Universiteit Leiden
The Netherlands

Discovering the preference hypervolume: an interactive model for real world computational co-creativity

Hagg, A.

Citation

Hagg, A. (2021, December 7). *Discovering the preference hypervolume: an interactive model for real world computational co-creativity*. Retrieved from https://hdl.handle.net/1887/3245521

Version:	Publisher's Version
License:	Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from:	https://hdl.handle.net/1887/3245521

Note: To cite this publication please use the final published version (if applicable).

CHAPTER 5

The Preference Hypervolume

In this chapter I treat the question of how to integrate the user into the co-creative process. The objective is to find the *preference hypervolume*, the part of solution space that contains solutions preferred by the user.

This chapter consists of three parts. First, solutions are compressed into a smaller set of representatives, which form a Jungian extraverted intuition for the human (see Jung (1923)). Due to the nature of evolutionary algorithms, small genetic variations tend to lead to a population consisting of genetic clusters, situated around elite solutions. Evidence for this effect was provided by Vassiliades and Mouret (2018). In Section 5.1.3, solutions are compressed using an unsupervised clustering method and then presented to the user as prototypes. The communication of prototypes to the human provides a Hegelian reflection (see Hegel (1842)). After the user 'shops' for their preferred solutions, QD is restarted from the set of selected prototypes.

The rest of the chapter is devoted to constraining QD directly, through genetic constraints, and indirectly, through phenotypic constraints. In Section 5.2, QD is given an inherent genetic bias towards moving closer to the selected solutions. Finally, in Section 5.3, QD is given a phenotypic bias, which is more in line with findings from previous chapters, producing results that are closer to the expected outcome of a user's selection.

In order to fulfill the requirements A1 ("The process should be iterative"), A2 ("The process should be interactive and contain (at least one) objective participant"), A4 ("The process should capture design knowledge through designer interaction") and D1 ("Communication should be evocative through features such as visualization and

abstraction") from Tables 2.1–2.4, the following research questions are answered in this chapter:

Research Questions

- VII Can QD results be summarized using representatives? (requirement D1)
- VIII Can QD be influenced by the user by their selected representatives across domains? (requirements A1 and A2)
- IX Can selected prototypical genomes be modeled? (requirements A4 and D1)
- X Can we constrain parameters by penalizing QD's objective? (requirement A4)
- XI Can we constrain phenotypes by penalizing QD's objective? (requirement A4)

The main insights highlighted in this chapter are that QD results can be compressed and modeled, the user can take influence on QD by selecting from that compressed set, and user constraints provide more natural results by comparing phenotypic instead of genetic selection.

5.1 Genetic Prototypes

Divergent algorithms can find a large number of possible solutions, but at the same time this can hinder the engineer's ability to select interesting designs. As automated diversity gives too many solutions, their more concise presentation makes QD more useful to designers. With the ability to efficiently create a diverse set of solutions, the co-creative system is ready to communicate with the human. Results need to be presented to the users in a concise way, enabling them to select and influence the underlying search.

Instead of being presented with all solutions, in the design by shopping paradigm (see Balling (1999) and Section 2.3.2) the users are offered a compressed version of the solution set. Their selection is then used to influence the next divergent stage.

Research questions VII ("Can QD results be summarized using representatives?") and VIII ("Can QD be influenced by the user by their selected representatives across domains?") are answered in this section. The interactive co-creative process

introduced here uses genetic similarity to influence QD on a parameter level, aligned with more classical approaches of interactive evolutionary systems.

Vassiliades and Mouret (2018) showed that the elite hypervolume, the part of the parameter/genome space that contains QD solutions, is sometimes less spread out than the elites are in behavior space. It can be hypothesized that although phenotypic diversity might be high according to the phenotypic features used in literature, QD still acts like a classical EA, producing species that are genetically sensitive, covering a larger part of phenotypic space than of genetic space (see Section 3.1.2). Evidence for this was originally found by Lehman (2012), reporting on the benefit of pressure towards novelty. But when novelty produces such species, a direct consequence is that in genetic space, the population will contain clusters of individuals. So in order to compress the solution set produced by QD, it seems obvious to find these clusters and pick one representative each.

According to prototype theory, objects are part of the same class based on similarity. Wittgenstein (1953) questioned whether classes can be rigidly limited and implied that there is such a thing as a distance to a class. Rosch (1975) introduced prototype theory, stating that natural classes consist of a representatives and non-prototypical examples, which can be ranked in terms of distance to the prototype.

In this section, which is based on work by Hagg et al. (2018), genetic prototypes are determined with unsupervised clustering and are then used as seeds, or starting points, for QD. Finally, the interaction between QD's results and a user selection is analyzed. A quantitative analysis is performed on a two-dimensional airfoil optimization domain containing an inexpensive objective function, and a qualitative analysis is then completed on an expensive three-dimensional car mirror optimization problem.

5.1.1 Unsupervised Clustering

As shapes of optimal regions (the elite hypervolume) are not known beforehand, an unsupervised clustering technique that can handle concave as well as convex clusters is necessary to find them. The most commonly used density-based clustering algorithm is density-based spatial clustering of applications with noise (DBSCAN), proposed by Ester et al. (1996), which uses a maximum neighborhood distance ϵ to determine whether points belong to the *core* of a cluster or are a border point (see Fig. 5.1). A core point of a cluster has more than n neighbors within a radius of ϵ.

A border point is still part of a cluster, as it is connected to a core point, but has less than n neighbours. A point that is not reachable via a core point is considered either to be noise or a point of another cluster. The technique is able to be applied to data sets with different local densities and find clusters that are convex as well as concave. The parameter ϵ is found using the L-method by Salvador and Chan (2003), which determines the best trade-off between the number of clusters and the classification error. The minimum number of core points per cluster is set to four.

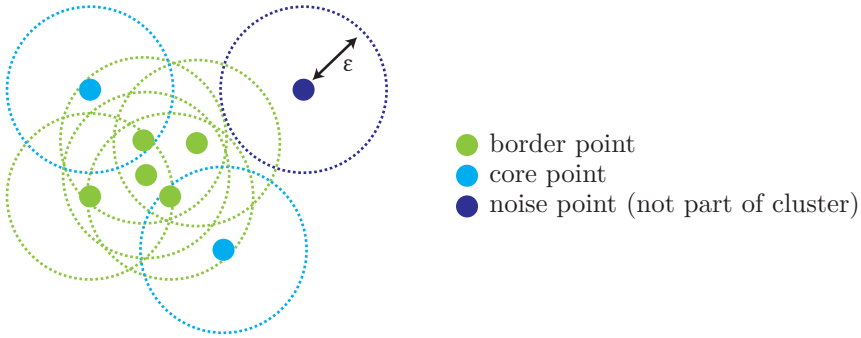

Figure 5.1: Density-based spatial clustering of applications with noise, introduced by Ester et al. (1996). The minimum number of neighbors n for a core point is four.

High Dimensionality Clustering depends on a notion of distance between points. The *curse of dimensionality* dictates that the relative difference of the distances of the closest and farthest data points goes to zero as the dimensionality increases (see Beyer et al. (1999a)). Often, the genome in evolutionary techniques consists of many parameters, causing distance metrics like Euclidean distance to become meaningless, which can happen for as few as 10–15 dimensions, shown by Beyer et al. (1999b). Clustering methods using a distance metric break down and cannot differentiate between points belonging to the same or to other clusters (see Tomašev and Radovanović (2016)). DR methods are applied to deal with this problem. Data is often located at or close to a manifold of lower dimension than the original space. DR transforms the data into a lower-dimensional space, enabling the clustering method to better distinguish clusters. Common DR methods are:

1. Principal component analysis (PCA), which uses eigenvector decomposition to find axes that are aligned to highest variance in data (in order of variance

magnitude)).

2. An alternative to PCA, kernel principal component analysis (kPCA) (see Schölkopf et al. (1997)), uses kernel[7] trick to transform n data points to n-dimensional kernel space that allow linear separation.

3. Isomap (see Tenenbaum et al. (2000)), which is a non-linear method that uses eigenvector decomposition on geodesic distances.

4. T-distributed Stochastic Neighborhood Embedding (t-SNE), see Maaten and Hinton (2008), which is an iterative process that maps points to two-dimensional space and keeps local neighborhood intact. It respects the local structure of high-dimensional data while discovering global structure (clusters at different scales). The method is explained in Appendix B.

5. Autoencoders (AE), proposed by Hinton and Salakhutdinov (2006), which are symmetrical neural networks with a central neural bottle neck.

5.1.2 Evaluation of Dimensionality Reduction and Clustering

The DR methods are analyzed as to whether they improve the clustering behavior of DBSCAN compared to applying clustering on the original dimensions.

The comparison needs to be robust w. r. t. differences in data dimensionality. Therefore metrics that use distance as a direct measure are not used. For example, the Silhouette index by Salvador and Chan (2003), a commonly used metric, when applying to qualitatively equivalent datasets of various dimensionalities (shown in Figure 5.2), converges to zero as the dimensionality increases.

Similar results were described by Tomašev and Radovanović (2016) in a survey on metrics for clustering in high dimensional spaces. The authors compared artificial datasets of different dimensionality but sampled from similar distributions (Gaussian, with diagonal covariance matrix). The indexes were run both on ground truth cluster configurations as well as k-means clustering. Taking into account the robustness w.r.t. dimensionality and variance stability of the indices, the $\bar{G}_+$ index is a good pick, according to Tomašev and Radovanović (2016), as it is practically independent of the dimensionality.

[7] non-linear function

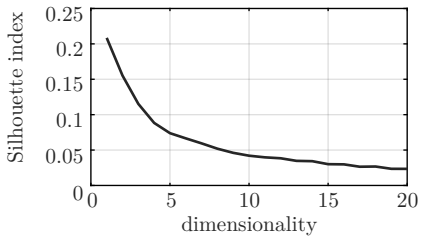

Figure 5.2: Silhouette index for a random cloud of samples in 1D to 20D. Two fixed Gaussian clusters sampled with diagonal covariance, one unit (in one dimension) apart from each other were used.

