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).

Discovering the Preference Hypervolume

an interactive model for real world computational co-creativity

Alexander Hagg

Discovering the Preference Hypervolume

an Interactive Model for Real World

Computational Co-creativity

Proefschrift

ter verkrijging van
de graad van doctor aan de Universiteit Leiden,
op gezag van rector magnificus prof.dr.ir. H. Bijl,
volgens besluit van het college voor promoties
te verdedigen op dinsdag 7 december 2021
klokke 11.15 uur

door

Alexander Hagg

geboren te Delft, Nederland
in 1979

Promotiecommissie

Promotor:	Prof. Dr. T.H.W. Bäck	
Promotor:	Prof. Dr. A. Asteroth	
Co-promotor:	Dr. M. Preuss	
Overige leden:	Prof. Dr. A. Plaat	(voorzitter, LIACS, NL)
	Prof. Dr. J. Batenburg	(secretaris, LIACS, NL)
	Prof. Dr. E. Hart	(Edinburgh Napier University, UK)
	Prof. Dr. S. Colton	(University of London, UK)
	Dr. R. Saunders	(LIACS, NL)
	Dr. A. Kononova	(LIACS, NL)

Copyright © 2021 Alexander Hagg.

This work received funding from the German Federal Ministry of Education and Research, and the Ministry for Culture and Science of the state of North Rhine-Westphalia (research grants 03FH012PX5 and 13FH156IN6).

Acknowledgements

I would like to dedicate this work to my second supervisor, Alexander Asteroth, whom I met in 2006 as my trade school teacher. During an evaluation meeting, after having read a report I had submitted, he put it aside without really discussing it and asked me why I did not go and study. This moment changed my life as I followed him to the university of applied sciences where he became professor. During my entire academic career he has been an inspiration, enabler, critic, and friend. I will be forever in your debt.

I would like to thank my promotor Thomas Bäck and my coauthors Adam Gaier, Dominik Wilde, Jörg Stork, Martin Zaefferer, Maximilian Mensing, Sebastian Berns, Mike Preuss, and Simon Colton for the inspiring discussions and our work together. Furthermore, I would like to thank my proofreaders, specifically Carolin Hoffmann, Evgeniya Ovchinnikova, Hanna Heinrich, and Bärbel Schomers. Carolin, you have become my sister and without you, my life would have been wildly different and certainly not as happy.

Finally, I would like to thank all the people in my life for being there for me. Writing this thesis is a milestone in my life and without your inspiration I would have not been able to keep going. I love you all.

Abstract

In this thesis, the ideas of Guilford about divergent thinking, Jung on intuition and Sartre on reflection by others are combined to create a Hegelian creative process. It is posed that the central object of preference discovery is a co-creative process in which the Other can be represented by a machine, as is often done in the computational creativity community. This thesis explores efficient methods to enhance introverted intuition using extraverted intuition's communication lines.

Possible implementations of such processes are presented using novel algorithms that perform divergent search to feed the users' intuition with many examples of high quality solutions, allowing them to take influence interactively. In this process, the machine feeds and reflects upon human intuition, combining both what is possible and preferred. The machine model and the divergent optimization algorithms are the motor behind this co-creative process, in which machine and users co-create and interactively choose branches of an ad hoc hierarchical decomposition of the solution space.

The proposed co-creative process consists of several elements, which are treated in this thesis in the following order: a formal model for interactive co-creative processes, evolutionary divergent search, diversity and similarity, data-driven methods to discover diversity, limitations of artificial creative agents, matters of efficiency in behavioral and morphological modeling, visualization, a connection to prototype theory, and methods to allow users to influence artificial creative agents.

We tend to portray human creativity as having few boundaries. It might seem folly for a computer scientist to venture into such a domain where we have biases about human capabilities and are not capable of quantifying results in an objective manner without involving the human we aim to partially replace. This dissertation

is an effort to connect the field of computer science with the fields of philosophy, psychology, and biology, to enhance, not replace human creativity.

The ethical, social and political consequences of replacing humans by algorithms are part of a wider debate about how we organize society in the light of artificial intelligence's deep impact. But instead of replacing the human, this thesis explicitly chooses to embrace human-computer interaction in creative design, putting the human back into the loop of algorithmic design in generative AI and optimization.

Replicators [...] should be thought of as having extended phenotypic effects, consisting of all its effects on the world at large, not just its effects on the individual body in which it happens to be sitting.

(Dawkins 1982)

There is a power and utility to regarding the gene as the unit of selection, but equally there is value to seeing the organism as the unit of niche construction.

(Laland 2004)

Can There Ever Be Too Many Options?

(Scheibehenne 2010)

No one can tell what the painting of tomorrow will be like; one cannot judge a painting until it is done.

(Sartre and Elkaïm-Sartre 1946)

Contents

Abstract **i**

List of Symbols

1 Introduction **1**
1.1 Creativity and Optimization 1
1.2 Examples Drive Creative Processes 2
1.3 Outline of the Dissertation 3

2 The Creative Process **7**
2.1 Creativity 7
2.1.1 Formalisms 9
2.1.2 Iterative Processes 11
2.2 Creative Cognition 12
2.2.1 Divergent Thinking 12
2.2.2 Priming 14
2.2.3 Convergent Thinking 14
2.2.4 Reorganization of Information 15
2.2.5 Dissonance 16
2.2.6 Resonance: Experience, Intuition and Bias 16
2.3 Computational Creativity 17
2.3.1 Requirements of Co-creative Systems 19
2.3.2 Related Co-creative Systems 21
2.4 Process Model 23
2.5 Chapter Summary 26

3 Divergent Search **29**
3.1 Extending the Phenotype 31

3.1.1 Genome, Phenotype and Behavior in Nature 32
3.1.2 Encoding in Evolutionary Computation 34
3.1.3 Conclusions 38
3.2 Multi-Solution Optimization Methods 38
3.2.1 Diversity in Objective Space 39
3.2.2 Diversity in Genetic Space 40
3.2.3 Diversity in Phenotypic Space 41
3.2.4 Conclusions 49
3.3 Comparing Divergent Search Methods 50
3.3.1 Niching with Voronoi Tessellation 51
3.3.2 Diversity Metrics 53
3.3.3 Evaluation 55
3.3.4 Conclusions 60
3.4 Phenotypic Features 60
3.4.1 Predefined Features 61
3.4.2 Learning Features 62
3.4.3 Evaluation 65
3.4.4 Conclusions 65
3.5 Limitations of Generative Models 66
3.5.1 Variational Autoencoders 67
3.5.2 Study Setup 69
3.5.3 Experiments 72
3.5.4 Discussion 78
3.5.5 Conclusions 80
3.6 Chapter Summary 81

4 Efficiency **83**
4.1 Surrogate Modeling of Phenotypic Features 84
4.1.1 Bayesian Optimization 84
4.1.2 Surrogate-Assisted Quality Diversity 85
4.1.3 Surrogate-Assisted Phenotypic Niching 87
4.1.4 Quantitative Comparison 90
4.1.5 Use Case: Wind Nuisance in Architecture 93
4.1.6 Conclusions 99
4.2 Surrogate Modeling of Neural Behaviors 100
4.2.1 Related Distance Kernels 102

4.2.2 Sampled Phenotypic Distance . . . 103
4.2.3 Evaluation . . . 106
4.2.4 Conclusions . . . 112
4.3 Chapter Summary . . . 113

5 The Preference Hypervolume 115
5.1 Genetic Prototypes . . . 116
5.1.1 Unsupervised Clustering . . . 117
5.1.2 Evaluation of Dimensionality Reduction and Clustering . . 119
5.1.3 Genetic Seeding . . . 122
5.1.4 Quantitative Analysis . . . 124
5.1.5 Qualitative Analysis . . . 126
5.1.6 Conclusions . . . 128
5.2 Modeling User Selection Drift . . . 129
5.2.1 User Decision Hypersurface Model . . . 130
5.2.2 Experimental Setup . . . 134
5.2.3 Selection on Hypersurface . . . 137
5.2.4 Influence of Penalty Weight on Drift . . . 138
5.2.5 Comparing Seeding and Modeling . . . 139
5.2.6 Conclusions . . . 142
5.3 Phenotypic Drift . . . 143
5.3.1 Comparing Genetic and Phenotypic Models . . . 143
5.3.2 The Preference Hypervolume . . . 145
5.3.3 Demonstration . . . 147
5.3.4 Conclusions . . . 150
5.4 Chapter Summary . . . 151

6 In Conclusion 153

Appendix A Evolutionary Algorithms 161
A.1 Selection . . . 163
A.2 Crossover . . . 163
A.3 Mutation . . . 163
A.4 Diversity Management . . . 164
A.5 Representation . . . 165

Appendix B T-Distributed Stochastic Neighborhood Embedding 166

Appendix C Gaussian Process Regression **168**

Bibliography 171

Acronyms 191

Summary 195

Samenvatting 197

About the Author 201

List of Symbols

k	number of nearest neighbors
$\mathcal{X}$	population matrix with one member per row
$\mathcal{A}$	phenotypic archive used as a niching method by quality diversity algorithm
$\mathcal{D}$	phenotypic feature coordinate matrix with one member per row
$\mathcal{N}(\mathbf{m}, \sigma)$	Gaussian random variable with mean m and variance σ^2
A	area
l	circumference length
P	point symmetry
$\mathbf{dR}$	radial deviation of polygon key point
$\mathcal{D}\theta$	angular deviation of polygon key point
$L(x, \hat{x})$	evidence lower bound between predicted output x and ground truth $\hat{x}$
$C(\cdot, \cdot)$	binary cross-entropy between two entrees
$K(\cdot, \cdot)$	Kullback-Leibler divergence between two distributions
β	regularization factor scalar in Kullback-Leibler divergence
γ	annealing factor in Kullback-Leibler divergence
μ	learning rate for optimization algorithms
p	significance value
κ	parameter that controls exploration and exploitation in upper confidence bound acquisition function within Bayesian optimization context

