A Visualizable Test Problem Generator for Many-Objective Optimization

DOI:
10.1109/TEVC.2021.3084119

Document Version
Accepted author manuscript

A Visualizable Test Problem Generator for Many-Objective Optimization

Jonathan E. Fieldsend, *Member, IEEE*[1], Tinkle Chugh[2], Richard Allmendinger *Member, IEEE*[3], and Kaisa Miettinen[4]

[1]University of Exeter, Exeter EX4 4QD, U.K. Email: J.E.Fieldsend@exeter.ac.uk
[2]University of Exeter, Exeter EX4 4QD, U.K. Email: t.chugh@exeter.ac.uk
[3]University of Manchester, Manchester M15 6PB, U.K. Email: richard.allmendinger@manchester.ac.uk
[4]University of Jyvaskyla, Faculty of Information Technology, P.O. Box 35 (Agora), FI-40014 University of Jyvaskyla, Finland. Email: kaisa.miettinen@jyu.fi

Abstract

Visualizing the search behavior of a series of points or populations in their native domain is critical in understanding biases and attractors in an optimization process. Distance-based many-objective optimization test problems have been developed to facilitate visualization of search behavior in a two-dimensional design space with arbitrarily many objective functions. Previous works have proposed a few commonly seen problem characteristics into this problem framework, such as the definition of disconnected Pareto sets and dominance resistant regions of the design space. The authors' previous work has advanced this research further by providing a problem generator to automatically create user-defined problem instances featuring any combination of these problem features as well as newly introduced ones, such as landscape discontinuities, varying objective ranges, and neutrality. This work makes a number of additional contributions including the proposal of an enhanced, open-source feature-rich problem generator that can create user-defined problem instances exhibiting a range of problem features — some of which are newly introduced here or form extensions of existing features. A comprehensive validation of the problem generator is also provided using popular multi-objective optimization algorithms, and some problem generator settings to create instances exhibiting different challenges for an optimizer are identified.

Keywords: Multi-objective test problems, evolutionary optimization, benchmarking, test suite, visualization.

1 Introduction

Visualizing the search behavior of a population-based multi/many-objective optimizer in its native domain is a challenging task that is essential in algorithm analysis and design. To support algorithm designers and practitioners in the development and selection of effective optimization algorithms, this work introduces and validates a feature-rich visualizable test problem generator for population-based many-objective optimization. Before we put the work into context with existing literature and describe the generator in more detail, we will briefly define Pareto optimality and dominance. These concepts are necessary to understand the structure of the problem instances.

For multi- and many-objective optimization problems, without loss of generality, we seek to simultaneously minimize K objectives: $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_K(\mathbf{x}))$, where each objective depends upon a vector $\mathbf{x} = (x_1, \ldots, x_N)$ of N design (or decision) variables. These problems may also include equality and inequality constraints that define $\mathcal{X}$, a feasible design set in the design space $\mathbb{R}^N$. Relatedly, the image of $\mathcal{X}$ in the objective space $\mathbb{R}^K$ is denoted by $\mathcal{Y}$, and termed the feasible objective set. When there is more than one objective to be minimized, solutions may exist for which performance on one objective cannot be improved without reducing performance on at least one other. Such solutions are said to be *Pareto optimal*. The set of all Pareto optimal solutions is said to form the *Pareto set* $\mathcal{P}$, whose image in the objective space is known as the *Pareto front* $\mathcal{F}$. $\mathcal{F}$ consists of objective vectors and their components are called objective values. Identifying such solutions relies on Pareto *dominance*. A feasible design vector $\mathbf{x}$ is said to dominate another feasible design $\mathbf{x}'$ iff

$$f_k(\mathbf{x}) \leq f_k(\mathbf{x}'),\ k = 1, \ldots, K \ \text{ and } \ \mathbf{f}(\mathbf{x}) \neq \mathbf{f}(\mathbf{x}'). \tag{1}$$

This is often simply denoted as $\mathbf{x} \prec \mathbf{x}'$ rather than $\mathbf{f}(\mathbf{x}) \prec \mathbf{f}(\mathbf{x}')$. It is useful when examining the properties of an optimizer to look at the distribution of designs in its approximation of $\mathcal{P}$ and $\mathcal{F}$. One should note that Multi-objective Evolutionary Algorithms (MOEAs) cannot guarantee Pareto optimality but they generate approximations that are mutually non-dominated, i.e., no solution dominates any other.

Over the years, numerous performance indicators have been proposed to measure convergence, spread, and/or distribution capabilities of a population of solutions searching over a multi-dimensional objective space (see e.g. [1, 2]). However, gaining an effective understanding about search bias, trade-offs and other attractors in the design and objective space relies primarily on visualizing the movement of the population to the Pareto set/front. This task becomes particularly informative if the Pareto set/front is known rather than being approximated and can be visualized [3]. Unfortunately, this task becomes more difficult as the dimensions of the design and the objective space increase.

Various techniques can be found in the literature to visualize search performance [3]. Scatter plots are a popular choice for problems with 2 and 3 objectives but suffer from poor scalability. Beyond 4 objectives/design variables, scatter plot approaches rely on mapping a higher-dimensional space to a 2 or 3-dimensional space, resulting in a loss of Pareto dominance relation information between solutions [4, 5, 3]. Alternatively, if pair-wise plots i.e., scatter plot matrices are used, the number of plots required becomes rapidly overwhelming (as $(K^2 - K)/2$ plots are needed for K objectives). Approaches commonly used to visualize the distribution of solutions in high-dimensional spaces are

parallel coordinate plots and visualizations based on heatmaps potentially coupled with dimensionality reduction methods (e.g. principal component analysis) (see e.g. [6] for this). However, also for these approaches, identifying relationships quickly becomes more difficult as the dimensionality increases. The set of solutions to compare also tends to grow with the number of objectives.

A range of multi- and many-objective test problems have been proposed in the literature to validate and further expand our understanding of the search behavior of optimization algorithms for different problem characteristics. Popular problems in the continuous domain include test suites such as DTLZ [7], WFG [8], and the BBOB problems [9]. Commonly used many-objective test problems are, for example, ones proposed in [10, 11]. Most recently, a proposal for a more realistic many-objective test suite was made using a coefficient matrix [12]. Classical discrete problems include multi-objective knapsack problems [13] and NK-landscapes [14], while commonly studied permutation problems include multi-objective traveling salesman problems [15] and flowshop scheduling [16]. Mixed-integer problems, such as variations of NK-landscapes [17], have received attention too. Moreover, the literature includes also many-objective test problems designed to address particular challenges, such as a dynamically changing fitness landscape [18] or difficult-to-approximate Pareto front boundaries [19]. Many of these problems are tunable, such as the dimension of the design/objective space, rate and degree of dynamic changes, and the shape of the Pareto front. Nevertheless, they lack the ability to visualize the movement of the population in its native domain, which is important to understand search performance [20, 21].

The motivation of distance-based multi- and many-objective optimization problems [22, 23] is to facilitate visualization of the search behavior in a multi-dimensional space. Here we use the acronym DBMOPP as shorthand for a distance-based multi/many-objective point problem. DBMOPPs are problems that can have arbitrarily many objectives but inherently have a two-dimensional design space allowing the Pareto set to be identified easily by eye. In its standard form, the Pareto set of a DBMOPP is defined on a regular polygon in a two-dimensional design space, and the objective functions represent distances between each of the vertices of the polygon and a solution (which is a two-dimensional point); these distances are to be minimized. Thus, the number of objective functions is equal to the number of vertices of the polygon, and the solutions inside the polygon are the Pareto optimal solutions. Visualizing the movement of solutions in this two-dimensional design space allows for a convenient analysis of their *convergence* and *distribution* towards the Pareto set.

Subsequent work on DBMOPPs extended the concept to disconnected Pareto sets [24] and different shapes [25], non-identical disconnected Pareto sets [25], arbitrarily large design spaces that could be projected back (using orthogonal projection vectors) to the two-dimensional visualization space [26], alternative distance metric (Manhattan distance) [27, 28], dominance resistance regions [29], dynamic variants of the problem [30], local fronts [31], and different types of constraints [32]. The authors' previous work [33] introduced further extensions to DBMOPPs including landscape discontinuities and varying objective ranges, amongst others.

DBMOPPs can be translated into practical applications, for example, into location selection problems [25, 34, 35]. Here, the task could be to decide about the location of a new building (e.g. school), such that it is close to a number of existing landmarks like a train station, shopping center, etc. In a DBMOPP, these landmarks would be represented by the vertices, and the Pareto optimal locations of the new building(s) would be defined by the relative locations of these landmarks. In reality, the distances (i.e. the objective functions) between each of the vertices of the polygon may be substituted with travel times or the cost to get from one location to another. It is worth noting that the related concept of line-based-distance problems was introduced in [36, 37]. However, our work focuses on point-based formulations (which is also largely the focus of existing literature). This work makes the following contributions to the state-of-the-art in DBMOPPs beyond our earlier work [33]:

1. Extension of a number of existing problem features.

2. Proposal of additional problem characteristics including hard constraints and non-intersecting performance disconnected Pareto sets.

3. An enhanced feature-rich automatic problem instance generator for creating user-defined problem instances. It encapsulates all existing features and the newly proposed characteristics for the Euclidean distance metric.

4. An open-source repository containing object-oriented MATLAB code for the generator, which includes methods for the visualization and verification of problem instances.[1]

5. A validation of the generator involving (i) different forms of statistical and visual analysis of the problem instance space to better understand relationships between the different problem features and (ii) an investigation of the search behavior of several popular multi- and many-objective evolutionary algorithms (MOEAs) representing different search concepts (NSGA-II [38], MOEA/D [39], IBEA [40], NSGA-III [41], and random search). Due to the large number of features embedded in the generators, the empirical study will be limited to a representative set of features.