The $\bar{G}_+$ metric $\bar{G}_+$ indicates whether members of the same cluster are closer together than members of different clusters. To this end, the index measures discordance among pairs of distances. A pair of distances is said to be concordant in case the distance between objects from the same cluster is lower than the distance between objects belonging to different clusters. Therefore, a discordant pair of distances signalizes that the intra-cluster structure, or ranking, is not well separated. The index is calculated as follows:

$$\bar{G}_+ = \frac{2S_-}{t(t-1)}$$

with S_- being the total number of discordant pairs in the data with respect to the partitioning induced by the clustering algorithm and t the number of data points. $\bar{G}_+$ therefore is equal to the number of discordant pairs relative to the total number of data point pairs in the set. The metric is robust w.r.t. differences in dimensionality, shown by Tomašev and Radovanović (2016). A low value (≥ 0) indicates a high quality of clustering.

Evaluation A high-performing two-dimensional airfoil is optimized using free-form deformation, with ten degrees of freedom (see Fig. 5.3). This classic design problem is similar to the one defined by Gaier et al. (2017), but using a different representation. The base shape, an RAE2822 airfoil, is evaluated in XFOIL[8], at an angle of attack of 2.7°at Reynolds number 10^6 (Mach number 0.5).

[8] `web.mit.edu/drela/Public/web/xfoil/`

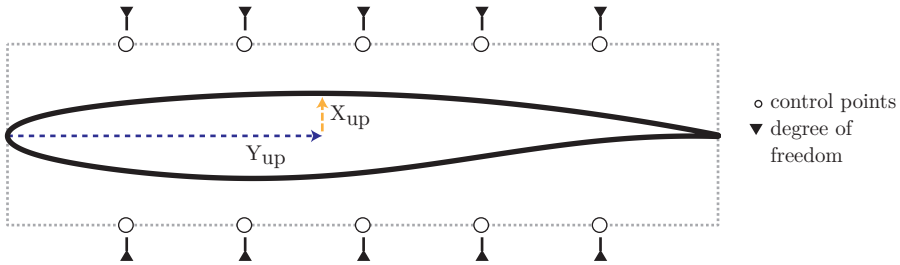

Figure 5.3: Two-dimensional airfoil encoding and features (X_{up}, Y_{up}).

The objective is to find diverse deformations, minimizing the drag coefficient c_D while keeping a similar lift force and area, described by Eq. 5.1–5.2

$$\text{fit}(x) = \text{drag}(x) \times p_{c_L}(x) \times p_A(x) \tag{5.1}$$

$$\text{drag} = -log(c_D(x)) \tag{5.2}$$

with A the area of the foil, and Eq. 5.3–5.4. The feature map, consisting of the x and y coordinates of the highest point on the foil (X_{up} and Y_{up}, see Fig. 5.3), is divided into a 25x25 grid, producing 625 solutions.

$$p_{c_L}(x) = \begin{cases} \frac{c_L(x)}{c_{L_0}}^2, c_L(x) < c_{L_0} \\ 1, \text{ otherwise } . \end{cases} \tag{5.3}$$

$$p_A(x) = \left(1 - \frac{|A - A_0|}{A_0}\right)^7 \tag{5.4}$$

PCA, kPCA, Isomap, t-SNE, and an AE are compared using DBSCAN on the latent spaces. All methods are used to reduce the dataset to a dimensionality of two. This increases explainability to the end user as it lets them visually inspect the results. The AE contains four consecutive, fully connect layers, consisting of 17, 7, 3, and 2 nodes. For t-SNE, perplexity is set to 50.

SAIL is evaluated 30 times on the two-dimensional airfoil domain. For every run of SAIL, the dimensionality of the resulting predicted optima is reduced with the various methods and the optima are clustered with DBSCAN. Table 5.1 shows that t-SNE allows DBSCAN to perform about an order of magnitude better than using other methods. Although t-SNE is not a convex method, it shows no variance,

indicating that it is quite robust. The number of clusters found is about twice as high as using other methods, and since the cluster separation is of higher quality, t-SNE is selected to reduce dimensionality in the rest of this evaluation.

Table 5.1: Quality of DR methods.

	Original	PCA	kPCA	Isomap	t-SNE	AE
mean G_+ score	0.36	0.33	0.22	0.30	**0.05**	0.454
mean number of clusters	4	5	7	4	**10**	4

An example t-SNE/DBSCAN clustering result is shown in Fig. 5.4. The clusters certainly contain solutions optically similar to their prototypes, yet some problematic classifications can be distinguished. The representation contains too much neutrality and/or sensitivity.

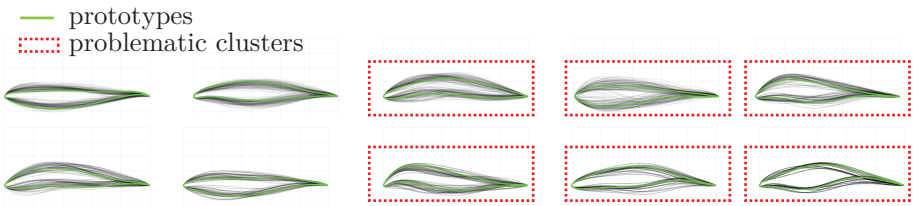

Figure 5.4: Example set of clusters found using t-SNE and DBSCAN on the airfoil domain. Prototypes are shown in green. Red boxes show problematic clusters, due to the wide variety of shapes contained within the cluster.

5.1.3 Genetic Seeding

Prototype discovery using quality diversity (PRODUQD) (pronounced: [prəˈdʌkt]), which performs a representative selection of designs, enables engineers to make design decisions more easily and to influence the search for optimal solutions.

A precise definition of PRODUQD can be found in Fig. 5.5 and Alg. 8. Initially, the design space is explored with a QD algorithm (line 6). In this section SAIL (see Alg. 7) is used, creating high-performing examples of designs.

A similarity space is constructed (line 7) using t-SNE (introduced by Maaten and Hinton (2008)), because this was shown to be the most appropriate dimensionality reduction method for this application (see Section 5.1.2). In this space, similar

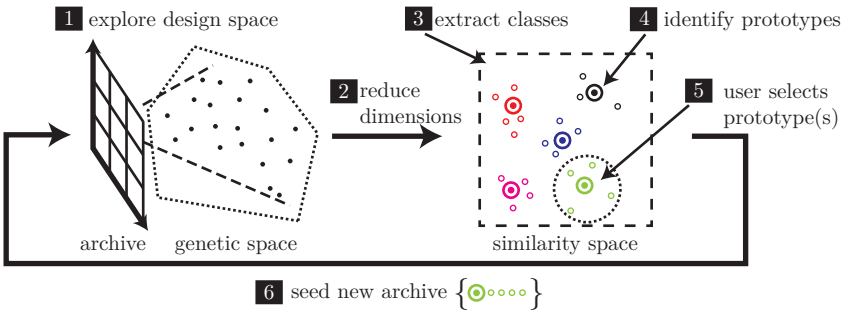

Figure 5.5: PRODUQD cycle. The design space is explored with QD (1). The archive's members' genomes are projected into a low-dimensional similarity space (2). In this space, classes are extracted (3) and one member of each class is chosen as a representative prototype (4). The user then selects one or more prototypes (5). The class members of all selected prototypes are used to seed the archive (6), after which QD is applied again to update the archive.

Algorithm 8 Prototype Discovery using Quality Diversity.

1: **procedure** PRODUQD
2: $\mathcal{X} \leftarrow \textsc{Sobol}(dim(\mathcal{X}))$
3: $\mathbf{f} \leftarrow \textsc{evaluate}(\mathcal{X})$
4: $budget \leftarrow budget/iterations$
5: **for** 1 to $iterations$ **do**
6: $(\mathcal{X}, \mathbf{f}_{pred}) \leftarrow \text{QD}(\mathcal{X}, \mathbf{f}, budget)$
7: $\mathcal{X}_{red} \leftarrow \textsc{dimensionality-reduction}(\mathcal{X})$
8: $\mathbf{c} \leftarrow \textsc{clustering}(\mathcal{X}_{red})$
9: $\mathbf{p} \leftarrow \textsc{prototype}(\mathcal{X}, \mathbf{c})$
10: $\mathbf{p}_{sel} \leftarrow \textsc{select}(\mathcal{X}(\mathbf{p}), \mathbf{f}_{pred}(\mathbf{p}))$ ▷ Performed by the user.
11: $\mathcal{X} \leftarrow \mathcal{X}(\mathbf{p}_{sel})$
12: **end for**
13: **end procedure**

solutions are clustered (line 8) into classes using DBSCAN (see Section 5.1.1). A prototype is extracted for every class (line 9) [9]. The most representative solution of a class is the member of a cluster that has the minimum distance to other

[9] In prototype theory, described by Rosch (1975), prototypes are those members of a class, "*with the most attributes in common with other members of the class and least attributes in common with other classes*".

members. The medoid is taken rather than a mean of the parameters, as the latter could be located in a non-optimal or even invalid region of the design space.

The prototypes are presented to the user (line 10), offering them a concise overview of the diversity in the generated designs. After the user selects one or more prototypes, the affiliated class members are used as seeds for the next QD iteration, serving as initial solutions in the acquisition and prediction maps (line 11). Initializing QD with individuals from the chosen class forces it to start its search within the class boundaries. Using a subset of untested solutions of a particular class stands in contrast to SAIL, which focuses on searching the entire design space, seeding both the acquisition and the prediction map with actual samples. Within each QD iteration, the surrogate model is retrained whenever new solutions are evaluated.

By clustering similar solutions into classes and representative prototypes, the optimization process is guided by extracting seeds from the classes selected by the user and reinserting them into QD as initial solutions for the next run, zooming in on a particular region in design space. This ideation process explores the design space while taking into account on-line design decisions.

5.1.4 Quantitative Analysis

To show PRODUQD's ability to produce designs based on a chosen prototype, it is replicated five times, selecting a different class in each run. In every iteration of PRODUQD, ten iterations of SAIL are run to acquire 100 new airfoils. The first iteration starts with an initial set of 50 samples from a Sobol sequence. Then, the five classes containing the largest number of optima are selected, and the algorithm is continued in separate runs for each class. After each iteration, the prototype that is closest to the one that was selected in the first iteration is marked. PRODUQD runs are compared to the original SAIL algorithm, using the same number of samples, a total of 500.