$d()$	pointer to function that returns phenotypic descriptors (features) of a solution in a divergent optimization context
$f()$	pointer to function that evaluates a solution in optimization
$\mathbf{M}_f$	surrogate model that predicts fitness
$\mathbf{M}_d$	surrogate model that predicts phenotypic descriptors
u_{max}	maximum air velocity in a flow field
E	enstrophy: a turbulence metric of a flow field
t	point in time
Re	Reynolds number of a flow field
$\mathbf{s}$	input vector of a neural network
$\mathbf{o}$	output vector of a neural network
p_{c_L}	lift penalty used in objective function
p_A	area penalty used in objective function
$\mathcal{H}$	decision hypersurface based on members of QD archive
$\mathcal{T}$	projection of archive members into similarity space
$\hat{\mathcal{T}}$	projection model
δ	distance measure
$\mathcal{S}$	selected solutions
$\overline{\mathcal{S}}$	deselected solutions
$\mathcal{P}$	binary selection partition
$\mathcal{M}$	user decision hypersurface model
E_s	symmetry error
$p(x)$	constraint penalty of a candidate solution x, selected by a user

CHAPTER 1

Introduction

There is a tension between artificial intelligence (AI) systems and responsibility. An increasing number of decisions are left to AI systems that use non-transparent models. The consequences of these decisions potentially have a large impact on society. Although explainability of AI as a subfield is growing, it has a hard time to keep up with the rate of adoption, due to the myriad of economically sound possibilities offered to us by AI's speed, generalizing and generating power. At the same time, decision and model credibility issues in deep learning (DL) and optimization techniques are hard to uncover by individuals. Humans need to keep control of decisions made by AI systems. The creative process is an example where generative AI systems are creeping into a domain that is classically seen as purely human. It is in this domain where I will answer questions about how to design fruitful interactions between generative AI and human decision makers.

1.1 Creativity and Optimization

Fundamentally, every creator strives to discover their true self, while acknowledging that they will never be able to fully and concisely grasp it. The function of creativity and art is to allow reflection of both the spectator and the creator. They are necessarily interactive concepts, according to Hegel (1842), and thus inherently iterative. Creativity is a domain in which decision making inevitably takes place on human time scales, updating intuitions through reflection, which go hand in hand.

Contrarily, in real world optimization, algorithms tend to be created that solve engineering problems under the assumption that quality, constraints and preferences

can be fully and concisely formalized a priori. Much of the work in optimization research tries to find global optima to such 'perfectly' formalizable problems. But the processes of problem solving and creativity are much more complex and lead to unexpected usage of such algorithms. In a 2014 interview study by Bradner et al. (2014) on the real world usage of automation in design optimization, "*participants reported consulting Pareto plots iteratively in the conceptual design phase to rapidly identify and select interesting solutions*". Yet, the employed algorithms are written with only a single design iteration in mind.

The tension between creativity (as Hegel understood it) and engineering design in practice seems to be obvious. Optimization research tends to take objectives and preferences as a given and as complete, whereas creativity is naturally open-ended. I pose that developing algorithms that embrace this discourse should be a central objective of optimization research. We should accept uncertainties as a given and build algorithms that support creative processes which help to discover preferences. The field of optimization has come a long way, tackled many issues and the advances in machine learning are opening up novel ways of dealing with the intangible: what does the engineer really want and how can we efficiently develop their understanding and support innovative thought?

1.2 Examples Drive Creative Processes

In everyday life, humans can feel insecure or even anxious about questioning their preferences and intuition. We either choose to ignore novel ideas, taking a more conservative stance towards problem solving and building upon solutions we know and understand, or rely on our intuition and trial-and-error-based techniques to develop new ideas. Those intuitions are based upon direct experience, on cross-connections we make based upon unrelated experiences, on familiarity but also on our problem solving skills, as was described in an empirical study by Raidl and Lubart (2001). We even experience a physical sensation as a reaction to these intuitions, or more precisely, introverted intuition, as defined by Jung (1923), which is paramount to discovering one's own preferences.

The artist's or creative engineer's search is not performed in a vacuum. We might have initial ideas. The ability to diverge from this starting point enables us to leave the realm of known solutions to find innovations. Yet, we need others to reflect upon

ourselves and gain true insight into what our preferences are and how they relate to those of others, creating 'anchors' that allow us to more confidently explore the space of (unknown) solutions. Sartre and Elkaïm-Sartre (1946) emphasized that the Other is needed for reflection, but every artist and creator, indeed everyone, has the responsibility to choose. In a creative process, especially when *reflecting* with others, we use the Jungian ability of extraverted intuition (brainstorming or group divergent thinking, as coined by Runco (2010)) to come up with and evaluate novel solutions that might not coincide with our own intuition, or trigger our intuition in unexpected ways.

In order to discover and communicate our preferences with ourselves as well as with others we create examples that are supposed to capture abstract ideas. Without examples, communicating novel ideas can be difficult. As Sartre and Elkaïm-Sartre (1946) stated: "*No one can tell what the painting of tomorrow will be like; one cannot judge a painting until it is done*". So creative thought, which Guilford (1967) defined as the ability to perform *divergent* thinking, is about generating many examples and iteratively discovering and converging towards our preferences. It is through these examples that we can both explore and communicate our preferences.

1.3 Outline of the Dissertation

In this thesis, the ideas of Guilford about divergent thinking, Jung on intuition and Sartre on reflection by others are combined to create a Hegelian creative process. It is posed that the central object of preference discovery is a co-creative process in which the Other can be represented by a machine, as is often done in the computational creativity community. This thesis is about exploring efficient methods to enhance introverted intuition using extraverted intuition's communication lines.

Possible implementations of such processes are presented, akin to the generative-explorative creative model introduced by Ward et al. (1999), using novel algorithms that perform divergent search to feed the users' intuition with many examples of high quality solutions, allowing them to take influence interactively. In this process, the machine feeds and reflects upon human intuition, combining both what is possible and preferred. The machine model and the divergent optimization

algorithms are the motor behind this co-creative process, in which machine and users co-create and interactively choose branches of an ad hoc hierarchical decomposition of the solution space.

The proposed co-creative process consists of several elements, which are treated in this thesis in the following order.

In Chapter 2, the theoretical frameworks of cognitive psychology and co-creativity are explored. A formal model for an interactive co-creative process is introduced.

To automatically generate solutions, evolutionary techniques are used, which are widely applied in divergent optimization settings. Based on natural phenomena such as behavior encoding, mutation, and heredity, they provide a natural means to search through large and high-dimensional search spaces. The biological and ecologic concepts of the extended phenotype and diversity are applied to evolutionary encodings in Chapter 3 to answer research question I: "Are solutions best compared using their genomes or their expressed phenotype or behavior?". A case is made to use a phenotypic search method called quality diversity (QD), a divergent search method which will be explained in Chapter 3, which is compared to state of the art multi-solution optimization algorithms. These algorithms can fulfill the divergent and convergent component from the AI's side. The comparison answers research question II: "What multi-solution optimization method produces the highest phenotypic diversity?".

The chapter also discusses what types of diversifying features can be used in phenotypic search to compare and produce a diversity of solutions. Research question III: "Can we produce more diverse solution sets when learning phenotypic niching from data instead of using predefined features?" is used to determine, how we can use generative model (GM) to increase solution diversity. GM can be used both to increase diversity of solution sets generated with predefined encodings as well as to provide the encodings altogether. Research question IV: "What are the limitations of generative models in terms of the possible diversity of the solutions they create?" is answered to decide how to use GM in a co-creative setting. See Table 1.1 for a list of publications Chapter 3 consists of.

Due to the fact that the QD method is generally computationally intense when applied to real world engineering processes, efficiency aspects that arise when applying the co-creative model to real world problem domains are discussed in

Table 1.1: Publications for Chapter 3

Hagg, A., M. Preuss, A. Asteroth, and T. Bäck (2020). An Analysis of Phenotypic Diversity in Multi-Solution Optimization. In Proceedings of the 9th International Conference on Bioinspired Optimisation Methods and Their Applications - BIOMA 2020.
Hagg, A., S. Berns, A. Asteroth, S. Colton, and T. Bäck (2021). Expressivity of Parameterized and Data-driven Representations in Quality Diversity Search. In Proceedings of the Genetic and Evolutionary Computation Conference - GECCO 2021.
Hagg, A. (2021). Phenotypic Niching using Quality Diversity Algorithms (accepted). In M. Epitropakis, X. Li, M. Preuss, and J. Fieldsend (Eds.), Metaheuristics for Finding Multiple Solutions. Springer Press.

Chapter 4. For a fluid dynamics domain, behavioral features of fluids are expensive to calculate. In order to increase efficiency, research question V: "Can we model behavioral features in a surrogate-assisted way by sampling based on optimality alone?" is answered. Neural encodings are used in a number of generative approaches, which is why in a robotics domain research question VI: "Can we model neural encodings' behavior ex situ by sampling their outputs and using a behavior-kernel?", is answered. See Table 1.2 for a list of publications Chapter 4 consists of.

Table 1.2: Publications for Chapter 4

Hagg, A. (2017). Hierarchical surrogate modeling for illumination algorithms. In Proceedings of the Genetic and Evolutionary Computation Conference - GECCO 2017.
Hagg, A., M. Zaefferer, J. Stork, and A. Gaier (2019). Prediction of neural network performance by phenotypic modeling. In Proceedings of the Genetic and Evolutionary Computation Conference - GECCO 2019.
Hagg, A., D. Wilde, A. Asteroth, and T. Bäck (2020). Designing air flow with surrogate-assisted phenotypic niching. In Proceedings of the 16th International Conference on Parallel Problem Solving from Nature - PPSN 2020.