6. Identification of particular problem generator settings to create problem instances with feature combinations that pose different challenges to popular multi-objective optimizers.

We want to emphasize that our focus is not on developing new algorithms to solve DBMOPPs. Instead, we focus on the development and validation of a feature-rich DBMOPP problem generator to encourage take up by the optimization community and facilitate understanding of the problem instances generated, respectively.

The remainder of this paper proceeds as follows. The next section provides a description of existing features in the DBMOPP literature. The existing DBMOPP framework is extended in Section 3 by introducing a set of new tunable problem characteristics. Section 4 then introduces a problem generator capable of automatically creating DBMOPP problem instances that feature a range of desired characteristics. In Section 5, we validate the problem generator by producing and analyzing a large number of problem instances with different features. We also analyze the performance of popular MOEAs on these instances, and identify a suite of generator settings that translate into problem instances that pose different challenges to these MOEAs. Finally, we conclude and discuss future research directions in Section 6.

[1] Available at `https://github.com/fieldsend/DBMOPP_generator`.

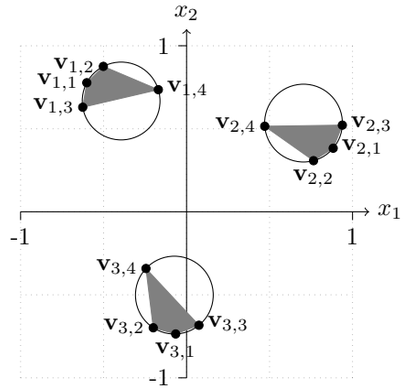

Figure 1: Illustration of Pareto regions that together define the Pareto set, $V_i = \{\mathbf{v}_{1,i}, \ldots, \mathbf{v}_{3,i}\}$, and $\mathcal{R}_j$ is defined by the convex hull of $\{\mathbf{v}_{j,1}, \ldots, \mathbf{v}_{j,4}\}$.

2 Visualizable Euclidean distance-based test problems

In conventional visualizable distance-based test problems, we have $\mathcal{X} \subseteq \mathbb{R}^2$. For point-based formulations in this domain, there are K sets of *attractor* vectors defined, where the kth set, $V_k = \{\mathbf{v}_{1,k}, \ldots, \mathbf{v}_{m_k,k}\}$ (see illustration in Figure 1), is used to assign the quality of a putative design vector $\mathbf{x} \in \mathcal{X}$, on the kth objective. Calculation of this objective is usually

$$f_k(\mathbf{x}) = \min_{\mathbf{v} \in V_k} (\texttt{dist}(\mathbf{x}, \mathbf{v})).$$

As m_k is the number of elements of V_k, i.e. $|V_k|$, and this term may vary with each k, it is possible for $|V_i| \neq |V_j|$. However, $|V_i| \geq 1$ for all i must hold. The distance function $\texttt{dist}(\mathbf{x}, \mathbf{v})$ typically returns the Euclidean distance between $\mathbf{x}$ and $\mathbf{v}$, although alternative metrics have been used in the literature, such as the Manhattan distance [27, 28].

Until the release of our preliminary generator in [33], DBMOPP problems appearing in the literature had been designed by hand. The use of a generator to automatically construct DBMOPPs facilitates empirical analysis based on test problem sampling, rather than just a single suite of limited size (whose particular configuration may eventually be 'exploited' to the detriment of generalized algorithm design). See e.g. [42] for a discussion on the use of generators for generalizable results.

2.1 Glossary of terms

Before we overview the range of properties we can automatically embed in a DBMOPP instance, we will first set out a glossary of terms, which we use here to help define the construction process of a DBMOPP, and distinguish between different DBMOPP features.

- *Regions*: A contiguous area in the two-dimensional DBMOPP design space, whose elements share some property of interest.
- *Local front sets (local non-dominated sets)*: Region(s) where the image of the elements in objective space all define a *local* Pareto front [43].
- *Pareto regions*: Region(s) where the image of the elements in objective space all define elements of a *global* Pareto front. These are typically convex polygons in $\mathcal{X}$.
- *Attractor regions*: Regions whose existence is due to attractor vectors, i.e. $\mathbf{v}$.
- *Region centres*: Where a region is constructed as a convex polygonal shape with attractors on the corners, these are placed on the diameter of circle with a particular center, denoted the region center. Note that this does not necessary coincide with the center of the constructed polygonal shape.
- *Penalty regions*: Regions where members experience a penalty on one or more objective values.
- *Constraint regions*: Regions where members exhibit a soft or hard constraint violation.

We now outline the process by which the generator distributes the set V_k as this will be used in the schematic illustrations throughout. Consider the illustration in Figure 1 with three attractor regions $\mathcal{R}_j = \{\mathbf{v}_{j,1}, \ldots, \mathbf{v}_{j,4}\}$, which together define the Pareto set. It is important to distinguish between the Pareto set (which defines the optimal set of solutions for the problem), and a *Pareto region* which is a polygonal shape in this construction, whose interior members are Pareto set members, but which might not, by itself, describe the complete Pareto set. Note that not all $\mathcal{R}_j$ will define Pareto regions — they may alternatively define locally rather than globally optimal solutions.

Let us consider the simplest distance-based problem formulation using points, where $|V_k| = 1$ for all k, as shown in the top left panel of Figure 2 for a $K = 4$ example. In our generation procedure, we place circles in the 2D design space, and place V_k elements on the circumference of a circle (middle column of Figure 2). The relative angles used to place the vectors $\mathbf{v}$ on circles defining the regions need to be consistent between them, the circle radii however do not need to be the same. The region will define elements of a local or global Pareto set (see the bottom row of panels in Figure 2), depending on the circle radius used. Such local configurations of points are our *regions* $\mathcal{R}_j$ — each $\mathcal{R}_j$ being defined by an element from 1 to K of the V_k sets. By placing fewer than the full complement of K different vectors $\mathbf{v}$ attractors on the circumference, dominance resistance areas can be induced [29].

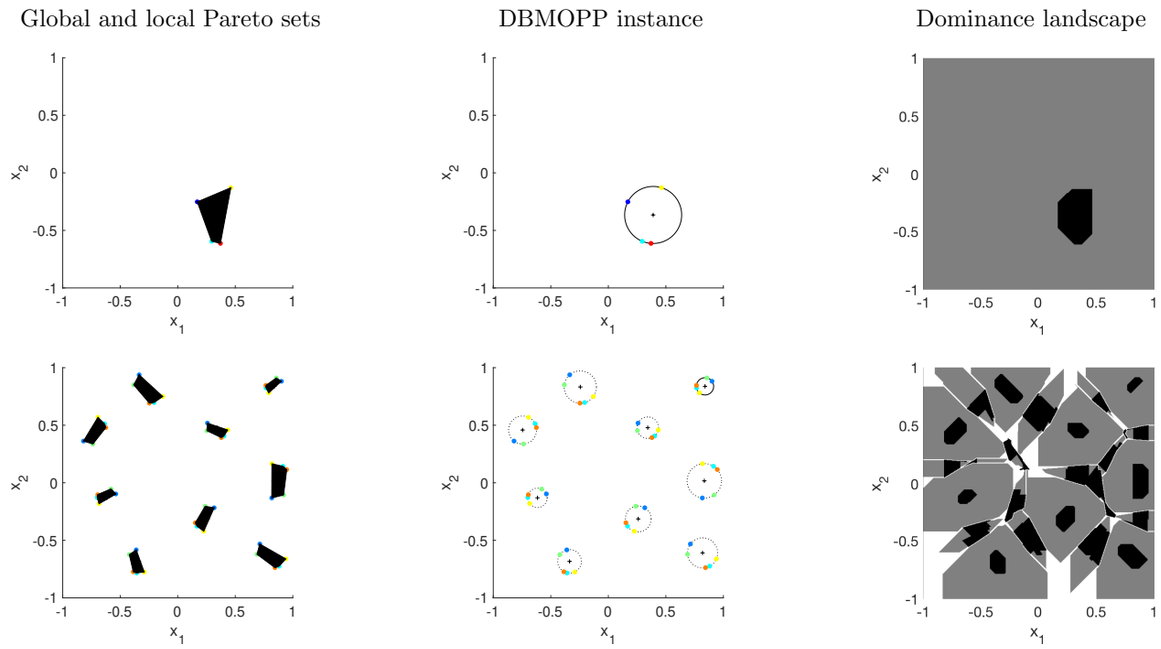

Figure 2: DBMOPP illustrations. *Top row*: a 4-objective problem with a single connected Pareto set. *Bottom row*: a 5-objective problem with a single connected Pareto set situated in the top right corner of $\mathcal{X}$, and numerous local non-dominated sets. *Left-column*: Pareto set and locally dominated sets in black. *Middle-column* Test problem configuration showing V_k, circles and centers. *Right column*: Local dominance landscape approximated by sampling $\mathcal{X}$ on a 500×500 grid.

2.2 Problem instance generator

Given the wide range of features that can be incorporated in a DBMOPP test problem instance, generating one automatically and ensuring the desired properties are all present is non-trivial. In [33], we observed that $\mathcal{X}$ may be partitioned into areas concerned with providing instances of each of the various properties desired. These are largely determined by sets of points defining the different region types.

The properties of problems generated are directly verifiable. For instance, finely discretizing the space and commencing a Pareto local search [44] from a single location in each of the local front sets will verify they match the problem as initialized. We additionally provide tools for such verification in the generator code repository.

2.3 Placing region centers

We allocate the centers defining each of the regions at random, but subject to lying at least $4r$ from the closest next region for all attractor regions (see [33] for a derivation of this limit to ensure instances behave as desired). Here, r is the *largest* radius employed by any individual attractor region (i.e. a region defined by elements of the V_k sets). Additionally, all such region centers must be at least r from the domain boundary. We employ a Monte Carlo circle placement with rejection sampling for this (as in [33]). For non-attractor regions, i.e. penalty/constraint regions (which we discuss further in the next section), these may be placed immediately adjacent to attractor regions, as they cannot induce Pareto optimal regions if placed too close (unlike the other region types).