The similarity (distance in two-dimensional t-SNE similarity space) of designs to prototypes of optima found in four separate runs, selecting a different prototype in each one, are shown in Fig. 5.6. The usage of seeds does not always prevent the ideation algorithm of finding optima outside of the selection, but PRODUQD produces solutions that are more similar to the selected prototype than SAIL. The parameter spread in solutions found with PRODUQD is lower than with SAIL.

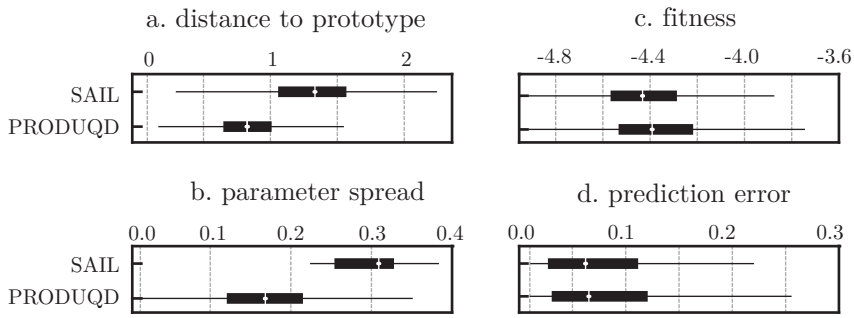

Figure 5.6: PRODUQD produces designs that are more similar to the selected prototype than using SAIL (the distance to prototypes is smaller). This is also visible in the smaller parameter spread. The produced designs have similar performance compared to SAIL's and the surrogate model is equally accurate.

Yet the true fitness scores and surrogate model prediction errors of both SAIL and PRODUQD are very similar.

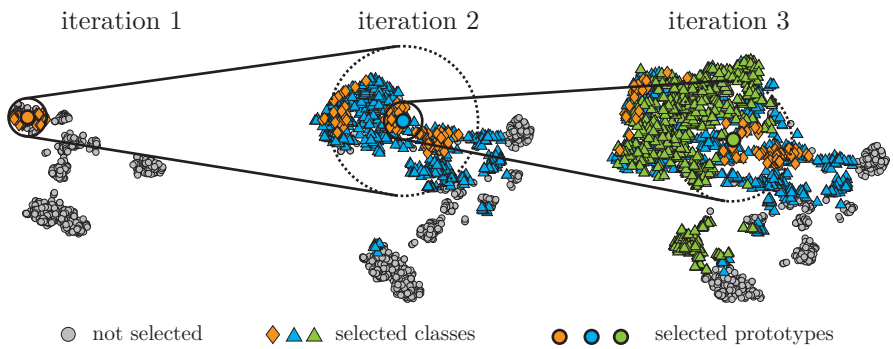

Figure 5.7: The region around the selected class is enlarged in similarity space and structure is discovered as more designs are added in later iterations. In each iteration the archive is filled with solutions from the selected class.

Fig. 5.7 shows the similarity space of three consecutive iterations. The effect of selection, zooming in on a particular region, can be seen by the fact that later iterations cover a larger part of similarity space, close to the prototype that was selected. Some designs still end up close to non-selected classes (in gray), which cannot be fully prevented without using constraints. PRODUQD is able to successfully illuminate local structure of the objective function around a prototype. It finds optima within a selected class of similar fitness to optima found in SAIL

using no selection, and is able to represent the solutions in a class in a more concise way, using prototypes as representatives, shown by the decreased variance within classes (see Fig. 5.6).

The performance of PRODUQD's designs is comparable to SAIL while directing the search towards design regions chosen by the user.

5.1.5 Qualitative Analysis

The second domain, a three-dimensional fluid dynamics problem from the automotive industry, is used to show results in a qualitative manner.

In the domain employed for qualitative analysis, the side mirror of the DrivAer car model (see Heft et al. (2012)) is optimized with a 51 parameter free-form deformation (see Fig. 5.8). The objective is to find many diverse solutions while minimizing the drag force (in Newtons) of the mirror. The numerical solver OpenFOAM[10] is used to determine flow characteristics and calculate the drag force. The feature map, consisting of the curvature of the edge of the reflective part of the mirror and the length of the mirror in flow direction, is divided into a 16x16 grid.

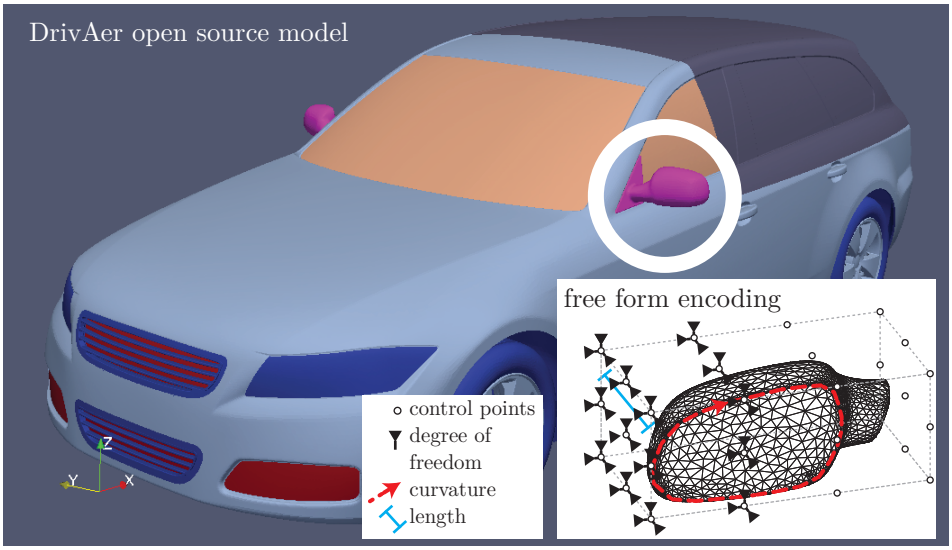

Figure 5.8: Three-dimensional mirror encoding and features (curvature and length).

[10] `openfoam.org`, simulation at 11 m/s.

A two-dimensional feature map, consisting of the curvature and the length of the mirror in flow direction (see Fig. 5.3), is illuminated from an initial set of 100 car mirror designs from a Sobol sequence. After acquiring 200 new samples with SAIL, a final archive is produced. From this set of solutions the two prototypes having the greatest distance to each other are selected. Then, PRODUQD is continued in two separate instances, sampling another 100 examples. Then the newly discovered prototype that is closest to the one first selected is used to perform two more iterations, resulting in a surrogate model trained with 600 samples.

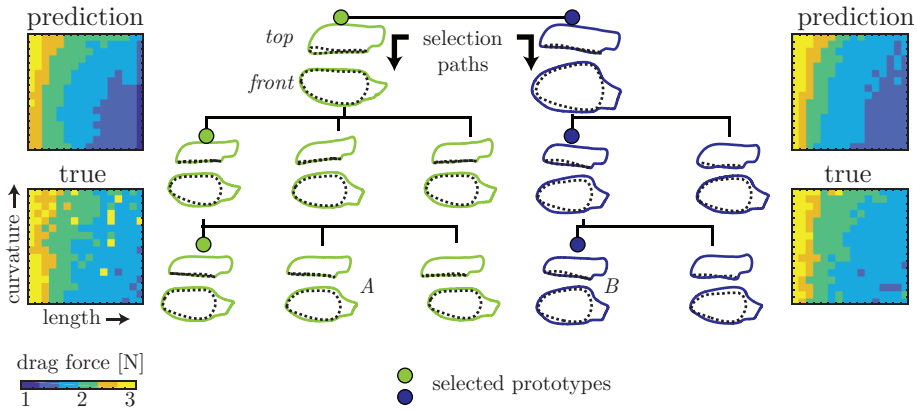

Figure 5.9: Phylogenetic tree of two PRODUQD runs diverging after first iteration, and predicted drag force maps (ground truth values are shown underneath).

The two resulting runs are shown in Fig. 5.9. Every branch in the phylogenetic tree of designs represents a selected prototype and every layer contains the prototypes found in an iteration. On average, 18 prototypes were found in each iteration.

Although the prototype selection does seem to influence the general shape of the mirrors, the results drift away from the selection. A clear example is the similarity between mirrors which are marked A and B in the figure. Although in the left run (in green), the red prototype was deselected, mirror A does resemble it more than it resembles the green prototype.

The predicted performance of the maps after the final selection is shown at the top. Every resulting mirror was evaluated with OpenFOAM to receive their respective ground truth fitness values. Ground truth fitness maps are shown underneath the predicted maps. The surrogate model gives an accurate prediction of the drag force, as the general fitness trends are captured in the maps.

5.1.6 Conclusions

QD can produce a large array of solutions, possibly impeding an engineer's capabilities of making a design decision. This section introduced computer-aided ideation, using QD in conjunction with a state of the art dimensionality reduction and a standard clustering technique, grouping similar solutions into classes and their representative prototypes. These prototypes can be selected to constrain QD in a next iteration of design space exploration by seeding it with the selected class.

This section gave evidence for an answer to research questions VII ("Can QD results be summarized using representatives?") and VIII ("Can QD be influenced by the user by their selected representatives across domains?"). Answering these questions was necessary to fulfill requirements A1 ("The process should be iterative"), A2 ("The process should be interactive and contain (at least one) objective participant") and D1 ("Communication should be evocative through features such as visualization and abstraction").

QD results are summarized using representative genetic prototypes, answering question VII in a positive manner. By using the fact that the population consists of clusters in genetic space, unsupervised clustering on a dimensionality reduced space can be used to determine how many prototypes are found by QD. The fact that the QD archive contains a relatively small set of design classes provides more evidence that is in line with the analysis by Vassiliades and Mouret (2018).

The similarity space can be used continuously as it is decoupled from the feature map. This allows the diversity metric, the feature characterization, to change between iterations, although the effects have not been analyzed here. Finally, although the median solution might be most similar to all solutions within a class, one indeed might choose the fittest solution of a class as its representative.

