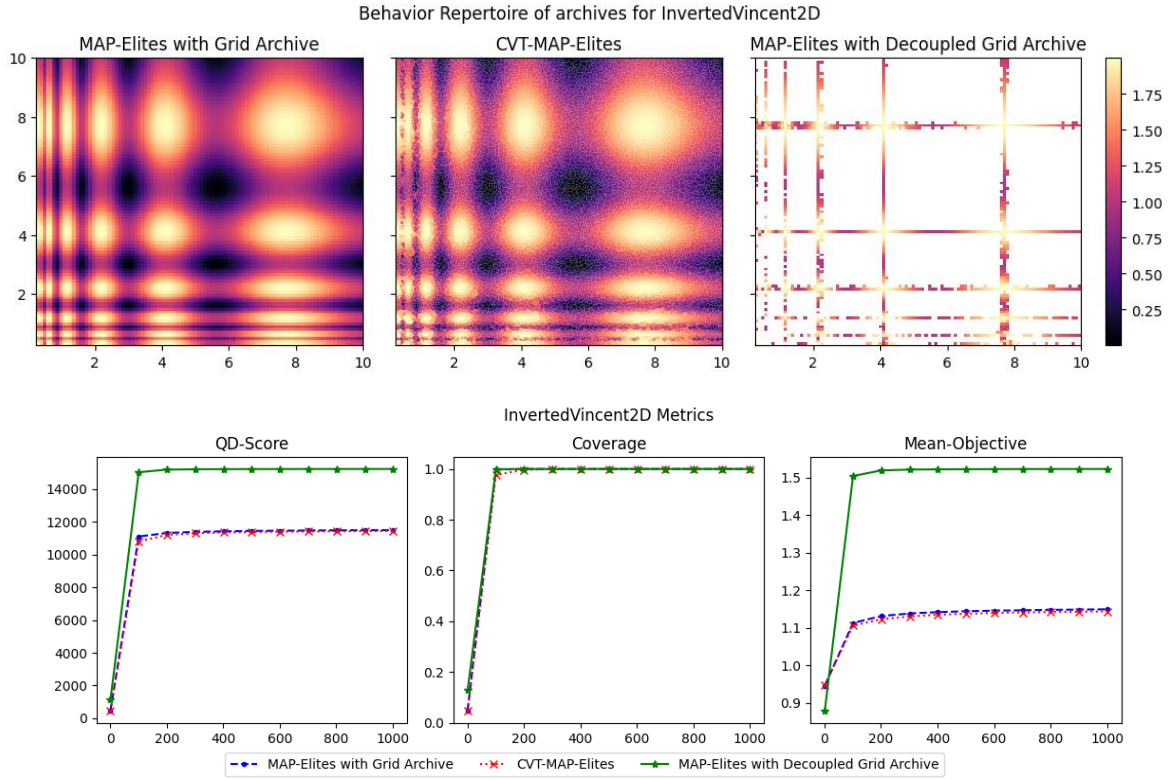


Figure 8

4.7 Other attempts at Returning the BR

4.7.1 Redefining the "return_elite_with_behaviour" function

During experimentation it was observed that the decoupled grid was consistently incapable of returning a complete behavioural repertoire for a given problem. Consequently, it was assumed that the algorithm was encountering the necessary solutions, to further illuminate the behavioural space, yet was opting them out due to their small fitness value. In the spirit of further experimentation, it was assumed that the presented collection was one version out of many other that could potentially be used to return the elite which best reflects a given behaviour characteristic.

To reiterate, the Decoupled Grid – as any other QD archive – can be asked to return the elite, which best reflects a given behaviour characteristic. Fundamentally, based on the way it was defined the Decoupled Grid returns the elite with the **greatest objective value** within a set S. In the experiments presented before, S is the set of elites that are returned by the indexing of the given behaviour in each dimension separately. That is, the N (possible) elites occupying the cells with indices that correspond to each of the cells of the One-Dimensional grids within the Decoupled Grid archive.

Several other versions have been created to possibly aid the collection in returning the behavioural repertoire. The following are other versions of the archive redefining the behaviour of the returned elite (function return_elite_with_behaviour). Evidently, none of the versions below seemed to sufficiently solve the problem in a complete and low-cost manner.

Redefinitions of the returned elite when given a behaviour characteristic:

- DeGrid-V1: Returns the elite that when simulated has the closest behaviour with the given one, among **the N elites that correspond with the given one**.
- DeGrid-V2: Returns the elite that when simulated has the closest behaviour with the given one, among **the elites in the collection** (not just the corresponding N).
  The distance between two behaviours is measured using the Euclidian distance.
- DeGrid-V3: Returns the elite that when simulated has the closest behaviour with the given one, among **the elites from the union of the N corresponding lists**. Assuming each index in the Decoupled Grid corresponds to N lists with possibly M elites per list.
- DeGrid-V4: Returns the elite that when simulated has the closest behaviour with the given one, among **the elites in the storage**. Assuming that 'storage' is a superset of the collection storing any elites that have once been added into the collection.
- DeGrid-V5: Returns an **ad-hoc elite matching the given behaviour and being created using RBF interpolation** of the elites in the collection.

4.7.2 Reducing the selection pressure

Due to the nature of the Decoupled Grid, within the collection there might be N (at max) duplicate solutions. This intrinsic property of the archive increases the selection pressure. The selection pressure is an informal term that indicates the strength of the strategy with which individual genomes are chosen from a population for later breeding.

The archive was slightly modified for a set of experiments, such that it would filter the duplicate solutions, by selecting on a uniformly distributed set of unique solutions. The hypothesis being that by using a uniformly distributed selection over a biased distribution the selection pressure would decrease and potentially allow for further illumination of the behavioural space. As the figures in Appendix A suggest, there are no apparent differences between a unified and a non-unified selection, when using the Decoupled Grid archive.

4.8 Analysis concerning the BR problem

It appears that the MAP-Elites algorithm with the proposed Decoupled Grid archive was not capable of completely illuminating the behaviour repertoire for the Robotic Arm, Inverted Sphere, Inverted Rastrigin and Inverted Vincent functions. Consequently, this persistent behaviour could be an indication that the proposed collection is unable in returning the behavioural repertoire for any given problem.

The proposed collection seems to discover solutions with high fitness value, as indicated by the persistent high value of the Mean-Objective in each of the examined problems. It is worth highlighting that the Mean-Objective of the collections being compared to the Decoupled Grid (that is the N-Dimensional Grid and the CVT-MAP-Elites' archive) were nearly a quarter as much as the Mean-Objective of the Decoupled Grid for all examined problems.

The collection seems incapable in finding all the maxima of a given problem (that includes both the local and the global maxima) as it can be seen from the Inverted Rastrigin function. It does however seem to locate most, if not all, the maxima of a given problem (as indicated by the Inverted Vincent function).

A notable observation was the grid-like layout of the solutions within the collection. That is, the Decoupled Grid seems to store the solutions in such a way to create grid-like layouts where in each intersection lies a global optimum. A reasonable explanation, as to why this effect happens is as follows. The Decoupled Grid decouples each dimension. This means that each dimension acts as an independent One-Dimensional archive – a list – that attempts to find the global maxima. Moreover, the problems being tested are all separable but the Arm repertoire.

> *Separable functions*: Function that can be solved by decomposing them into D One-Dimensional functions and aggregating (in our case with summation) the obtained optima, there is no interaction between the different variables [6].

With all these information in mind, each dimension of the Decoupled Grid searches for the same solution, since the problems being tested are all separable. Consequently, all solutions gather in a neighbourhood near the global optima, leading to the cross-markings.

Due to these findings, further examination has been conducted to further study the behaviour of the Decoupled Grid in higher dimensions. However, our attention turned towards determining whether the new collection could be beneficial in returning multiple global optima of a given multimodal problem.

Chapter 5: Testing within Multimodal Optimization framework

5.1 Multimodal Optimization

Testing a collection with regards to Multimodal Optimization means testing the collection in its ability to find the maxima of a given problem – that includes both local and global maxima.

It was evident that the Decoupled Grid was not capable of maintaining all the maxima of a given problem, yet it could locate most global maxima (as indicated by the Inverted Rastrigin and Inverted Vincent functions). Consequently, the goal of this section is to determine whether the collection can find solely the global maxima of a multimodal optimization problem.

Since any illumination algorithm can also be used as an optimization algorithm, the illumination algorithms are considered a superset of optimization algorithms [3]. Thus, the MAP-Elites with a Decoupled Grid archive has been tested with the CVT-MAP-Elites algorithm in a Multimodal Optimisation framework.

5.2 MMO Problems

These were the problems that have been used to test the capability of the archive in finding the global maxima: (1) GM-25-Random (2) GM-25R-Random (3) GM-7-Random.

5.2.1 Gaussian Mixture 25-Random function (GM-25-Random)

The Gaussian Mixture 25-Random function has been defined as the sum of the Gaussians with $\mu = x_{opt}$, $\sigma = radi = 0.1$, and maximum objective value $obj_{opt} = 1.00$ for all solutions. In which x_{opt} is a $25xN$ matrix with fixed values for the first two dimensions and randomly selected values in any other higher dimension. It is worth noting that the function simulates to some extend the behavioural space of the Inverted Rastrigin function.

Genotype: A solution is given by the N real-valued variables used to compute the fitness function $\hat{x} = (x_1, \dots, x_N)$ such that $\forall x_i \in [0, 1]$

Descriptor: Each solution is described by the same values of the N variables: $(x_1, \dots, x_N)$

Fitness: The fitness is given by the Gaussian Mixture 25-Random function:

$$f(\hat{x}) = \sum_{i=1}^{K} G\left(x_i, x_{opt_i}, obj_{opt_i}, radi_i\right)$$

$$G\left(x, x_{opt}, obj_{opt}, radi\right) = obj_{opt} \times e^{-\frac{\left(x - x_{opt}\right)^2}{2 \times radi^2}}$$

Where K is the number of created solutions, in this case K = 25.

5.2.2 Gaussian Mixture 25R-Random function (GM-25R-Random)

The Gaussian Mixture 25R-Random function has been defined to be the same as the GM-25-Random. The only distinction is a random rotation along the axis such that two solutions do not project on the same dimension. It is worth noting that the function simulates to some extend the plane of the Inverted Rotated Rastrigin function.

Genotype: A solution is given by the N real-valued variables used to compute the fitness function $\hat{x} = (x_1, \dots, x_N)$ such that $\forall x_i \in [0, 1]$

Descriptor: Each solution is described by the same values of the N variables: $(x_1, \dots, x_N)$

Fitness: The fitness is given by the Gaussian Mixture 25R-Random function:

$$f(\hat{x}) = \sum_{i=1}^{K} G\left(x_i, x_{opt_i} \times M, obj_{opt_i}, radi_i\right)$$

$$G\left(x, x_{opt}, obj_{opt}, radi\right) = obj_{opt} \times e^{-\frac{\left(x - x_{opt}\right)^2}{2 \times radi^2}}$$

Where K the number of created solutions, in this case K = 25 and M is an orthogonal $25x25$ matrix.

5.2.3 Gaussian Mixture 7-Random function (GM-7-Random)

The Gaussian Mixture 7-Random function has been defined as the sum of the Gaussians with $\mu = x_{opt},\ \sigma = 0.2$, and maximum objective value $obj_{opt} = 1.00$ for all solutions. In which x_{opt} is a $7xN$ matrix with fixed values for the first two dimensions and randomly selected values in any other higher dimension.

Genotype: A solution is given by the N real-valued variables used to compute the fitness function $\hat{x} = (x_1, \dots, x_N)$ such that $\forall x_i \in [0, 1]$

Descriptor: Each solution is described by the same values of the N variables: $(x_1, \dots, x_N)$

Fitness: The fitness is given by the Gaussian Mixture 25-Random function:

$$f(\hat{x}) = \sum_{i=1}^{K} G\left(x_i, x_{opt_i}, obj_{opt_i}, radi_i\right)$$

$$G\left(x, x_{opt}, obj_{opt}, radi\right) = obj_{opt} \times e^{-\frac{\left(x - x_{opt}\right)^2}{2 \times radi^2}}$$

Where K is the number of created solutions, in this case K = 7.

5.3 Metrics used for finding the maxima of MMO problems

The performance of the archives has been measured along QD and MMO metrics. In detail, the following set of metrics have been used:

QD-Score: The total objective values of all the elites in an archive.

Coverage: The percentage of cells that are occupied compared to the total number of cells (Archive Size) that was initially allocated to the archive.

Success-Rate: The percentage of runs in which all the desired peaks are located.

Mean-Maxima-Found: The average number of maxima (peaks) in the archive.

Success-Performance: The average number of function evaluations divided by the *Success-Rate*. Notice that the Success Performance can only be obtained if the success rate is not zero.

Max-Peak-Ratio: The sum of all the fitness values of the maxima in the final population divided by the sum of all the values of real optima of the objective function. Assuming a maximization problem and that all values are positive. A large MPR value indicates a better performance of the particular algorithm.

5.4 Challenges in finding the maxima of MMO problems

Collecting the metrics of the Multimodal Optimization problems came with certain challenges. Most MMO metrics required the prior knowledge of the problem's optima solutions. That information was not easily obtainable, nor scalable depending on the problem.

Challenges:

1. *Needed to know the optima within an archive.*
   Reason: Due to metric definitions
2. *Needed to know the optima of the problems.*
   Reason: Due to metric definitions
3. *Needed to have a small number of optima.*
   Reason: To be easily scalable

In the process of finding a method that would identify the optima solutions within an archive, a heuristic was used. The Nearest Better Clustering (NBC) [15] was altered in such a way to find the elites that were considered to be maxima. A high-level algorithm of NBC can be seen in the figure bellow:

Algorithm 1: Nearest-better clustering (NBC)

```
1 compute all search points mutual distances;
2 create an empty graph with num(search points) nodes;
  // make spanning tree:
3 forall the search points do
4 |   find nearest search point that is better; create edge
  |   to it;
  // cut spanning tree into clusters:
5 delete edges of length > φ · mean(lengths of all edges) ;
  // find clusters:
6 find connected components;
```

Figure 9

Having said that, as the project progressed it was evident that the metrics required both finding the maxima of an archive and finding the maxima of the problem. NBC was not only primarily focused on the former but was susceptible to false negatives. That is, NBC could not identify all the optima of the archive, when the solutions were not in an evenly spaced grid-like layout (as is the case with the Inverted Vincent function). Thus, NBC was considered to be not ideal and was set aside.

In the process of finding a method that would identify both the optima solutions within an archive and the solutions of a given problem, a mathematical approach was used. The gradient

of the fitness function could be easily calculated using the autograd library [2] and then equated with zero to determine the optima solutions. Later on, it was determined that this approach required near perfect solutions for them to be considered as optima – leading to abnormally high arithmetic tolerance values. Moreover, the arithmetic tolerance that was set to prune the furthest solutions would at times leave more than one solution near the optima, due to symmetry. For these reasons, this method was considered not ideal and was set aside as well.

The MMO problems that were to be tested would initially be the same as the ones used in the QD framework – for consistency. However, it was clear, from their definitions, that as the number of dimensions increased, the number of optima also increased exponentially. For instance, the multimodal Inverted Rastrigin function has only 121 fixed optima in two dimensions, but 161,051 in five. Consequently, storing the fixed values of high-dimensional high-multimodal problems would come with an unacceptable memory cost that should not be taken.

Alternatively, the MMO problems, needed to be easily scalable in high dimensions and their number of optima needed to be relatively small. For this reason, a tool was created that would allow the creation of such problems. The *Gaussian Mixture toolkit (GM-toolkit)* sums Gaussian functions to create simple multimodal domains, with a fix number of maxima.
Having a small, fixed number of maxima for each problem, helped in locating the optima of the archive. This was done by querying the archive's solutions for the nearest neighbour of the problem's optima solutions.

[2] https://github.com/HIPS/autograd efficiently computes derivatives of numpy code.

5.5 Implementation

The pyribs library was extended to support the collection of multimodal optimisation metrics. As such, a new method was implemented in all archives (the basis of the archive class) that when given a set of values (maxima of the problem), would return the objective values of the elites with solutions that matched these values. In essence, querying the archive's solutions for the nearest neighbours of the problem's optima solutions using a kd-tree provided by the scipy library [3]. That way, any algorithm could utilise this method to evaluate the MMO metrics that required knowledge of the maxima within the archive. It is worth nothing that, a parameter (arithmetic tolerance) was used to indicate the maximum allowed distance that a solution can diverge from the generated solutions of the problem to be considered as a maximum. This parameter was set to be 0.1 for all executed experiments. Hence, solutions that are within a 0.1 apart from an actual maximum are considered maxima.

The GM-toolkit was used to simulate the behaviour of the problems that were to be tested by creating corresponding functions for each one. The GM-toolkit requested a set of solutions that corresponded to the optima solutions of the problem (x_{opt}), a set of optima values that corresponded to the objective value of each given solution (obj_{opt}), and a set of radii that corresponded to the influence of the optima towards the created landscape (radi).

The MMO problems were not implemented in a Functional Programming pattern, due to the need of added functionality. Instead, Object Oriented Programming was used. Essentially, each class of "Problem" consisted of the following main attributes and functions (the full implementation of the class can be found in Appendix C):

Problem:

- *x_dim:* The number of genotypical dimensions to be used.
- *beh_dim:* The number of phenotypical dimensions to be used.
- *xl, xh:* The minimum and maximum bounds of a genotype. Assuming that each genotype is the same no matter dimension.
- *yl, yh:* The minimum and maximum values (range) of the objective/fitness function.
- *x_opt:* The optima (maxima) values of the objective/fitness function.
- ***fit(xs, beh_dims)****:* The fitness function returns the objective values of each batch of solutions, along with their corresponding behavioural values.
- ***get_maxima_idx(xs, atol)****:* The maxima indices that correspond to the given solutions, based on the specified arithmetic tolerance.

[3] https://scipy.org an open-source software for mathematics, science, and engineering.

5.6 Experimental Scenarios

The Decoupled Grid archive has been tested with only CVT-MAP-Elites' Archive in finding the optima of a given problem. The MAP-Elites' Grid Archive has not been tested due to exponential amount of memory requirements.

Following the pattern described in QD framework, the experiments in this framework are generated from a single run with a predefined seed equal to one.

Each experiment was conducted using a predefined number of evaluations. In other words, each experiment was calling upon the fitness function a certain number of times. The tested values range from 10^6 to 10^8 evaluations which equates with 10^3 to 10^5 iterations.

The executed experiments were using the following parameters:

Evaluations = 10^6

Problem	**Genetic Dimensions**	**Iterations**	**Batch size**	**Emitters**
GM-25-Random	2-6	10^3	10^2	10
GM-25R-Random	2-6	10^3	10^2	10
GM-7-Random	2-7	10^3	10^2	10

Evaluations = 10^7

Problem	**Genetic Dimensions**	**Iterations**	**Batch size**	**Emitters**
GM-25-Random	2-10	10^4	10^2	10
GM-25R-Random	2-10	10^4	10^2	10
GM-7-Random	2-10	10^4	10^2	10

Evaluations = 10^8

Problem	**Genetic Dimensions**	**Iterations**	**Batch size**	**Emitters**
GM-25-Random	2-10	10^5	10^2	10
GM-25R-Random	2-10	10^5	10^2	10
GM-7-Random	2-10	10^5	10^2	10

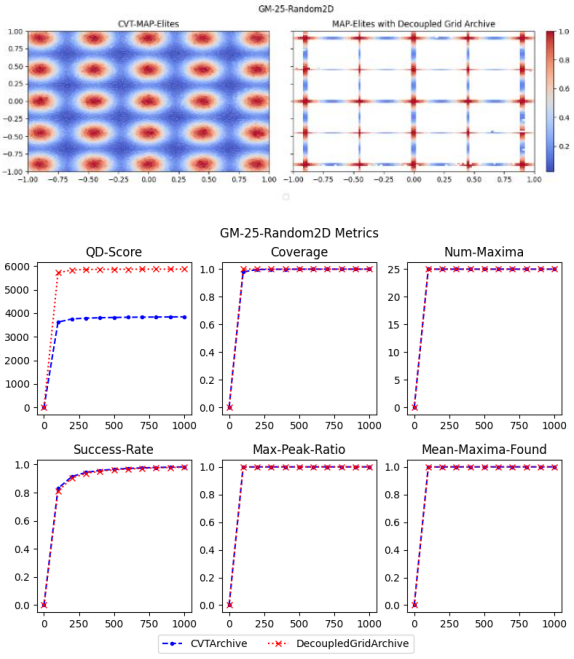
The number of dimensions of the behavioural characteristics was set to be the same as the number of dimensions of the genotype. Additionally, the archive size was set to be 10.000 niches for all collections.