As well as being able to visualize the Pareto set and local non-dominated sets directly in $\mathcal{X}$, in visualizable problems, we can also look at the landscape induced by them. One approach is to view the *dominance landscape* [45]. These are shown in the right-most panels of Figure 2. The black regions in the local dominance landscape are comprised of cells (squares) in the discretized space, where the center of all eight immediate neighboring cells (the Moore neighborhood) are mutually non-dominating with the location at the center of the middle cell. This denotes a cell as belonging to dominance-neutral region — i.e. all local moves being mutually non-dominating. These may be identified by point-based Pareto hill-climbing [46], but note that a contiguous region of such local optima is not guaranteed to be composed entirely of members that are mutually non-dominating (a local Pareto set), as construction of these relies on a set-based rather than point-based hill-climb (see e.g. [44]). Instead, the black regions describe a locally dominance-neutral region, where all local moves estimated at the discretization used are mutually non-dominating. The gray regions in the plot are made up of cells which have at least one dominating neighbor (i.e. lie on a dominance hill-climb path, rather than the end of a path). All dominating movement paths from neighbors in gray regions lead to the same local optima region (i.e. same black region). As such, the gray regions denote those basin components which lead to the same dominance-neutral attractor. White regions are comprised of cells whose neighbors lead to multiple different attractor regions (and therefore denote boundary regions/saddle-points).

Note the complex interactions in the landscape in the bottom-right panel of Figure 2. The local dominance-neutral regions include the Pareto set and the regions denoting specified local fronts from the left panels, but also additional dominance-neutral regions lying between these have been induced by the V_k points. These generally have much smaller basins (and in some cases no basin at all). Note the dominance-neutral regions in the dominance landscapes shown in the right panel may be larger than the corresponding Pareto/local non-dominated regions shown in the left panels of Figure 2. This is because the dominance-neutrality is *local* to the neighborhood of each cell in the right-hand side figures, rather than calculated with respect to *every* member of the region (denoting the landscape observed by a local greedy dominance-based hill-climber).

Table 1: An overview of existing and new features embedded in the proposed DBMOPP generator.

Feature	Year	Ref.	Extended here
Arbitrarily many objectives	2005	[22]	
Disconnected Pareto sets	2010	[24]	
Non-identical disconnected Pareto sets	2011	[25]	✓
Arbitrarily many variables	2014	[26]	
Dominance resistance regions	2016	[29]	
Time-varying problem settings	2017	[30]	✓
Local Pareto optimal fronts	2018	[31]	
Discontinuous objective functions	2019	[33]	
Varying solution density in Pareto sets	2019	[33]	
Varying objective scales	2019	[33]	✓
Neutrality	2019	[33]	
Soft constraints (real-valued)	2019	[32]	✓
Hard constraints	2020	Here	
Non-intersecting disconnected Pareto sets	2020	Here	

2.4 DBMOPP features

We now describe the features enabled in the DBMOPP generator, and highlight the relevant literature. Table 1 summarizes these supported features in the generator.

2.4.1 Arbitrarily many objectives

Because there is no limit on the size of V, the formulation allows a user defined and effectively unbounded number of objectives in an instance.

2.4.2 Disconnected Pareto sets

Disconnected Pareto sets were introduced in DBMOPP by [24]. In our generator, we can populate a problem instance with arbitrarily many disconnected set regions. These include the following three distinct subtypes.

- Duplicate sets: Regions $\mathcal{R}_j$ that have equivalent performance, meaning all Pareto optimal objective vector combinations are exhibited by Pareto set members in each region (three such regions are shown in Figure 1).
- Partially overlapping performance sets: Regions $\mathcal{R}_j$ that each describe *part* of the Pareto front, but can partially overlap in the mapping through $\mathbf{f}$ (i.e. a particular Pareto optimal objective vector combination may be obtainable via more than one $\mathcal{R}_j$, see e.g. [25]).
- Non-intersecting performance sets: Newly introduced in the generator in this work — where the Pareto front may only be determined by finding *all* disconnected Pareto regions, and there is no duplication of performance between these regions.

Examples of these three types are shown in Figure 3.

2.4.3 Arbitrarily large design spaces

Using two orthogonal projection vectors $(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2)$ forming a basis, as proposed in [26], a design space of arbitrarily many dimensions may be projected into a 2-dimensional space, and subsequently evaluated under $\mathbf{f}$. A design vector $\mathbf{z} \in \mathcal{Z}$, $\mathcal{Z} \subset \mathbb{R}^N$, $N > 2$, can be mapped to a respective $\mathbf{x}$ using N-dimensional projection vectors (or basis) $\boldsymbol{\pi}_1$ and $\boldsymbol{\pi}_2$. After a projection back into $\mathbb{R}^2$, the corresponding $\mathbf{x}$ can be evaluated and visualized:

$$\mathbf{x} = \frac{(\mathbf{z} \cdot \boldsymbol{\pi}_1)}{||\boldsymbol{\pi}_1||} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \frac{(\mathbf{z} \cdot \boldsymbol{\pi}_2)}{||\boldsymbol{\pi}_2||} \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

A single N-dimensional space with multiple regions $\mathcal{R}_j$ may be projected via two orthogonal vectors down to 2 dimensions, but it is also possible to project to multiple *different* 2-dimensional spaces, with *different* orthogonal vector pairs of the same dimension and evaluate $\mathbf{z}$ using each of these projections. This allows the different regions $\mathcal{R}_j$ to be oriented differently in $\mathcal{Z}$ (and therefore be more distant than in the single projection case) [26].

2.4.4 Dominance resistance regions

The typical generation of a DBMOPP results in all solutions that minimize any *individual* objective f_k also being Pareto optimal. In [29], region constructions were introduced which could overcome this limitation and supply designs which were dominance resistant [47] (i.e. solutions which were dominated by other designs in $\mathcal{X}$ but with optimal values for some of the objectives when compared to the objective vectors of Pareto set members). Specifically, dominance resistance regions have points whose V_k match those in the Pareto set, but which are described by at most $K - 1$ of the points used to define a Pareto region $\mathcal{R}_j$. This ensures that each solution in a dominance resistance region is dominated by at least one design in the Pareto set. This generation process means there are numerous designs in $\mathcal{X}$ whose quality may be optimal under one or more f_k, but when evaluated under $\mathbf{f}$ are still located very far from $\mathcal{F}$.

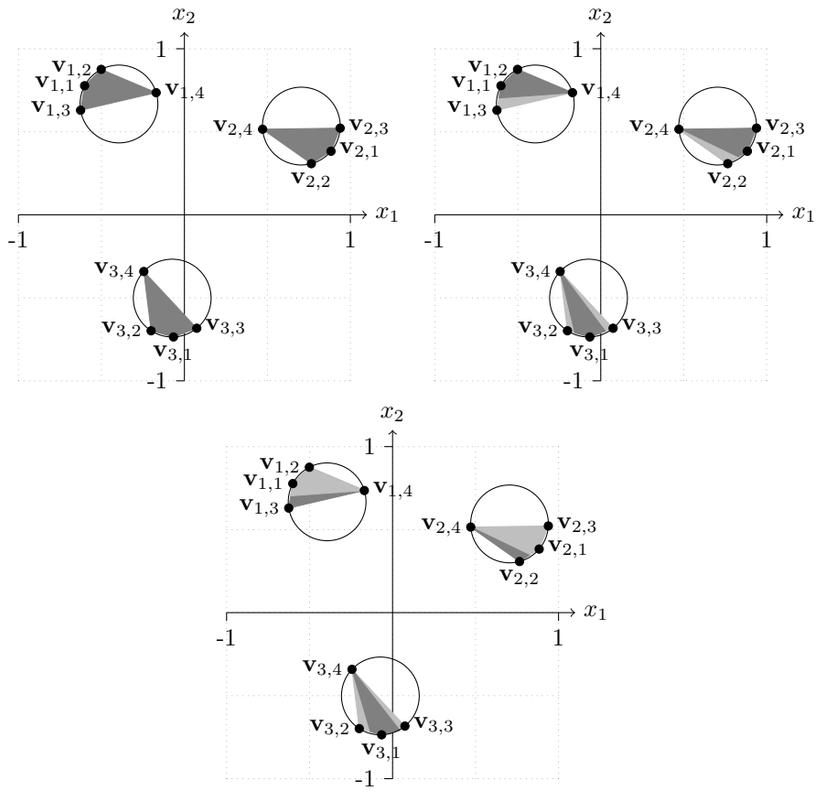

Figure 3: Illustration of disconnected Pareto set region types. Regions in light gray are subject to an additive penalty on the objective vectors, which ensures that they are not Pareto optimal. *Left*: Duplicate sets — the Pareto front may be described completely on identifying a single region. *Middle*: Partially overlapping performance sets — no single disconnected region defines the entire Pareto front, however there is some duplicated performance between regions. *Right*: Non-intersecting performance set — the entire Pareto front is only obtainable once all disconnected regions are identified, and there is no duplicated objective vectors mapped to by $\mathbf{f}$ between regions.

2.4.5 Local fronts

Local fronts in multi-objective optimization problems act much like local optima in single-objective problems. As illustrated earlier in Figure 2, these generate their own local basins of attraction in $\mathcal{X}$. These may be easily constructed in our framework by using the angles selected for the placement of the attractor points in the Pareto region $\mathcal{R}_j$, but applying a larger radius when distributing the corresponding attractor points for local regions. Note that these may also be rotated as a group around the circle center defining each region without affecting their properties (as long as the regions are placed sufficiently apart, see [33] for more details).