In Chapter 5 the user selection is modeled and incorporated into the AI agent's learning and adaptation phase. In order to achieve this, the following research

questions are answered: “Can QD results be summarized using representatives?”, to communicate results to the human, “Can QD be influenced by the user by their selected representatives across domains?” and “Can selected prototypical genomes be modeled?”,to allow humans to take influence, and finally: “Can we constrain parameters by penalizing QD’s objective?” and “Can we constrain phenotypes by penalizing QD’s objective?”, to finally produce the full co-creative process. See Table 1.3 for a list of publications Chapter 5 consists of.

Table 1.3: Publications for Chapter 5

Hagg, A., A. Asteroth, and T. Bäck (2018). Prototype discovery using quality-diversity. In Proceedings of the 16th International Conference on Parallel Problem Solving from Nature - PPSN 2018, pp. 500-511. Springer.
Hagg, A., A. Asteroth, and T. Bäck (2019). Modeling User Selection in Quality Diversity. In Proceedings of the Genetic and Evolutionary Computation Conference - GECCO 2019.
Hagg, A., A. Asteroth, and T. Bäck (2020). A Deep Dive Into Exploring the Preference Hypervolume. In Proceedings of the International Conference on Computational Creativity - ICCC 2020.

In Chapter 6 I summarize the findings in this thesis. The appendices contain background information on basic methods that are used in this work. They discuss evolutionary algorithms, a dimensionality reduction method, and a widely used statistical modeling technique.

We tend to portray human creativity as having few boundaries. It might seem folly for a computer scientist to venture into such a domain where we have biases about human capabilities and are not capable of quantifying results in an objective manner without involving the human we aim to partially replace. This dissertation is an effort to connect the field of computer science with the fields of philosophy, psychology, and biology.

The ethical, social and political consequences of replacing humans by algorithms are part of a wider debate about how we organize society in the light of artificial intelligence’s deep impact. But instead of replacing the human, this thesis explicitly chooses to embrace human-computer interaction in creative design, putting the human back into the loop of algorithmic design in generative AI and optimization.

CHAPTER 2

The Creative Process

A first important step is to analyze formalisms of human creativity and the components of such processes. The aim of this chapter is to find out which components can be replaced or extended by computer algorithms. It serves as a connection between insights from psychology on the components and traits of creativity. The chapter describes requirements for systems in the field of computational co-creativity. Two main questions about the application of computational techniques on creative processes are answered: during which phases can and should we apply computational techniques and what psychological traits should be taken into account. The chapter discusses the various formalisms of creativity and co-creative processes that were described by theorists, what psychological effects can encourage and discourage creativity, what requirements computationally creative design processes have, which computer aided design processes already exist, and where they fail to meet those requirements. Finally, a formalized co-creative process is introduced, designed to extend human creativity in engineering using an AI agent.

2.1 Creativity

The 'standard definition' of creativity by Runco and Jaeger (2012) requires a combination of originality and effectiveness, on equal footing. Without originality or novelty, something is not considered to be creative. If we reuse or replicate old solutions to the same problems, we might rather call this replication. The advantage of replicating a solution is that it has proven to be effective. If a solution or product is ineffective, we would not see it being used, or probably at least see less use of it. This Darwinian view is certainly not a general rule, but is an accepted heuristic in engineering design.

Then again, if we come up with novel solutions, be it by transforming known solutions, transferring them from other domains or by coming up with entirely novel ideas, they certainly have to be functional and effective. This was already established by Stein (1953): "*The creative work is a novel work that is accepted as tenable or useful or satisfying by a group in some point in time*". However, Stein stated on novelty: "*The extent to which a work is novel depends on the extent to which it deviates from the traditional or the status quo*". Novel solutions are found at a distance from known solutions, yet only the latter are identified to be useful and represent engineers' expert knowledge.

Novelty and (known) usefulness are in direct conflict (see Gardner (1993)). Within this conflict emerges functional creativity, combining new elements with existing knowledge. When we want to find novel solutions, we have to venture into unknown territory and deal with uncertainty. Old solutions are well-understood and there is evidence of their usefulness. A random solution we have not seen before is more novel than those that are known but there is no documented evidence for its usefulness. However, randomization certainly acts as a pressure to create novel solutions. So how do we go about in a functionally creative process? How do we deal with uncertainty, knowledge and novelty?

Runco and Jaeger (2012) acknowledged that there is an ongoing debate about how creative processes work. Simonton (2007) presented two competing theories concerning the creative process using Picasso's production of the painting *Guernica* as an example. The 'systematic view' describes an expert-driven (monotonic) creative process whereas the opposing theory is a non-monotonic Darwinian blind-variation-and-selective-retention theory, which is much more unpredictable. In the latter, free association of creators is a combination of blind variation, but steered by unconscious biases called selective retention. Simonton showed that an expert panel, consisting of neutral, pro-monotonic and pro-non-monotonic experts, agreed that the process of creating *Guernica* was non-monotonic. Weisberg and Hass (2007) found that blindness, the inability to predict the outcome of a change, is a common theoretical element in both monotonic and non-monotonic processes. Again, two competing drivers of creativity can be distinguished: prediction of outcome and variation towards new solutions.

Creativity is a balancing act between blind variation to explore novel solutions and being able to predict the outcome of this variation. I pose that this balancing act can be 'resolved' by computational systems. If a machine could be programmed to

combine these abilities inherently difficult for humans, we could create a system that allows humans to concentrate on the task of judging the outcome with less bias than in purely human creative processes. What follows now is an overview on formalisms of functional creativity.

2.1.1 Formalisms

Formalisms of functional creativity and creative processes were subdivided into three main classes by Lubart (2001): factorial, associative and stage theories. *Factorial theories* describe creativity as an ability that consists of a set of skills and factors, or traits (Wang and Nickerson (2017)). Amabile (1988, 1996) developed a factorial theory which describes creativity as having three components: domain-relevant skills (knowledge, technical skills, special talents), creativity-relevant skills (coping with complexity, heuristics to generate novel ideas, e.g. trying counterintuitive ideas when stuck, persistence and sustained attention to a task) and task motivation. *Associative theories* define creativity as a process of combination of mental elements and associative thinking to connect those elements (Mednick (1962)). Finally, *stage theories* define creativity as a process consisting of fixed steps. This is certainly the most interesting set of theories when designing co-creative algorithms, as they describe processes that can be formalized into algorithms. For this reason, stage theories are described here in more detail.

The classical four-stage model of the creative process by Wallas (1926) consists of the phases of preparation, where information is gathered, incubation, where ideas churn in the person's head, illumination, where what seems to be a solution becomes apparent, and verification, where the individual checks out the apparent solution. Some stage theories show a similarity between creativity and biological evolution, for example Campbell (1960) discussed two stages of creative thought. In the first, wide variations of ideas are produced, and in the second a selection from those variations takes place. Production and selection are both relevant for creative processes. Cropley and Cropley (2010) extended the classical model of the creative process to the following phases: preparation, activation, generation, illumination, verification, communication and validation. But they stressed that this phase model "*is not an exact and concrete description of the process of the emergence of functionally creative products*". Cropley's model can be linked to the classical four-stage model by having the activation and generation phase subsumed

into the incubation phase. The other phases either exist in the classical model as well, or they describe parts of the transitions between phases.

Limitations Guilford (1967) criticized the four-stage model because "*it tells us almost nothing about the mental operations that actually occur*". It was criticized in other work (see Lubart (2001)), for example in work by Goldschmidt (1991), who analyzed the sketching protocol of architectural designers. She found that the design process is much more complex than what the four-stage process allows for: "*The dialectics of sketching is the oscillation of arguments which brings about gradual transformation of images, ending when the designer judges that sufficient coherence has been achieved*". This and other evidence from creative engineering processes leads to other, more 'advanced' models and more research on what actually happens during the phases.

Connection to Associative Theories Various authors described stage theories that are connected to associative theories' mental elements. In the first stage, an external stimulus results in a search cue in the mind that leads to the activation of knowledge, both directly and through association. This search cue acts as a pulse that activates knowledge through the (connectivist model of the) brain. This effect can be translated into the context of machine creativity by the neighbourhood relations used in neural representations. In a second step, the combination of knowledge from different areas produce creative ideas. The more distant the connections that those associations activate are, the higher the originality of ideas, but at the cost of higher cognitive load (Santanen et al. (2004); Nijstad and Stroebe (2006)). Machines do not suffer from cognitive load and can be used to help humans find more distant solutions. Wang and Nickerson (2017) suggested that providing diverse and unrelated stimuli may improve idea originality.

This thesis does not limit itself to a particular stage theory but uses the model as a structural basis to develop co-creative processes. I explore the possibilities that new developments in the fields of optimization and machine learning allow for. It is important to find a 'natural' way to integrate machines into the creative process, using strengths of machines where humans are ill-equipped to perform certain tasks. In the following sections and subsections, requirements for co-creative processes are summarized (in boxes).

2.1.2 Iterative Processes

Cropley and Cropley (2010) acknowledged that the phases of a human creative process are not written in stone. Rather, they follow an idealized pattern that is not always directly representable in real world settings. Phases can be merged, recombined and looped. Creative processes often may not follow a linear pathway because new insights can arise while creating. For example, the activation phase might be revisited many times until a fully effective solution is developed. New insights, errors of judgment and the need for compromise due to differences in preferences of creative teams happen all the time. Looped or iterative creative processes can deal with these issues, as they allow us to reflect and communicate, and decompose the problem we are trying to solve. Therefore, looped creative processes are of special interest in this thesis.

Arieti (1976) defined the creative process as an iterative sequence of steps or phases. This looped process can be encoded using a Darwinian interpretation of creativity (see Dasgupta (2004) and Cropley and Cropley (2010)). Another example is the Geneplore model by Finke et al. (1996) in which a two-stage subdivision is presumed: in the first, generative stage (or phase), based on memory retrieval, association and analogical transfer, a preinventive structure is generated. In the second, exploratory stage, attributes and interpretations about that structure are found, out of which a meaningful creative outcome emerges. Both Arieti and Finke apply the idea of iterative search to the creative process.