QD can be influenced by using selected classes as seeds, but the answer to question VIII remains inconclusive. A problem persists with the resulting mirror shapes drifting away from the selected prototypes, due to QD not being constrained. This is probably due to the neutrality and sensitivity effects that were described in Section 3.1.2. These problems are addressed in the next sections.

5.2 Modeling User Selection Drift

In the last section, the initial overview of the design space produced by QD was analyzed by clustering QD's resulting genomes in an unsupervised manner, forming design classes and prototypical representatives. Seeding the archive guides QD towards the selected prototype. Yet, it is not sufficient to guarantee that QD only produces solutions within their classes. In this section, based on work by Hagg et al. (2019), the feasibility and accuracy of modeling the user's selection and constraining QD based on that model is evaluated for both a closed form encoding as well as a neural encoding. I answer research questions IX ("Can selected prototypical genomes be modeled?") and X ("Can we constrain parameters by penalizing QD's objective?"). The model and constraints here will be applied to phenotypes in the next section.

The unconstrained objective function still allows QD to find solutions within non-selected regions. As a design process consists of many design decisions, it is unclear how the seeding approach would be successful in that case. The subspace that contains solutions fulfilling all of the user's decisions becomes smaller and more complex as more design decisions are added, but QD is divergent and will discover solutions outside of the selection. QD has to be constrained by modeling the user's selection decisions.

The genome-based selection process is subject to another problem. When a sensitive, non-linear phenotypic mapping is used or the phenotype's behavior is reactive, e.g. when evolving neural network controllers or producing shapes from neural representations, as done by Clune and Lipson (2004), small changes in the genome can lead to large changes in the phenotype/behavior. The designer expects that the continuation of the design process will produce similar solutions. But when using genetic similarity, such a non-linear mapping causes unexpected behaviors to be discovered, which would be counterintuitive to the designer. Therefore, in this section the user's selection is modeled, which allows comparing candidate solutions to the selected ones.

5.2.1 User Decision Hypersurface Model

A user's selection needs to be modeled to properly and continuously constrain the optimization process. QD fills its niching archive based on the behavior or phenotype of solutions (see Fig. 5.10a), but the search itself still takes place in genetic space (see Fig. 5.10b). Explicit search constraints need to be defined on the genome.

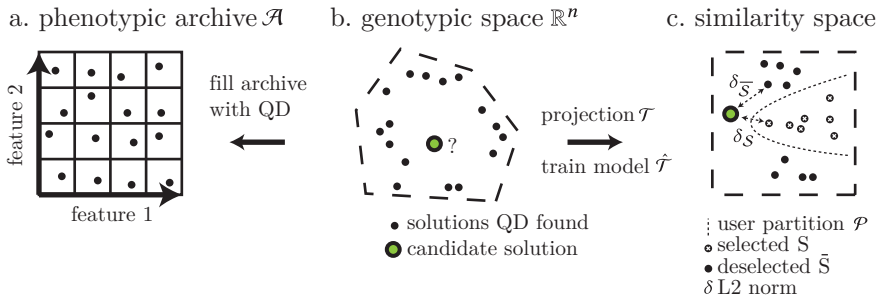

Figure 5.10: QD searches through genetic space $\mathbb{R}^n$ (b) to fill an archive $\mathcal{A}$ of diverse, high-performing phenotypes (a) in a low-dimensional phenotypic (or behavior) space. The genetic dimensionality n can be very high. By projecting the archive's members onto a low-dimensional similarity space (c), the user's selection can be modeled. The projection model $\widehat{\mathcal{T}}$ allows making comparisons of candidate solutions to the user selection $\mathcal{S}$ on the hypersurface by using an L2 norm.

Shaham and Steinerberger (2017) and Hagg et al. (2018) showed that when the dimensionality of genetic space is reduced with t-SNE, more structure in the objective landscape can be discovered by clustering techniques that are based on the notion of distance. This provides evidence that distances measured in the similarity space resulting from t-SNE are more effective in describing genetic similarity than those measured in the original genetic space.

The user's decision is modeled in the similarity space. This space is created based on a t-SNE mapping applied to solutions that have been placed in the QD archive. This creates a snapshot of the space at the time in which a decision was made. The set of members in the archive represent a hypersurface that unfolds in $\mathbb{R}^n$ as new solutions are added to empty niches and moves through $\mathbb{R}^n$ as solutions in niches are exchanged. Every niche in the archive can exclusively contain points of a subspace of $\mathbb{R}^n$, but a particular instance of an archive can be mapped into a

lower dimensional space that retains the structure of the archive, while allowing to measure distances on this approximation of the decision hypersurface.

The user decision hypersurface model (UDHM) is formalized as follows:

$$
\begin{aligned}
\mathcal{A} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}, \mathbf{x} \in \mathbb{R}^n &: \text{QD archive with } m \text{ points} \\
\mathcal{H} = \text{span}(\mathcal{A}) &: \text{decision hypersurface} \\
\mathcal{T} : \mathcal{A} \to \mathcal{A}' \subseteq \mathbb{R}^d &: \text{projection into similarity space} \\
\widehat{\mathcal{T}} : \mathcal{H} \subseteq \mathbb{R}^n \to \mathcal{H}' \subseteq \mathbb{R}^d &: \text{projection model} \\
\delta : \mathbb{R}^d \to \mathbb{R} &: \text{distance measure, e.g. L2 norm} \\
\mathcal{S}, \overline{\mathcal{S}} &: \text{selected/deselected solutions} \\
\mathcal{P} = \{\mathcal{S}, \overline{\mathcal{S}}\} &: \text{binary selection partition} \\
\mathcal{M} = (\widehat{\mathcal{T}}, \delta, \mathcal{P}) &: \text{user decision hypersurface model}
\end{aligned}
$$

The span of the points in the archive $\mathcal{A}$ defines the decision hypersurface $\mathcal{H}$ that is projected into similarity space $\mathbb{R}^d$ using t-SNE ($\mathcal{T}$). A dimensionality of $d = 2$ is chosen for visualization purposes and because the projection method, t-SNE, has been robustly tested for this case.

The dimensionality reduction method, t-SNE, does not provide a model, but merely maps high-dimensional points onto a lower-dimensional space. It is sensitive to local optima of its cost function, and therefore is not guaranteed to produce the same result in multiple runs. Due to this sensitivity and the performance cost of the calculation, t-SNE's mapping is turned into a predictive model using GP.

Separate GP models for each of the d similarity space coordinates result in the projection model $\widehat{\mathcal{T}}$. The GP models are trained using the archive members based on which a decision is made. The models describe the knowledge that was present at that point in time. The models' isotropic Matérn kernel is applied on the coordinates of the model's training samples in $\mathbb{R}^d$. The length scale priors to training are set to the mean Euclidean distance between the samples in $\mathbb{R}^d$.

The projection model $\widehat{\mathcal{T}}$, consisting of the two GP models, is used to penalize the fitness of a solution based on its distance to selected solutions. The model prevents unexpected drift that can be caused by changes that would arise due to recalculation of the t-SNE mapping.

The user decides which solutions they would like to further investigate, represented by the binary selection partition $\mathcal{P}$ (see Fig. 5.10c). The UDHM $\mathcal{M}$, which contains the projection model $\widehat{\mathcal{T}}$, a metric δ and the user's selection $\mathcal{P}$ allows us to compare candidate solutions to the user's selection based on their genomes, in similarity space. A decision metric can now be defined that determines whether a candidate solution is closer to $\mathcal{S}$ or to $\overline{\mathcal{S}}$.

User Selection Drift In order to measure how close a solution is to the user selection, a metric called *user selection drift* $d_{\mathcal{M}}$ is introduced. It compares the distance of a candidate solution $\mathbf{x}_c$ to the set of selected $\mathcal{S}$ and to the set of deselected examples $\overline{\mathcal{S}}$:

$$\delta_{\mathcal{S}} = min(\delta(\mathcal{T}(\mathbf{x}_c), \mathcal{S})) : \text{min. distance to selected}$$
$$\delta_{\overline{\mathcal{S}}} = min(\delta(\mathcal{T}(\mathbf{x}_c), \overline{\mathcal{S}})) : \text{min. distance to deselected}$$
$$d_{\mathcal{M}}(\mathbf{x}_c) = \frac{\delta_{\mathcal{S}}}{(\delta_{\mathcal{S}} + \delta_{\overline{\mathcal{S}}})}, 0 \leq d \leq 1 : \text{(normalized) user selection drift}$$

The parameter $d_{\mathcal{M}}$ measures the distance $\delta_{\mathcal{S}}$ between $\mathbf{x}_c$ and the closest point in $\mathcal{S}$, and distance $\delta_{\overline{\mathcal{S}}}$ between it and the closest point in $\overline{\mathcal{S}}$ (see Fig. 5.11). The normalized user selection drift equals zero when it exactly matches a selected point and equals to one when the candidate has the same coordinates as a deselected point. The UDHM and user selection drift can be used to augment the objective function with, for example, a penalty metric.

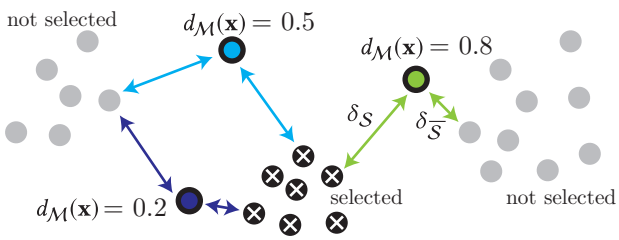

Figure 5.11: User selection drift $d_{\mathcal{M}}$ is based on the distance to the closest selected point $\delta_{\mathcal{S}}$ and the distance to the closest deselected point $\delta_{\overline{\mathcal{S}}}$.