5.7 Experimental Results and Analysis

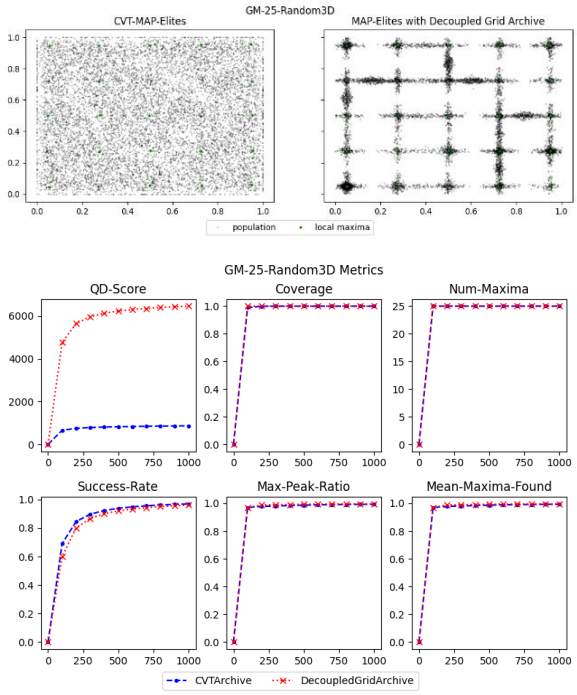
5.7.1 GM-25-Random with 10^6 Evaluations

Considering the results from the QD framework, the Decoupled grid archive seemed to perform well in 2D problems, regarding the finding of the global maxima. When being tested on GM-25-Random with 10^6 evaluations both archives (CVT-MAP-Elites' archive and Decoupled grid archive) performed very well in 2D, as expected, with the Decoupled Grid having significantly higher QD-Score. When steadily increasing the dimensionality of the problem whilst keeping the number of evaluations constant the two archives started to diverge from one another, with respect to their metrics. A noticeable change was detected when the number of dimensions was 5. At that point neither of the two archives managed to identify all the maxima of the problem. Interestingly, CVT-MAP-Elites' archive managed to find more maxima than the Decoupled grid, by a difference of 5. When the dimensionality of the problem was further increased it was clear that neither of the two archives could find all the maxima of the problem. With a dimensionality of 6 the Decoupled grid archive was once again relatively better than the CVT-MAP-Elites' archive, by finding almost twice as much maxima as the CVT-MAP-Elites' archive.

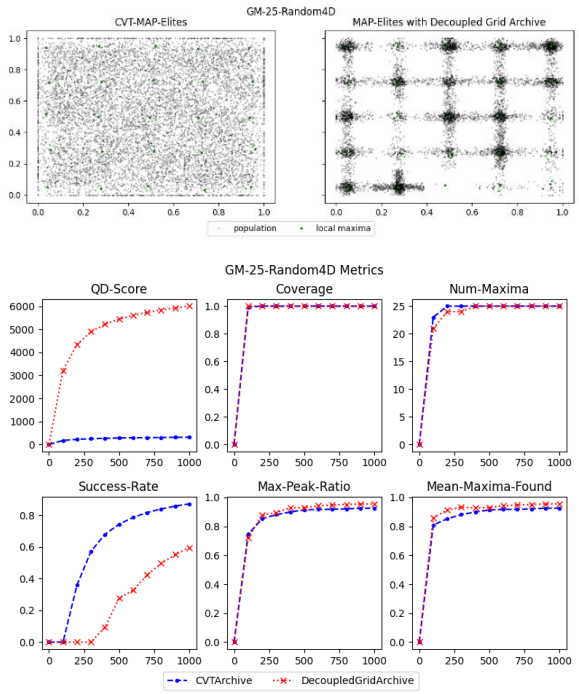
Behaviour in 2D



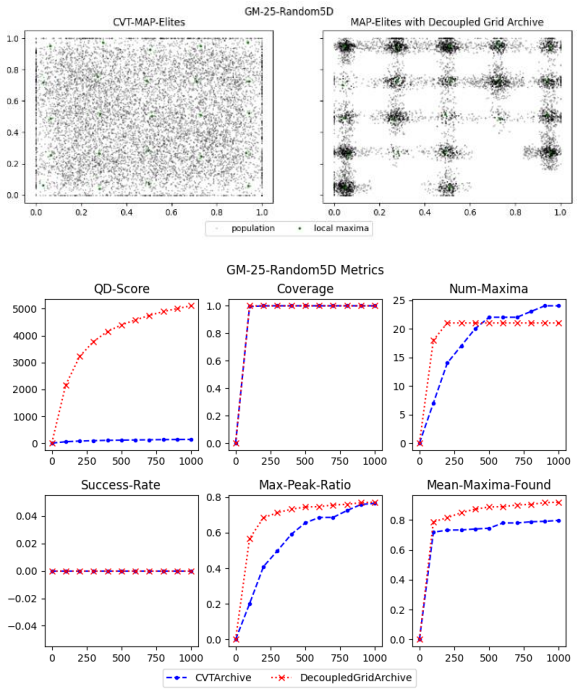
Behaviour in 3D



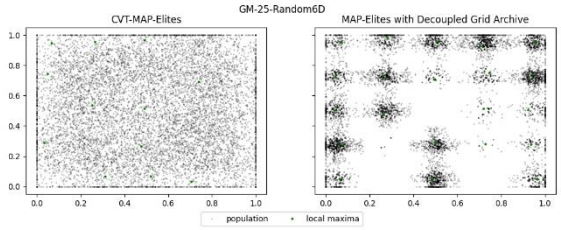
Behaviour in 4D

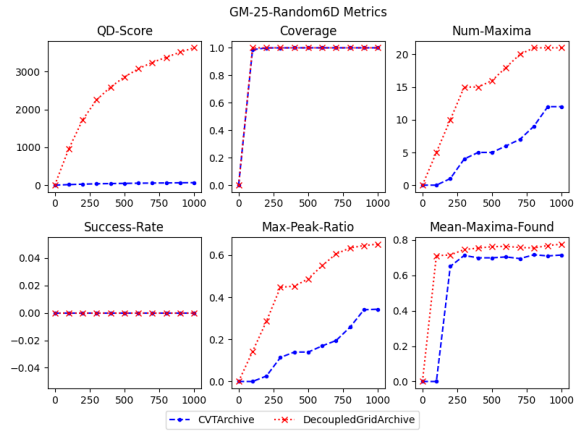


Behaviour in 5D



Behaviour in 6D



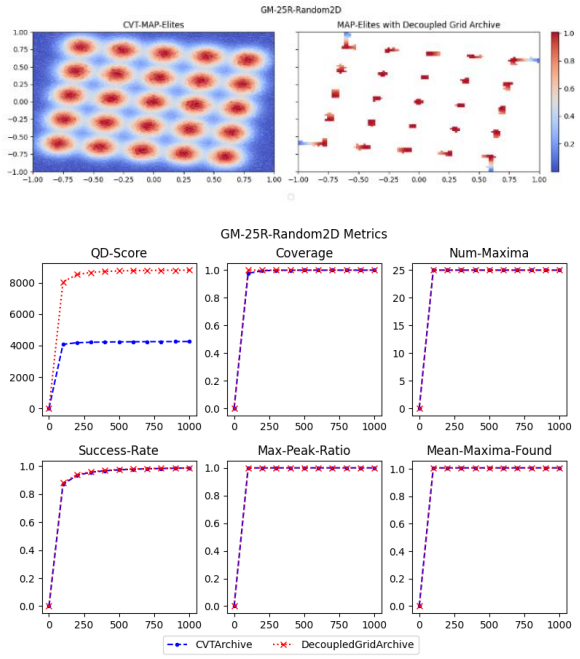


5.7.2 GM-25R-Random with 10^6 Evaluations

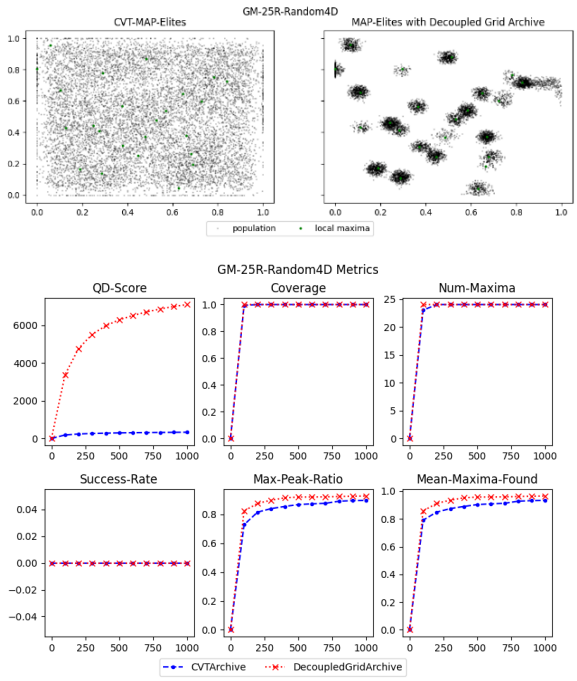
When being tested on GM-25R-Random with 10^6 evaluations both archives performed very well in 2D. It was a clear sign that the Decoupled grid archive was not restricted by the formation of the solutions in any dimension. When steadily increasing the dimensionality of the problem whilst keeping the number of evaluations constant the two archives started to diverge from one another, with respect to finding the global maxima, once the dimensionality of the problem was 4. At that point, neither of the archives managed to successfully find all the maxima of the problem. Interestingly, their metrics (besides QD-Score) were pointing towards complete success. That led to the belief that there was an implication with the randomisation of the angle of rotation. Possibly, certain optima were being mapped close enough that the GM-toolkit created a single optimum rather than individual optima, which conflicted with the number of predefined solutions. With that in mind, the GM-25R-Random problem was only used for in-between comparisons among the archives and not the problem itself, as certain metrics were not exact. With that being said, due to implications with the GM-toolkit, the next

noticeable change between the two archives was noticed in 6D. At that point, the Decoupled Grid managed to find nearly twice as much optima as the CVT-MAP-Elites' archive.

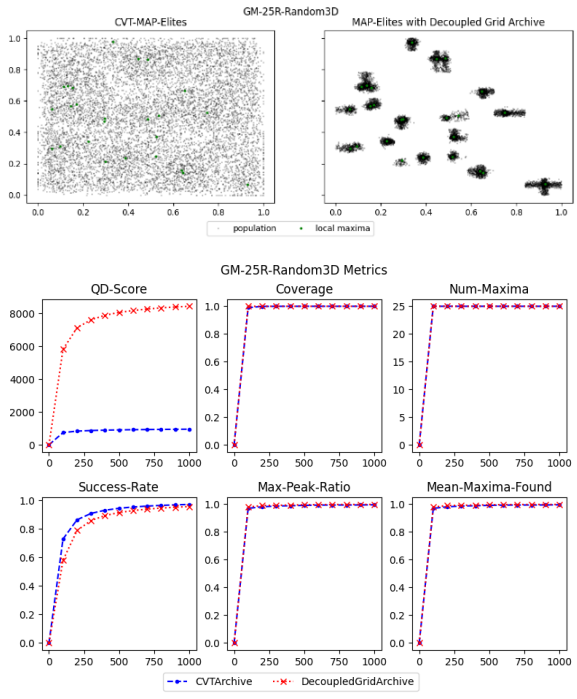
Behaviour in 2D



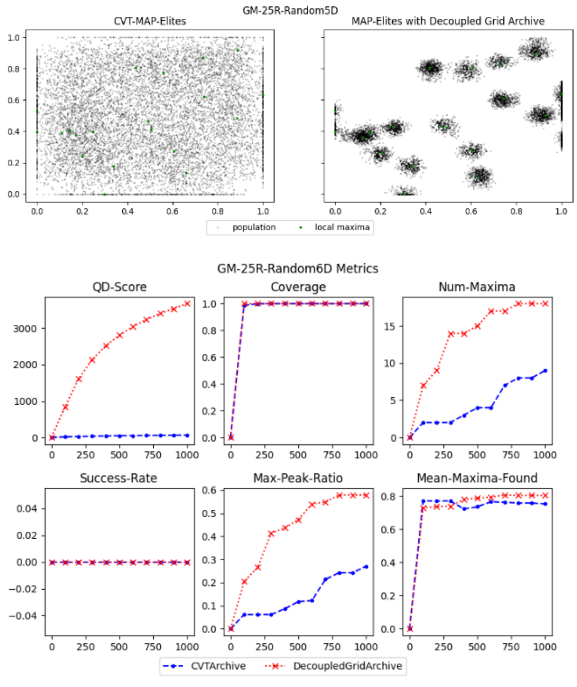
Behaviour in 4D



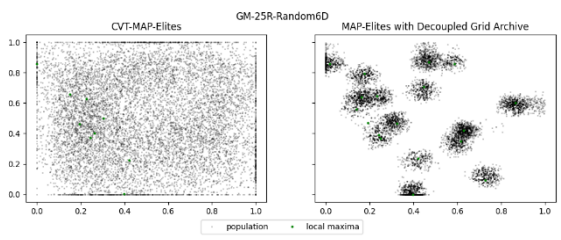
Behaviour in 3D

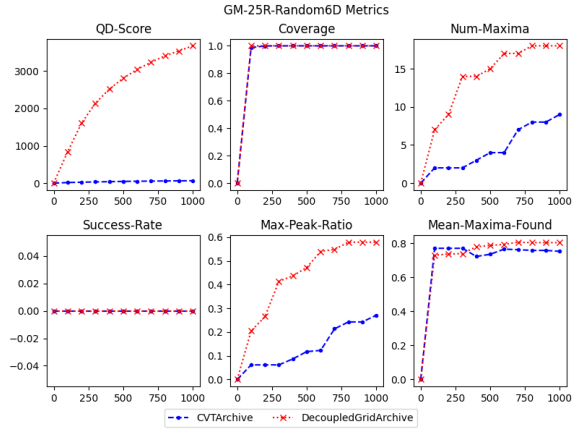


Behaviour in 5D



Behaviour in 6D

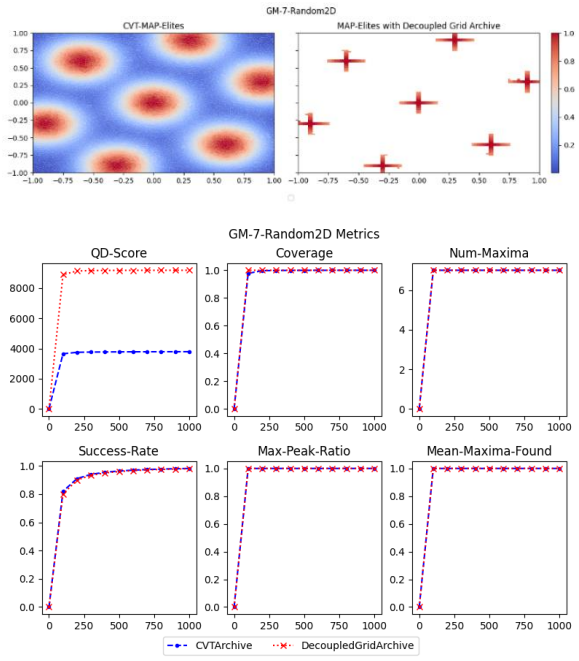




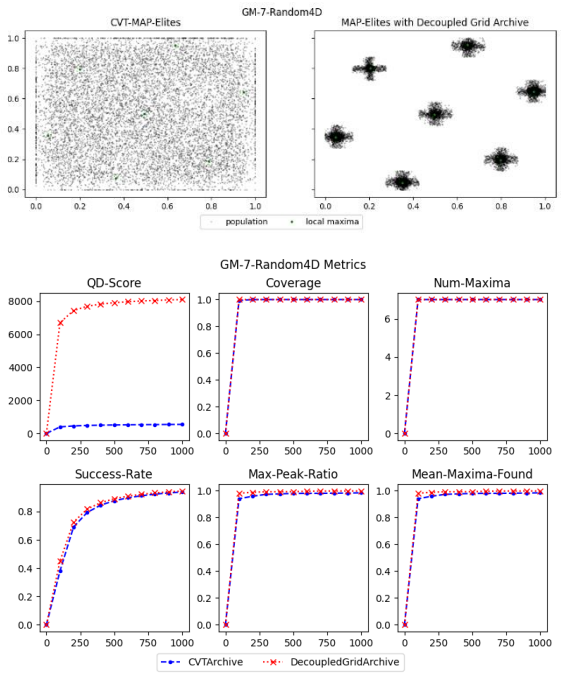
5.7.3 GM-7-Random with 10^6 Evaluations

When being tested on GM-7-Random with 10^6 evaluations both archives performed very well in 2D. When steadily increasing the dimensionality of the problem whilst keeping the number of evaluations constant, the two archives started to diverge from one another, with respect to finding the global maxima, once the dimensionality of the problem was 5. At that point the Decoupled Grid found all the global maxima of the problem, whereas the CVT-MAP-Elites archive found all but one. The same behaviour was observed in the next increase of the dimensionality with 6D. In 7D both archives were not able to locate all the optima of the problem. Interestingly, the Decoupled grid managed to locate all but two, whereas the CVT-MAP-Elites' archive was not able to locate any maxima.