2.4.6 Constraints

Beyond box-constraints defining the feasible design set $\mathcal{X}$, Nojima et al. [32] have recently shown how more complex constraints may be incorporated in DBMOPPs. Their work defines four main types of constraint regions:

1. Vertex: A set of vertices are distributed, typically on the polygon defining the Pareto set region. All locations within a fixed radius of the vertices are subject to a constraint violation value, inversely proportional to their distance from the vertex.

2. Center: The circular constraint region is centered on the Pareto set polygon and covers the polygon entirely. This has the effect of pushing the feasible Pareto set to the circumference of the constraint region.

3. Moat: The circular constraint region is centered on the Pareto set polygon and covers the polygon entirely. However, it is not applied *within* the polygon. This has the effect of placing a 'moat' of constrained space around the Pareto set, which must be traversed if Pareto optimal designs are to be identified

4. Checker: The design space is gridded, with each alternative cell having a constraint vertex centered within it – effectively generating a checkerboard effect.

We further build on this in our framework allowing constrained regions of heterogeneous size and area or volume.

We follow the methodology of [32] for incorporating vertex, center and moat-style soft constraints (and introducing their hard constraint counterpart). For the checker-style, however, we introduce a more flexible and heterogeneous region construction, where each of the penalty centers can be placed *arbitrarily* as long as their particular radius means they do not intersect with a Pareto (or local Pareto) region. This means the number and size of these regions can be chosen so that a given proportion of $\mathcal{X}$ is constraint violating, and the penalty regions themselves can intersect, making more complex shapes of the infeasible/constraint-violating design space. We illustrate our implementation of these four different constraint region types in Figure 4. Constraint violations are assessed after projection into $\mathcal{X}$ when larger design spaces are employed.

2.4.7 Discontinuous objective functions

The use of the sets of attractor points V in the construction of a DBMOPP results in smooth objective landscapes. Discontinuities can be introduced (as outlined in [33]) via *penalty regions* $\mathbf{p}$. These may be used to apply a fixed

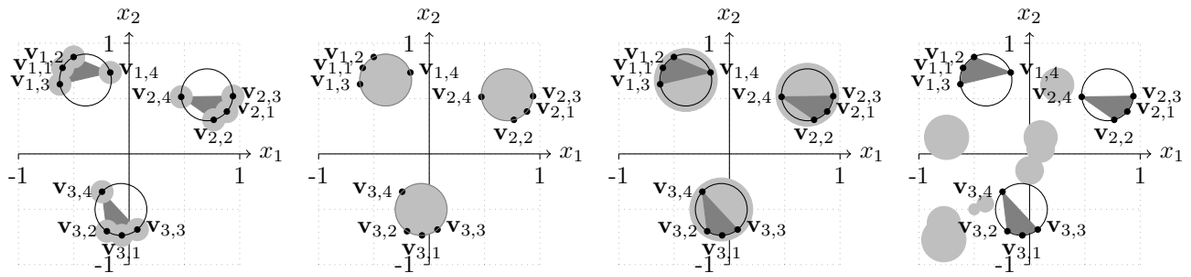

Figure 4: Illustration of constraint region types as implemented in the generator. *Left to Right*: Vertex, Center, Moat and (extended) Checker-type. Light gray areas denote regions where the constraints are violated.

or varying non-zero penalty to one or more objective values for all locations within the region. The effect is to also induce discontinuity in the landscape of those objectives at all locations that lie on the perimeter of the penalty region — causing a step-change (discontinuity) in the objective function response at the region boundary.

In [33], we used circular penalty regions defined by a center (location) and a radius and they were kept spatially distinct. However, here we permit them to be placed arbitrarily close (and indeed intersecting) allowing the composition of complicated and non-convex penalty regions to be formed.

2.4.8 Varying solution density in Pareto sets

In [33], we showed how varying the relative lengths of the orthogonal projection vectors used to generate arbitrarily large design sets allows us to vary the density of the solutions mapped back to the 2D representation in $\mathcal{X}$. This can in turn make some regions $\mathcal{R}_j$ and regions of the Pareto front more difficult to attain than others.

2.4.9 Varying objective scales

In standard formulations of DBMOPPs, the range of each objective does not vary greatly, and the minimum of all objectives is 0. As we first outlined in [33], the objective ranges can easily be shifted to be arbitrarily wide or narrow, with any maxima and minima via a multiplication and shift term $f_k^{rescaled}(\mathbf{x}) = a_k + b_k f_k(\mathbf{x})$.

2.4.10 Neutrality

Neutral (flat) regions of the objective/dominance landscape can be generated using the penalty region approach detailed in Section 2.4.7, where instead of an additive or multiplicative penalty on the objective(s) associated with designs in the region, a constant value is used to replace objective values. This has the effect of making all design vectors in the region express identical objective values for the set of objectives affected. Neutrality is common in combinatorial problems, but can also exist in continuous ones, for instance, the labor cost or time of manufacture may not change *at all* between similar engineering designs.

2.4.11 Variable difficultly in attaining Pareto regions

Recent work has explored varying the difficulty of finding Pareto regions in DBMOPPs [48]. Two routes to this are identified:

1. Ensuring solutions close to one $\mathcal{R}_i$ are more likely to dominate solutions equivalently close to another $\mathcal{R}_j$.
2. Ensuring there is a varying difficult/complexity in discovering different $\mathcal{R}_i$.

We have not (yet) implemented the objective value modifications outlined in [48] for the first route, however we note the second route is achieved naturally through the generator in instances with $N > 2$. This is because $\mathcal{R}_i$ further away from the origin in $\mathcal{X}$ are mapped to by smaller volumes of Z, and therefore are more difficult to attain (as density is lower).

3 Illustration on some popular MOEAs

We ran four different MOEAs: NSGA-II [38], MOEA/D [39], IBEA [40] and NSGA-III [41] on problems with different feature values (lower and upper bounds are given in Table 2). As mentioned earlier, we limit our analysis here to a representative set of features as opposed to carrying out a superficial analysis across all features embedded in the generator. A total of 5,760 problem instances (i.e. 5,760 feature vectors obtained by all combinations of features in Table 2) were used to test and analyze the performances[2]. For each feature vector, we used 11 different instances to consider the stochastic nature of the generator when realizing an instance with specific properties (similar to NK landscapes [49]). For instance, a problem with a feature vector (0,0,2,2,0,0,0) (i.e., each feature set to its lower bound) was generated 11 times with the generator. All instances were optimized for 200 generations (experimental settings are detailed in the supplementary material). In addition to the four algorithms mentioned above, we applied random search on all problem instances as a baseline approach; this approach draws solutions at random from the uniform distribution spanning the search domain each generation.

To analyze the performances, we used the hypervolume [1] ratio (i.e., the hypervolume of the obtained non-dominated solutions/hypervolume of the Pareto front). Furthermore, we compared different algorithms statistically using the Wilcoxon pairwise test with a Bonferroni correction. These experiments can provide insight for (i) selecting an algorithm based on the known problem characteristics or features, and (ii) designing problems to test an algorithm developed.

[2]See supplementary material for details of the derivation of 5,760.

Table 2: Features, comprising the feature vector, with their lower and upper bounds (lb and ub, respectively) and type.

Feature	lb	ub	type
Non-identical Pareto sets	0	1	binary
Varying density	0	1	binary
Number of objectives	2	10	int
Number of variables (dimensions)	2	20	int
Number of disconnected Pareto sets	0	6	int
Number of local fronts	0	6	int
Number of dominance resistance regions	0	6	int

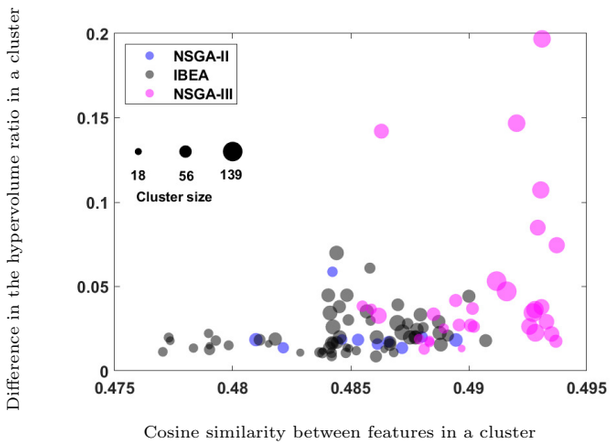

Figure 5: Average cosine similarity between features in a cluster and differences in hypervolume ratio. The sizes of the bubbles are proportional to the size of the clusters. The color indicates the best performing algorithm in a cluster whereby 'best' is measured in terms of majority voting.

To get insights into the large amount of data in the results, we clustered the different feature vectors using their cosine similarity and plotted against the differences in the hypervolume ratio in Figure 5. The feature vectors with similar cosine similarity indicate that they are in the same direction. Therefore, the problems with similar cosine similarities have a similar complexity. The difference in the hypervolume ratio is the sum of differences in the hypervolume of solutions obtained by all algorithms. A small difference indicates that all algorithms performed similarly to each other and a large difference indicates that at least one of the algorithms performed significantly better or worse than the others. A number of observations can be made from Figure 5:

- The presence of bubbles (feature clusters) with multiple colors implies that there is no single best algorithm across the test instances generated. This provides evidence that the proposed generator is able to produce problem instances of varying (and controllable) complexities.
- Each feature cluster corresponds to only one of three colors (MOEAs), NSGA-II, IBEA or NSGA-III, indicating that they performed best (in terms of majority vote) across the problem instances. Although MOEA/D and random search did not perform best for any feature cluster, there are individual feature vectors within a feature cluster for which they outperformed the others.
- The similarity between feature vectors in a cluster is proportional to the cluster (bubble) size. This is because a large size cluster has a large variance in feature vectors compared to a small sized cluster. Moreover, the spread of the clusters with respect to the differences in the hypervolume ratio increased with the cosine similarity. This is because many problem instances contained in larger clusters (located in the far right of the plot) leads to more significant differences in algorithm performance.
- IBEA performed the best for low values of the cosine similarity between features in a cluster with NSGA-III becoming the superior algorithm beyond a similarity level of around 0.48. There was a sweet-spot around a medium similarity level where NSGA-II performed well too. This observation confirms that the generator can create problem instances that pose challenges to different types of algorithms.