Integration of Model To make use of the UDHM, PRODUQD is extended by including the UDHM. The algorithm is formalized in Algorithm 9. The projection model $\widehat{\mathcal{T}}$ is trained using the archive members' genomes, the selection vector (partition $\mathcal{S}$) and the vector that contains the solutions that were not selected by the user (partition ($\overline{\mathcal{S}}$)) in line 11. The user selection is turned into a penalty function and the objective function is adjusted in line 12, according to the following equation:

$$
\begin{aligned}
d_{\mathcal{M}}(\mathbf{x}) &: \text{drift penalty} \\
w_p &: \text{penalty weight} \\
f'(\mathbf{x}) = f(\mathbf{x}) \cdot (1 - w_p \cdot d_{\mathcal{M}}(\mathbf{x})) &: \text{adjusted objective (maximization)}
\end{aligned}
$$

The penalty, a value between zero and one, is used to penalize a solution's fitness.

Algorithm 9 PRODUQD with UDHM.

1: **procedure** PRODUQD/UDHM
2: $\mathcal{X} \leftarrow \textsc{Sobol}(dim(\mathcal{X}))$
3: $\mathbf{f} \leftarrow \textsc{evaluate}(\mathcal{X})$
4: $budget \leftarrow budget/iterations$
5: **for** 1 to $iterations$ **do**
6: $(\mathcal{X}, \mathbf{f}_{pred}) \leftarrow \text{QD}(\mathcal{X}, \mathbf{f}, budget)$
7: $\mathcal{X}_{red} \leftarrow \textsc{dimensionality-reduction}(\mathcal{X})$
8: $\mathbf{c} \leftarrow \textsc{clustering}(\mathcal{X}_{red})$
9: $\mathbf{p} \leftarrow \textsc{prototype}(\mathcal{X}, \mathbf{c})$
10: $\mathbf{p}_{sel} \leftarrow \textsc{select}(\mathcal{X}(\mathbf{p}), \mathbf{f}_{pred}(\mathbf{p}))$ ▷ Performed by the user.
11: $\widehat{\mathcal{T}} \leftarrow \textsc{train}(\mathcal{X}, \mathbf{c}, \mathbf{p}_{sel}, \neg\mathbf{p}_{sel})$ ▷ Train user selection model based on archive member genomes, clustering and selection vector.
12: $f() \leftarrow (w_p, \widehat{\mathcal{T}})$ ▷ Use UDHM to penalize fitness.
13: $\mathcal{X} \leftarrow \mathcal{X}(\mathbf{p}_{sel})$
14: **end for**
15: **end procedure**

The UDHM determines how far a solution might be mutated away from a known selected one by measuring the distance to solutions from the state of the archive at the time the decision was made. The measurement is taken in similarity space, in our case created by t-SNE. As this method compresses the parameters into

a lower dimensional space, the space is not homogeneous and measurements in that space cannot be translated to a Euclidean measurement in parameter space. Neither a simple threshold can be used to determine whether a candidate is closer to one solution or the other, nor can it be assumed that there is a universal penalty function that works in all domains and in all t-SNE projections. Therefore, a simple linear penalty function is used, which is zero at points in $\mathcal{S}$ and equals weight w_p at a known point in $\overline{\mathcal{S}}$, to influence the objective function. The penalty is used to scale the objective function. It is dependent on the range of the fitness function values as well as the structure of the hyperspace. For this reason the penalty has to be parameterized for each domain and task. A solution's fitness is penalized when it is unlikely to belong to $\mathcal{S}$, but it will still be accepted when there is no alternative solution, because in the conceptual phase of an engineering task it is better to show alternatives in a niche than showing no solutions at all. By showing the user selection drift per niche, they can be informed about its supposed distance to the selection.

In the following section, the model is evaluated for a linear genome to phenotype mapping as well as a nonlinear, reactive case. Both representations' behaviors are measured in a similar behavior space.

5.2.2 Experimental Setup

QD uses an objective function that is adjusted with a penalty based on UDHM. Whether less user selection drift takes place than only using seeding needs to be evaluated. This takes place in two tasks that are both defined in the same domain using the same selection criteria, objective function and diversity measure. Because design decisions are based on the way a solution is expressed in the problem domain, which can be a shape or a behavior, the selection process that is introduced is based on the behavior of solutions.

The domain that is encoded both in a geometric and a neural manner, has yet the same phenotypic space. The multimodal maze presented in Fig. 4.13 is used again. In this domain, solutions are expressed as paths through the maze but through two different encodings.

The first task is a path planning problem in which the genome to phenotype mapping is very simple (see Fig. 5.12c). A solution is encoded by a sequence of seven (x, y) nodes that, when connected, form a path. The range of the node

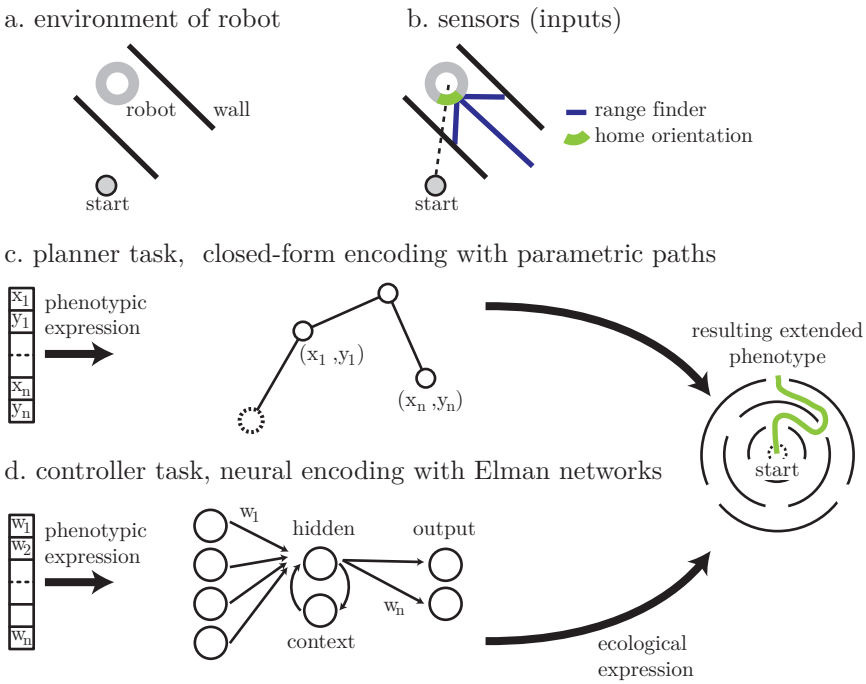

Figure 5.12: Closed form (c) and neural encoding using an Elman network (d) of maze problem (a, b) with the same (extended) phenotypic space.

coordinates is limited between -200 and 200. Diversity is aligned to the maze and the closed form encoding ensures that solutions showing the same phenotypic behavior, e.g. take similar paths, are also similar in terms of their genomes. For this task phenotypes and behavior are the same. Because a small change in the genome causes a small change in the phenotype, behavioral similarity matches genetic similarity and the UDHM should perform well.

In the second task (see Fig. 5.12d), optimization of neural robot controllers, small changes in the genome lead to large changes in the phenotype due to the sensitive encoding of the neural controller. Solutions are evaluated in the same maze as the path planning task, using the simulation that was created by Mouret (2011a). A robot is equipped with three range finders that are able to detect the distance to the nearest walls, and a home beacon that identifies the quadrant in which the direction to the start position of the robot lies. The sensory information is used as an input to a neural network. The robot is controlled by a derivation of a recurrent Elman network (see Elman (1990)) which controls two outputs: forward/backward

and rotational movement. The network contains five hidden neurons and five context neurons, whereby the weights to the context layer are evolved as well, for a total of 92 weights. The weights' range lies between minus three and three. The simulation is run for 1000 time steps.

The paths taken by the robots are not directly encoded in the genome, but result from the interaction of the phenotype, the neural network, with the environment. Therefore, similar controllers could display different behavior, which should make the comparison of solutions by their genomes less effective. The task is thus useful to show limits of the comparison in similarity space when using a non-linearly coupled genome, phenotype and behavior. With this final task the limits of the approach are tested.

The MAP-Elites archive is set to contain 900 (30 x 30) elites to ensure that the original t-SNE implementation is able to converge within a relatively short time. For larger archives it is recommended to use Barnes-Hut t-SNE (see Maaten (2014)) which is able to deal with much larger data sets. MAP-Elites is initialized with 2000 (planner) or 200 (controller) solutions. Each configuration is repeated six times for all three exits to account for stochastic effects. The initial orientation of the robot is changed by 60° steps starting at 30° between runs of the algorithm. In the path planning task the population is initialized using a normal distribution with a small σ to ensure that most initial paths are within the center area of the maze to prevent many invalid solutions. In the control task the controller weights are chosen from a space-filling Sobol sequence. Each initial archive created with a MAP-Elites run takes 8192 generations in the path planning task, and 2048 in the control task. In every generation, 32 children are created through normally distributed mutation with $\sigma = 5\%$ for the planning and $\sigma = 1\%$ for the control task. Parent selection is done by randomly choosing solutions from the archive.

The GP models for both coordinates use an isotropic Matérn kernel. The length scale hyperparameter prior is set to the mean Euclidean distance between the points in the archive in $\mathbb{R}^n$ (see Rasmussen (2004)). The length scale prior can make or break the model's ability to correctly compare solutions, as the training of the GP models will not converge. When the length scale is too short, the penalty will be too high for candidate solutions close to $\mathcal{S}$. When the length scale is too long, the penalty will be too low for those close to $\overline{\mathcal{S}}$.

After filling the archive with an initial set of solutions, the user will select those that escape the inner ring of the maze through a particular exit in the inner ring. With a trained UDHM model, MAP-Elites is run for 4096 generations in the second constrained iteration based on the user's selection. The expectation is that the solutions in the second iteration take the same exit as the one selected by the user.

5.2.3 Selection on Hypersurface

Some example results from both tasks are shown here in more detail. Fig. 5.13 shows paths that are found by QD in the first iteration without user selection in the path planning task (top row). Paths that do not get out are marked in gray. The user selects the solutions that take the preferred exit in the inner ring of the maze, in this case the lower left exit, marked in red. The hypersurface, or similarity space, shows that solutions that are close together tend to take the same exit.