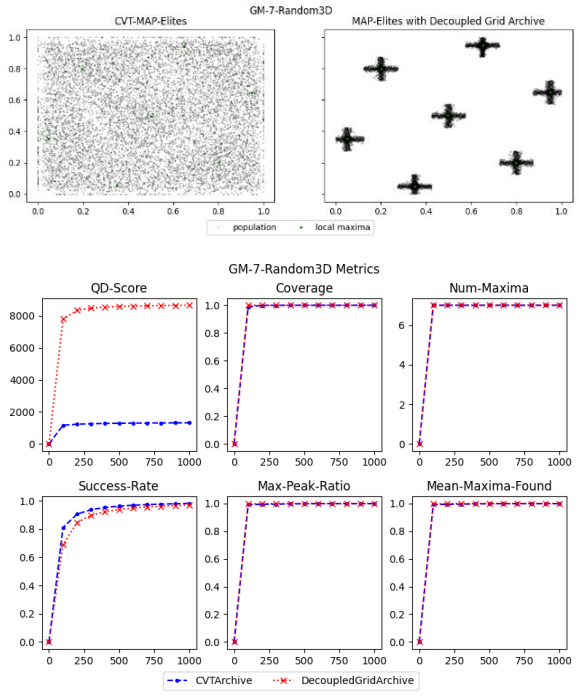
Behaviour in 2D



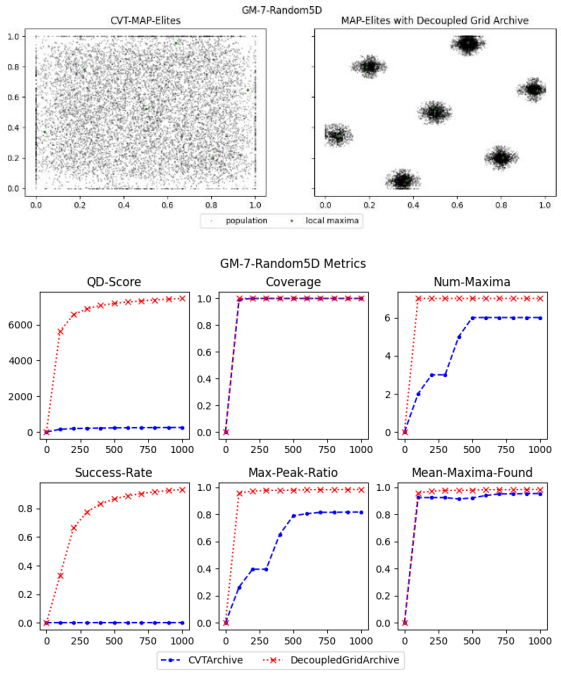
Behaviour in 4D



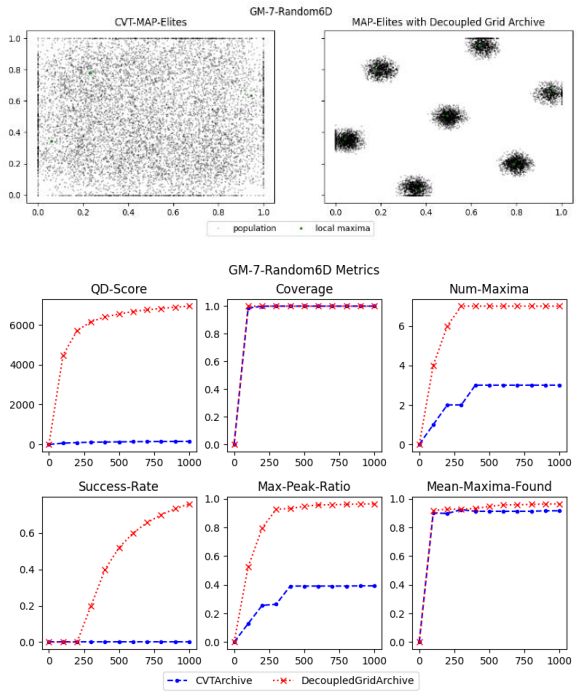
Behaviour in 3D



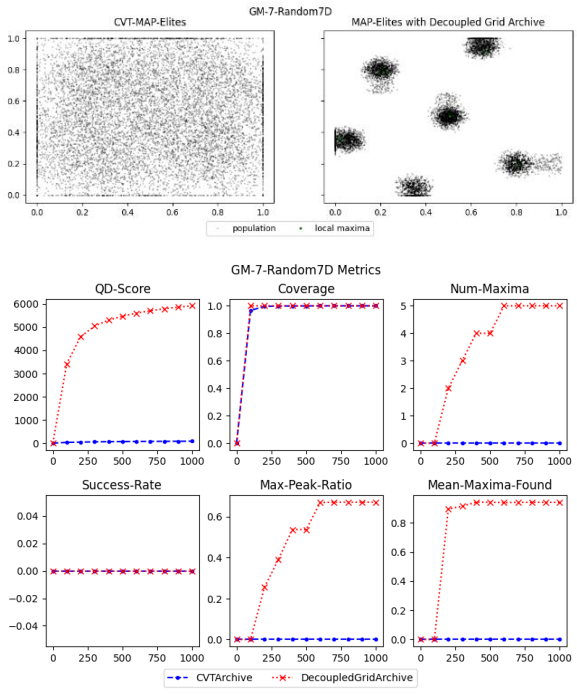
Behaviour in 5D



Behaviour in 6D



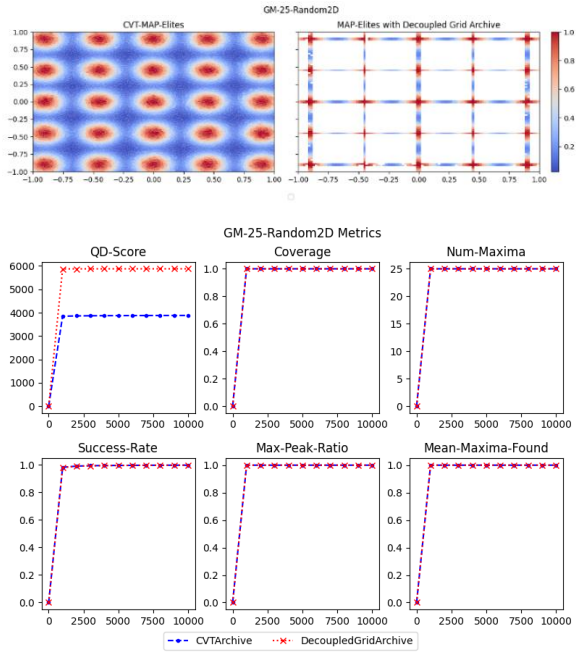
Behaviour in 7D



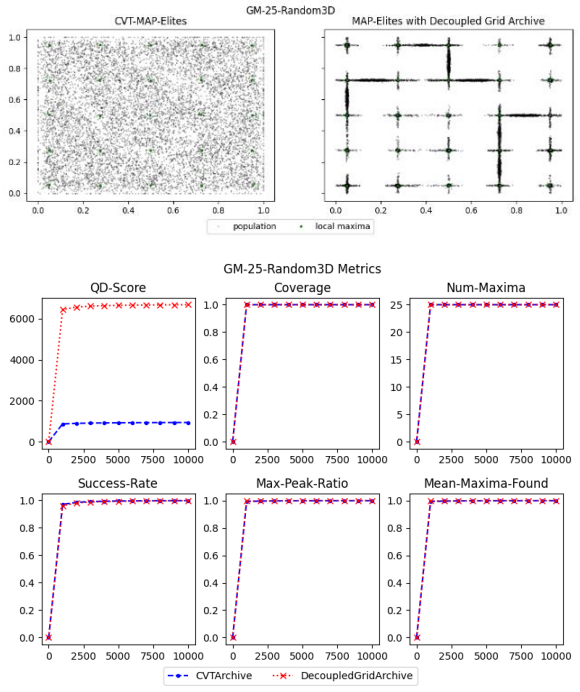
5.7.4 GM-25-Random with 10^7 Evaluations

When being tested on GM-25-Random with 10^7 evaluations (an increase by factor of 10, in relation to the previous set of experiments) both archives performed relatively the same in low dimensions. When being tested on GM-25-Random with 10^7 evaluations in 5 dimensions, the CVT-MAP-Elites' archive managed to locate all the maxima of the problem, whereas the Decoupled Grid managed to locate all but 4 maxima. In 6 dimensions the CVT-MAP-Elites' archive managed to locate all the maxima of the problem, whereas the Decoupled Grid managed to locate all but 3 maxima. In 7 dimensions neither of the archives managed to locate all the maxima, with both finding relatively the same number of maxima (Decoupled grid found one less than CVT-MAP-Elites' archive). In 8 dimensions once again, neither of the archives managed to locate all maxima, yet the Decoupled Grid archive managed to locate twice as many maxima compared to the CVT-MAP-Elites. In higher dimensions, CVT-MAP-Elites' archive could not locate any maxima, whereas the Decoupled Grid archive managed to locate a couple of solutions – even in 10 dimensions.

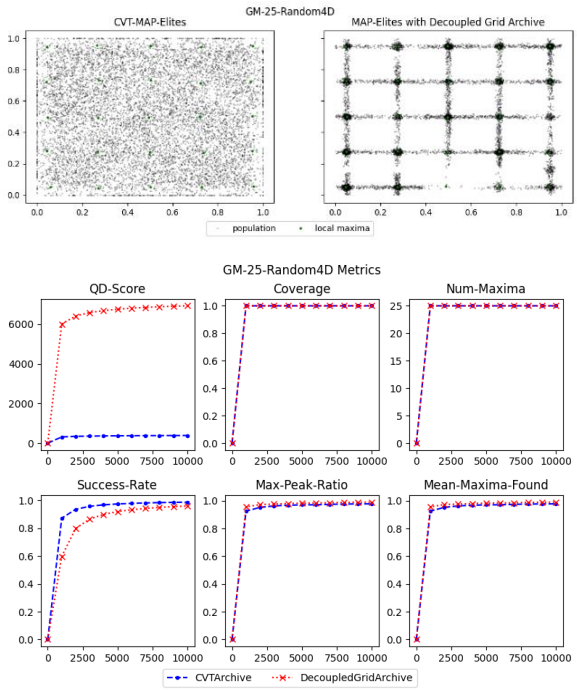
Behaviour in 2D



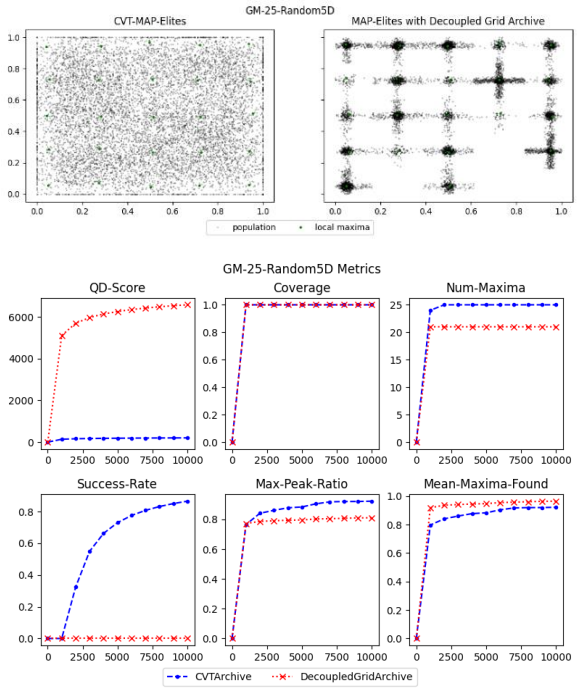
Behaviour in 3D



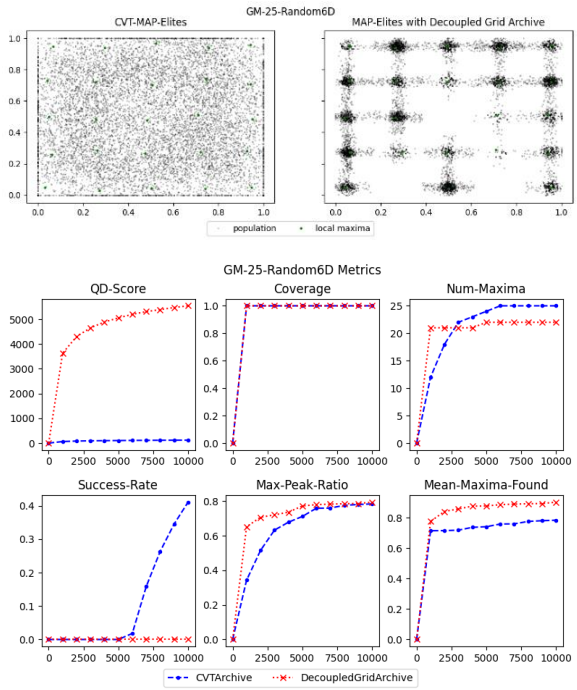
Behaviour in 4D



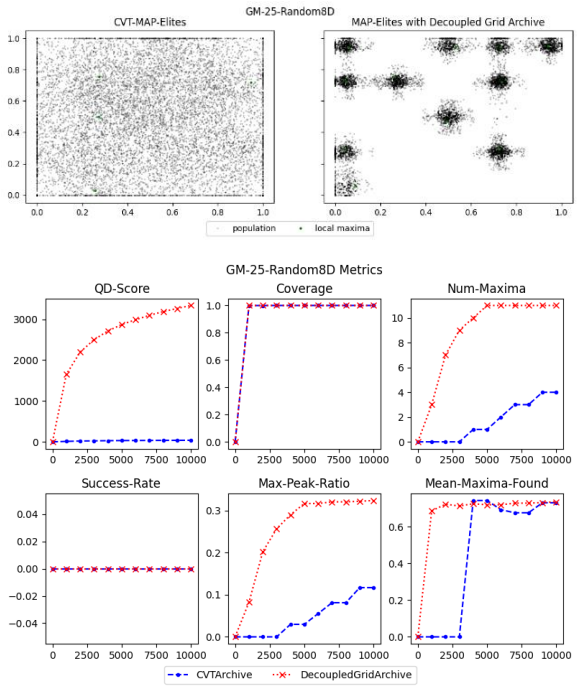
Behaviour in 5D



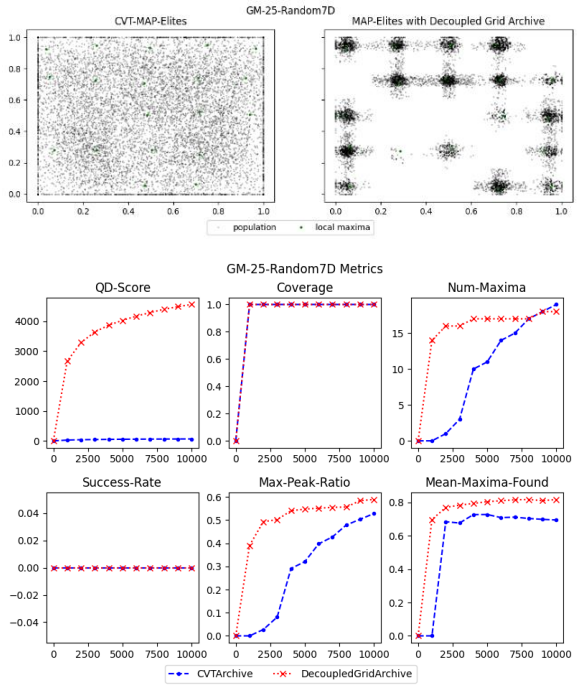
Behaviour in 6D



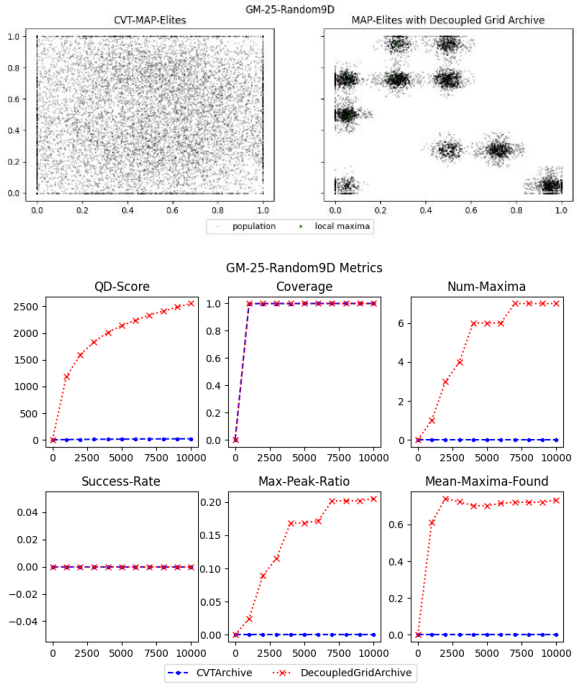
Behaviour in 8D



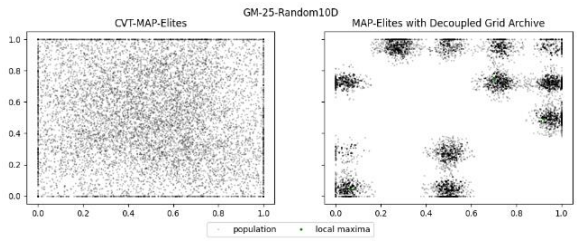
Behaviour in 7D

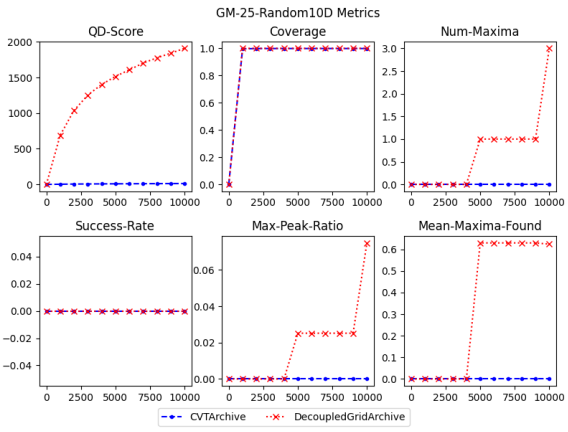


Behaviour in 9D



Behaviour in 10D

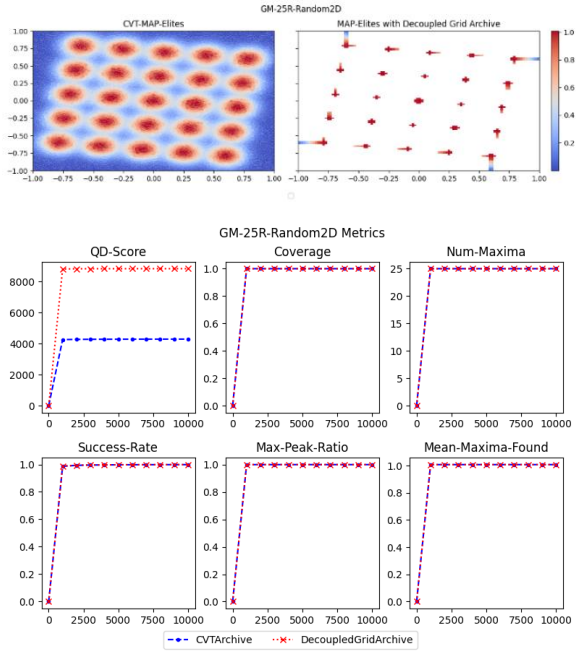




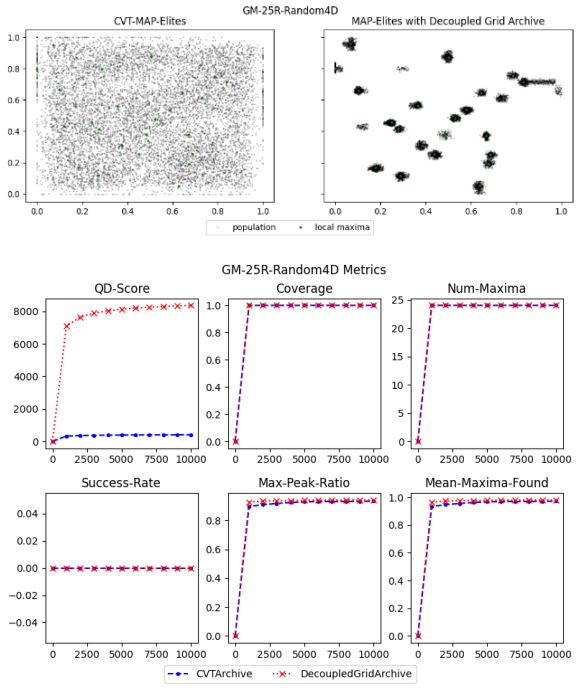
5.7.5 GM-25R-Random with 10^7 Evaluations

When being tested on GM-25R-Random with 10^7 evaluations both archives performed relatively the same in low dimensions. When being tested on GM-25R-Random with 10^7 evaluations, in 8D the Decoupled grid archive managed to find twice as many maxima as the CVT-MAP-Elites' archive. In higher dimensions the CVT-MAP-Elites' archive could not find any maxima, whereas the Decoupled Grid managed to locate a couple.