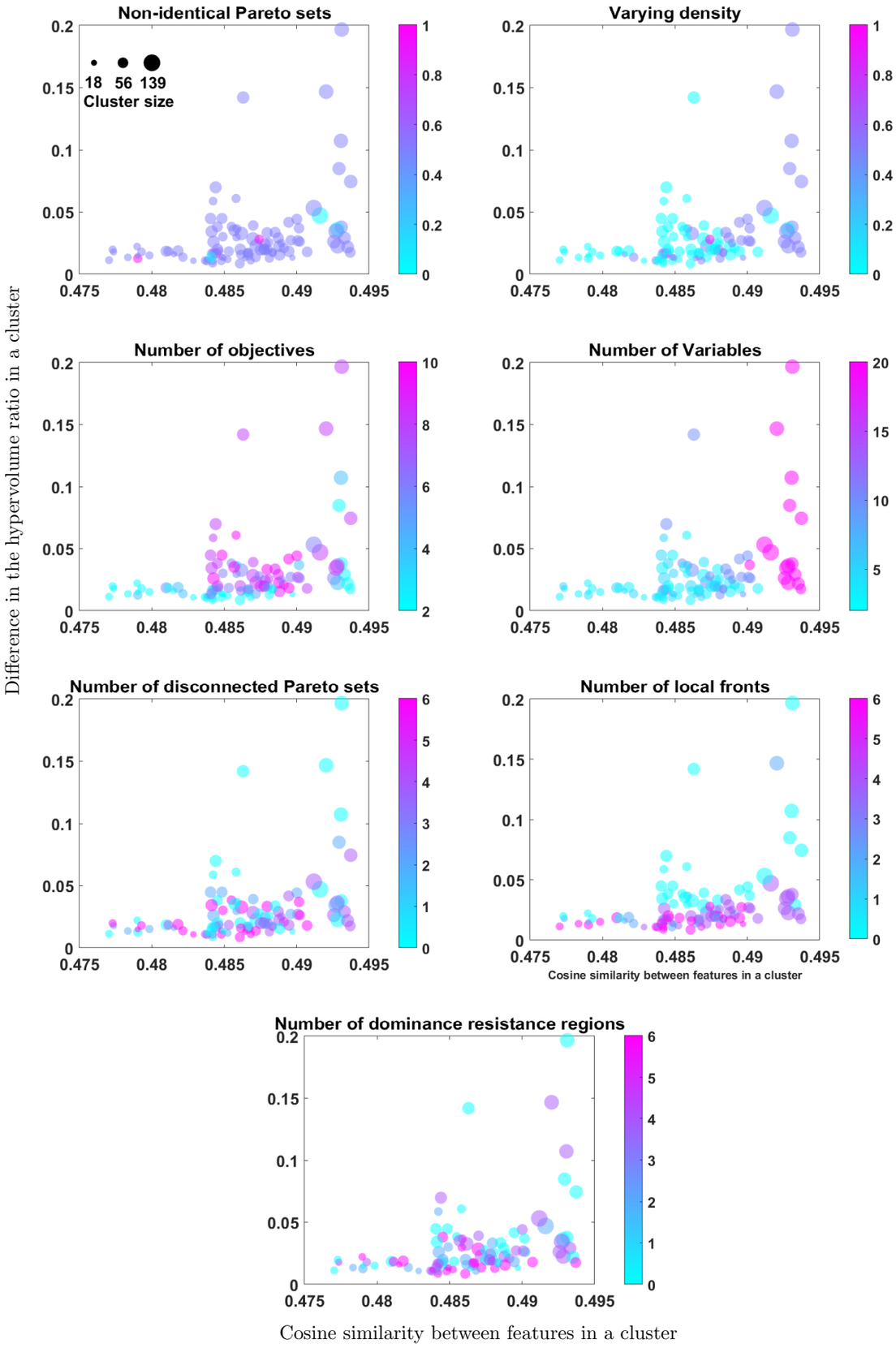

Figure 6: Cosine similarity between features in a cluster and differences in hypervolume ratio for different features. The sizes of the bubbles are proportional to the size of the clusters. The color indicates the median feature value of the feature vectors in a cluster.

To further analyze the relationship between different feature vectors and algorithm performance, we plotted the differences in hypervolume ratio with their cosine similarity, and colored the clusters based on a particular feature as depicted in Figure 6. Keeping in mind the best performing MOEA for different clusters (Figure 5), we can observe the following trends from Figure 6:

- NSGA-III tended to perform best for problem instances with many design variables (top row, fourth column), many objectives (top row, third column), high varying density (top row, second column), and no or few dominance resistance regions (bottom row, right column).
- IBEA tended to perform best for problem instances with many objectives, many local Pareto fronts (bottom row, middle column), many disconnected Pareto sets (bottom row, left column), and many dominance resistance regions.
- NSGA-II tended to perform best for problem instances with many local Pareto fronts.

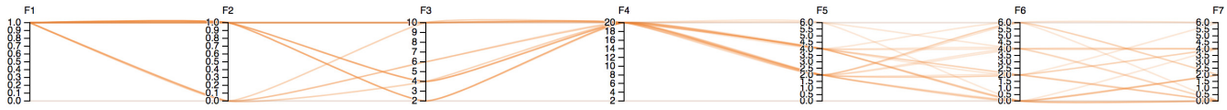

Figure 7: Features on which random search performed the best, F1 to F7 are features from top to bottom in Table 2

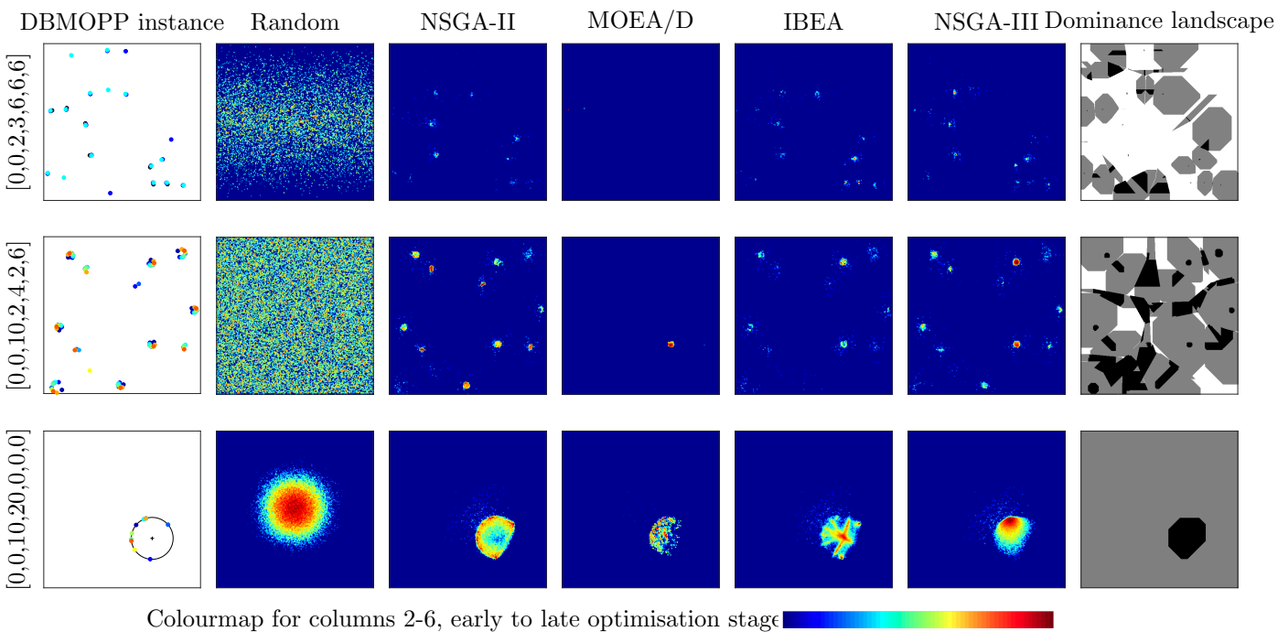

Figure 8: Trace generation plots of solutions with different optimisers. The best performed algorithm from top to bottom rows is: IBEA, NSGA-II, and NSGA-III. The features of different problem matrices are mentioned in parentheses in the order [non-identical Pareto sets, varying density, number of objectives, number of decision variables, number of disconnected Pareto sets, number of local fronts, number of dominance resistance regions].

Some of these observations are less obvious. In particular, the fact that IBEA and NSGA-II performed better than NSGA-III for problem instances with many local Pareto fronts is because local fronts do not lie in the same direction as the global front in many visualizable test problems instances. This is in contrast to widely used benchmarking suites, such as ZDT and DTLZ, where such fronts are 'lined up' in decision space. Therefore, dominance and indicator-based MOEAs had an advantage over decomposition-based MOEAs when solving visualizable test problem instances with many local fronts. Another, perhaps surprising observation, is that dominance-based MOEAs were not always performing poorly on problems with many objectives, highlighting the fact that algorithm performance is a function of multiple problem features.

As mentioned above, the generator also created problem instances on which random search outperformed the other algorithms. Figure 7 shows a parallel coordinate plot of features values, where this is the case. It can be seen that the problem instances needed to have non-identical Pareto sets and many design variables for random search to do well. Parallel coordinate plots of feature values favorable for the other algorithms can be found in the supplementary material.