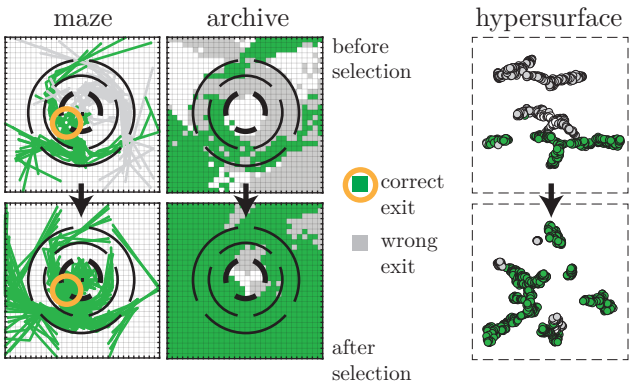

Figure 5.13: Path planning task with closed form encoding before (top) and after (bottom) selection of the 3rd exit. Left: example paths (every 5th). Center: end points of paths in QD archive with color assigned depending on whether exit was selected. Right: points projected onto the decision hypersurface.

An example archive that is produced after selection of the lower left exit and continuing MAP-Elites for another 4096 generations is shown in the bottom row of Fig. 5.13. MAP-Elites uses the adjusted objective function (Section 5.2.1), resulting in an archive that is mostly filled by paths that take the chosen exit

(marked in green). The most right column of Fig. 5.13 shows the archive's contents projected onto the hypersurface before and after MAP-Elites has been adjusted for selection. Solutions in the archive are well separated according to their behavior, the exit they took in the inner ring, which is to be expected with a direct genome to phenotype mapping.

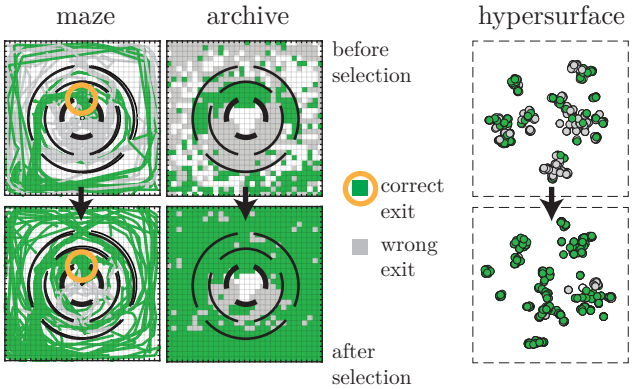

Figure 5.14: Neurocontrol task with neural encoding before (top) and after (bottom) selection of the 1st exit. Left: example paths (every 5th). Center: end points of paths in QD archive with color assigned depending on whether exit was selected. Right: points projected onto the decision hypersurface.

The solutions in the robot neurocontrol task are not as well separated on the hypersurface (right column of Fig. 5.14). This is most likely due to the encoding's predicted degree of neutrality and sensitivity. The paths the robots take look qualitatively different from the ones in the path planning task and the archive does not fill up in the same way. Yet the user selection is still effective, as can be seen by the archive being filled up almost entirely by solutions that take the preferred exit (marked in green).

5.2.4 Influence of Penalty Weight on Drift

The penalty weight's efficacy is evaluated with a mutation distance of 5% of the range of the genes for 4096 generations. The percentage point improvement of selected and deselected solutions that are found against baseline runs with the weight set to zero is measured. Fig. 5.15 shows those runs for the planner and control tasks, with six replicates per penalty weight setting.

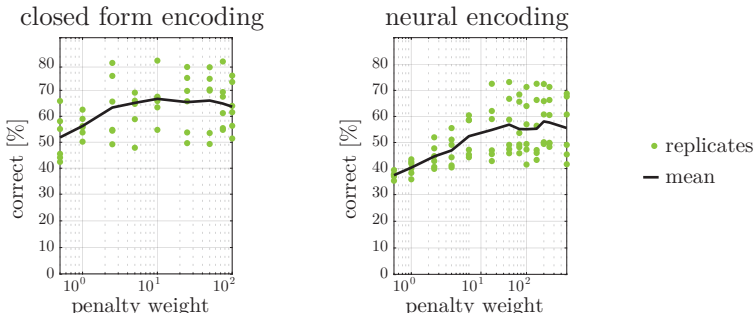

Figure 5.15: Influence of drift penalty weight on percentage of correct exits for the closed form and neural encodings. Three exit selections were replicated six times, totaling 18 runs.

The penalty derived from the UDHM behaves in a similar fashion for both tasks, although the optimal weight is higher in the control task. Because the fitness function range is the same for both tasks, the difference has to be fully explained by the structure of the hypersurface. As was already visible in Fig. 5.14, the solutions are not as well separated in the case of the control task. The linear penalty function therefore has to be set to a more conservative, higher value, in order for it to be effective. The optimal weight setting for this domain is equal to 10 for the planning task and 120 for the control task, after which the effect of increasing the penalty weight suffers from diminishing returns.

5.2.5 Comparing Seeding and Modeling

In this section, UDHM is compared against the QD seeding approach that was introduced in the previous section. It is expected to have less user selection drift, as it constrains QD to find solutions closer to those that were originally deselected by the user. The amount of user selection drift should be correlated to the mutation distance used in QD. Higher mutation distances should lead to more drift in the unconstrained seeding approach as they will allow solutions to be mutated to such a degree that they can jump over to the next basin of attraction. Because solutions are not removed from the archive when only using an UDHM-adjusted objective function, the random sampling in MAP-Elites will lead to discovering new valid solutions less often than when seeding, as deselected solutions will still be picked for mutation as often as selected ones.

The user selection drift of the seeding and UDHM approaches and their combination is evaluated by varying the QD mutation distance and evaluating the user selection drift that was defined in Section 5.2.1. The higher the mutation distance, the larger the drift is expected to be, especially when not using the UDHM to constrain QD. Neither UDHM nor seeding can fully prevent discovering novel solutions that do not fulfill the selection criterion, so drift should occur in both approaches, but it should be lower for the UDHM model.

Path Planning Task with Closed Form Encoding As becomes clear from the results of the planning task on the left of Fig. 5.16, using a UDHM in QD enables suppressing user selection drift. Since PRODUQD starts its search at all locations in $\mathcal{S}$ and $\overline{\mathcal{S}}$, the higher the mutation distance is, the more often a deselected solution can mutate towards $\mathcal{S}$. A large selection drift can be seen for small mutation distances when only using the UDHM, because without seeding, and with only a limited number of generations, the adjusted PRODUQD algorithm can not mutate solutions far enough away from the deselected set.

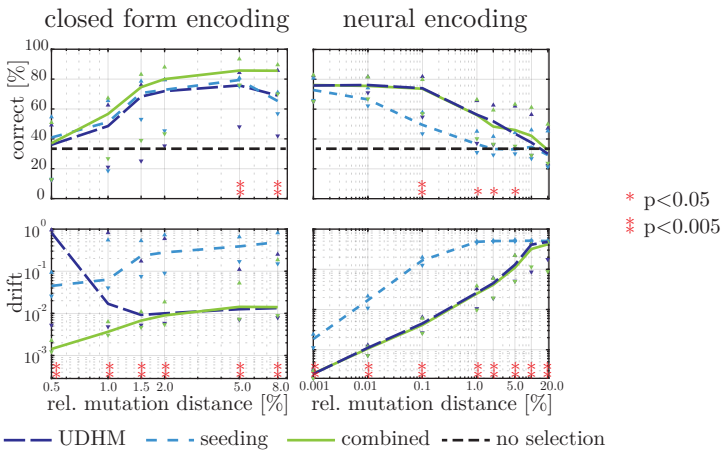

Figure 5.16: Median percentage of correct solutions and median user selection drift in both tasks using UDHM, the seeding method and a combination of the two. The baseline (dotted gray lines) uses no selection. The triangles show the 25%/75% percentiles. Significance results from a two-sample Kolmogorov-Smirnov test are shown with asterisks.

In the seeding approach, QD always starts exactly at the selection $\mathcal{S}$. The approach shows less drift for very small mutation distances because it simply does not get the

opportunity to mutate away far enough to generate solutions that use a different exit. It however does not find many new solutions, as on average, it ends up at about the same number of correct solutions compared to not using selection at all (33%), which is the expected value in a non-biased maze with three exits in the inner ring. With increasing mutation distance, the seeding approach drifts away from the user selection. The combination of the two approaches performs best, as the search starts at the selected locations and is suppressed from moving too far away.

	none	UDHM	seeding	combined
Closed form encoding				
correct %	33	58	65	**72**
drift $d_{\mathcal{M}}$	-	0.01	0.17	**0.00**
Neural encoding				
correct %	33	**60**	44	**60**
drift $d_{\mathcal{M}}$	-	0.03	0.48	**0.02**

Table 5.2: Median percentage of correct exits taken after running QD with selected exit. Baseline results without selection under "none". User selection drift is shown as well.

The median values are shown in Table 5.2. Of all solutions found, 72% take the correct exit. A two-sample Kolmogorov-Smirnov test was performed to show the mutation distances at which the combined approach produces significantly different results than the seeding approach.

Control Task with Neural Encoding The results for the non-linear and reactive control task show a qualitatively different behavior. In this case, small mutation distances are beneficial to all three user selection variants. It is not hard to see why this can happen. If the weights to the output neuron that is controlling rotational movement are a bit higher, the robot will rotate more in the beginning of trajectory and might select another exit. A small change in a neural controller might lead to the robot selecting a very different path. With increasing mutation distance, the controllers move further away from the initial archive. The UDHM is able to hold the selection much longer (up to a mutation distance of 0.1%), but at some point it gives way to the pressure exerted by mutation. The combined approach benefits only from the UDHM as it shows the same behavior.

The mean values are shown in Table 5.2. The low mutation distances in this case still allow jumping the gap between basins of attraction because of the non-linear mapping of genome to phenotype and behavior. This explains why all approaches perform worse than in the planning task. The combined approach performs best but very similar to the UDHM alone.

Discussion The objective in the experiments does not contain any information on what exit in the inner ring should be taken. Instead, the user can select the preferred solutions. A combination of seeding and the UDHM leads to a robust selection model within the adjusted PRODUQD algorithm. By capturing the user's selection in a model, as opposed to the seeding method, the UDHM adds continuous control over the QD search and less user selection drift takes place. The genome-to-phenotype mapping influences whether a small or large mutation distance should be used in QD. In general, the mutation distance for an indirect, neural genome-to-phenotype mapping has to be lower, as similar genomes are more likely to produce dissimilar phenotypes. Constraining QD by adjusting the objective function allows QD to find new solutions that still adhere to a user's selection.