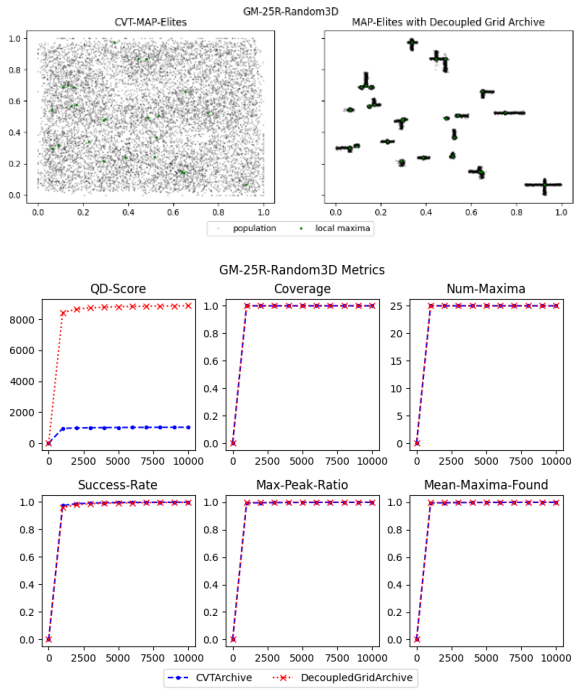
Behaviour in 2D



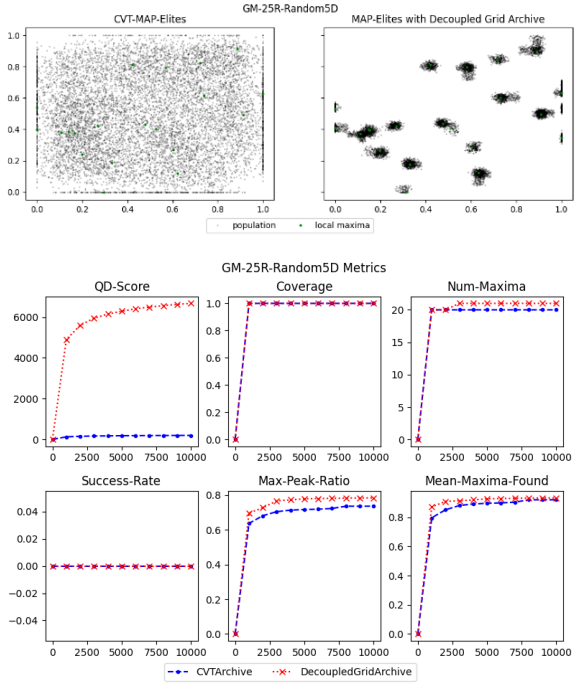
Behaviour in 4D



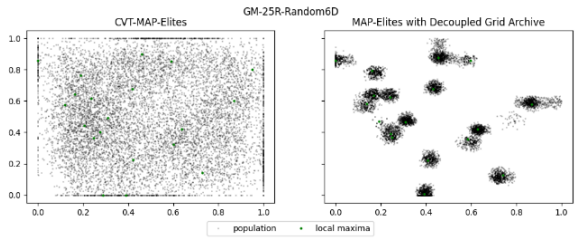
Behaviour in 3D

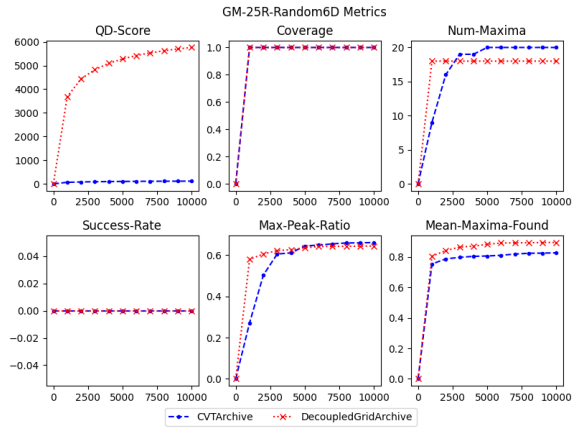


Behaviour in 5D

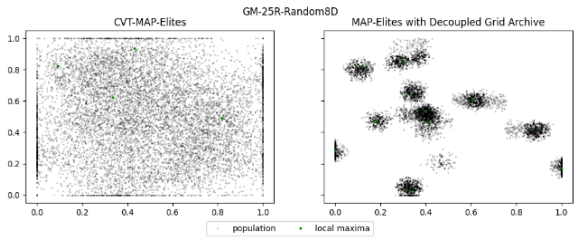


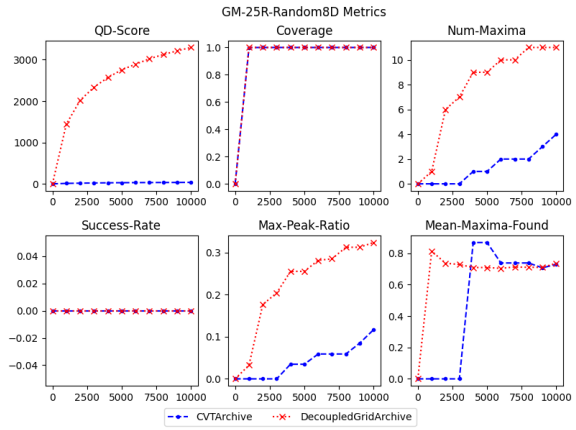
Behaviour in 6D



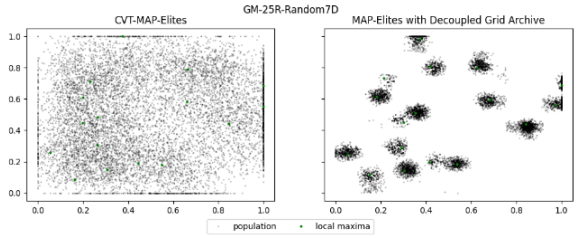


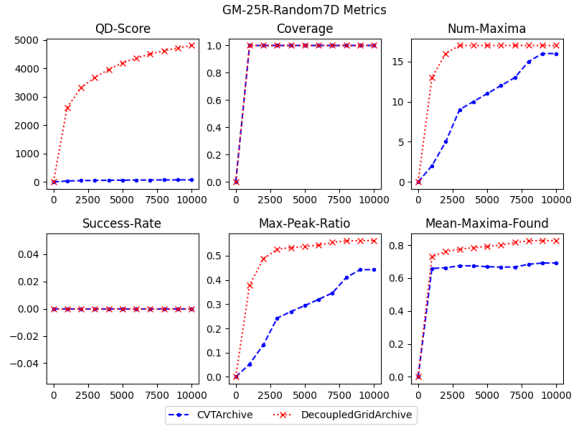
Behaviour in 8D



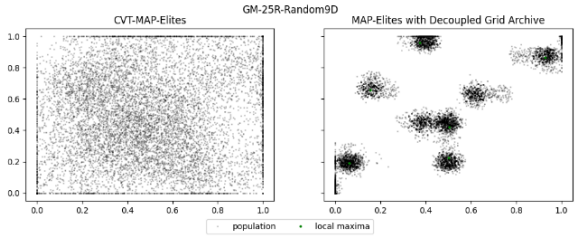


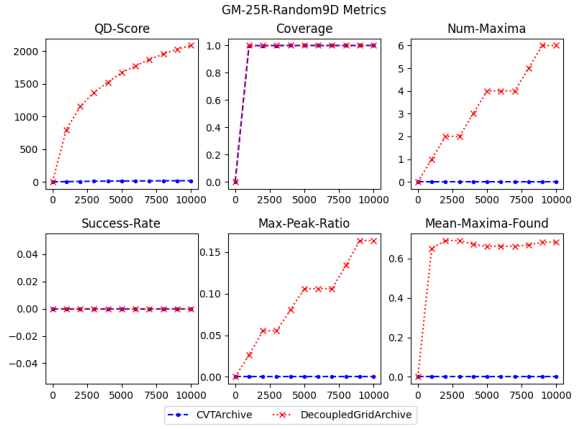
Behaviour in 7D



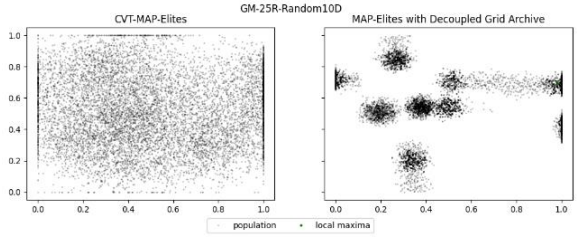


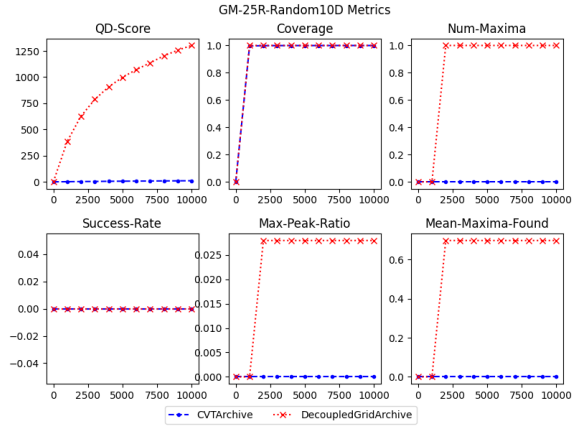
Behaviour in 9D





Behaviour in 10D

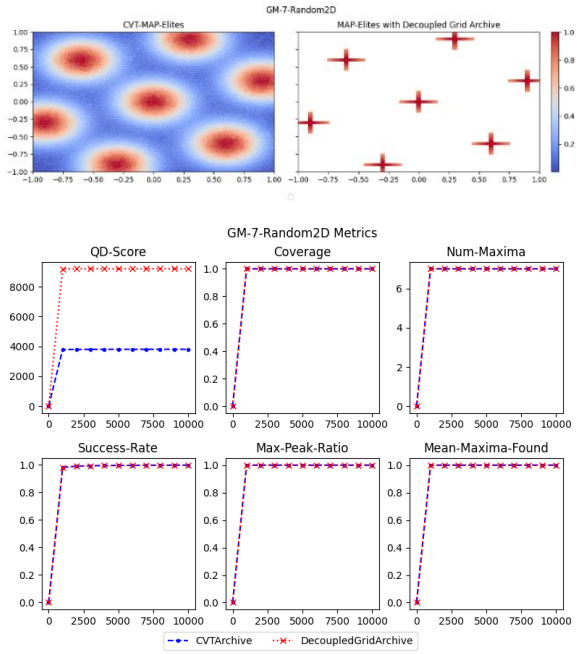




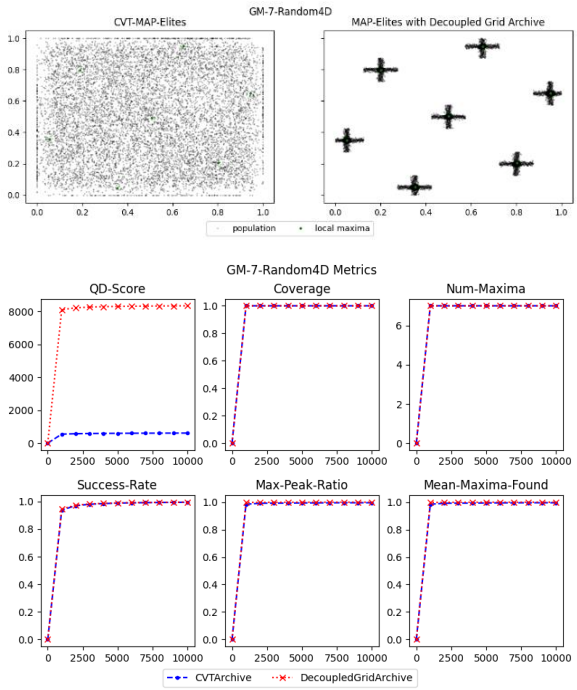
5.7.6 GM-7-Random with 10^7 Evaluations

When being tested on GM-7-Random with 10^7 evaluations both archives performed relatively the same in low dimensions. When being tested on GM-7-Random with 10^7 evaluations, in 7D, neither of the archives managed to locate all the maxima of the problem. The Decoupled Grid seemed to find one maximum more than the CVT-MAP-Elites' archive. In 8D, the CVT-MAP-Elites could not locate any maxima, whereas the Decoupled Grid archive located all the maxima of the problem. In 9D, neither archive managed to find all the maxima. The Decoupled Grid found 6 and the MAP-Elites' archive found only one. In 10D, the Decoupled Grid was the only archive that managed to find maxima.

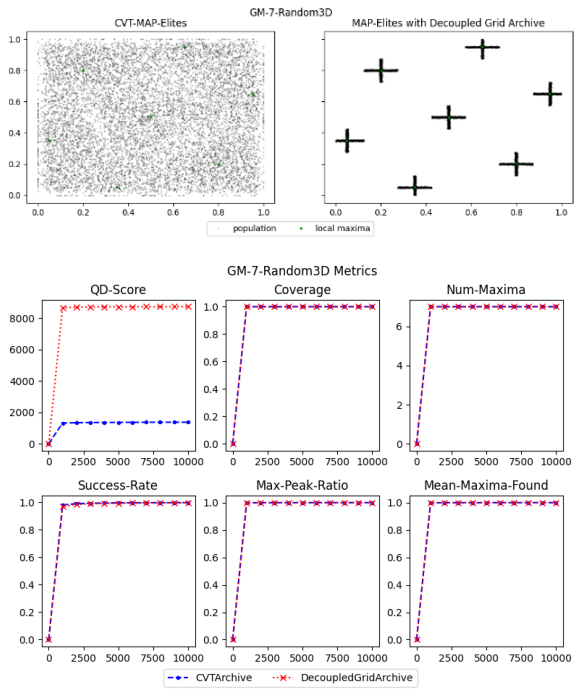
Behaviour in 2D



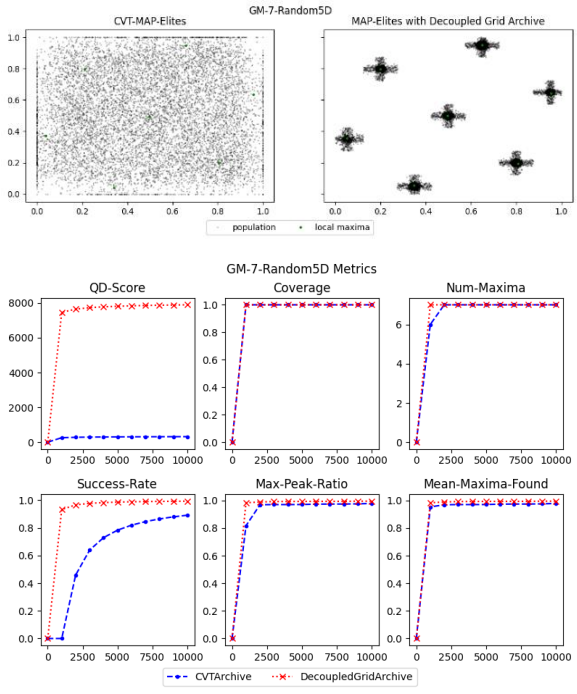
Behaviour in 4D



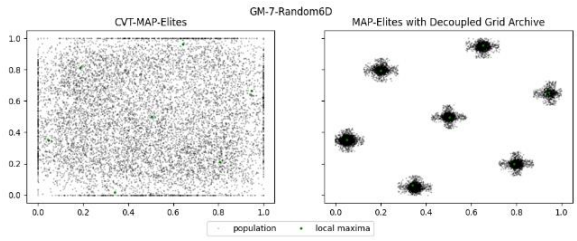
Behaviour in 3D

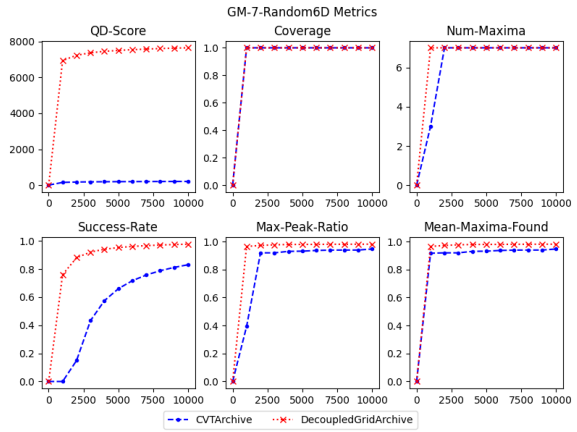


Behaviour in 5D

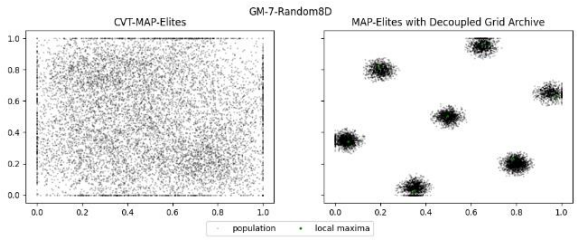


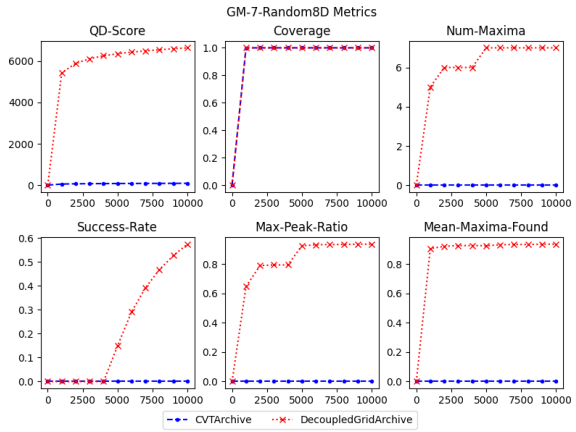
Behaviour in 6D



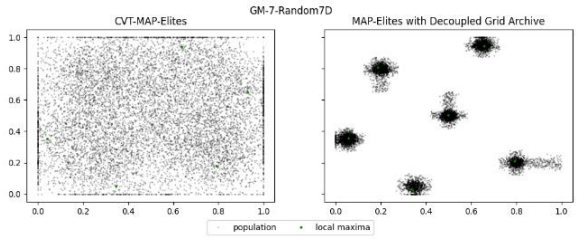


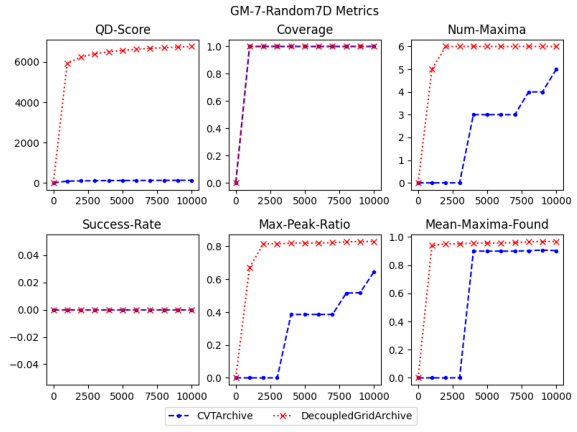
Behaviour in 8D



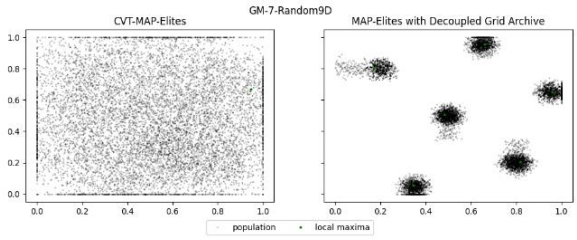


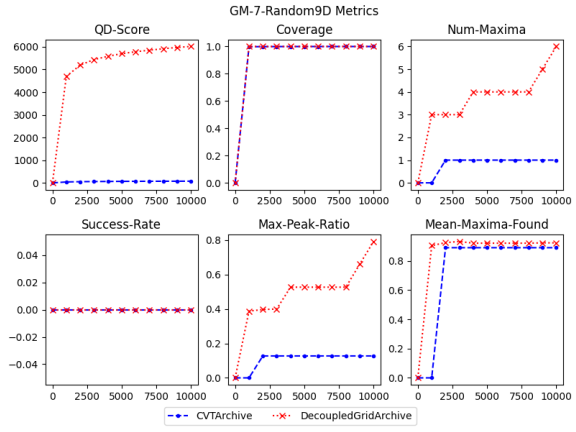
Behaviour in 7D



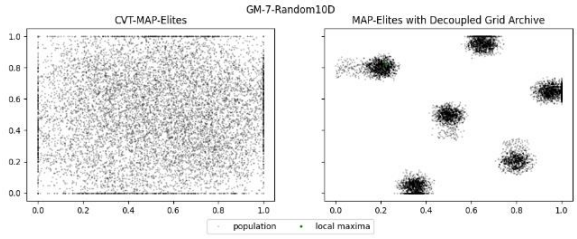


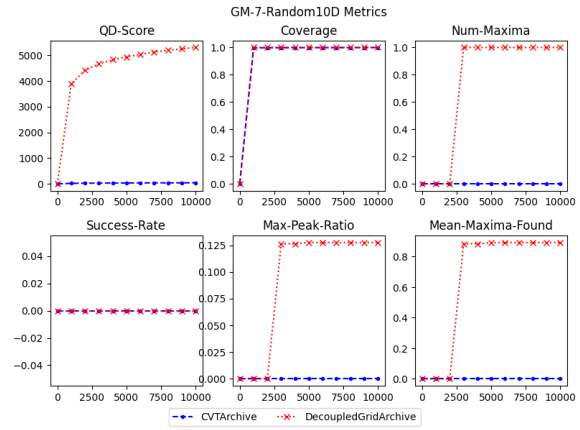
Behaviour in 9D





Behaviour in 10D

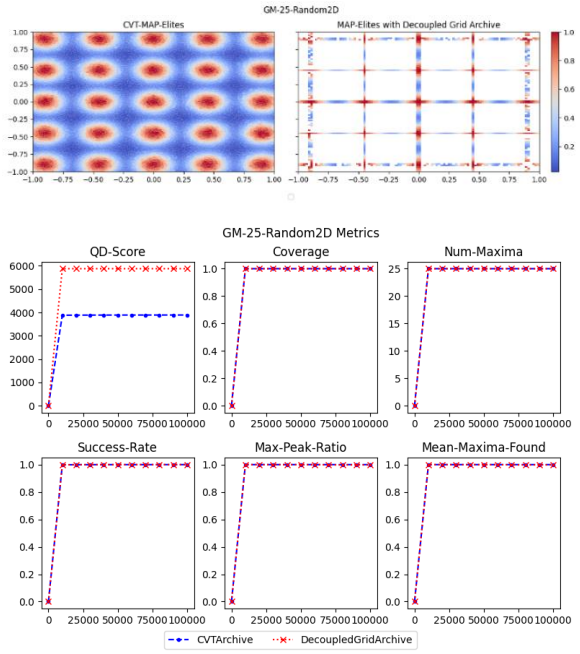




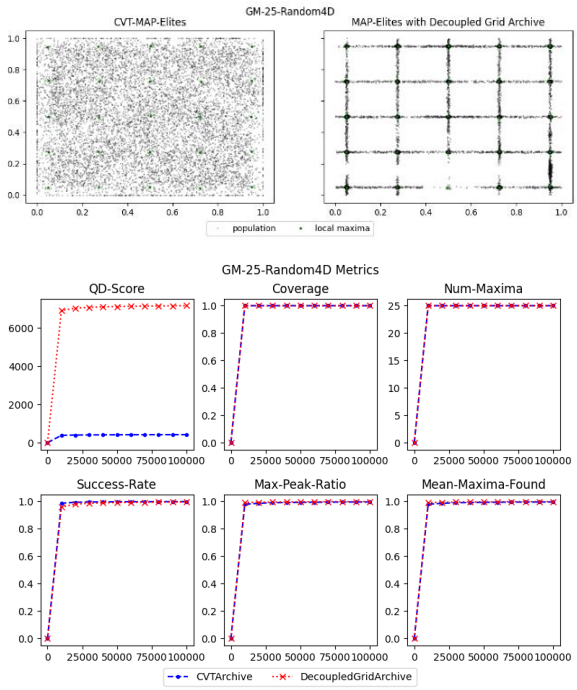
5.7.7 GM-25-Random with 10^8 Evaluations