One of the main advantages of using DBMOPPs is that the search behavior of a population can be visualized in its native domain. This property is explored in Figure 8, which shows the trace generation plots of the MOEAs and random search (columns) for three selected problem instances (rows). We selected these instances randomly from the cluster on the extreme left (IBEA performed the best), on the extreme right (NSGA-III performed the best) and the cluster of features on which NSGA-II performed the best in Figure 5. The far-right column shows the local dominance landscape of a problem instance, and the first column of the figure shows the distribution of the attractor vectors in $\mathcal{X}$; the global Pareto fronts are indicated by the circles with a black cross in the center (zoom in to spot the circles in the first and second row).

The problem instance in the first row of Figure 8 has only two objectives and three design variables but many (six) disconnected Pareto regions, local fronts and dominance resistance regions, which makes the problem difficult to solve. IBEA and random search were able to find most of the disconnected Pareto regions. However, IBEA was able to achieve a higher hypervolume than random search. An important observation is that IBEA was able to outperform all algorithms used in this work on difficult problems. Similar results were also observed in [50], where IBEA performed better than other algorithms. More work (which is out of scope here) is needed to understand the effectiveness of IBEA on such difficult problems which induce a complicated dominance landscape.

The problem instance in the second row of Figure 8 has many objectives (K=10), disconnected Pareto regions (four) and dominance resistance regions (six), and few (two) design variables and local fronts. It can be seen that NSGA-II was able to find most of the disconnected Pareto regions, indicating that NSGA-II outperformed the other algorithms in the presence of many objectives because of the effects of the other features. (A similar observation was also made in [51].) The problem instance in the third row has many objectives (K=10) and design variables (N=20) but no disconnected Pareto set and none of the other features. NSGA-III was able to outperform the other algorithms on this problem instance and found solutions closest to the global front. This pattern is in alignment with the observation made in Figure 6 (top row, right column), suggesting that NSGA-III performed best on DBMOPPs with many design variables.

4 Conclusions

Visualizing the search behavior of an evolutionary algorithm in its native domain, be it in the design or objective space, is critical to understanding the search biases and attractors a problem may have. Ultimately, this understanding supports algorithm designers and practitioners in the process of developing, validating and selecting a suitable optimization algorithm for the problem at hand. Visualization becomes even more challenging as the dimension of the search domain increases. To address this challenge, we have presented, implemented, made openly available and validated a feature-rich visualizable test problem generator for many-objective optimization. The generator is based on the concept of distance-based multi/many-objective point problems (DBMOPPs), which are a type of problem that can have arbitrarily many objectives but inherently have a two-dimensional design space.

This work extends previous work of the authors on DBMOPPs by augmenting the generator with novel features including hard constraints and non-intersecting disconnected Pareto regions, which have not been considered in the DBMOPP literature so far. Furthermore, the generator has also been extended and includes an extensive range of features drawn from the DBMOPP literature, and some further feature extensions (such as hard constraints, and constraint regions with more complicated shapes).

We have validated the proposed generator by analyzing the problem instance space in terms of diversity, complexity, and difficulty for several widely used multi- and many-objective evolutionary algorithms. Key findings of the analysis are that the generator is able to produce test problem instances of different complexity, and that algorithm performance is a function of several problem features and not only of the number of objectives (which is mistakenly often assumed for dominance-based MOEAs). Most importantly, we can identify 'sweetspots' in the instance space that are more suitable for certain types of MOEAs (perhaps surprisingly, including sweetspots where random search performed best). The last finding in particular is of great value to practitioners: the suitability of DBMOPPs to model real location selection problems (as mentioned in the introduction) allows practitioners, such as local authorities and urban planners, to use their knowledge about the urban landscape to select a suitable algorithm to solve the (location selection) problem at hand.

Gaining an understanding of the relationship of all features incorporated in the proposed generator and their impact on algorithm performance is a difficult task, and this work has only scratched the surface in this regard. Future work will investigate the performance of existing methods for selected combinations of features (while switching off the other features or setting them be non-challenging as informed by this study), such as constraint-handling strategies combined with dynamic optimization methods. Insights gained from these studies are expected to inform the design of more efficient optimization methods including surrogate-assisted [52] and other data-driven methods [53], not considered in this work.

Acknowledgments

This work was supported by the Engineering and Physical Sciences Research Council [grant number EP/N017846/1]. This research is related to the thematic research area DEMO (jyu.fi/demo) of the University of Jyväskylä.

References

[1] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca, "Performance assessment of multiobjective optimizers: an analysis and review," Tech. Rep. 2, 2003.

[2] M. Li and X. Yao, "Quality evaluation of solution sets in multiobjective optimisation: A survey," *ACM Comp. Surv.*, vol. 52, no. 2, p. Article 26, 2019.

[3] T. Tušar and B. Filipič, "Visualization of Pareto front approximations in evolutionary multiobjective optimization: A critical review and the prosection method," *IEEE Trans. Evol. Comp.*, vol. 19, no. 2, pp. 225–245, 2015.

[4] M. Köppen and K. Yoshida, "Visualization of Pareto-sets in Evolutionary multi-objective optimization," in *Proc. of the 7th Int. Conf. on Hybrid Intelligent Systems*, 2007, pp. 156–161.

[5] J. E. Fieldsend and R. M. Everson, "Visualising high-dimensional Pareto relationships in two-dimensional scatterplots," in *Evol. Multi-Criterion Optimization, Proc*, R. Purshouse, P. Fleming, C. Fonseca, S. Greco, and J. Shaw, Eds., 2013, pp. 558–572.

[6] M. J. Walter, D. J. Walker, and M. J. Craven, "Visualising evolution history in multi-and many-objective optimisation," in *Parallel Problem Solving from Nature – PPSN XVI, Proc.* Springer, 2020, pp. 299–312.

[7] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable test problems for evolutionary multiobjective optimization," in *Evol. Multiobjective Optimization: Theoretical Advances and Applications*, A. Abraham and R. Goldberg, Eds. Springer, 2005, pp. 105–145.

[8] S. Huband, P. Hingston, L. Barone, and L. While, "A review of multiobjective test problems and a scalable test problem toolkit," *IEEE Trans. Evol. Comp.*, vol. 10, no. 5, pp. 477–506, 2006.

[9] T. Tusar, D. Brockhoff, N. Hansen, and A. Auger, "COCO: the bi-objective black box optimization benchmarking (bbob-biobj) test suite," *CoRR*, 2016. [Online]. Available: http://arxiv.org/abs/1604.00359

[10] D. K. Saxena, Q. Zhang, J. A. Duro, and A. Tiwari, "Framework for many-objective test problems with both simple and complicated Pareto-set shapes," in *Evol. Multi-Criterion Optimization, Proc*, R. Takahashi, K. Deb, E. Wanner, and S. Greco, Eds. Springer, 2011, pp. 197–211.

[11] Y.-M. Cheung, F. Gu, and H.-L. Liu, "Objective extraction for many-objective optimization problems: Algorithm and test problems," *IEEE Trans. Evol. Comp.*, vol. 20, no. 5, pp. 755–772, 2016.

[12] W. Chen, H. Ishibuchi, and K. Shang, "Proposal of a realistic many-objective test suite," in *Parallel Problem Solving from Nature – PPSN XVI, Proc.* Springer, 2020, pp. 201–214.

[13] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach," *IEEE Trans. Evol. Comp.*, vol. 3, no. 4, pp. 257–271, 1999.

[14] H. E. Aguirre and K. Tanaka, "Working principles, behavior, and performance of MOEAs on MNK-landscapes," *Eur. J. Oper. Res.*, vol. 181, no. 3, pp. 1670–1690, 2007.

[15] D. W. Corne and J. D. Knowles, "Techniques for highly multiobjective optimisation: some nondominated points are better than others," in *GECCO'07: Proc. of the Genetic and Evol. Comp. Conf.* ACM, 2007, pp. 773–780.

[16] H. Ishibuchi, T. Yoshida, and T. Murata, "Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling," *IEEE Trans. Evol. Comp.*, vol. 7, no. 2, pp. 204–223, 2003.

[17] R. Li, M. T. Emmerich, J. Eggermont, E. G. Bovenkamp, T. Bäck, J. Dijkstra, and J. H. Reiber, "Mixed-integer NK landscapes," in *Parallel Problem Solving from Nature – PPSN IX, Proc.* Springer, 2006, pp. 42–51.

[18] M. Farina, K. Deb, and P. Amato, "Dynamic multiobjective optimization problems: test cases, approximations, and applications," *IEEE Trans. Evol. Comp.*, vol. 8, no. 5, pp. 425–442, 2004.

[19] Z. Wang, Y.-S. Ong, J. Sun, A. Gupta, and Q. Zhang, "A generator for multiobjective test problems with difficult-to-approximate Pareto front boundaries," *IEEE Trans. Evol. Comp.*, vol. 23, no. 4, pp. 556—571, 2019.

[20] M. Köppen and K. Yoshida, "Visualization of Pareto-sets in evolutionary multi-objective optimization," in *Proc. of the 7th Int. Conf. on Hybrid Intelligent Systems.* IEEE, 2007, pp. 156–161.

[21] S. Zapotecas-Martínez, C. A. Coello Coello, H. E. Aguirre, and K. Tanaka, "A review of features and limitations of existing scalable multi-objective test suites," *IEEE Trans. Evol. Comp.*, vol. 23, no. 1, pp. 130–142, 2019.

[22] M. Köppen, R. Vicente-Garcia, and B. Nickolay, "Fuzzy-Pareto-dominance and its application in evolutionary multi-objective optimization," in *Evol. Multi-Criterion Optimization, Proc*, C. A. Coello Coello, A. H. Aguirre, and E. Zitzler, Eds. Springer, 2005, pp. 399–412.