5.2.6 Conclusions

This section gave evidence for an answer to research questions IX ("Can selected prototypical genomes be modeled?") and X ("Can we constrain parameters by penalizing QD's objective?"), fulfilling requirements A4 ("The process should capture design knowledge through designer interaction") and D1 ("Communication should be evocative through features such as visualization and abstraction"). The user's selection can be modeled in relation to the presented hypersurface of high-performing solutions and QD can be constrained through its objective function, using these models.

User selection drift was introduced, which compares the distance of a candidate solution to the sets of selected and deselected examples. The drift can be used to penalize solutions that are too close to those that were explicitly not selected. A user driven QD algorithm is formalized that adopts the penalty in its objective function. A model-based constraint of QD is compared against an approach that seeds the QD archive with the selected solutions.

Evaluation was performed in a new multimodal benchmark domain. The tasks defined in it allow the comparison of effects of using differently coupled representations that are hard to quantify in a pure design task, but results can be transferred back to a design case, as was shown by Hagg et al. (2018). Both tasks demonstrate that QD can be influenced towards a user's selection. The structure of the decision hyperspace created using t-SNE is permissive to compare solutions of various dimensionality. UDHM is most effective when combined with seeding. Depending on the representation and the genome to behavior/phenotype mapping, the mutation distances can be high or should be more conservative. UDHM, especially when combined with the seeding approach, is able to influence QD to discover new solutions that adhere to the user's decision. Of course the occasional misclassification by the user will cause unexpected behavior, and multiple selection rounds might have to be applied. The introduced models open up the possibility of using QD in an interactive optimization process; QD can be used within the design by shopping paradigm.

5.3 Phenotypic Drift

The limits of the approach for nonlinear, neutral, sensitive and ecologic representations introduces the necessity to measure similarity in a different way, not based on the genome but rather on the phenotype or behavior itself. Finally, after having introduced models and constraints to QD on a genetic level, here I answer the final research question XI ("Can we constrain phenotypes by penalizing QD's objective?"). In this section, neural representations are used to compare solutions and build a phenotypic similarity space. These neural representations enable using the same phenomenon of search cues described in stage theories of creativity (see Section 2.1.1).

5.3.1 Comparing Genetic and Phenotypic Models

In a two-dimensional shape domain, consisting of local interpolating splines as defined by Catmull and Rom (1974), genetic and phenotypic models are compared. The splines are encoded by a polar-coordinate-based genome (see Fig. 5.17). It (see Fig. 5.17) consists of 16 parameters controlling the polar coordinate deviation of the polygon control points. The first eight genes determine the deviation of

the radius of the polygon's control points, the second eight genes determine their angular deviation. The polygon shapes are then transformed into their final shape by using locally interpolating splines. By controlling the radius r and angle θ, a large variety of convex and concave shapes can be created.

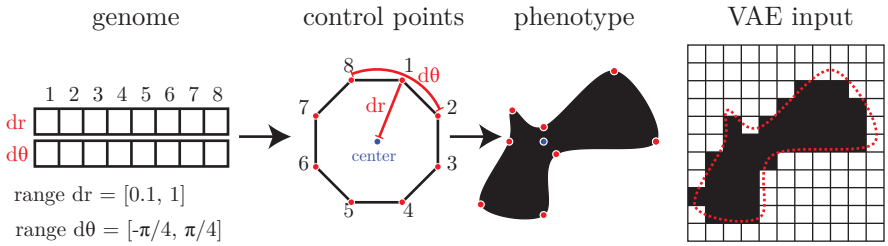

Figure 5.17: Shapes are encoded with 16 polar control points that get transformed into the final shape using locally interpolating splines. The shape is then discretized to serve as the input to the latent model.

The encoding (see Fig. 5.17) is quite sensitive and contains many neutral solutions. A triangle rotated by 120° is still the same shape.

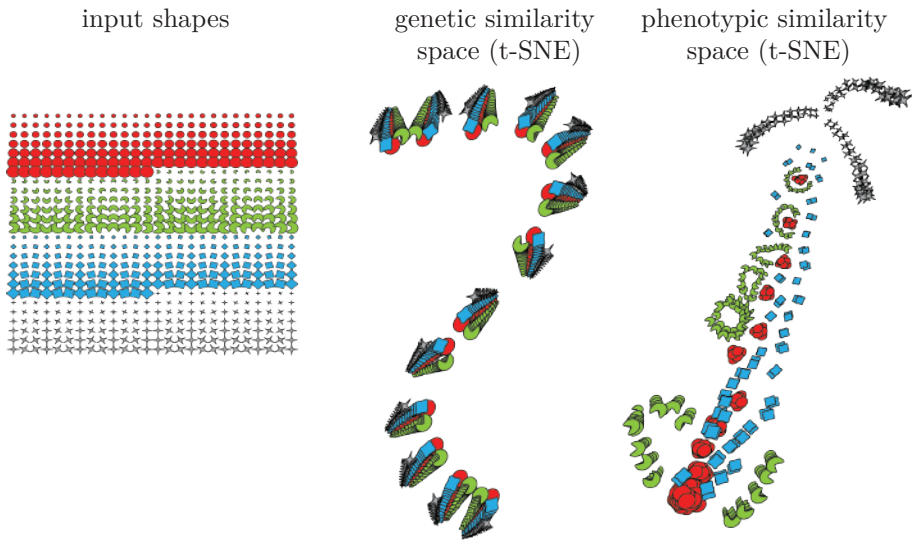

Figure 5.18: Genetic (16D) and phenotypic space (4096D) show the effect of neutral and sensitive, non-linear mappings on a simple polygon encoding domain.

A shape set consisting of 4096 shapes is created. The input shapes are rotated in 30° steps. The shape set therefore contains multiple neutral morphological

copies, except for the Pacman shapes. The shape set is projected into a latent two-dimensional space usign t-SNE (Maaten and Hinton (2008)), applied to the shape set's genetic space, which is 16-dimensional, and phenotypic space, which is 4096-dimensional (64x64 pixels), see Fig. 5.18.

Similar (neutral) polygons get different positions assigned in the resulting genetic similarity space, but are correctly positioned in phenotypic space. Neutrality is therefore not an issue when similarity is measured in phenotypic space. Sensitivity is noticeable as well. The shapes are oftentimes positioned close to other classes in genetic space, although they are qualitatively in phenotypic space, which is much better at separating the shape classes.

Phenotypic similarity is not dependent on the encoding. It creates a more natural similarity space in which solutions can be compared.

5.3.2 The Preference Hypervolume

Building on PRODUQD, an interactive, co-creative process, determining the preference hypervolume (HyperPref), is introduced (see Fig. 5.19 and work by Hagg et al. (2020)). The central process consists of two alternating steps: I) the computer initiates the process by producing a diverse set of high quality solutions and II) the users select the solutions they fancy, based on the phenotypic expression, after which the computer updates the set of solutions that are preferred as well as high-performing.

A closed form encoding needs to be created by the user first. Then, an initial pool of random solutions is generated and evaluated using a user-defined objective, which can be an optimality criterion or a more general factor about the appearance of solutions, throwing a wider net for more 'free' thinking. The genomes are then expressed into their phenotypes. A latent model is trained to compress the phenotypes into a low-dimensional description. This allows us to determine how similar solutions are and perform phenotypic niching despite of the high dimensionality of the phenotypes. QD creates such a latent niching archive, consisting of high-performing solutions (according to the user-defined objective function) and triggers a first intuition of what good solutions can look like.

Users select their preferred solutions from the archive, and a snapshot of the latent model and preferences is saved. The similarity metric is based on the latent

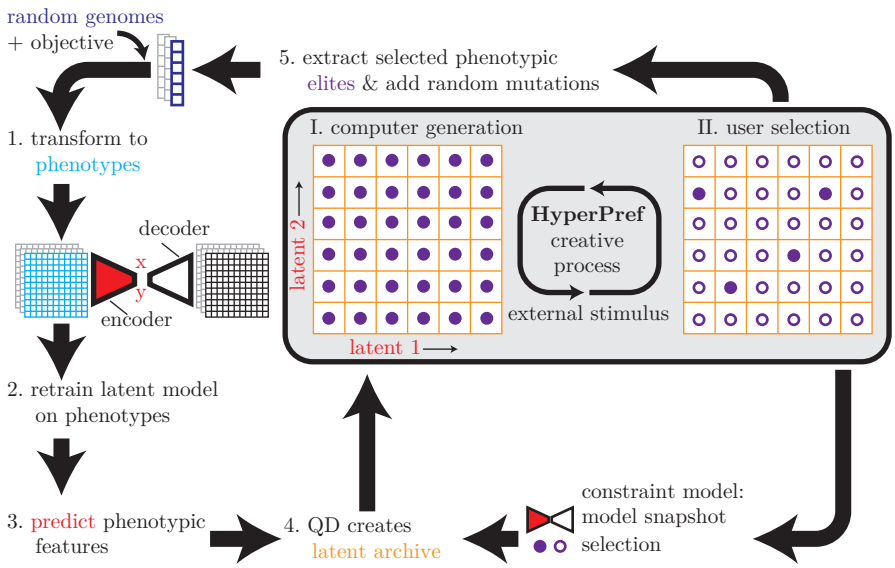

Figure 5.19: Discovering the preference hypervolume with HyperPref. The co-creative process (gray box) consists of computer-generated solution sets (step I) that are influenced by the users' selection (step II). A latent model is trained on the phenotypes (1, 2) of a random set of solutions. The model predicts phenotypic features (3) while creating diverse solutions using quality diversity (4), enhancing the intuition of the user. The user can now select/deselect solutions. The selection and a snapshot of the latent model form a constraint model. In the next iteration, the selected solutions are extracted and a new population is created by adding small mutations to the selection (5). The objective function is adjusted with a constraint penalty and the process resumes at (1). The phenotypic latent model (not the constraint model) is updated (2) and an intuition about what is high-performing and within the user's selection is expressed (4).

distance of new candidate solutions to the preferred and non-preferred solutions. The metric determines whether a new candidate solution is (likely to be) part of the preference hypervolume or not. The preferred solutions are used to create a new set of initial solutions by perturbing the original solutions, augmenting the data set with possibly new innovations. The preference hypervolume usually consists of disconnected regions in the search space. By increasing the mutation strength (the σ of the normal distribution from which the amount of perturbation is chosen), the search can be forced to be more explorative. With a low σ, the search is more exploitative.