When being tested on GM-25-Random with 10^8 evaluations both archives performed relatively the same for low dimensions. When being tested on GM-25-Random, both archives managed to locate all the maxima up to the 4th dimension. In 5D CVT-MAP-Elites' archive found all the maxima of the problem, whereas the Decoupled grid was missing 4. In 6D, the Decoupled Grid was missing 2 maxima and in 7D was missing 7. In 8D, where neither of the archives managed to locate all the maxima, the Decoupled Grid archive found nearly twice as many maxima as the Decoupled Grid. The same holds true for 9D. In 10D, the Decoupled Grid locates half of the problem's maxima, whereas the CVT-MAP-Elites only one.

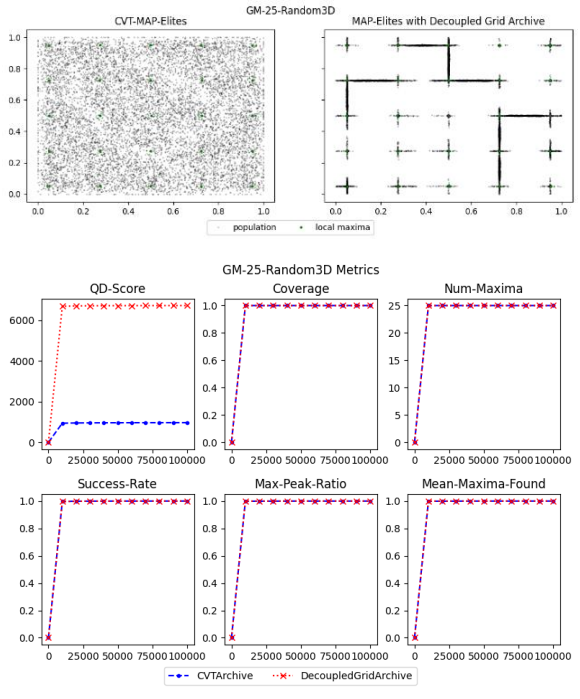
Behaviour in 2D



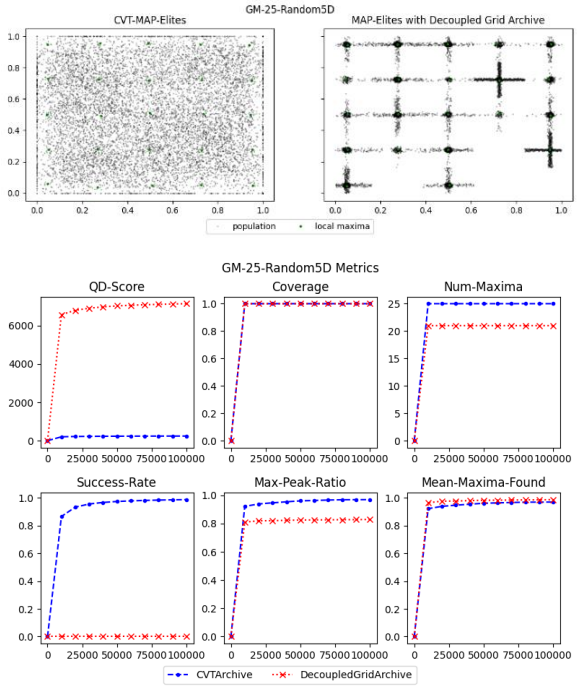
Behaviour in 4D



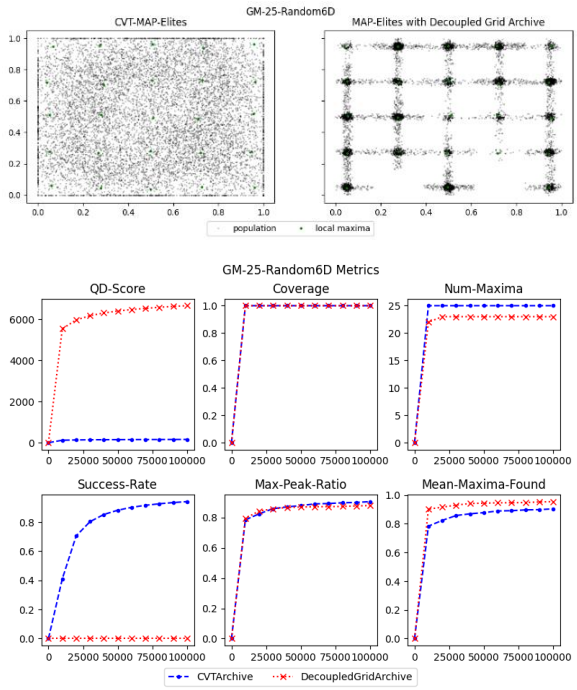
Behaviour in 3D



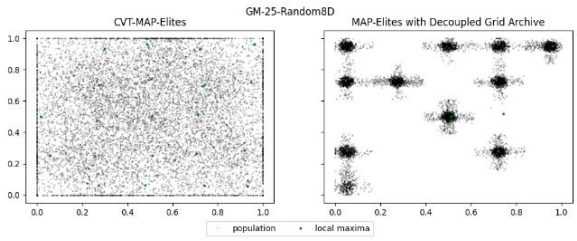
Behaviour in 5D

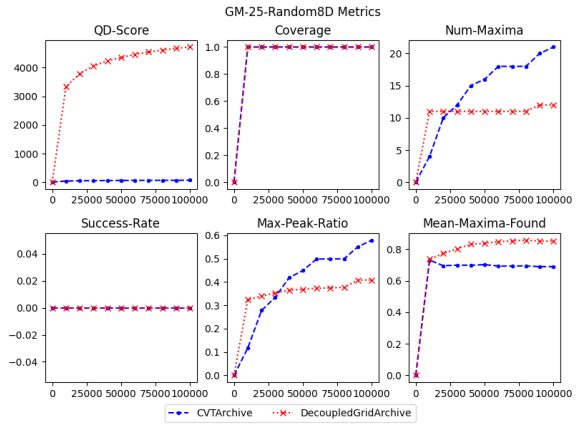


Behaviour in 6D

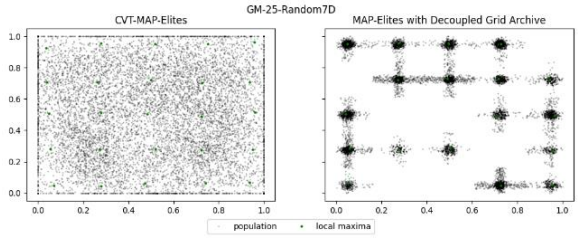


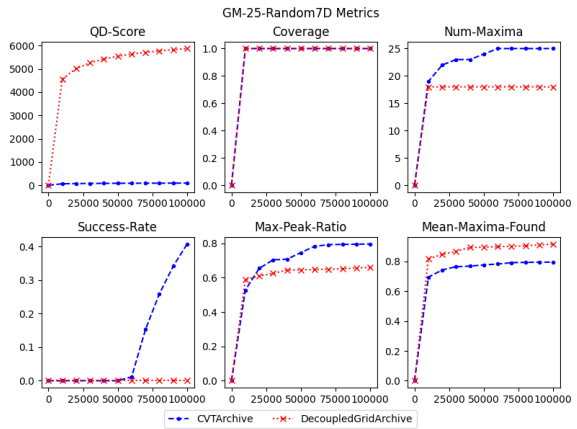
Behaviour in 8D



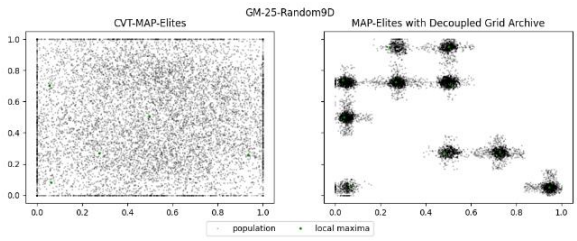


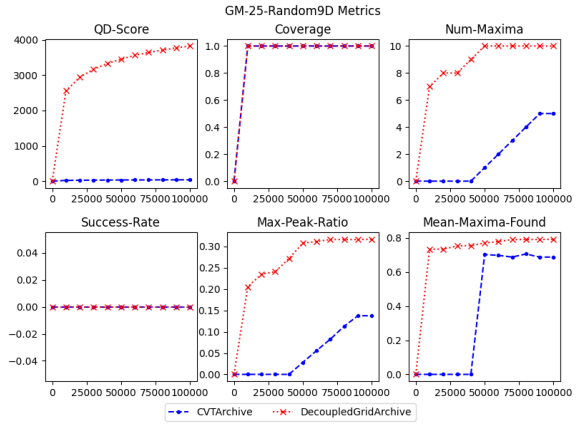
Behaviour in 7D



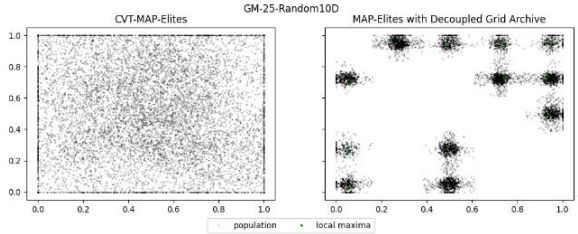


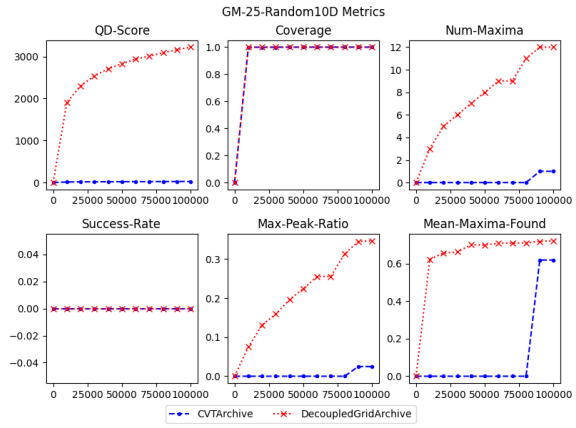
Behaviour in 9D





Behaviour in 10D

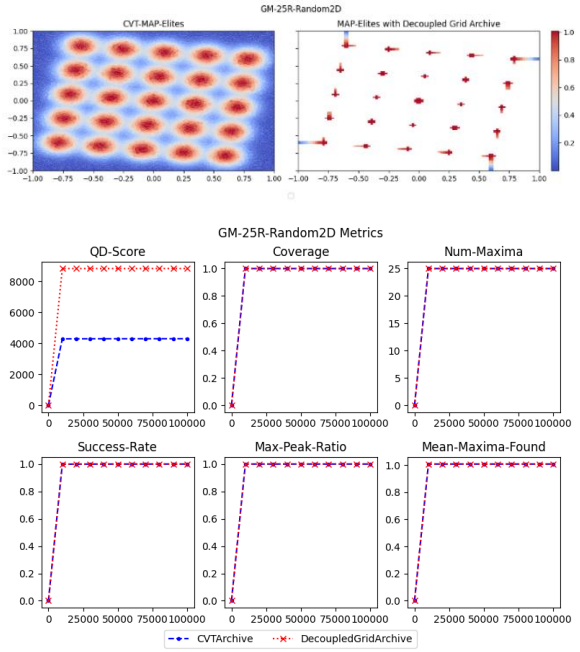




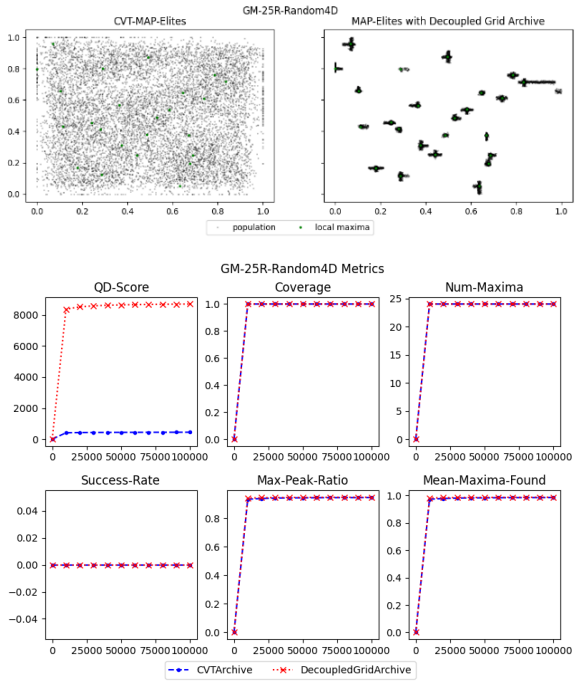
5.7.8 GM-25R-Random with 10^8 Evaluations

When being tested on GM-25R-Random with 10^8 both archives performed relatively the same for low dimensions. In 9D, the Decoupled Grid manages to locate more maxima than the CVT-MAP-Elites and in 10D the Decoupled Grid is the only archive that manages to find any maxima.