[23] M. Köppen and K. Yoshida, "Substitute distance assignments in NSGA-II for handling many-objective optimization problems," in *Evol. Multi-Criterion Optimization, Proc*, S. Obayashi, K. Deb, K. Poloni, T. Hiroyasu, and T. Murata, Eds. Springer, 2007, pp. 727–741.

[24] H. Ishibuchi, Y. Hitotsuyanagi, N. Tsukamoto, and Y. Nojima, "Many-objective test problems to visually examine the behavior of multiobjective evolution in a decision space," in *Parallel Problem Solving from Nature – PPSN XI, Part II*, R. Schaefer, C. Cotta, J. Kolodziej, and G. Rudolph, Eds. Springer, 2010, pp. 91–100.

[25] H. Ishibuchi, N. Akedo, and Y. Nojima, "A many-objective test problem for visually examining diversity maintenance behavior in a decision space," in *GECCO'11: Proc. of the Genetic and Evol. Comp. Conf.* ACM, 2011, pp. 649–656.

[26] H. Masuda, Y. Nojima, and H. Ishibuchi, "Visual examination of the behavior of EMO algorithms for many-objective optimization with many decision variables," in *IEEE Congress on Evol. Comp., Proc.* IEEE, 2014, pp. 2633–2640.

[27] H. Zille and S. Mostaghim, "Properties of scalable distance minimization problems using the Manhattan metric," in *2015 IEEE Congress on Evol. Comp., Proc.* IEEE, 2015, pp. 2875–2882.

[28] J. Xu, K. Deb, and A. Gaur, "Identifying the Pareto-optimal solutions for multi-point distance minimization problem in Manhattan space," Michigan State University, Tech. Rep. COIN Report 2015018, 2015.

[29] J. E. Fieldsend, "Enabling dominance resistance in visualisable distance-based many-objective problems," in *GECCO'16: Proc. of the Genetic and Evol. Comp. Conf. Companion.* ACM, 2016, pp. 1429–1436.

[30] H. Zille, A. Kottenhahn, and S. Mostaghim, "Dynamic distance minimization problems for dynamic multi-objective optimization," in *2017 IEEE Congress on Evol. Comp., Proc*, 2017, pp. 952–959.

[31] Y. Liu, H. Ishibuchi, Y. Nojima, N. Masuyama, and K. Shang, "A double-niched evolutionary algorithm and its behavior on polygon-based problems," in *Parallel Problem Solving from Nature – PPSN XV, Proc.* Springer, 2018, pp. 262–273.

[32] Y. Nojima, T. Fukase, Y. Liu, N. Masuyama, and H. Ishibuchi, "Constrained multiobjective distance minimization problems," in *GECCO'19: Proc. of the Genetic and Evol. Comp. Conf.* ACM, 2019, pp. 586–594.

[33] J. E. Fieldsend, T. Chugh, R. Allmendinger, and K. Miettinen, "A feature rich distance-based many-objective visualisable test problem generator," in *GECCO'19: Proc. of the Genetic and Evol. Comp. Conf.* ACM, 2019, pp. 541–549.

[34] C. Günther and C. Tammer, "Relationships between constrained and unconstrained multi-objective optimization and application in location theory," *Math. Meth. Oper. Res.*, vol. 84, pp. 359–387, 2016.

[35] S. Alzorba, C. Günther, N. Popovici, and C. Tammer, "A new algorithm for solving planar multiobjective location problems involving the manhattan norm," *Eur. J. Oper. Res.*, vol. 258, no. 1, pp. 35–46, 2017.

[36] M. Li, S. Yang, and X. Liu, "A test problem for visual investigation of high-dimensional multi-objective search," in *2014 IEEE Congress on Evol. Comp., Proc.* IEEE, 2014, pp. 2140–2147.

[37] M. Li, C. Grosan, S. Yang, X. Liu, and X. Yao, "Multiline distance minimization: A visualized many-objective test problem suite," *IEEE Trans. Evol. Comp.*, vol. 22, no. 1, pp. 61–78, 2018.

[38] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comp.*, vol. 6, no. 2, pp. 182–197, 2002.

[39] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *IEEE Trans. Evol. Comp.*, vol. 11, pp. 712–731, 2007.

[40] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *Parallel Problem Solving from Nature – PPSN VIII, Proc.* Springer, 2004, pp. 832–842.

[41] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints," *IEEE Trans. Evol. Comp.*, vol. 18, no. 4, pp. 577–601, 2014.

[42] T. Bartz-Beielstein, "How to create generalizable results," in *Springer Handbook of Computational Intelligence*, J. Kacprzyk and W. Pedrycz, Eds. Berlin, Heidelberg: Springer, 2015, pp. 1127–1142.

[43] K. Miettinen, *Nonlinear Multiobjective Optimization.* Boston: Kluwer Academic Publishers, 1999.

[44] L. Paquete, T. Schiavinotto, and T. Stützle, "On local optima in multiobjective combinatorial optimization problems," *Annals of Operations Research*, vol. 156, pp. 83–97, 2007.

[45] J. E. Fieldsend and K. Alyahya, "Visualising the landscape of multi-objective problems using local optima networks," in *GECCO'19: Proc. of the Genetic and Evol. Comp. Conf. Companion*, 2019, pp. 1421–1429.

[46] S. Verel, A. Liefooghe, L. Jourdan, and C. Dhaenens, "Pareto local optima of multiobjective NK-landscapes with correlated objectives," in *Evol. Comp. in Combinatorial Optimization, Proc*, P. Merz and J.-K. Hao, Eds. Springer, 2011, pp. 226–237.

[47] T. Hanne, "On the convergence of multi objective evolutionary algorithms," *Eur. J. Oper. Res.*, vol. 117, pp. 553–564, 1999.

[48] Y. Liu, H. Ishibuchi, G. G. Yen, Y. Nojima, and N. Masuyama, "Handling imbalance between convergence and diversity in the decision space in evolutionary multimodal multiobjective optimization," *IEEE Trans. Evol. Comp.*, vol. 24, no. 3, pp. 551–565, 2020.

[49] T. Weise, S. Niemczyk, H. Skubch, R. Reichle, and K. Geihs, "A tunable model for multi-objective, epistatic, rugged, and neutral fitness landscapes," in *GECCO'08: Proc. of the Conf. on Genetic and Evol. Comp.* ACM, 2008, pp. 795–802.

[50] L. C. T. Bezerra, M. Lopez-Ibanez, and T. Stützle, "A large-scale experimental evaluation of high-performing multi- and many-objective evolutionary algorithms," *Evol. Comp.*, vol. 26, no. 4, pp. 621–656, 2018.

[51] H. Ishibuchi, T. Matsumoto, N. Masuyama, and Y. Nojima, "Many-objective problems are not always difficult for Pareto dominance-based evolutionary algorithms," in *Proc. of the 24th European Conf. on Artificial Intelligence*, 2020, pp. 291–298.

[52] R. Allmendinger, M. Emmerich, J. Hakanen, Y. Jin, and E. Rigoni, "Surrogate-assisted multicriteria optimisation: Complexities, prospective solutions, and business case," *J. Multi-Crit. Dec. Anal.*, vol. 24, no. 1-2, pp. 5–24, 2017.

[53] Y. Jin, H. Wang, T. Chugh, D. Guo, and K. Miettinen, "Data-driven evolutionary optimization: An overview and case studies," *IEEE Trans. Evol. Comp.*, vol. 23, no. 3, pp. 442–458, 2019.

Supplementary Material: A Visualizable Test Problem Generator for Many-Objective Optimization

Jonathan E. Fieldsend, *Member, IEEE*[1], Tinkle Chugh[2], Richard Allmendinger *Member, IEEE*[3], and Kaisa Miettinen[4]

[1]University of Exeter, Exeter EX4 4QD, U.K. Email: J.E.Fieldsend@exeter.ac.uk
[2]University of Exeter, Exeter EX4 4QD, U.K. Email: t.chugh@exeter.ac.uk
[3]University of Manchester, Manchester M15 6PB, U.K. Email: richard.allmendinger@manchester.ac.uk
[4]University of Jyvaskyla, Faculty of Information Technology, P.O. Box 35 (Agora), FI-40014 University of Jyvaskyla, Finland. Email: kaisa.miettinen@jyu.fi

1 Experimental settings

The following settings were used in running different MOEAs:

1. Population size: 100, 120, 132, 156 and 276 for two, four, six, eight and 10 objectives, respectively
2. Number of reference vectors in MOEA/D: Same as the population size except 275 for 10 objectives
3. Total number of generations: 200
4. Crossover type: Simulated binary crossover
5. Crossover probability and distribution index: 0.8 and 20, respectively
6. Mutation type: Polynomial mutation
7. Mutation probability and distribution index: $\frac{1}{\text{number of decision variables}}$ and 20, respectively

We used the method used in [1] to generate 275 reference vectors for 10 objectives to be used in MOEA/D. As NSGA-II typically works with a population size of an even number, we fixed the population size to 276 in all algorithms. This is the reason why the population size is different from the number of reference vectors for 10 objectives.

2 Number of instances tested

In this paper, we used the following instances and their all possible combinations to test different algorithms. In

Table 1: Instances of DBMOPP used in testing different algorithms

Feature	Instance
Non-identical Pareto sets	0, 1
Varying density	0, 1
Number of objectives	2, 4, 6, 8, 10
Number of variables	2, 3, 5, 10, 20
Number of disconnected Pareto sets	0, 2, 4, 6
Number of local fronts	0, 2, 4, 6
Number of dominance resistance regions	0, 2, 4, 6

Table 1, the varying density equal to one is not applicable when the number of decision variables is equal to two. Therefore, the number of possible combinations is 5760.