Note that in contrast to other work, the latent space is not directly used for the search, but to compare phenotypic similarity of solutions. Although the latent model would also allow this, it would constrain the search to the interpolated space between selected solutions. This is only sensible when the first latent model from which the users select solutions is trained on a representative set of feasible and relatively high-performing solutions. Bootstrapping a model without a closed form encoding is not feasible due to the vast dimensionality of phenotypic spaces. Instead, in the proposed co-creative process, QD is the generator on which the model is trained, not vice versa. We can speculate that using a closed form encoding, not limiting the search to the latent space, will allow more innovative solutions to be found once users constrain the search to their preferences, considering solutions that would not be considered by the first model.

The computer updates the latent model, which now describes the similarities within the preference hypervolume. By adjusting the objective function, adding the constraint model, QD's results are updated. This process can continue until the users are satisfied. In contrast to previous work that used a combination of NS (as opposed to QD) and AE, by Liapis et al. (2013a), QD uses an explicit external objective. The proposed process searches for quality as well, and the latent model is used not only as a way to enhance novelty but also to capture the user's choice.

Due to the evidence against using GM as a search space, which was shown in Section 3.5, HyperPref only uses GM as a niching method and its latent space to model the user's preferences. The representations, QD's search space, are kept in their closed form.

5.3.3 Demonstration

Two use cases (see Fig. 5.17 and Fig. 5.20) demonstrate the capabilities of HyperPref. In an artistic case, the users are looking to design a ninja star, starting from *centrally symmetric* shapes. Another situation, which is closer to creative engineering, starts out with *unbalanced* shapes with the aim to find wing profiles. The first objective prefers solutions that are point symmetric through the center point of the shape. The shape is sampled at $n = 100$ equidistant locations on its circumference, after which the symmetry metric is calculated. It is based on the symmetry error E_s,

the sum of Euclidean distances of all $n/2$ opposing sampling locations to the center (see Section 5.3.1):

$$f_P(\boldsymbol{x}) = \frac{1}{1 + E_s(\boldsymbol{x})}, E_s(\boldsymbol{x}) = \sum_{j=1}^{n/2} \left\| \boldsymbol{x}_j - \boldsymbol{x}_{j+n/2} \right\|$$

The second objective maximizes the distance between the center of mass and the center of the bounding box around the shape.

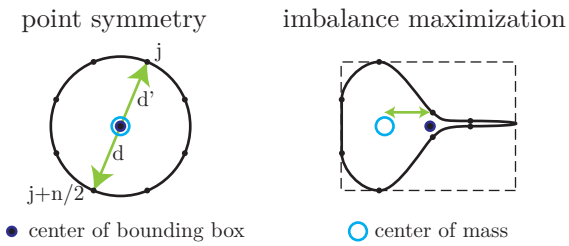

Figure 5.20

The shapes are converted into 128x128 pixel images to serve as training data to a cVAE (see Fig. 3.20). The use of a cVAE creates a more evenly distributed projection of the shapes onto the latent space. The second reason to use a variational variant of an AE is that it allows sampling from the latent space, which can be of use when interpolating between known shapes. However, this feature will not be used within the scope of this work.

For simplicity, a two-dimensional latent space is used which only captures the similarity of solutions based on the largest phenotypic variance. The shape genome consists of 16 genes. QD produces 32 new child solutions for 1024 generations, by using a normally distributed mutation operator with $\sigma = 10\%$ of the parameters' range as a generator of diversity. The archive holds 20×20 solutions.

The cVAE is trained on a GPU with 128×128 pixel representations of the shapes and the ADAM training method (see Kingma et al. (2015)). The encoder consists of two convolutional and non-linear (reLu) layers, eight filters of size three with a stride of one. Training is performed with a learning rate of 0.001 and maximizes the evidence lower bound, thereby minimizing the Kullback-Leibler divergence between the original and the latent distribution.

The solution set is updated using 64 perturbed ($\sigma = 10\%$) versions of the selected shapes. The constraint penalty, which is multiplied with the original fitness function, is based on the user selection drift that was introduced in Section 5.2.1, with the minimal distance s of a candidate solution x to a selected solution and $\bar{s}$ to a deselected solution:

$$p(x) = \begin{cases} 1, & \text{if } \frac{s}{(s+\bar{s})} < 0.5 \\ 1 - 2 \cdot (\frac{s}{(s+\bar{s})} - 0.5), & \text{otherwise} \end{cases}$$

Results Fig. 5.21 shows the initial computer-generated solution set on the left.

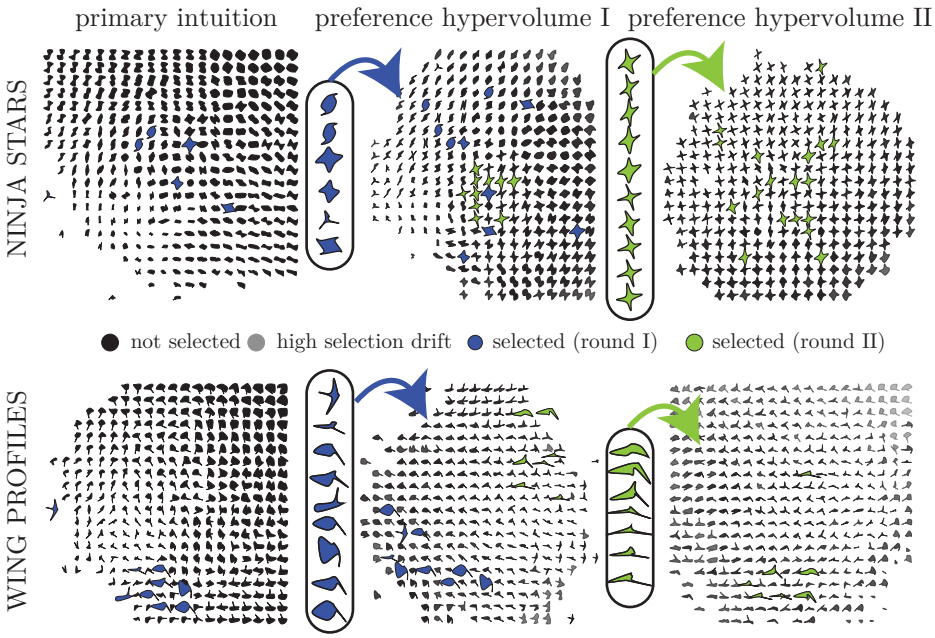

Figure 5.21: Top: starting with centrally symmetric shapes to design ninja stars. Bottom: starting with unbalanced shapes to design wing profiles.

The diversity of the sets is clearly visible. A group of creators is simulated that all have different preferred shapes (shown in the center). The selected shapes are shown in blue. After selection, the computer updates the set, reflecting the combination of the creators' choices and the general objective (center of the figure). This process of user selection is repeated once more. The resulting sets of ninja

stars and wing profiles are shown in the second preference hypervolume on the right.

Discussion The primary solution set contains a large and diverse number of shapes, offering users inspiration and feeding their intuition about how shapes could look like. With a specific goal in mind, namely designing ninja stars or wing profiles, users and computers can co-create in an intuitive creative process. The process offers reflection, by combining preferred shapes and zooming in on the preference hypervolume. Only two steps are necessary to create shapes that are close to what one could and would expect from such a creative process.

5.3.4 Conclusions

This section gave evidence for an answer to research question XI ("Can we constrain phenotypes by penalizing QD's objective?"), fulfilling requirement A4 ("The process should capture design knowledge through designer interaction"). It was shown how to combine the divergent search of QD to trigger the user intuition about what solutions are possible and high-performing. This allows creators to select shapes they prefer by shopping for designs. The machine reflects upon that selection, incorporating the preferences through a constraint model and discovering the preference hypervolume in an intuitive, co-creative manner. Rather than hand-crafting the features on which the initial set is based, or finding them via genetic similarity, they are generated by a variational autoencoder, trained on the phenotypical expression of the solutions, rather than hand-crafted features or genetic similarity. The constraint model is based upon a snapshot of that model in combination with the set of selected and non-selected solutions.

The resulting creative process, which continuously visualizes and updates the creators' intuition, was demonstrated in a simple two-dimensional shape domain. The updates can be fast, depending on the GPU used for training the cVAE and the number of QD updates and cVAE prediction speed. The current bandwidth of GPUs is such that the method is close to being on-line.

5.4 Chapter Summary

This chapter discussed methods to capture a user's preferred shapes in the context of a QD solution set. QD results were first summarized into representative prototypes inside a genetic similarity space. This answered research question VII ("Can QD results be summarized using representatives?"). The resulting prototypes were shown to be usable in a co-creative process (PRODUQD) to influence a QD algorithm to produce solutions that were similar to the preferred shapes. Evidence was given for a positive answer to research question VIII ("Can QD be influenced by the user by their selected representatives across domains?").

The user selection was then modeled by using a Gaussian process model that correctly predicts the similarity space coordinates of new candidate solutions. This answered research question IX, "Can selected prototypical genomes be modeled?".

Explicit constraints could then be defined by the user implicitly by selecting prototypes they prefer. The user's influence was then shown on multiple domains. The model of the user's preferences was used to penalize new solutions accordingly, by a simple adjustment to QD's objective function. This answered research question X, "Can we constrain parameters by penalizing QD's objective?".

The weakness of the approach is the dependency on the encoding and the inability to take into account neutrality and sensitivity effects. Phenotypic space was therefore modeled using a GM, which corrected the weakness. The model's similarity space was again used to influence QD results after the user's selection. Finally, the answer to research question XI, "Can we constrain phenotypes by penalizing QD's objective?", is positive.