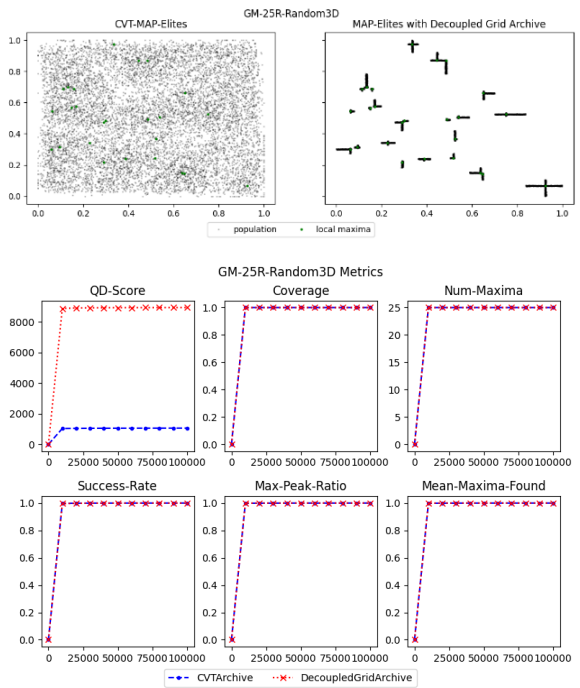
Behaviour in 2D



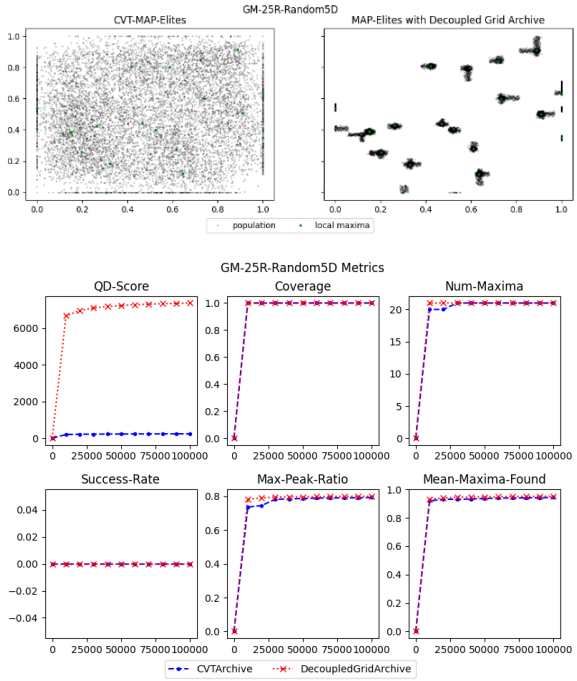
Behaviour in 4D



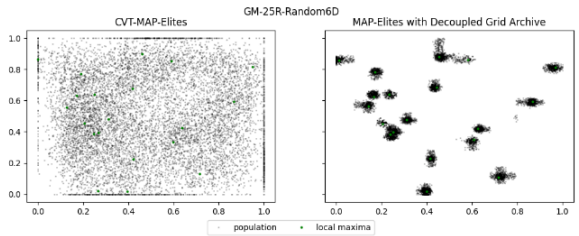
Behaviour in 3D

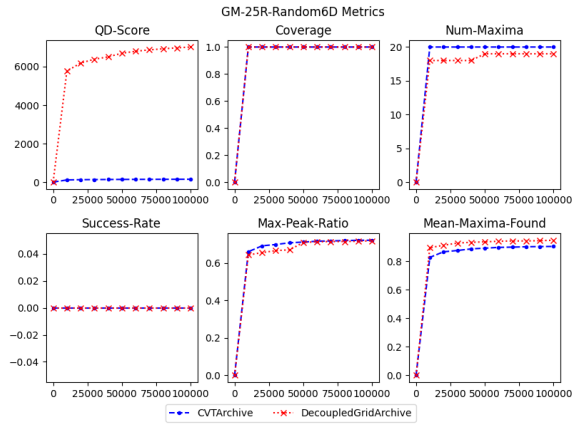


Behaviour in 5D

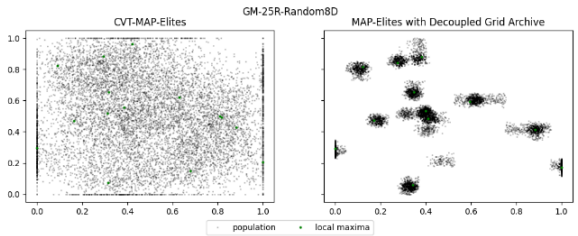


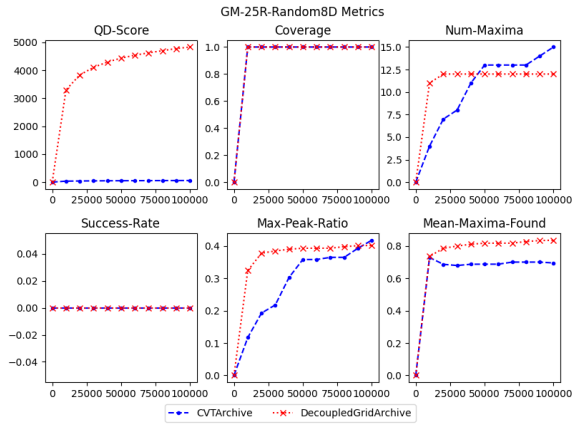
Behaviour in 6D



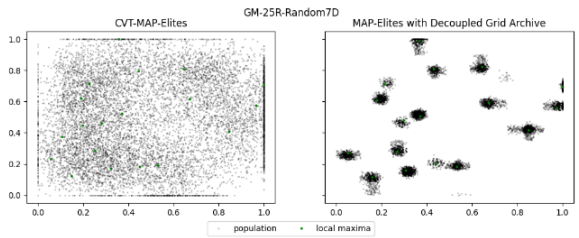


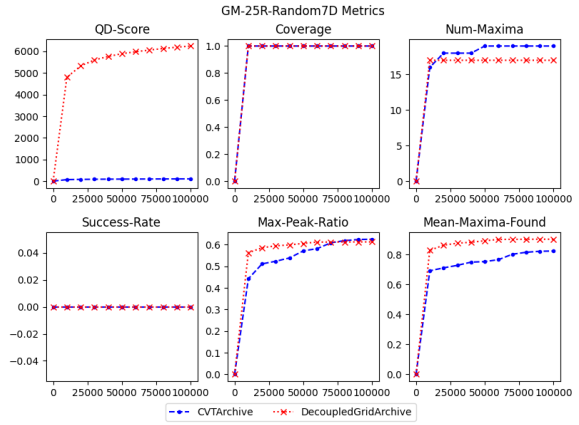
Behaviour in 8D



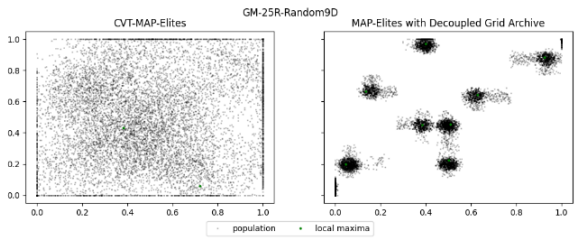


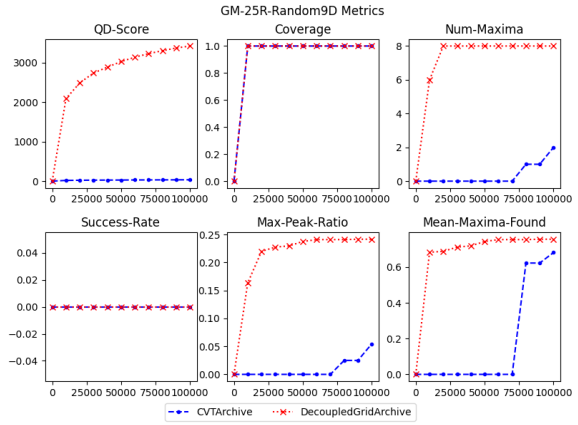
Behaviour in 7D



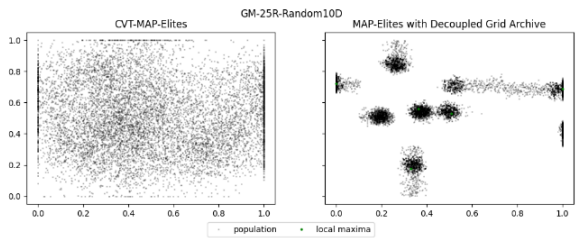


Behaviour in 9D





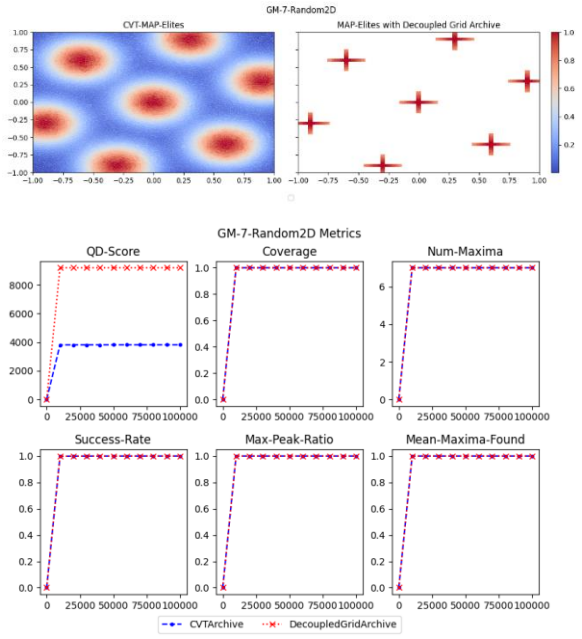
Behaviour in 10D



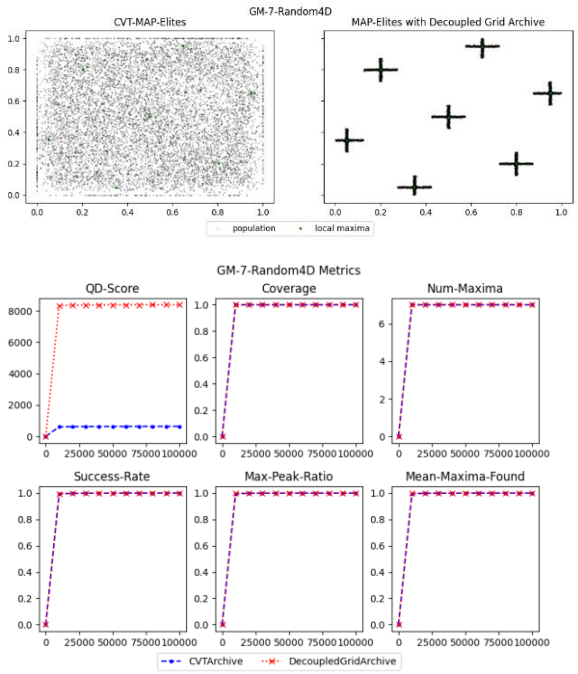
5.7.9 GM-7-Random with 10^8 Evaluations

When being tested on GM-7-Random with 10^8 both archives performed relatively the same for low dimensions. In 7D the CVT-MAP-Elites' archive manages to locate all the maxima, whereas the Decoupled Grid is missing a single maximum. In 8D the Decoupled Grid manages to locate all the maxima, whereas the CVT-MAP-Elites' archive is missing 3 maxima. In 9D neither of the archives manages to locate all the maxima of the problem. The Decoupled Grid is missing a single maximum, whereas the CVT-MAP-Elites' archive is missing 3 maxima. In 10D the Decoupled Grid is the only archive that manages to locate any maxima.

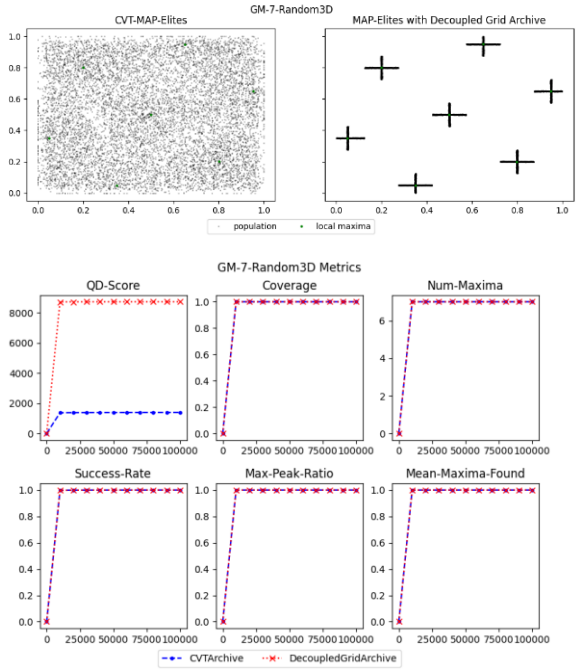
Behaviour in 2D



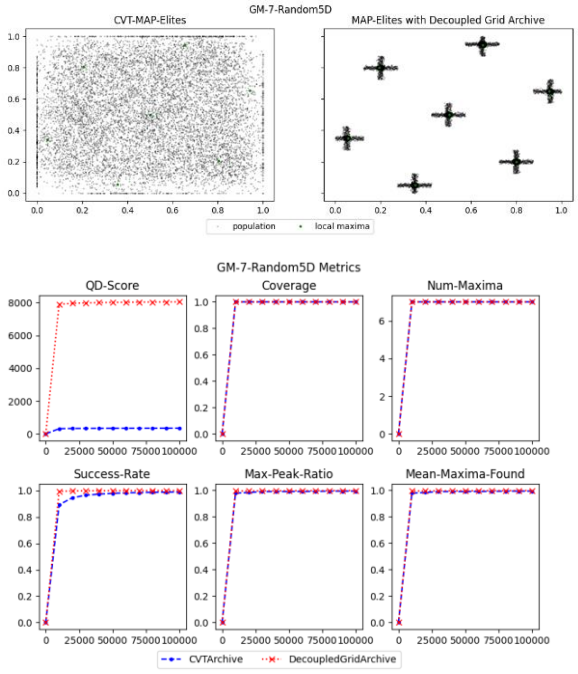
Behaviour in 4D



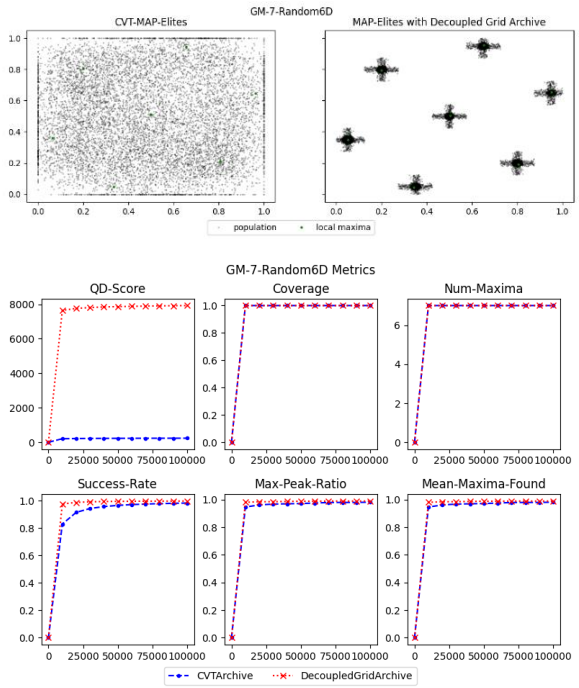
Behaviour in 3D



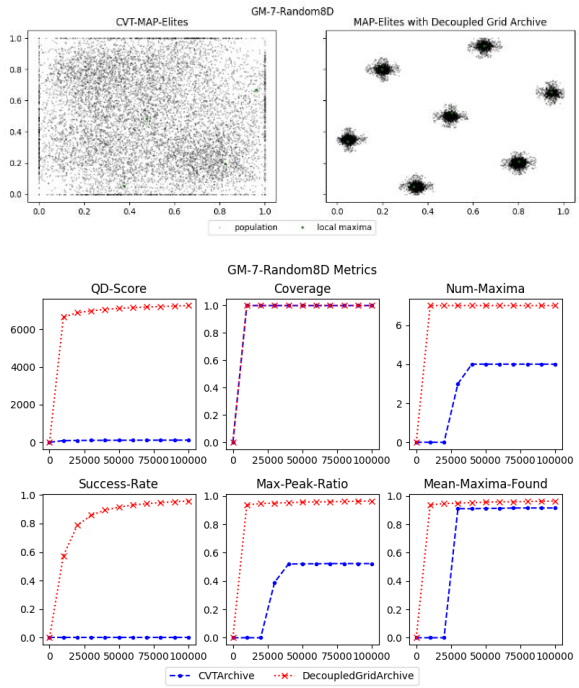
Behaviour in 5D



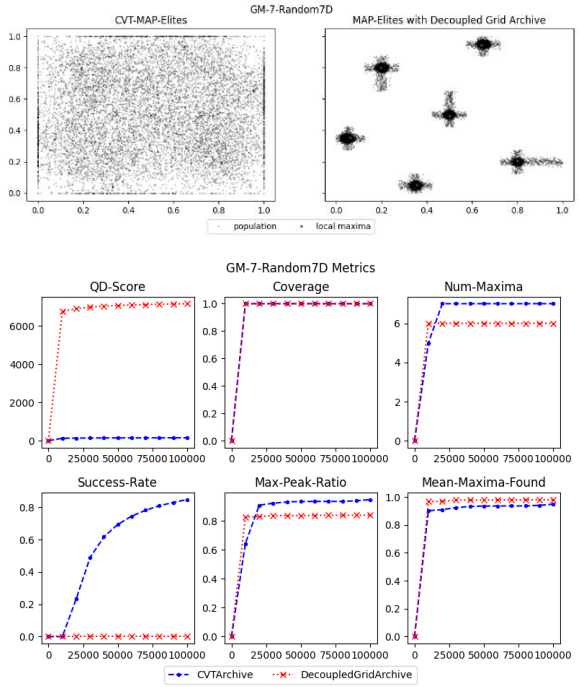
Behaviour in 6D



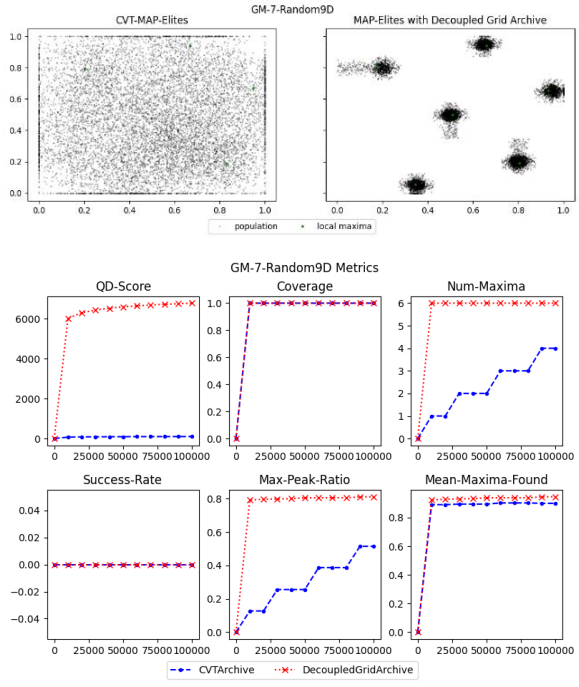
Behaviour in 8D



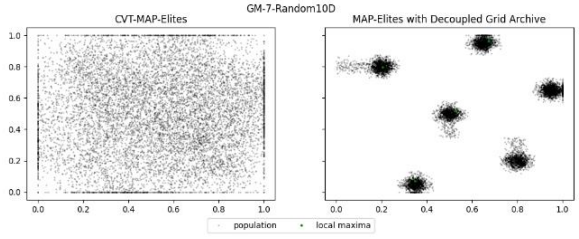
Behaviour in 7D

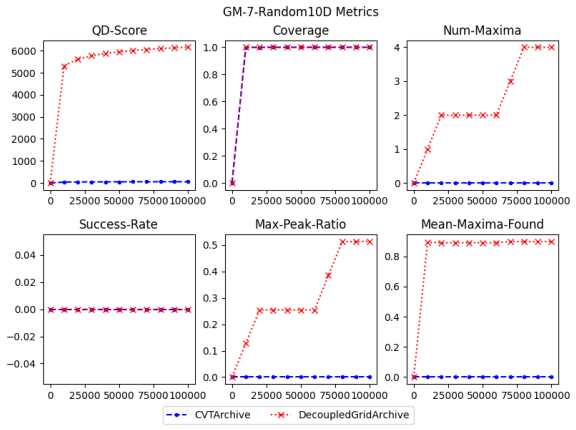


Behaviour in 9D



Behaviour in 10D





5.8 Analysis concerning MMO framework

Each problem needs a certain number of evaluations for an optimisation algorithm to successfully detect the optima of that problem. As the number of dimensions of the problem increases, the number of evaluations needs to increase as well (for the same algorithm to maintain its performance). Consequently, if the number of evaluations is fixed then the algorithms can be compared based on the number of maxima they manage to locate. The table below shows the archive that has manage to either locate most of the maxima or locate the maxima faster than the competing archives – for the specified number of evaluations and the problem at hand.

Evaluations	Problem	Dimensions	Advantageous Archive
10^6	GM-25-Random	2-3 4-5 6	Both CVT-MAP-Elites' Decoupled Grid
10^6	GM-25R-Random	2-5 - 6	Both CVT-MAP-Elites' Decoupled Grid
10^6	GM-7-Random	2-4 - 5-7	Both CVT-MAP-Elites' Decoupled Grid
10^7	GM-25-Random	2-3, 7 4-6 8-10	Both CVT-MAP-Elites' Decoupled Grid
10^7	GM-25R-Random	2-6 - 7-10	Both CVT-MAP-Elites' Decoupled Grid
10^7	GM-7-Random	2-4 - 5-10	Both CVT-MAP-Elites' Decoupled Grid
10^8	GM-25-Random	2-4 5-8 9-10	Both CVT-MAP-Elites' Decoupled Grid
10^8	GM-25R-Random	2-6 7-8 9-10	Both CVT-MAP-Elites' Decoupled Grid
10^8	GM-7-Random	2-6 7 8-10	Both CVT-MAP-Elites' Decoupled Grid

It appears that the Decoupled grid is mostly outperformed by the CVT-MAP-Elites' archive. That is, the CVT-MAP-Elites' archive is consistent in returning the maxima of the problem for an average range of up to six consecutive dimensions. Contrary, the Decoupled Grid archive performs well for a smaller average range of up to five consecutive dimensions.

However, the Decoupled Grid is capable in finding maxima even in dimensions where the given number of evaluations is relatively small for an algorithm to properly examine the domain. In essence, it was observed that the proposed collection was capable in discovering

more maxima than the CVT-MAP-Elites' archive at such scenarios. That is, even at times where the CVT-MAP-Elite would return none. Moreover, it was observed that the Decoupled grid would always have at least a single maximum in its collection – at least for the experiments being tested.

The proposed collection was tested with the GM-25-Random problem to determine whether it was capable in scaling in higher dimensions. Then with the GM-25R-Random problem to determine whether the grid-like layout of the solutions was influencing the performance of the archive; and lastly, with the GM-7-Random problem to further examine the archive in higher dimensions.

The GM-25R-Random problem was used to examine the behaviour of the Decoupled Grid when the solutions of the given problem were not aligned in a grid-like layout. The Decoupled Grid seemed to perform in similar manner as the CVT-MAP-Elites' archive. Consequently, the location of the solutions in the domain of the problem do not influence the ability of the archive in finding the maxima solutions of the problem.

It has been tested with fewer solutions with the help of the GM-7-Random problem, to further examine its behaviour in higher dimensions. The Decoupled Grid seemed to cope with the increase in complexity as it was able to store at least a single maximum in its collection.

It has been noticed that the MAP-Elites algorithm with the Decoupled Grid archive was twice as fast as the CVT-MAP-Elites algorithm with its corresponding archive, in terminating for a specific number of evaluations. That observation was noticed in all the performed experiments.

Despite all that, there are no strong conclusions concerning the Multimodal Optimization framework, and that is because the archive has not been tested among other MMO algorithms nor with a high enough number of dimensions. With the data that has been presented in this section, the Decoupled Grid seems to combine characteristics of both QD and MMO. In that it finds regions in the dimensional space with solutions that are close to the global optima.