Shaw (1989) believed that the loops and phases could be modeled mathematically. He foresaw the introduction of powerful new computational techniques that would support creative processes based on these models (see Aldous (2017)). These looped creative processes give way to the idea of computer aided design and ideation, which are central concepts used in this thesis.

Requirement

The creative process should be iterative and looped.

Before we dive into what creative machine agents can do for and with us in an iterative process, we need to establish what traits human creators need to have to be creative and what traits inhibit creativity. The mental elements that determine

the ability to be creative consist of cognitive skills and traits that are described in the next section.

2.2 Creative Cognition

Some criticism arose about models of creativity, especially on the stage models, first formulated by Guilford (1967) when he wrote about the rather abstract phase of incubation: "*It is not incubation itself that we find of great interest. It is the nature of the processes that occur during the latent period of incubation, as well as before it and after it.*".

The interest in what processes happen in the mind gave rise to creative cognition, a subfield of cognitive psychology. Creative cognition aims to advance the understanding of creativity and fundamental cognitive operations that produce creative thought, and to provide experimental evidence for such cognitive processes (see Finke et al. (1996) and Ward et al. (1999)).

To be creative, to perform a creative act, we need certain traits and abilities. These include problem definition and redefinition, divergent thinking, reorganizing of information, feature mapping, analysis, analogical thinking, selective combination, evaluation, thinking styles (being more or less detail oriented), personality traits (tolerance of ambiguity, openness, perseverance) and domain-relevant knowledge (see Lubart (2001, 2005)).

As shown in the following, some traits of our mind prevent us from being creative, which is where computers can help us.

2.2.1 Divergent Thinking

As mentioned before, novelty and usefulness, the two aspects of functional creativity, are extremes on the scale of evidence. On the one hand, old solutions are well understood and can be used with greater certainty about their usefulness. On the other hand, we do not have direct evidence for the usefulness of novel solutions. When we venture into finding novel solutions, we have to deal with this lack of evidence. So the question arises how we traverse the space of all solutions. This process has been described as divergent thinking (see Guilford (1967)): the generation of a large quantity and diversity of solutions, which may therefore

be original, in contrast to a process that converges and usually leads only to convention.

Mednick (1962) introduced an *associative theory* that describes the process of moving from one idea to another through association, e.g. through *functional proximity*. These idea chains form a structure in divergent thinking with the more original ideas, so called *remote associates*, produced late in this form of ideation, far removed from the starting point.

Requirement

Comparisons between solutions should be based on their functional expression, their *phenotype*.

Volle (2018) presented evidence that divergent thinking is connected to two modes of cognitive processes: associative and cognitive control processes. The first activate remote ideas by spontaneous *spreading of semantic activation* from close to distant associates (representatives). The control processes allow for voluntary activation, elaboration and selection of remote ideas.

Requirement

To support activation of remote ideas by semantic spreading, close as well as distant associates need to be perceived by the user.

Divergent thinking is an indicator of creative potential, yet only a moderate indicator of creative performance. Runco (2010) suggested that it is beneficial to consider a large number of ideas. They have to be produced in an efficient way in real world co-creative systems, which is important because long evaluation times are common in these settings.

Requirement

The user should be efficiently presented with a large number of ideas.

Divergent thinking is not reserved for the individual alone. Brainstorming, using Jungian extraverted intuition, is a form of group divergent thinking. Runco (2010) performed experiments with participants in brainstorming sessions. They were "*told to (a) postpone judgment, (b) focus on quantity of ideas (fluency) and not quality of ideas, and (c) use each other's ideas as springboards for one's own*

thinking.". Runco noticed that "*when in groups, social loafing is likely, and when working in a group, the most original ideas tend to be risky precisely because they are original. They are risky in the sense that an individual is taking a chance by sharing something that other people may not understand or appreciate.*"

Since brainstorming with others can be restrictive to creativity, original ideas might survive longer if individuals can perform brainstorming with an 'objective' participant, a computer that introduces novel ideas but also reflects and varies the user's ideas.

Requirement

The creative process should contain an objective participant to reduce the risk of restriction through misappreciation of ideas in a group of human agents.

2.2.2 Priming

A lack of creativity when generating ideas was found to be connected to recently activated knowledge and examples. The hypothesis "*that a broad or narrow scope of perceptual attention engenders an analogously broad or narrow focus of conceptual attention, which in turn bolsters or undermines creative generation*" was tested by Friedman et al. (2003). Their experiments provided support for "*the attentional priming hypothesis, suggesting that situationally induced variations in the scope of perceptual attention (and simple cues associated with such variations) may correspondingly expand or constrict the focus of conceptual attention within the semantic network, thereby improving or diminishing creativity*". Presenting a diverse set of examples will therefore increase creativity in the human.

Requirement

A diverse set of examples primes the user for an increase of creativity.

2.2.3 Convergent Thinking

Divergent thinking is necessary to find novel solutions, yet their usefulness is not a primary goal in the process. Solutions might not be as useful as they could be. In order to innovate, ideas need to be optimized towards reaching goals to be

useful. As stated by Müller-Wienbergen et al. (2011): "*The human mind is prone to reproduce what it is used to. [...] Hence, IT systems supporting creative work have to support creative individuals by extending their personal knowledge while, at the same time, preventing them from merely walking down beaten tracks*". They "*propose a design theory for IT systems that support both convergent and divergent thinking*".

Both convergent and divergent thinking are involved in functional creativity. A computer aided ideation process should therefore be designed to help with both divergent as well as convergent thinking.

Requirement

The artificial agent should perform both divergent as well as convergent thinking.

2.2.4 Reorganization of Information

The process of reorganizing information is part of creative thinking as well. Reorganization, according to Lubart (2001), consists of "*problem construction, information encoding [...], category search [...], specification of best fitting categories, combination and reorganization of category information to find new solutions*". Mumford et al. (1991) postulated that "*information is stored and interpreted in categorical structures [...]. These categorical structures or schemata represent an organized interrelated set of discrete pieces of information, where organization is derived from the central features of the category. Information about solutions is reorganized alongside feature categories*". The organization of divergent search results along meaningful features is an inherently important feature to communicate them to the user.

Requirement

Solution candidates have to be presented along meaningful feature dimensions.

2.2.5 Dissonance

A link was found between cognitive dissonance and creativity. Dissonance causes unease and discomfort, but is shown to be a possible precursor to creativity, simulating human cognition. As Runco and Pritzker (2020) described: "*Since dissonance is discomforting, it often functions as a source of arousal. The dissonance provides most people with a directional goal in terms of which they can either search for memories to support the desired conclusions or search out new knowledge to create a new explanation. The latter is actually how cognitive dissonance leads to the motivated reasoning resulting in creativity.*"

So how can we trigger dissonance, possibly even in the minds of 'less creative' people? I pose that by having a computer generate novel but functional solutions, the minds of engineers can be triggered, even if they seek to avoid these triggers, by codifying the induction of cognitive dissonance within the creative process.

Requirement

A machine should ideally trigger cognitive dissonance in the user by generating novel and functional solutions.

2.2.6 Resonance: Experience, Intuition and Bias

Psychologists have observed an interesting effect of experience when they tried to correlate it with creativity. On the one side, a layman without any experience usually has no way to recognize problems or novel solutions. One needs to be somewhat of an expert to be able to perform activation, due to the knowledge base necessary to be manipulated to yield effective novelty. However, weathered experts can have a "*vested interest in maintaining the status quo*", as Cropley and Cropley (2010) stated. Even without such a conscious interest, functional fixity and confirmation bias can lead to an unwillingness or inability to recognize new problems and solutions. There is quite some evidence that the "*relationship between level of preexisting knowledge and creativity is U-shaped: Both very high (great expertise) and very low (ignorance) levels of preexisting knowledge may inhibit Activation*" (see Cropley and Cropley (2010), Mumford and Gustafson (1988), and Martinsen (1995)).

Humans thus tend to generate solutions that are based upon those they have encountered before. This makes the phase particularly hard for those that lack that experience. They might tend to perform domain substitution, using solutions from other domains, but only after they have understood the problem at least to a certain extent. Experts are more able to generate solutions, but here may arise a problem again, as too much experience could lead to a more conservative stance on design.

So with this in mind, computer aided creative processes should be designed to support dealing with both extremes of the scale of expertise. A need for creative processes that can support both novice as well as experts exists. As Bonnardel and Zenasni (2010) emphasized: "*novices and expert designers differ in their creative process and the kind of help that a computer system could best provide depends on the user's level of expertise*".

Requirement

The creative process should be designed to support both extremes of the scale of expertise.

2.3 Computational Creativity

After describing how creativity can be decomposed in interacting phases, what cognitive traits encourage or discourage creativity, and how these traits should be supported by computers, in this section computational creativity and computer aided ideation are discussed.

Computational creativity, according to Colton et al. (2012), is the "*philosophy, science and engineering of computational systems which, by taking on particular responsibilities, exhibit behaviours that unbiased observers would deem to be creative*". This is a more strict interpretation of computational creativity. As the Association for Computational Creativity (2020) defined: "*The goal of computational creativity is to model, simulate or replicate creativity using a computer, to achieve one of several ends: to construct a program or computer capable of human-level creativity, to better understand human creativity and to formulate an algorithmic perspective on creative behavior in humans, and to design programs that can enhance human creativity without necessarily being creative themselves*".

The difference between computationally creative systems and creative support tools lies in the degree of *responsibility* of the system we create (see Colton et al. (2012)). In the former case, the system itself should be deemed creative, in the latter case it only assists the user in being creative. Maher (2012) introduced a new perspective on creativity as being ascribed "*to a computational agent, an individual person, collectives of people and agents and/or their interaction*". A computer aided design system was first introduced by Kamentsky and Liu (1963). It featured an early interactive creative process involving humans and a computer.