3 Details of Clustering

In this paper, we used k-means clustering [2] using cosine similarity. The cosine similarity estimates the similarity between two vectors by measuring cosine of angle (θ) between them. Given two vectors $\mathbf{u}$ and $\mathbf{v}$, the cosine similarity is estimated as:

$$cos(\theta) = \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}||||\mathbf{v}||},$$

where $||\mathbf{u}||$ and $||\mathbf{v}||$ represent the norms of vectors $\mathbf{u}$ and $\mathbf{v}$, respectively. The range of similarity lies between -1 and 1; -1 means vectors are in the opposite direction, +1 means vectors are exactly the same and in the same direction and 0 means vectors are orthogonal to each other. This results in clusters having vectors in the same direction and thus similar features are grouped together in Figures 5 and 6. In this work, we used 100 clusters based on trial and error and working on finding an appropriate number is one of our future research topics. For more details about k-means clustering using cosine similarity, see [3].

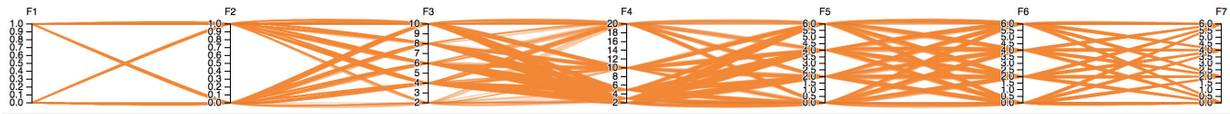

Figure 1: Features on which NSGA-II performed the best

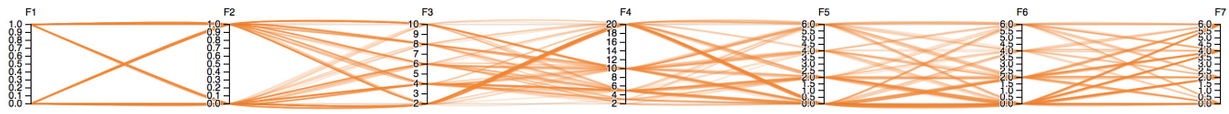

Figure 2: Features on which MOEA/D performed the best

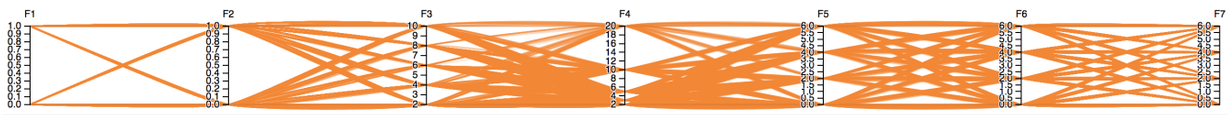

Figure 3: Features on which IBEA performed the best

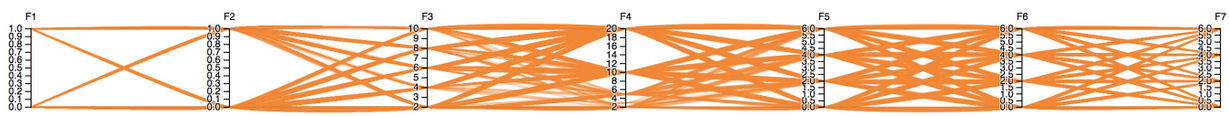

Figure 4: Features on which NSGA-III search performed the best

4 Parallel coordinate plots of solutions with different algorithms

We provide parallel coordinate plots of features with respect to the algorithm performances. Figures 1, 2, 3 and 4 show the feature vectors on which an algorithm performed the best. The observations from these results are the same as mentioned in the main manuscript.

5 The generator

Algorithm 1 sets out at a high-level the sequence of operations in constructing a DPMOPP instance in the generator. Prior to line 1 are listed the arguments to the object constructor, the collection of which (`args`) are checked for legality in line 1. They are subsequently used to initialize various matrices and structures to hold the state of the object, and encapsulated within the instance being constructed (line 4). Given the number of regions to be generated, and the proportion of the 2D space to be dedicated to checker-type constraints and objective function neutrality, the maximum radius that can be used for an attractor region is determined, and the regions are randomly placed sequentially, via rejection sampling (line 5, and Algorithm 2).

Once the attractor regions have been legally placed, the rotations to be applied to each attractor region are then set (line 6), and then the attractor points are placed on the circumference of each attractor region type (line 7). Following this a uniformly distributed set of N_{MC} points are generated in the visualizable plane, denoted M (line 8). All members of M falling inside the attractor regions are removed (line 9). This enables us to estimate the proportion of the domain $\mathcal{X}$ which are within the attractor regions ($\frac{|M|-N_{MC}}{N_{MC}}$), and that which is remaining to be available to be used (if required) for example for checkerboard constraint and neutral regions.

Lines 10-33 outline a set of operations, only a subset of which will ever be employed when setting up a single instance. If the `constraint_type` is either `soft_checker` or `hard_checker` (line 10), then elements of M are randomly selected, a checker constraint region centered on it and a radius for it is randomly selected from (0,min(c, r)], where c is the distance to the closest *attractor* region to the selected point from M, and r is the radius of the circle whose area would encompass the remaining proportion of checker type constraint to be allocated. Once the radius is selected, all elements lying within the region are removed from M, and the process is repeated until the proportion of checker type constrained space is achieved (line 11). A similar approach is used in line 14 for placing the neutral regions. Following this any discontinuous regions are placed (line 17), the vertex type constraint regions are placed (line 20), followed by center constraint type regions (line 23), and moat constraint regions (line 26). Finally the projection vectors used are set and the objective rescaling variables (lines 28-33).

References

[1] K. Deb and H. Jain, "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, 2014.

[2] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[3] S. Al-Anazi, H. AlMahmoud, and I. Al-Turaiki, "Finding similar documents using different clustering techniques," *Procedia Computer Science*, vol. 82, pp. 28–34, 2016.

Algorithm 1 DPMOPP instance generator

Require: K ▷ Number of objectives (integer, ≥ 2)
Require: N ▷ Number of variables (integer, ≥ 2)
Require: N_{lpf} ▷ Number of local Pareto fronts (integer, ≥ 0)
Require: N_{drr} ▷ Number of dominance resistance regions (integer, ≥ 0)
Require: N_{gps} ▷ Number of global Pareto sets (integer, ≥ 1)
Require: `prop_constrained_checker` ▷ Proportion of constrained 2D space if checker type (continuous, ≥ 0 & ≤ 1)
Require: `pareto_set_type` ▷ Disconnected Pareto set type, i.e. {`duplicate`, `partial`, `non_intersecting`}
Require: `constraint_type` ▷ Constraint type
Require: N_{dof} ▷ Number of discontinuous objective function regions (integer, ≥ 0)
Require: `vary_density` ▷ Vary solution density (Boolean)
Require: `vary_objective_scales` ▷ Vary objective scales (Boolean)
Require: `prop_neutral` ▷ Proportion of neutral space (continuous, ≥ 0 & ≤ 1)
Require: N_{MC} ▷ Number of samples used for approximating checker and neutral space coverage, (integer, ≥ 1000)
if `self.illegal_input_combination(args)` **then** ▷ Check arguments in correct ranges, and not in conflict
  return Error message ▷ Inform user of issue with constructor arguments
end if
Assign arguments to object state (`self`) ▷ Encapsulate argument as part of object state, and initialize other elements
`self.set_up_attractor_regions()` ▷ Calculate max maximum region radius given problem properties
`self.assign_attractor_region_rotations()` ▷ Set up rotations to be used by each attractor region
`self.place_attractor_points()` ▷ Randomly place attractor regions in 2D space
$M :=$ `self.uniform_sample_from_2D_domain()` ▷ Generate n_{MC} uniform samples in 2D
$M :=$ `self.remove_samples_in_attractor_regions(`M`)` ▷ Remove any samples falling in attractor regions
if `self.constraint_type` $\in$ {`soft_checker`, `hard_checker`} **then**
  $M :=$ `self.place_checker_constraint_locations(`M`)` ▷ Place checker constraint regions, update M
end if
if `prop_neutral` > 0 **then**
  `self.place_neutral_regions(`M`)` ▷ Place neural regions
end if
if $N_{dof} > 0$ **then**
  `self.place_discontinuous_regions()` ▷ Place regions whose boundaries cause discontinuities in objectives
end if
if `self.constraint_type` $\in$ {`soft_vertex`, `hard_vertex`} **then**
  `self.place_vertex_constraint_locations()` ▷ Place constraints located at attractor points
end if
if `self.constraint_type` $\in$ {`soft_center`, `hard_center`} **then**
  `self.place_center_constraint_locations()` ▷ Place center constraint regions
end if
if `self.constraint_type` $\in$ {`soft_moat`, `hard_moat`} **then**
  `self.place_moat_constraint_locations()` ▷ Place moat constraint regions
end if
if `self.vary_density` **then**
  `self.set_projection_vectors()` ▷ Set $\boldsymbol{\pi}_1$ and $\boldsymbol{\pi}_2$
end if
if `self.vary_objective_scales` **then**
  `self.set_objective_rescaling_variables()` ▷ Set offset and multiplier for objectives
end if
return Problem instance

Algorithm 2 `self.set_up_attractor_regions()` method

Require: `self` ▷ Current partially constructed object state
`max_radius` $:= \frac{1}{2\sqrt{\texttt{self.}N}+1}$
`max_radius` $:=$ `max_radius` $\times (1 - ($`self.prop_neutral` $+$ `self.prop_constrained_checker`$))$
if $N_{lps} > 0$ **then**
  `max_radius` $:=$ `self.place_attractor_region_centres(max_radius)`
  `self.reduce_region_radii()` ▷ Halve the radii for global Pareto regions, and distribute the local region radii between this and `max_radius`
end if