Chapter 6: Conclusions

6.1 Summary

All the experiments were executed a single time with a pre-defined seed, due to time limitations. As a result, no rigorous conclusions can be drawn.

The persistent inability of the Decoupled Grid in returning a complete behavioural repertoire for a given problem among the Robotic Arm, Inverted Sphere, Inverted Rastrigin, and Inverted Vincent, is a possible indication that the proposed collection is unsuited for the QD framework. Its behaviour is rather peculiar as it creates a gird-like layout of solutions rather than fully illuminating the behavioural space.

With respect to the Multimodal Optimization nature of the archive, not much can be deduced. The collection seems to detect some local maxima of the given problem; however, it applies a high selection pressure to each of the N independent lists that form the archive. This results, in the fast and constant replacement of lesser fit solutions with fitter – even if the solutions are considered (local) maxima. Consequently, the archive inevitably seems to strive in finding the best solution, which corresponds to the global maxima of the problem.

Additionally, the proposed collection seems to outperform the CVT-MAP-Elites in terms of speed for finding all the global maxima of a given problem. That is mainly because the collection merely extends the MAP-Elites algorithm, which itself is faster than CVT-MAP-Elites in high dimensions, due to its faster ability in getting the index of the elite within the archive [13].

Lastly, the new collection might not entirely fall under the QD framework and questionably fit the MMO framework, yet it does seem to combine characteristics of the two. The reason being that the Decoupled Grid seems to find regions in the dimensional space with solutions that are close to the global optima.

6.2 Future work

The Decoupled Grid archive does not seem like a promising alternative to the N-Dimensional Grid of MAP-Elite's algorithm. However, this conclusion is based on experiments of a single execution with a predefined seed. Multiple executions in a fully randomised simulated environment, are needed for a definite conclusion concerning the framework of QD optimisation.

With regards to Multimodal Optimization, the proposed collection could be compared with other multimodal optimization algorithms such as the Clearing algorithm [16] or the Restricted Tournament Selection algorithm [17]. When testing the new collection in this framework it would be ideal if the problems being tested were not separable, by (for instance) testing the algorithms in simulated environments.

The QD algorithms in this work were tested with a uniform selection among the elites. One could examine the behaviour of the Decoupled Grid under the influence of other selections. That is to select parents in other ways that can make the convergence faster such as the "Iso+LineDD" [14].

The QD-Score and the Coverage metrics both seemed to favour the Decoupled Grid archive. That is mainly because the archive does not behave in similar manner to the N-Dimensional Grid. As such, for a better comparison, one could map all the solutions of the proposed collection into an N-Dimensional Grid and then measure the metrics based on that grid.

References

[1] K. Chatzilygeroudis, A. Cully, V. Vassiliades and J.-B. Mouret, "Quality-Diversity Optimization: a novel branch of stochastic optimization," in *Black Box Optimization, Machine Learning, and No-Free Lunch Theorems*, Springer, 2021, pp. 109-135.

[2] C. Antoine, C. Jeff, T. Danesh and B. M. Jean, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503-507, May 2015.

[3] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *arXiv preprint arXiv: 1504.04909*, 2015.

[4] V. Vassiliades, K. Chatzilygeroudis and J.-B. Mouret, "Using Centroidal Voronoi Tessellations to Scale Up the Multidimensional Archive of Phenotypic Elites Algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 22, pp. 623-630, 2018.

[5] X. Yu and M. Gen, in *Introduction to evolutionary algorithms*, Springer Science & Business Media, 2010.

[6] M. Preuss, "EA Techniques for Multimodal Problems," in *Multimodal Optimization by Means of Evolutionary Algorithms*, 1st, Ed., Springer Publishing Company, Incorporated, 2015.

[7] L. Joel and S. Kenneth, "Exploiting Open-Endedness to Solve Problems Through the Search for Novelty," *Artificial Life - ALIFE*, January 2008.

[8] J. Lehman and K. O. Stanley, "Evolving a Diversity of Creatures through Novelty Search and Local Competition," in *GECCO*, New York, 2011.

[9] J. K. Pugh, L. B. Soros and K. O. Stanley, "Quality diversity: A new frontier for evolutionary computation," *Frontiers in Robotics and AI*, vol. 3, p. 40, 2016.

[10] A. Cully and Y. Demiris, "Quality and diversity optimization: A unifying modular framework," *IEEE Transactions on Evolutionary Computation*, vol. 22, pp. 245-259, 2018.

[11] K. Chatzilygeroudis, V. Vassiliades and J.-B. Mouret, "Reset-free trial-and-error learning for robot damage recovery," *Robotics and Autonomous Systems*, vol. 100, pp. 236-250, 2018.

[12] A. Khalifa, S. Lee, A. Nealen and J. Togelius, "Talakat: Bullet Hell Generation through Constrained Map-Elites," in *Proceedings of the Genetic and Evolutionary Computation Conference*, New York, NY, USA, Association for Computing Machinery, 2018, p. 1047–1054.

[13] Q. Du, V. Faber and M. Gunzburger, "Centroidal Voronoi Tessellations: Applications and Algorithms," *SIAM Review*, vol. 41, no. 4, pp. 637-676, 1999.

[14] V. Vassiliades and J.-B. Mouret, "Discovering the Elite Hypervolume by Leveraging Interspecies Correlation," in *Proceedings of the Genetic and Evolutionary Computation Conference*, vol. abs/1804.03906, New York, NY, USA, Association for Computing Machinery, 2018, p. 149–156}.

[15] M. Preuss, "Niching the CMA-ES via Nearest-Better Clustering," in *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation*, New York, NY, USA, Association for Computing Machinery, 2010, p. 1711–1718.

[16] A. Petrowski, "A clearing procedure as a niching method for genetic algorithms," in *Proceedings of IEEE International Conference on Evolutionary Computation*, 1996, pp. 798-803.

[17] G. R. Harik, "Finding Multimodal Solutions Using Restricted Tournament Selection," in *Proceedings of the 6th International Conference on Genetic Algorithms*, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 1995, p. 24–31.

[18] A. Cully and J.-B. Mouret, “Behavioral Repertoire Learning in Robotics,” in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, ACM, 2013, p. 175–182.

[19] J. Clune, J.-B. Mouret and H. Lipson, “The evolutionary origins of modularity,” *Proceedings of the Royal Society B: Biological Sciences*, vol. 280, no. 1755, 2013.

[20] A. Cully, J. Clune, D. Tarapore and J.-B. Mouret, “Robots that can adapt like animals,” *Nature*, vol. 521, no. 7553, pp. 503-507, 2015.

[21] S. Doncieux and J.-B. Mouret, “Behavioral diversity measures for Evolutionary Robotics,” in *CEC. IEEE*, Barcelona, 2010.

[22] M. Flageat and A. Cully, “Fast and stable MAP-Elites in noisy domains using deep grids,” in *Proceeding of the Alife conference*, 2020.

[23] J. Lehman and K. O. Stanley, “Abandoning Objectives: Evolution Through the Search for Novelty Alone,” *Evolutionary Computation*, vol. 19, pp. 189-223, 2011.

[24] N. Casas, “Genetic algorithms for multimodal optimization: a review,” *arXiv preprint arXiv:1508.05342*, 2015.

[25] P. A. Vikhar, “Evolutionary algorithms: A critical review and its future prospects,” in *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, 2016, pp. 261-265.

[26] V. Vassiliades, K. Chatzilygeroudis and J.-B. Mouret, “Comparing Multimodal Optimization and Illumination,” in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, New York, NY, USA, Association for Computing Machinery, 2017, p. 97–98.

[27] J.-B. Mouret, “Novelty-based multiobjectivization,” in *New horizons in evolutionary robotics*, Springer, 2011, pp. 139-154.

Appendix A

The pyribs library was extended to support the Decoupled Grid archive. These are the contents of _decoupled_grid_archive.py

```
"""Contains the DecoupledGridArchive."""

import numpy as np
import numba as nb
from ribs.archives import ArchiveBase, AddStatus, Elite
from ribs.archives._archive_base import RandomBuffer, readonly
from numpy.linalg import norm

_EPSILON = 1e-6


class DecoupledGridArchive(ArchiveBase):
    """An archive that decouples each dimension into independent
    uniformly-sized bins.

    Args:
        dims (array-like of int): Number of bins in each dimension of the
            behavior space, e.g. ``[20, 30, 40]`` indicates there should be 3
            dimensions with 20, 30, and 40 bins. (The number of dimensions is
            implicitly defined in the length of this argument).
        ranges (array-like of (float, float)): Upper and lower bound of each
            dimension of the behavior space, e.g. ``[(-1, 1), (-2, 2)]``
            indicates the first dimension should have bounds :math:`[-1,1]`
            (inclusive), and the second dimension should have bounds
            :math:`[-2,2]` (inclusive). ``ranges`` should be the same length as
            ``dims``.
        seed (int): Value to seed the random number generator. Set to None to
            avoid a fixed seed.
        dtype (str or data-type): Data type of the solutions, objective values,
            and behavior values. We only support ``"f"`` / :class:`np.float32`
            and ``"d"`` / :class:`np.float64`.
    Raises:
        ValueError: ``dims`` and ``ranges`` are not the same length.
    """

    def __init__(self, dims, ranges, seed=None, dtype=np.float64):
        self._dims = np.array(dims)
        if len(self._dims) != len(ranges):
            raise ValueError(f"dims (length {len(self._dims)}) and ranges "
                             f"(length {len(ranges)}) must be the same length")

        ArchiveBase.__init__(
            self,
            storage_dims=tuple([len(self._dims), np.max(self._dims)]),
            behavior_dim=len(self._dims),
            seed=seed,
            dtype=dtype,
        )
```

```
        self._unique_occupied_indices = None
        self._unique_occupied_indices_cols = None

        ranges = list(zip(*ranges))
        self._lower_bounds = np.array(ranges[0], dtype=self.dtype)
        self._upper_bounds = np.array(ranges[1], dtype=self.dtype)
        self._interval_size = self._upper_bounds - self._lower_bounds

        self._boundaries = []
        for dim, lower_bound, upper_bound in zip(self._dims,
self._lower_bounds,
                                                 self._upper_bounds):
            self._boundaries.append(
                np.linspace(lower_bound, upper_bound, dim + 1))

    @property
    def dims(self):
        """(behavior_dim,) numpy.ndarray: Number of bins in each dimension."""
        return self._dims

    @property
    def lower_bounds(self):
        """(behavior_dim,) numpy.ndarray: Lower bound of each dimension."""
        return self._lower_bounds

    @property
    def upper_bounds(self):
        """(behavior_dim,) numpy.ndarray: Upper bound of each dimension."""
        return self._upper_bounds

    @property
    def interval_size(self):
        """(behavior_dim,) numpy.ndarray: The size of each dim (upper_bounds -
        lower_bounds)."""
        return self._interval_size

    @property
    def boundaries(self):
        """list of numpy.ndarray: The boundaries of the bins in each dimension.

        Entry ``i`` in this list is an array that contains the boundaries of the
        bins in dimension ``i``. The array contains ``self.dims[i] + 1`` entries
        laid out like this::

            Archive bins:   | 0 | 1 |   ...   |    self.dims[i]    |
            boundaries[i]:  0   1   2   self.dims[i] - 1     self.dims[i]

        Thus, ``boundaries[i][j]`` and ``boundaries[i][j + 1]`` are the lower
        and upper bounds of bin ``j`` in dimension ``i``. To access the lower
        bounds of all the bins in dimension ``i``, use ``boundaries[i][:-1]``,
        and to access all the upper bounds, use ``boundaries[i][1:]``.
        """
```

```
        return self._boundaries

    def initialize(self, solution_dim):
        """Initializes the archive by allocating storage space.

        Child classes should call this method in their implementation if they
        are overriding it.

        Args:
            solution_dim (int): The dimension of the solution space.
        Raises:
            RuntimeError: The archive is already initialized.
        """
        if self._initialized:
            raise RuntimeError("Cannot re-initialize an archive")
        self._initialized = True
        self._rand_buf = RandomBuffer(self._seed)
        self._solution_dim = solution_dim
        self._occupied = np.zeros(self._storage_dims, dtype=bool)
        self._solutions = np.empty((*self._storage_dims, solution_dim),
                                   dtype=self.dtype)
        self._objective_values = np.empty(self._storage_dims, dtype=self.dtype)
        self._behavior_values = np.empty(
            (*self._storage_dims, self._behavior_dim), dtype=self.dtype)
        self._metadata = np.empty(self._storage_dims, dtype=object)
        self._occupied_indices = []
        self._occupied_indices_cols = tuple(
            [] for _ in range(len(self._storage_dims)))
        self._unique_occupied_indices = set()
        self._unique_occupied_indices_cols = tuple(
            [] for _ in range(len(self._storage_dims)))

        self._stats_reset()
        self._state = {"clear": 0, "add": 0}

    @staticmethod
    @nb.jit(nopython=True)
    def _get_index_numba(behavior_values, upper_bounds, lower_bounds,
                         interval_size, dims):
        """Numba helper for get_index().

        See get_index() for usage.
        """
        # Adding epsilon to behavior values accounts for floating point
        # precision errors from transforming behavior values. Subtracting
        # epsilon from upper bounds makes sure we do not have indices outside
        # the grid.
        behavior_values = np.minimum(
            np.maximum(behavior_values + _EPSILON, lower_bounds),
            upper_bounds - _EPSILON)

        index = (behavior_values - lower_bounds) / interval_size * dims
        return index.astype(np.int32)
```

```
def get_index(self, behavior_values):
    """Returns indices of the behavior values within the archive's grid.

    First, values are clipped to the bounds of the behavior space. Then, the
    values are mapped to bins; e.g. bin 5 along dimension 0 and bin 3 along
    dimension 1.

    The indices can be used to access boundaries of a behavior value's bin.
    For example, the following retrieves the lower and upper bounds of the
    bin along dimension 0::

        idx = archive.get_index(...)  # Other methods also return indices.
        lower = archive.boundaries[0][idx[0]]
        upper = archive.boundaries[0][idx[0] + 1]

    See :attr:`boundaries` for more info.

    Args:
        behavior_values (numpy.ndarray): (:attr:`behavior_dim`,) array of
            coordinates in behavior space.
    Returns:
        tuple of tuples: The grid indices.
    """
    index = DecoupledGridArchive._get_index_numba(
        behavior_values,
        self._upper_bounds,
        self._lower_bounds,
        self._interval_size, self._dims
    )
    return tuple([(i, index[i]) for i in range(len(self._dims))])

def elite_with_behavior(self, behavior_values):
    """Gets the elite with behavior vals in the same bin as those specified.

    Since :namedtuple:`Elite` is a namedtuple, the result can be unpacked
    (here we show how to ignore some of the fields)::

        sol, obj, beh, *_ = archive.elite_with_behavior(...)

    Or the fields may be accessed by name::

        elite = archive.elite_with_behavior(...)
        elite.sol
        elite.obj
        ...

    Args:
        behavior_values (array-like): Coordinates in behavior space.
    Returns:
        Elite:
          * If there is an elite with behavior values in the same bin as
            those specified, this :namedtuple:`Elite` holds the info for
```

```
            that elite. In that case, ``beh`` (the behavior values) may not
            be exactly the same as the behavior values specified since the
            elite is only guaranteed to be in the same archive bin.
          * If no such elite exists, then all fields of the
            :namedtuple:`Elite` are set to None. This way, tuple unpacking
            (e.g.
            ``sol, obj, beh, idx, meta = archive.elite_with_behavior(...)``)
            still works.
        """
        best_elite_idx = None
        best_elite_dist = np.inf

        for index in self.get_index(np.asarray(behavior_values)):
            if self._occupied[index]:
                new_elite_dist = norm(
                    self._behavior_values[index] - behavior_values)
                if new_elite_dist < best_elite_dist:
                    best_elite_dist = new_elite_dist
                    best_elite_idx = index

        if best_elite_idx is None:
            return Elite(None, None, None, None, None)

        return Elite(
            readonly(self._solutions[best_elite_idx]),
            self._objective_values[best_elite_idx],
            readonly(self._behavior_values[best_elite_idx]),
            best_elite_idx,
            self._metadata[best_elite_idx],
        )

    def get_random_unique_elite(self):
        """Selects an elite uniformly at random,
        from one of the unique elites, of the archive's bins.

        Since :namedtuple:`Elite` is a namedtuple, the result can be unpacked
        (here we show how to ignore some of the fields)::

            sol, obj, beh, *_ = archive.get_random_elite()

        Or the fields may be accessed by name::

            elite = archive.get_random_elite()
            elite.sol
            elite.obj
            ...

        Returns:
            Elite: A randomly selected elite from the archive.
        Raises:
            IndexError: The archive is empty.
        """
```

```
        if self.empty:
            raise IndexError("No elements in archive.")

        random_idx = self._rand_buf.get(len(self._unique_occupied_indices))
        index = self._unique_occupied_indices[random_idx]

        return Elite(
            readonly(self._solutions[index]),
            self._objective_values[index],
            readonly(self._behavior_values[index]),
            index,
            self._metadata[index],
        )

    @staticmethod
    @nb.jit(locals={"already_occupied": nb.types.b1}, nopython=True)
    def _add_numba(new_index, new_solution, new_objective_value,
                   new_behavior_values, occupied, solutions, objective_values,
                   behavior_values):
        """Numba helper for inserting solutions into the archive.

        See add() for usage.

        Returns:
            was_inserted (bool): Whether the new values were inserted into the
                archive.
            already_occupied (bool): Whether the index was occupied prior
                to this call; i.e. this is True only if there was already an
                item at the index.
        """
        already_occupied = occupied[new_index]
        if (not already_occupied or
                objective_values[new_index] < new_objective_value):
            # Track this index if it has not been seen before -- important that
            # we do this before inserting the solution.
            if not already_occupied:
                occupied[new_index] = True

            # Insert into the archive.
            objective_values[new_index] = new_objective_value
            behavior_values[new_index] = new_behavior_values
            solutions[new_index] = new_solution

            return True, already_occupied

        return False, already_occupied

    def add(self, solution, objective_value, behavior_values, metadata=None):
        """Attempts to insert a new solution into the archive.

        The solution is only inserted if it has a higher ``objective_value``
        than any of the elites previously in the corresponding bins.
```