Lubart (2005) proposed four categories of human-computer interaction to promote creativity:

- computers may facilitate the management of creative work
- computers may facilitate communication between individuals collaborating on creative projects
- the use of creativity enhancement techniques
- the creative act through integrated human-computer cooperation during idea production

The third approach is dominant in most work (see Wang and Nickerson (2017)) but this thesis takes specific interest in defining creative support systems in which the computer agent and the human user or users cooperate on a somewhat equal footing. The system should be able to be used as a creativity support system for individuals as well as for groups, taking advantage of both introverted as well as extraverted intuition and incorporating the artificial agent as a colleague, or rather an inspiring subordinate.

Requirement

The creative process should incorporate an artificial agent as an inspiring subordinate, and allow one or multiple users to interact.

As was established by Lubart (2005), some of the creative artificial intelligence programs that were deemed a 'failure' because they were not fully autonomous and always involved some level of human interference, were actually examples of successful human-computer interactions that facilitate creativity. He stated that it is possible to conceive of computers as real partners. He continued to say that "*The most ambitious vision of human-computer interaction for creativity involves a*

real partnership, in which humans and computers work hand in hand". Machines *can themselves be creative, or contribute new ideas in a dialogue with humans*". Although this position on machine creativity can be criticized, because in the end we humans judge whether something is creative or not, or at least define the metrics that can measure creativity, the weaker proposition of machines contributing and communicating new ideas leads to co-creative processes. Requirements for an iterative co-creative process are described in the following section.

2.3.1 Requirements of Co-creative Systems

The requirements that were defined in Sections 2.1.2 – 2.3 can be summarized in three general requirements.

- The creative **process** should be evolutionary, iterative and interactive, containing (at least one) objective participant as well as one or more users.
- The **search method** used by the objective participant should generate a diverse set of novel and functional solutions to trigger both dissonance as well as resonance with novice and expert users.
- Solution **similarity** should be based on their functional expression, the way solutions solve a problem, and presented along meaningful feature dimensions.

Research on functional co-creativity has produced a large set of (possible) requirements. Two complementary works have had a large impact on co-creative design environments. Parmee and Bonham (2000) defined requirements for techniques that support innovative and creative design activity in an interactive design team and evolutionary search environment. Müller-Wienbergen et al. (2011) stressed the importance of knowledge in creative work and propose that creativity support systems should support knowledge acquisition.

The full list of requirements is based on the previous sections and requirements extracted from Parmee and Bonham (2000) and Müller-Wienbergen et al. (2011). The list is summarized in Tables 2.1, 2.2, 2.3, and 2.4.

Table 2.1: Process requirements for co-creative processes I

ID	Requirement
A1	The process should be iterative
A2	The process should be interactive and contain (at least one) objective participant
A3	The initial solution set presented to user should need as little guidance from the user as possible
A4	The process should capture design knowledge through designer interaction
A5	The process should be adaptable along the resonance-dissonance dimension
A6	The process should contain evolutionary components

Table 2.2: Knowledge requirements for co-creative processes II

ID	Requirement
B1	Solutions should be categorized
B2	Knowledge should be organized hierarchically
B3	User can filter knowledge base dynamically
B4	Process should provide diverse perspectives on existing knowledge

Table 2.3: Search requirements for co-creative processes III

ID	Requirement
C1	Search should efficiently generate a diverse set of novel, functional solutions
C2	Search should efficiently sample complex design spaces
C3	Search should identify regions of design feasibility and optimality
C4	Search should support addition, removal and/or variation of constraints, objectives and variable parameter bounds
C5	Solution similarity should be based on their functional expression and presented along meaningful feature dimensions

Table 2.4: Communication requirements for co-creative processes III

ID	Requirement
D1	Communication should be evocative through features such as visualization and abstraction

An overview of the state of the art in co-creative systems follows.

2.3.2 Related Co-creative Systems

In this section I will discuss some of the many co-creative support systems developed in the last few decades. It is by no means intended to be complete, but provides a rough overview. The systems described here are generative, presenting a set of solutions, and form an active participant in an interactive co-creative process involving AI and users.

A priori generation, parameter range driven selection. A typical work in interactive design, Parmee and Bonham (2000) and Parmee (2002) introduced an environment in which, by iterative adjustment of rule-based preferences (parameter, constraint and objective weight ranges), an evolutionary algorithm is used to rapidly discover design requirements. Bradner et al. (2014) found that *some* architecture firms use parametric models, associated parameters and simulations as a form of design documentation. Having a parameterized representation at some point in the design process allows designers to tune parameters a posteriori. This process also allows classical parameter-based optimization to take place. Yet, their finding that "*parameters tell the design story*", which is lifted into the article's title, is not concisely defined. Moreover, the finding could be rephrased to "*design parameters summarize the history of the design process and the encoding replaces the final object with a collection of objects that are connected through their parameters*". The question remains whether it is parameters that tell the design story or whether, alternatively, phenotypes and how they are related to each other through high level features give more insight into what solutions could look like, providing more understanding while allowing novel solutions that do not 'fit' the parameter space to be protected.

A priori generation, example driven selection. Opposite to a focus on finding parameter ranges, the 'design by shopping' paradigm by Balling (1999) and Stump et al. (2003) introduced an alternative way for users to express preferences in a process of *a posteriori articulation of preference*. A Pareto front of optima is created by a multi-objective optimization algorithm, after which engineers choose a solution to their liking. In a study by Bradner et al. (2014) the authors

reported that many participants consult "*Pareto plots iteratively in the conceptual design phase to rapidly identify and select interesting solutions*". Pareto plots offer alternative trade-off solutions, which implies that the dimensions of variation follow some style of objective. Herring et al. (2009) showed that examples are a cornerstone of creative practice and are utilized for many reasons throughout the design process.

Secretan et al. (2011) performed an online experiment in which they had a large crowd of users improve the quality and quantity of generated artifacts, using an underlying non-linear evolutionary encoding. Gerber et al. (2012) generated design alternatives according to user defined parameter ranges and introduced automatic quality analysis of these alternatives. Woolley and Stanley (2014) introduced a collaboration between *novelty search*, an optimization method without an objective other than finding novel solutions, and, through interaction, human users. They found that this combination, in which humans select robot behaviors from a computer-generated set, leads to a faster convergence on solutions to some deceptive problems. However, this method still aims to find a single solution to a task. Although it does not emphasize computational creativity, it has promising components. Similarly, Sørensen et al. (2016) showed that users can guide evolution with respect to their preferences. They performed an experiment where users iteratively selected Super Mario behaviors from a set of generated candidates. The participants were shown to be able to evolve diverse behaviors.

In reinforcement learning, creating a neural network model from human preferences makes it possible to find robust controllers without an explicit objective. Christiano et al. (2017) created a system that presents the user with pairs of examples, letting them pick which one they prefer. Bontrager et al. (2018) used evolutionary techniques to search the latent space of generative models and presented these results to participants in an experiment. The participants then selected those results that resemble a predefined target. The experiment showed that the combination of techniques allows a user to evolve complex objects to a specific goal by selection.

A posteriori generation, example driven selection. In an example of a posteriori user selection, Liapis et al. (2013b) introduced a constrained novelty search to generate alternatives to the user's design. A distance metric based on the Hamming distance between expressed game level maps was used to generate novel

alternatives. The authors than take the idea introduced by Schmidhuber (2007) that beauty can be expressed as the compressibility of objects. This quantitative method measures 'elegance' or regularity of an observation. Liapis et al. extended the approach using an autoencoder that compresses the set of solutions which is produced in an exploratory phase. They defined an 'interestingness' metric that measures "*where there is potential to improve the compression ratio, or in other words potential [...] to learn about this type of environment*". This springs from the idea that highly compressible data sets might be elegant but also empty of information. According to the authors, adding a new data point to a solution set should be dependent on how much it improves the compressibility by an autoencoder, uncovering elegance and regularity in the full data set.

The interestingness metric, based on the distance of a solution to known solutions in the autoencoder's latent space, was used by Liapis et al. (2013) to inform the exploratory step in the next iteration, producing newer, more interesting solutions. This approach was then used by Yannakakis et al. (2014) in an iterative mixed-initiative co-creative process. Alternatives to human design updates are suggested by the computer in real-time. Fitness dimension weights are adjusted to reflect the user's selection (see Liapis et al. (2014)). However, the mere suggestion of alternatives to the user's ideas might not be sufficient to trigger cognitive dissonance that might truly inspire the person. Difficult problem domains can have unexpected solutions, and relying on human preferences alone is not enough to reduce the effect of the u-shaped creativity curve. Furthermore, the initial set of solutions might have a large impact on the user's cognitive priming.

2.4 Process Model

A model for computational co-creativity is defined here. It is built around the reflective and provoking interaction between an objective participant, an AI agent, and one or more users.

The users formalize the representation and quality of solutions in closed form. The agent then performs divergent-convergent search to create a large and diverse set of useful solutions. The set is presented to the users to expand the way they are a priori primed, trigger cognitive dissonance in case they find unexpected solutions, and help discover their preferences. The agent is an active but objective partner

in the creative process, as it does not have preferences of its own, but rather tries to create as many solutions as possible. This brainstorming aid can make an inexperienced user understand what solutions might look like and inspire an experienced user to think outside of the box.

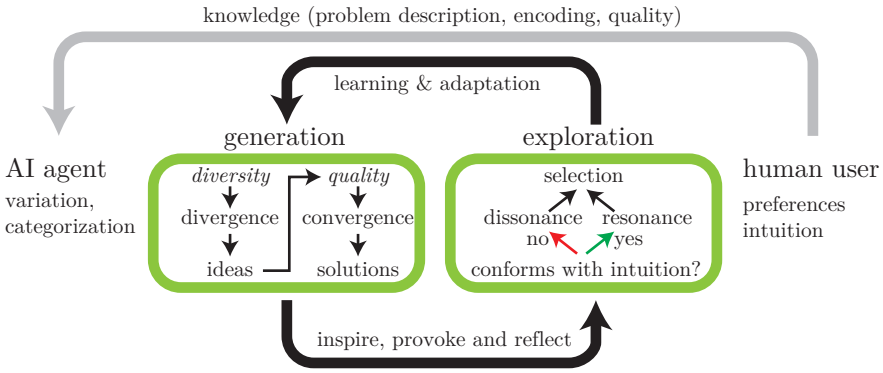

Figure 2.1: Model of the interaction between an objective, creative agent and a human user.