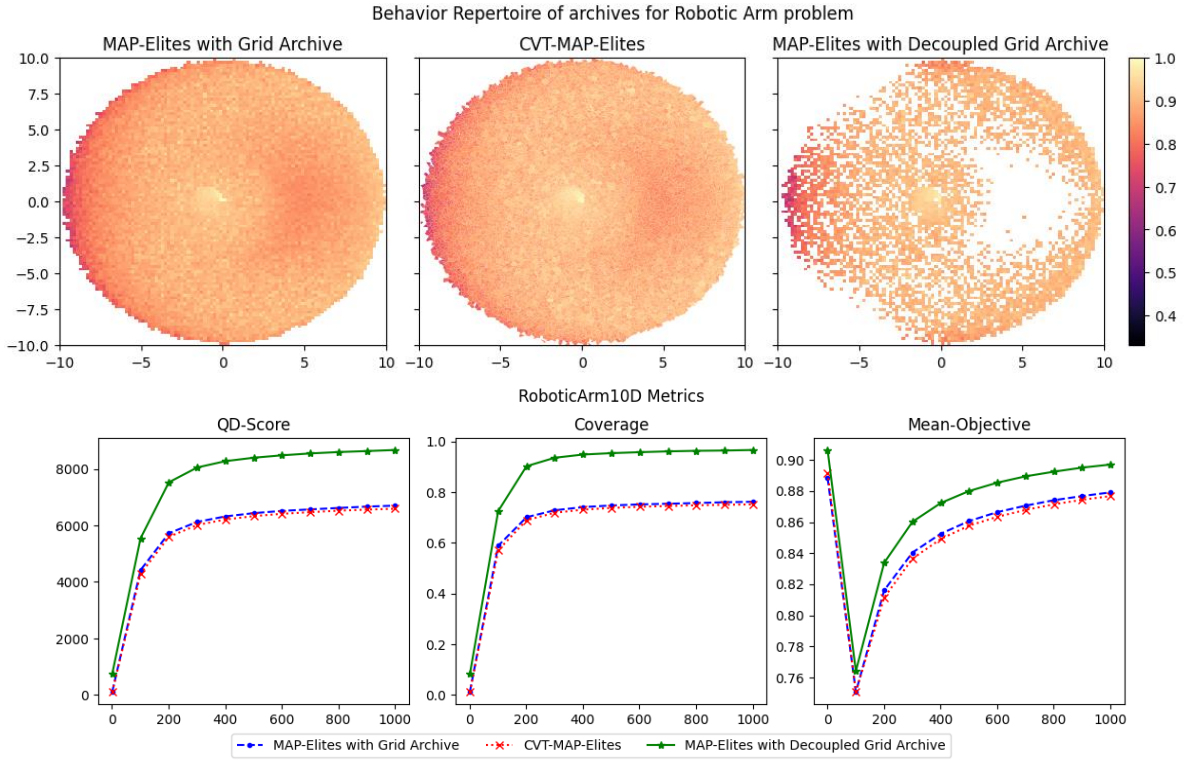
```
    Args:
        solution (array-like): Parameters of the solution.
        objective_value (float): Objective function evaluation of the
            solution.
        behavior_values (array-like): Coordinates in behavior space of the
            solution.
        metadata (object): Any Python object representing metadata for the
            solution. For instance, this could be a dict with several
            properties.
    Returns:
        tuple: 2-element tuple describing the result of the add operation.
        These outputs are particularly useful for algorithms such as CMA-ME.

            **status** (:class:`AddStatus`): See :class:`AddStatus`.

            **value** (:attr:`dtype`): The meaning of this value depends on
            the value of ``status``:

            - ``NOT_ADDED`` -> the "negative improvement," i.e. objective
              value of solution passed in minus objective value of the
              solution still in the archive (this value is negative because
              the solution did not have a high enough objective value to be
              added to the archive)
            - ``IMPROVE_EXISTING`` -> the "improvement," i.e. objective
              value of solution passed in minus objective value of solution
              previously in the archive
            - ``NEW`` -> the objective value passed in
    """
    self._state["add"] += 1
    solution = np.asarray(solution)
    behavior_values = np.asarray(behavior_values)
    objective_value = self.dtype(objective_value)
    status, values = AddStatus.NOT_ADDED, []

    index = self.get_index(behavior_values)

    for individual_index in index:
        old_objective = self._objective_values[individual_index]

        was_inserted, already_occupied = self._add_numba(
            individual_index, solution, objective_value, behavior_values,
            self._occupied, self._solutions, self._objective_values,
            self._behavior_values
        )

        if was_inserted:
            self._metadata[individual_index] = metadata

        if was_inserted and not already_occupied:
            self._add_occupied_index(individual_index)
            self._add_unique_occupied_index(individual_index)
            status = AddStatus.NEW
            values.append(objective_value)
```

```
                self._stats_update(self.dtype(0.0), objective_value)
            elif was_inserted and already_occupied:
                status = max(AddStatus.IMPROVE_EXISTING, status)
                values.append(objective_value - old_objective)
                self._stats_update(old_objective, objective_value)
            else:
                status = max(AddStatus.NOT_ADDED, status)
                values.append(objective_value - old_objective)

        return status, np.mean(values)

    def _add_unique_occupied_index(self, index):
        """Adds a new index to the lists of occupied indices."""
        self._unique_occupied_indices.add(index)

        # Some archives (e.g. CVTArchive) have a 1D index and use ints instead
        # of tuples, so we convert to a singleton tuple here.
        if not isinstance(index, tuple):
            index = (index,)

        for i, idx in enumerate(index):
            self._unique_occupied_indices_cols[i].append(idx)
```

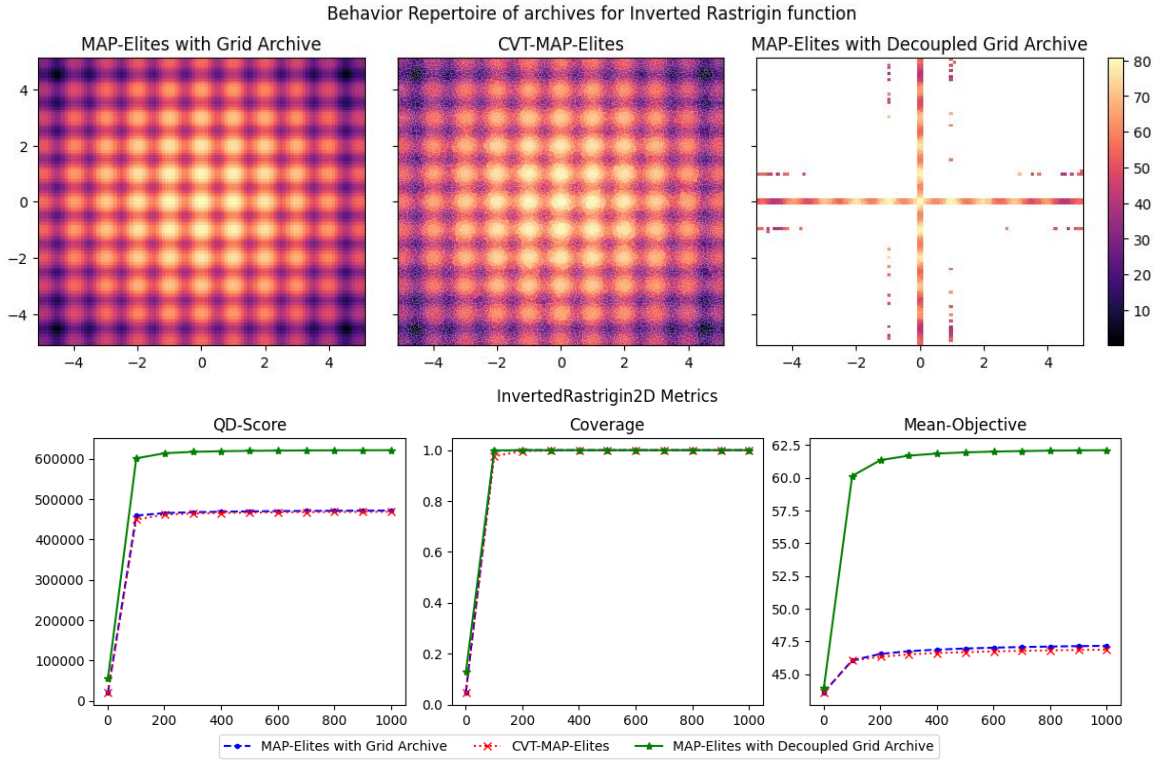
Appendix B

The results of the uniform selection, namely, when the decoupled grid filters the duplicate solutions.

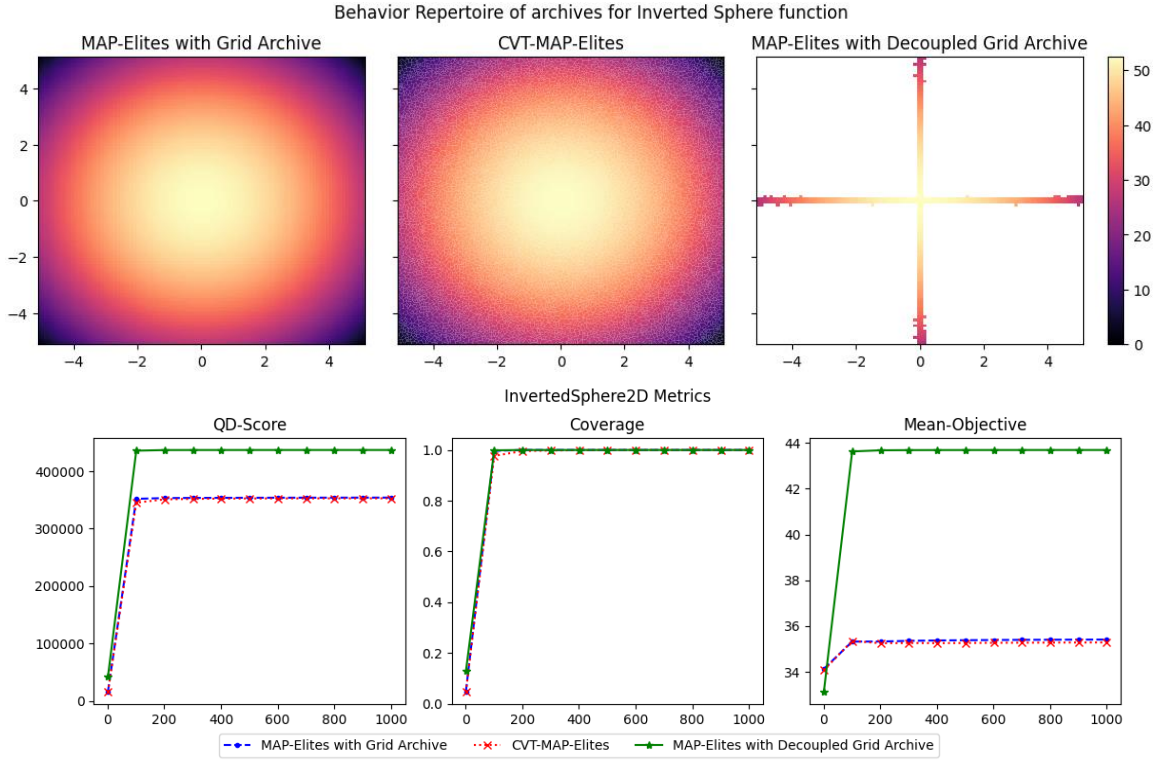
2D Robotic Arm



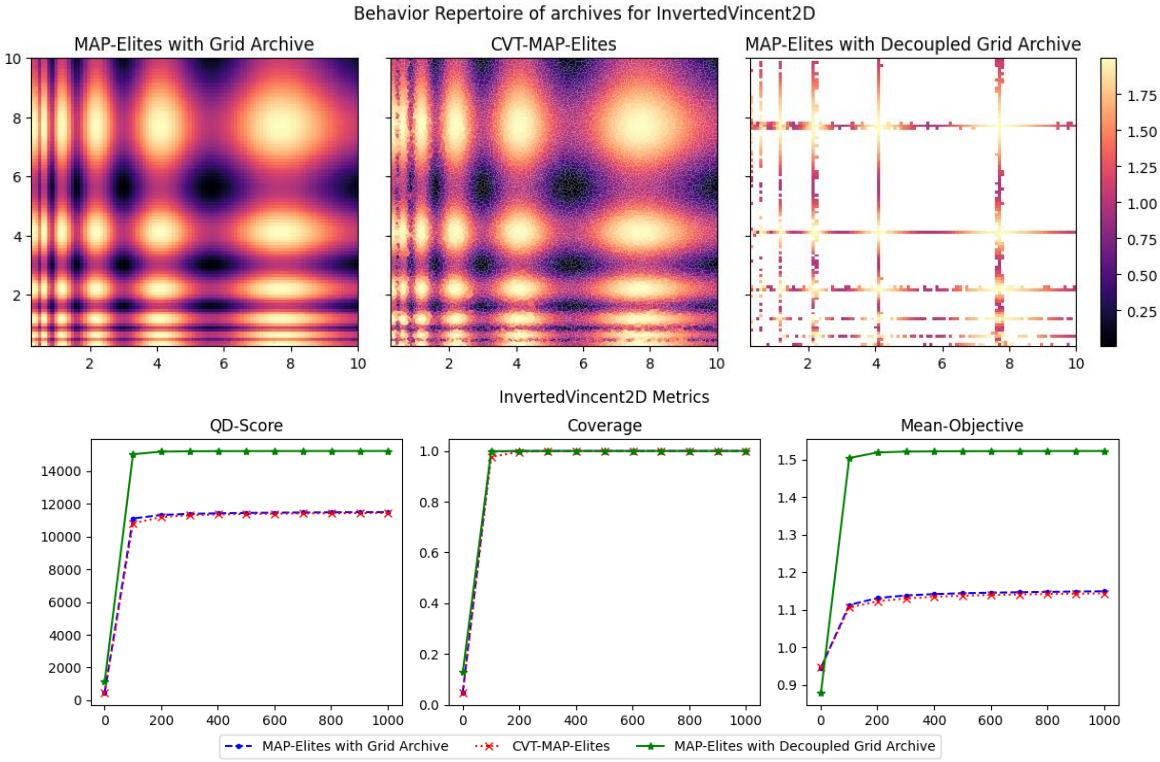
2D Inverted Rastrigin function



2D Inverted Sphere function



2D Inverted Vincent function



Appendix C

The problems used in QD and MMO of class Problem. The file Problems.py consisted of:

```
from itertools import product
from abc import abstractmethod
from scipy.stats import ortho_group
from numpy.random import default_rng
from scipy.spatial import KDTree
import autograd.numpy as np
from autograd import elementwise_grad as egrad
from autograd.numpy.linalg import norm
################################################################################
def gaussian(xs, x_opt, obj_opt, radi):
    return obj_opt * np.prod(np.exp(-(xs - x_opt) ** 2
                                    / (2 * radi ** 2)), axis=1)
################################################################################
class Problem:
    def __init__(self, x_dim, x_low, x_high, obj_low, obj_high, x_opt, **kwargs):
        self.x_dim = x_dim
        self.xl, self.xh = x_low, x_high
        self.yl, self.yh = obj_low, obj_high
        self.x_opt = np.array(x_opt)
        self.x_opt_norm = self.norm(self.x_opt)
        self.sum_maxima = np.sum(self.fit(self.x_opt_norm)[0])
        self.num_maxima = len(x_opt)

    @abstractmethod
    def evaluate(self, xs) -> np.array:
        pass

    def inv_norm(self, xs):
        xs = np.array(xs)
        xs = (self.xh - self.xl) * xs + self.xl
        return xs

    def norm(self, xs):
        xs = np.array(xs)
        xs = (xs - self.xl) / (self.xh - self.xl)
        return xs

    def fit(self, xs, beh_dims=None) -> np.array:
        xs = np.array(xs)
        xs = (self.xh - self.xl) * xs + self.xl  # inv_norm(0,1) -> (l, h)
        obj = self.evaluate(xs)
        obj = obj - self.yl
        beh_dims = beh_dims if beh_dims else self.x_dim
        return obj, xs[:, :beh_dims]  # beh subset_of xs

    def eval(self, xs) -> np.array:
        xs = np.array(xs)
        xs = (self.xh - self.xl) * xs + self.xl  # inv_norm(0,1) -> (l, h)
        obj = self.evaluate(xs)
        obj = obj - self.yl
```

```
        return obj

    def grad(self, xs):
        return egrad(self.eval)(xs)

    def is_optima(self, xs, atol=1e-8, rtol=1e-5):
        xs = np.array(xs)
        xs = xs[np.newaxis, :] if xs.ndim == 1 else xs
        return norm(egrad(self.eval)(xs), axis=1) < atol

    def is_maxima(self, xs, atol=1e-8, rtol=1e-5):
        xs = np.array(xs)
        xs = xs[np.newaxis, :] if xs.ndim == 1 else xs

        return np.logical_and(
            self.is_optima(xs, atol, rtol),
            np.all(egrad(egrad(self.eval))(xs) < 0, axis=1))

    def get_maxima_idx(self, xs, atol=1e-8):
        maxima_idx = KDTree(xs).query(self.x_opt_norm,
                                      distance_upper_bound=atol)[1]
        maxima_idx = maxima_idx[maxima_idx != len(xs)]
        return maxima_idx

    def domain(self):
        return [(self.xl, self.xh), ] * self.x_dim

    def name(self):
        return self.__class__.__name__

    def __str__(self):
        return f"{self.name()}{self.x_dim}D"


class InvertedVincent(Problem):
    """
    The Inverted Vincent function has the following properties.
    Sol Domain: [0.25, 10]
    Obj Range: [-1, 1]
    Max Optima: 6^dim
    Global Optima: 6^dim
    """

    def __init__(self, x_dim=2):
        x_1d_opt = np.array([  # 25 digit precision
                             0.33301843547196485837498 63,
                             0.62422843364856970835974 54,
                             1.17008878749642195040967 10,
                             2.19328005073801545655976 96,
                             4.11120714288535284208626 26,
                             7.70627725630577558586066 37,
                             7.70627725630577558586066 37,
                         ])
        x_opt = np.array(list(product(*[x_1d_opt for _ in range(x_dim)])))
```

```
        super().__init__(x_dim=x_dim, x_low=0.25, x_high=10,
                         obj_low=-1, obj_high=1,
                         x_opt=x_opt)

    def evaluate(self, xs) -> np.array:
        xs = np.array(xs)
        xs = xs[np.newaxis, :] if xs.ndim == 1 else xs
        return (1 / xs.shape[1]) * np.sum(np.sin(10 * np.log(xs)), axis=1)


class InvertedShubert(Problem):
    """
    The Inverted Shubert function has the following properties.
    Sol Domain: [-10, 10]
    Obj Range: [-l^dim, h * l^(dim-1)]
        where l =14.508007927195033117413042789881397076 71
        where h = 12.870885497725684895569773596085472611 71
    Max Optima: 2 * (19*20)^(dim/2) for even dims
    Global Optima: dim * 3^dim
    """

    def __init__(self, x_dim=2):
        super().__init__(
            x_dim=x_dim, x_low=-10., x_high=10.,
            obj_low=-14.508007927195033117413042789881397076 71 ** x_dim,
            obj_high=(12.870885497725684895569773596085472611 71 *
                      14.508007927195033117413042789881397076 71 ** (x_dim - 1)),
            x_opt=None)  # IDK x_opt of InvShubert

    def evaluate(self, xs) -> np.array:
        xs = np.array(xs)
        xs = xs[np.newaxis, :] if xs.ndim == 1 else xs
        return -np.prod(np.array([np.cos(2 * x + 1) + 2 * np.cos(3 * x + 2) +
                                  3 * np.cos(4 * x + 3) + 4 * np.cos(5 * x + 4) +
                                  5 * np.cos(6 * x + 5) for x in xs]), axis=1)


class GaussianMixture(Problem):
    def __init__(self, name, x_opt, obj_opt, radi):
        self.obj_opt = np.array(obj_opt)
        self.radi = np.array(radi)
        self.gm_name = name
        super().__init__(
            x_dim=len(x_opt[0]),
            x_low=-1., x_high=1.,
            obj_low=0., obj_high=np.sum(obj_opt),  # 1
            x_opt=x_opt
        )

    def evaluate(self, xs) -> np.array:
        xs = np.array(xs)
        xs = xs[np.newaxis, :] if xs.ndim == 1 else xs
        gauss_mix = 0
        for x_opt, obj_opt, radi in zip(self.x_opt, self.obj_opt, self.radi):
```

```
            gauss_sgl = gaussian(xs, x_opt, obj_opt, radi)
            gauss_mix = gauss_mix + gauss_sgl
        return gauss_mix

    def name(self):
        return f"GM-{self.gm_name}"


def create_gm_args(p, geno_dim):
    x_opt, obj_opt, radi = None, None, None

    if p == '7-Random':
        k = 7
        rng = default_rng(1)
        x_opt = 2 * rng.random((k, geno_dim)) - 1
        x_opt[:, 0] = np.linspace(-.9, .9, k)[[2, 5, 0, 3, 6, 1, 4]]
        x_opt[:, 1] = np.linspace(-.9, .9, k)
        obj_opt = np.ones(k)
        radi = np.ones(k) * 0.2
    elif p in ('25-Random', '25R-Random'):
        k = 25
        rng = default_rng(1)
        x_opt = 2 * rng.random((k, geno_dim)) - 1
        sols = np.linspace(-.9, .9, int(k ** .5))
        if p == '25R-Random':
            sols = np.linspace(-.7, .7, int(k ** .5))
        sols = np.array(list(product(sols, repeat=2)))
        x_opt[:, 0] = sols[:, 0]
        x_opt[:, 1] = sols[:, 1]
        obj_opt = np.ones(k)
        radi = np.ones(k) * 0.1
        if p == '25R-Random':
            x_opt = x_opt @ ortho_group.rvs(dim=x_opt.shape[1], random_state=2)

    return x_opt, obj_opt, radi
```