The process model is shown in Fig. 2.1. It is somewhat similar to the generative-explorative models employed by Finke et al. (1996) and Ward et al. (1999), as it consists of the two stages of generation and exploration. In the model introduced here, the generation stage is performed by the AI agent, which, based upon the problem representation defined by the user, creates ideas and turns them into categorized solutions using a diverge-converge operation. This operation is based on evolutionary optimization because of the amount of evidence that they foster diversity (see Hagg et al. (2017) and Lehman et al. (2020)).

The human user then explores these solutions, based upon their internal knowledge and intuition, being inspired by the dissonance caused by unexpected solutions. The user expresses their preferences through selection of solutions. The AI is responsible to inspire and provoke the human but also to reflect the user's preferences. It does so by collecting relevant information through learning and adaptation.

This by no means makes the AI component a fully creative, independent agent, as the knowledge and activation, as well as the exploratory phases, are still the responsibility of the user. They need to define an encoding of a solution and can influence the AI agent's search by adding objectives. However, the AI agent is responsible for generation (divergence), illumination (convergence), verification

(quality) and communication (presentation) of solutions to the user. By adding random variation and evolutionary search inside the agent, ideas (novelty) are added in an objective manner. The agent is also responsible to find high quality solutions and learn the user's preferences. The human is responsible to select solutions and determine when to end the process.

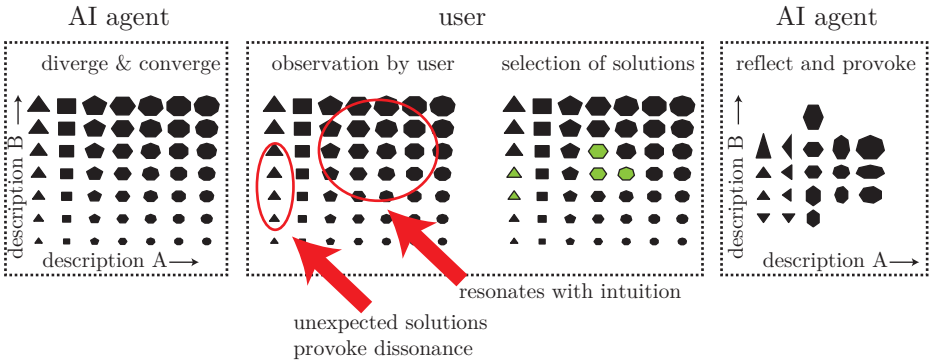

Figure 2.2: Example of a dialogue between an objective, creative agent and a human user. First, the AI agent uses divergent and convergent techniques to generate a diverse set of solutions along meaningful feature dimensions. The human agent is then confronted with this set. Some solutions will conform to their intuition, but unexpected solutions provoke dissonance. The human agent then selects the solutions according to their preferences. The AI agent in its turn reflects upon this selection and provokes the user again by finding variations on the selected solutions.

Fig. 2.2 shows an example of a partial dialogue between the AI agent and the user. The solutions presented by the agent are categorized and visualized in an ordered fashion, along descriptive feature dimensions. After the user's selection, the AI agent creates a model of the user's selection and returns a new solution set that is in line with the user's selection.

The process is iterative (req. A1), consists of interaction (req. A2) between user(s) and an objective AI agent (req. A3), retains information about relevance and features of (intermediate) solutions (req. B1), efficiently generates a diverse set of useful ideas (req. C1, C2, C3, C5), retains information about user(s) preferences (req. A4), generates variations of solutions (req. B4), communicates solutions to the user(s), allows the user to influence the representation of the solution set (req. B3), and allows a user guided, hierarchical decomposition of the solution space (req. B2). Finally, the process is adaptable along dimension resonance and dissonance

(req. A5). The scope of this thesis does not include req. C4, the addition, removal and or varation of constraints, objectives and parameter bounds.

2.5 Chapter Summary

This chapter discussed the creative process and connected it to cognitive effects that need to be taken into account when designing a co-creative process where AI and human agents work together. Requirements were extracted for co-creative systems and examples for such systems that can be found in literature were described.

A process model for computational co-creativity was defined that is built around divergent search. The process in itself is based on decades of research in cognitive psychology, creativity, and (evolutionary) optimization. The contribution is inspired by the ability and interest in novelty- and diversity-based search. Only in the past decade, methods have been developed that emphasize diversity as well as quality, and the understanding of very high-dimensional objects and spaces.

This work tries to provide a deeper connection between cognitive psychology and (divergent) optimization. An emphasis on high functional diversity of solutions is made to increase the probability of both dissonance and resonance. The use of an artificial, objective participant allows us to question the user's intuition, advancing their understanding of design problems. The objective participant can help to combat group pressure between peers and presents a large number of functional ideas to prime the user and trigger dissonance.

By using phenotypic features to structure the solution set shown to the user, the process allows the user as well as the AI agent to learn a model of the functional solution space over meaningful differences and better understand the user's preferences about *how* artifacts solve a problem. The process is capable of handling high-dimensional artifacts in complex and real world engineering settings.

The objective participant feeds and reflects upon the interaction with the user and merges the intuitions of what is possible and preferred in a model. The model and the divergent search algorithm are the motor behind the creative process, in which computer and users co-create a common understanding.

The following chapters of this dissertation will describe how to efficiently create a diverse solution set while taking into account the user's preferences.

CHAPTER 3

Divergent Search

One widely-used method to generate diverse solutions is to rely on random or semi-random search mechanisms to generate novel, unconventional ideas. Blind or unbiased variation is often used as part of an evolutionary algorithm (EA) (see Appendix A), which selectively retains perturbed solutions when indications exists that they are useful, as was described by Simonton (2007). Some stage theories of creativity, which were described in Section 2.1.1, contain a recombination step (see Santanen et al. (2004) and Nijstad and Stroebe (2006)) that causes high cognitive load in humans. To find more distant, novel, solutions, evolutionary computation should be used as the driver of variation in creative processes (see Lehman et al. (2020)). EA perform specifically well at finding 'haphazard recombinations' to generate novel solutions. In a recent article, Miikkulainen (2020) described that he expects evolutionary computation, combined with novel learning techniques, to be the 'next deep learning'. He pinpoints machine creativity as the upcoming milestone in artificial intelligence.

In this chapter I discuss divergent evolutionary optimization techniques that form the cornerstone of the objective participant in the process model introduced in Section 2.4. In the divergent search paradigm, we do not look to find the best solution, but instead find a diverse set of high-performing solutions. Diversity is the result of a process that finds novel solutions and is usually measured and enforced based on distance metrics on a set of descriptive feature dimensions. This chapter discusses solution diversity as treated throughout the history of evolutionary optimization algorithms. Three main paradigms are examined: multiobjective (or multicriterion) optimization, multimodal optimization, and phenotypic niching (also called quality diversity (QD)). The latter connects the concept of ecologic niches in biology to evolutionary optimization. Common diversity metrics are

used to compare these diversity maintenance techniques, whereby understanding the difference between genetic and phenotypic similarity helps us to create and select these divergent algorithms for use in co-creative processes. The solution sets that are created help to overcome the limits of our imagination (see Sartre and Elkaïm-Sartre (1946)), by showing a large variety of solutions.

In this chapter, the following requirements from Tables 2.1–2.4 are fulfilled:

- A3 ("The initial solution set presented to user should need as little guidance from the user as possible")
- A6 ("The process should contain evolutionary components")
- B1 ("Solutions should be categorized")
- B2 ("Knowledge should be organized hierarchically")
- B3 ("User can filter knowledge base dynamically")
- B4 ("Process should provide diverse perspectives on existing knowledge")
- C1 ("Search should efficiently generate a diverse set of novel, functional solutions")
- C3 ("Search should identify regions of design feasibility and optimality")
- C4 ("Search should support addition, removal and/or variation of constraints, objectives and variable parameter bounds")
- C5 ("Solution similarity should be based on their functional expression and presented along meaningful feature dimensions")

In order to achieve this, the following research questions need to be answered:

Research Questions

I Are solutions best compared using their genomes or their expressed phenotype or behavior? (requirements B1, B2, B3, and C5)
II What multi-solution optimization method produces the highest phenotypic diversity? (requirements A1, A6, B4, C1, C3, and C4)
III Can we produce more diverse solution sets when learning phenotypic niching from data instead of using predefined features? (requirements A3, B4, C1)
IV What are the limitations of generative models in terms of the possible diversity of the solutions they create? (requirements A3, B4, C1)

The main insight highlighted in this chapter is that in multi-solution optimization, a focus on providing functional diversity can be achieved by applying diversity maintenance based on artifacts' morphological and behavioral similarity.

What follows now is a connection between research in natural evolution and ecology on the importance of the phenotype, or expression of a genome, to evolutionary encodings. This is taken from work by Hagg (2021).

3.1 Extending the Phenotype

In this and the next section, an answer to research question I ("Are solutions best compared using their genomes or their expressed phenotype or behavior?") is sought. To achieve this, the manner in which solutions are encoded in (evolutionary) computation needs to be discussed. Let us take a step back and analyze what ideas from natural evolution, the origin and inspiration of evolutionary computation, tell us about computational encodings.

The two key features of natural evolution are survivability (quality, fitness, or optimality) and diversity. Quality is mainly driven by naturally occurring scarcity of nourishment, weather and safe environments. Only creatures that are fit enough will survive to create offspring. Mutations in genomes of the offspring of successful creatures cause the population to diverge and explore possible genomes. Diversity arises from the driving forces of random variation and protection by *niching* effects. Evolutionary niches shield species from having to battle for nourishment with all other species. Each species naturally converges to being fit enough within a niche, becoming specialized on surviving in that niche, or even by constructing their own niche (this last effect was described by Laland (2004)).

In classical EA, an emphasis is put on quality rather than diversity of outcome. Diversity only serves as a means to a goal. In the context of engineering optimization this makes sense. In the end we want solutions to perform as well as possible and computational methods are better at maximizing the optimality of large solution sets than human engineers are. Diversity measures like niching (see Schaaf et al. (2003)) are used to prevent a population-based optimization algorithm from 'getting stuck' in local optima, a sub-optimal region of the solution space. The population within the algorithm is artificially spread out by the combination of a mutation operator, which introduces random variations of the population, and a diversity

measure, subdividing the population between multiple attracting optima, as was discussed by Asteroth and Hagg (2015). Oftentimes a recombination operator is used as well to allow for jumps in the objective landscape, combining components of existing solutions into new artifacts.

Niching is often tightly connected to the concept of speciation. Only individuals that are located in or close to the same niche interact directly. These individuals are likely to be more genetically compatible and produce offspring. In evolutionary computation, speciation is often seen as a core property of artificial niching methods. "*Niching in evolutionary algorithms is a two-step procedure that a) concurrently or subsequently distributes individuals onto distinct basins of attraction and b) facilitates approximation of the corresponding (local) optimizers*", as summarized by Preuss (2006). Basins of attraction represent high fitness regions in the objective function out of which there is no 'escape path' that does not go through a low fitness region. A species will therefore 'nest' itself inside of a basin of attraction.

3.1.1 Genome, Phenotype and Behavior in Nature

Let us now look at the concept of fitness in an example from biology. A simple fitness metric can be defined as the amount of grass in kilograms eaten per day. This metric might be naive but it is not that different from the kind of naive metrics often used in computational evolution, for example the amount of wind resistance that is caused by an airplane wing or the number of crates stacked by a robot per hour. The amount of grass a cow eats is many times larger than what a cutworm is able to consume. Both species are specialized in their own niche, both have reached a local optimum, but when using the proposed fitness metric, their absolute fitness is vastly different. Yet, both species are in an evolutionary stable niche and have been able to survive for long periods of time. We can derive that a highly diverse set of solutions can emerge from natural evolution. In the example above, neither solution is 'preferred', as there is no invisible engineer selecting between those niches.

An ecologic niche can be seen as a basin of attraction in which only species with certain properties can survive. A species that is effective at surviving in a niche is less likely to adapt towards another basin, as it would first have to cross unfit genetic states, although this is not impossible and is often seen in bird species that migrate to islands and become fit with respect to another niche. This can lead

to large changes in the behavior of a species, for example birds losing their flight abilities due to genetic drift (see Wright et al. (2016)).

In computational evolution, a niche is often reduced to a basin of attraction in the objective function over the genome, or encoding. This is very common in benchmark comparisons between algorithms. However, this is a simplification compared to natural evolution. A natural niche can be filled by different genomes. It is not the cutworm alone that can eat grass. Cows can too. Both can digest grass and therefore have similarities in their phenotype, the expressed morphology (the ability to metabolize grass) and behavioral expression (searching grass for nourishment) of a genome. Describing a niche by looking at the genome's expression seems to make more sense than concentrating on its genome alone. The expressed genome, or phenotype, for example the number of jumps performed by a robot biped controller (a behavior), or the amount of curvature in an aerodynamic shape (shape morphology), surely is a closer description of the individual in terms of what niche it can occupy, how successful it can be inside the niche and how it achieves this success.

An example that entails more complex behavior is swimming. Even a genome that does not provide the ability to 'properly' swim, like a fire ant's, can cross rivers through the expression of that genome. Fire ants achieve this through cooperation with conspecifics by self-assembling into a raft, a population level behavior that was observed by Mlot et al. (2011). Thus, a beaver and a fire ant, both having genomes that occupy vastly different basins of attraction in genetic space, can occupy the same ecologic niche according to certain features of their phenotype (see Figure 3.1). They can both cross a river, so display behavioral similarity, and can occupy similar niches, at least those that contain rivers.

An important insight is that a beaver does not swim the same way a colony of fire ants does. The beaver's body performs swimming motions whereas the fire ants need to cooperate. But on a simplified descriptive level, the ability to cross water, the behavior is the same. On the other hand, when we take 'body size' as a description, a large body size will prevent the beaver from entering places fire ants can reach. Fire ants and beavers can be distinguished in terms of the two phenotypic diversity metrics. In one they fall into the same niche, in the other they do not.

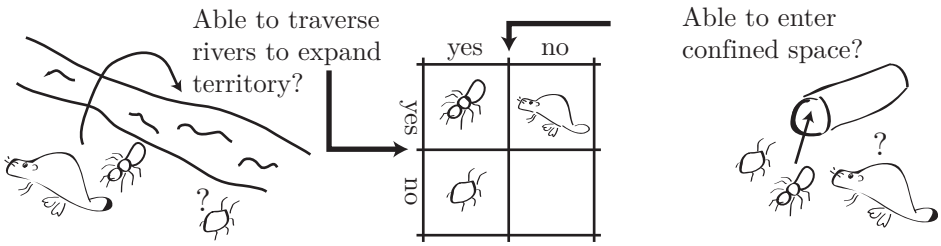

Figure 3.1: Some phenotypic features, like the ability to traverse a river, put the fire ant and the beaver into the same phenotypic niche (rows). Features like body size allows separation between the two species into separate niches (columns). Stink bugs can enter confined spaces like ants do, but they cannot swim, putting them into yet another niche. Both the morphology of the body as well as the behavior of a species determine what niche it can occupy.

It is the interaction between a creature and its environment, the behavior, not the genome, that ultimately defines a natural niche. Dawkins (1982) already posed that behavior can be subsumed into the *extended phenotype*. Non-behaviorial phenotypic traits influence the conditions under which a behavior can take place. A large animal will not be able to occupy niches that need a small body. The morphology of the species can limit or allow certain behaviors. Therefore, certain features of the extended phenotype, be it behavioral or morphological, are important for fitness and help to determine in what niche an individual can flourish.[1]

3.1.2 Encoding in Evolutionary Computation

What does the insight about the extended phenotype's importance mean for evolutionary computation? The distinction between genome and phenotype has been a key inspiration for evolutionary computing, as was described in Sterelny et al. (2001).

Solutions in evolutionary techniques are usually encoded as shown in Fig. 3.2. A discrete set of n parameters, the genome, constitutes the n-dimensional search

[1] Laland (2004) took Dawkins' idea one step further and introduces the idea of *niche construction*. By influencing its environment, a creature can create its own niche. This introduced the idea of evolution being a causally cyclical process. A creature creates a niche by acting on its environment, its genome adapts to the niche, which in itself causes changes in the environment, and so forth.

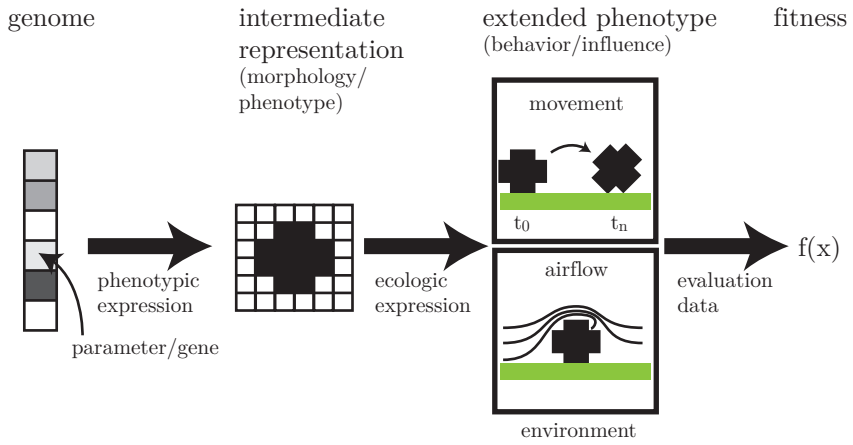

Figure 3.2: Evolutionary representation, an interpretation based on work by Whigham et al. (2017). A genome, consisting of a number of parameters is phenotypically expressed into an intermediate representation, e.g. a morphology. Through its ecologic expression, the behavior or influence on its environment, a solution is in a state where it can be evaluated using one or more fitness functions.

space in which solutions are evolved. Genomes are expressed into intermediate representations, the phenotype, for example a shape or a neural network. But the phenotype is not the full description of a solution. Only once the solution is inserted into the application environment on which it is supposed to act or interact with can we inform ourselves about its quality. One can argue that the 'genome-to-phenotype mapping'[2], the phenotypic expression, has to be extended with a 'phenotype-to-solution mapping'. From here on, separate terms for these mappings will be used: the phenotypic and ecologic expression. The fitness or quality of the phenotype is determined based upon a solution's ecologic expression, its interaction with the environment, or ecosystem.

Neutrality Although the expression function should ideally be one-to-one, it often does not prevent multiple genomes to be mapped to the same phenotype. The phenomenon of such a mapping is called genetic neutrality, which is akin to

[2]In evolutionary optimization, the term genotype is used instead of genome, although this constitutes a divergence from the nomenclature used in biology. There, a genotype is often used as a functional subset of a genome. In computer science, however, we do mean the mapping between the *entire* genome to a phenotype. Hence, the decision is made to use the less ambiguous biological term.

genetic neutrality in biology. In biology, a neutral mutation is understood to have no effect on the survivability of a life form. In evolutionary computation, a neutral mutation changes a genome in such a way that the phenotype is not changed. From here on out let us refer to genetic variants that lead to the same phenotype as neutral genomes (see Hu et al. (2011)).

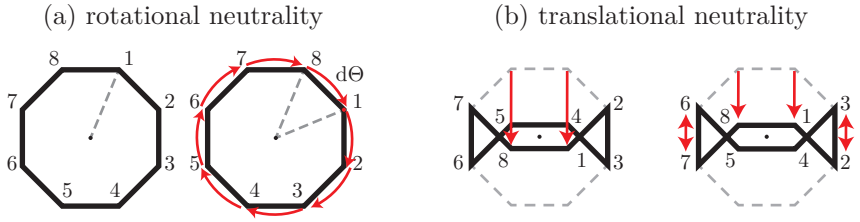

Figure 3.3: Genetic neutrality. The same phenotype is expressed when rotating the control points (numbered one through eight) by a $\frac{\pi}{8}$ angle (left) or by translating the control points as shown (right).

Fig. 3.3a shows an example of neutrality. For angles $\theta = 0°$ and $\theta = 45°$, phenotypically speaking, these shapes are the same. In this case, eight genomes all point to the same phenotype. Similarly, Fig. 3.3b shows how, through translations of the control points, a similar shape can appear based on different genomes.

A neutral genome-to-phenotype mapping (Fig. 3.4) can be highly non-linear. Neutral individuals contain different genomes but produce the same phenotype. A large change to the genome can lead to a small change in the phenotype.

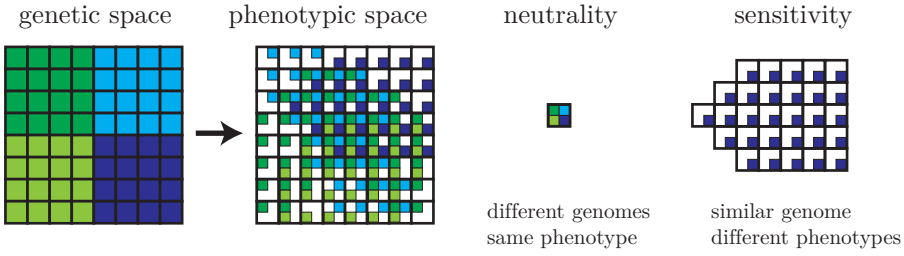

Figure 3.4: Due to genetic neutrality, different genomes (clarified by the use of different colors in the genetic space) can expose a similar phenotype and be put into the same phenotypic niche. Sensitivity can cause similar genomes to produce a relatively large range of phenotypes.

It is postulated that diversity maintenance in a neutral genetic space leads to lower phenotypic diversity than when using phenotypic niching. To increase diversity in phenotypic space, niching should be performed there.

Sensitivity A second effect arises when genomes are projected into phenotypic space in a non-linear way, as shown in Fig. 3.4. A small change in the genome can lead to large changes in the resulting phenotype (see Tarapore et al. (2016)). This effect is coined *sensitivity*[3]. The more sensitive a representation is, the larger phenotypic changes can be when applying small changes to the underlying genome.

Take a simple spline representation. The spline shapes are determined by control points. The shape of the spline can therefore extend beyond the control points' locations, creating a shape that is 'larger' than the space determined by the control points themselves. Fig. 3.5 shows three splines with very similar control point locations. Only the center control point's y value is varied, so the solutions are very close to each other in genetic space. The small changes in control point locations cause larger-scale changes in the phenotype. This, however, is not a mere scale difference, as it is obvious (for us humans) to see the qualitative differences between the solutions' expressed shapes.

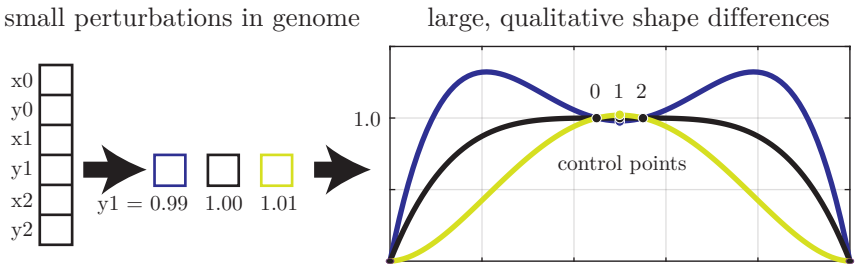

Figure 3.5: Three fifth order spline interpolations that have similar genomes.

The locations of control points relative to each other are more informative for predicting the final shape than their absolute locations. Hence, sampling the parameter space, which determines the control points' locations, does not give as much control over the search in morphological space, as when a search takes place in morphological space directly.

[3]This effect is connected to the concept of evolvability, described by Valiant (2009).

The observation that (genetic descriptions of) species sometimes fill more than one phenotypic niche is not restricted to non-closed form (non-parametric) or indirect encodings. The latter usually refers to neural representations in robotics where the mapping is highly non-linear and dependent on the environment. The effect can happen in any mapping between genome and phenotype. Some evidence was provided by Hagg et al. (2018), in which genomes are clustered in a low-dimensional space. The resulting solution *classes*[4] are not always mapped to a single phenotypic niche. In complex optimization tasks, like aerodynamics, bifurcations (abrupt large changes in the air flow when applying a small change to a shape within that flow) are very common in real world turbulent flow, as was shown by Carroll and Mehra (1982).

3.1.3 Conclusions

The evidence that neutrality and sensitivity occur in real world encodings and applications implies that, if we want to find a functionally diverse set of solutions, we need to search phenotypic or behavioral space, not genetic space. The concept of the extended phenotype and the effects of genetic neutrality and sensitivity provide evidence for an answer to research question I ("Are solutions best compared using their genomes or their expressed phenotype or behavior?"). I postulate that solutions are best compared using their expressed phenotype and behavior. To provide more evidence, the next section introduces the available multi-solution optimization paradigms after which, in Section 3.3, they are compared.

3.2 Multi-Solution Optimization Methods

In this section, three multi-solution paradigms are discussed to understand how the concept of diversity has been used in evolutionary computation. There is a myriad of arguments for diversity in optimization. The concept of diversity itself is considered to be the goal (see Basto-Fernandes et al. (2013)) for a number of reasons: generating alternatives for engineers to choose from, finding trade-offs between features, learning properties of the decision space (*innovization*, see Deb and Srinivasan (2006)), increasing robustness of optimization, allowing solutions

[4]Classes translate to species as defined by Basto-Fernandes et al. (2017).

to adapt to a changing fitness landscape (see Cully and Demiris (2017)), performing model-based diagnostics, or crossing the reality gap between simulation and reality (see Koos et al. (2012)). By understanding the functional diversity of high-performing solutions, we can find more unexpected solutions that provide users of co-creative systems with relevant dissonance effects to expand their intuition.

In this section, three multi-solution paradigms, depicted in Fig. 3.6, that have been developed in the last decades are shown. The paradigms are compared in the next section.

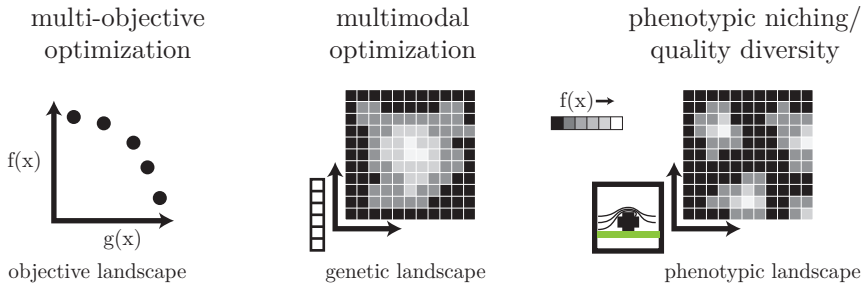

Figure 3.6: Three multi-solution paradigms. The multi-objective paradigm finds a Pareto front of non-dominating solutions along a number of objectives. The multimodal paradigm diversifies solutions according to their genetic distance. The phenotypic or quality diversity paradigm diversifies solutions in terms of phenotypic/behaviorial dimensions.

3.2.1 Diversity in Objective Space

In multi-objective (or multicriteria) optimization (MOO), the Pareto set of trade-off solutions with respect to two or more objective functions is discovered (see Fig. 3.6). One of the most successful methods is the non-dominated sorting genetic algorithm II (NSGA-II) that was introduced by Deb et al. (2002). The paradigm does not control the diversity of the genomes or their expression, only the diversity in objective space, by leveraging a crowding distance metric into the algorithm. However, solutions are expected to be 'different' as they fulfill different trade-offs between objectives. An assumption arises that the front consists of a diverse set of solutions, all with different qualities in terms of the objective functions. Genetic diversity has been established as an important factor in the success of

MOO algorithms, as was shown by Toffolo and Benini (2003) and Shir et al. (2009). As Ulrich (2010) noticed: "*sometimes it might be more important to find a structurally diverse set of close-to-optimal solutions than to identify a set of optimal but structurally similar solutions*".

Although MOO is arguably the first evolutionary technique that explicitly searches for a set of solutions, the field has taken most effort to find the Pareto optimal set and less into investigating the question of (phenotypic) solution diversity.

3.2.2 Diversity in Genetic Space

Computer science research up to the 1970's had already developed interest in population-based evolutionary approaches to optimization, but in that decade specifically, a number of approaches that increase genetic diversity were introduced. Arguably, genetic diversity was first used in evolutionary optimization not to find different solutions, but to deal with the problem of premature convergence to local optima. De Jong (1975) analyzed the problem of premature convergence to local optima in genetic space in his doctoral thesis in 1975. He recognized the need to integrate the concept of *niches* into the evolutionary optimization method. He introduced *crowding*, where child solutions only replace the most similar parent in the population, to improve performance on multimodal surfaces. Alternative techniques are *sharing*, introduced by Holland (1975), where fitness values are penalized due to proximity of other solutions, and *clearing*, introduced by Pétrowski (1996), where the population is subdivided into subpopulations according to their similarity in genetic space.

In the 1990s, multi-solution, multi-local or multimodal optimization (MMO) was put into focus. This paradigm has the explicit goal to find a diverse set of high quality locations in the search space, based on a single criterion (see Fig. 3.6). The first metric that was intended to be used on benchmark problems was peak ratio (see Ursem (1999)), the percentage of local optima that are found.

The field of MMO contains early work on restricted tournament selection to produce multiple solutions by Harik (1995) and was later more established by methods like basin hopping by Wales and Doye (1997), topographical selection by Törn and Viitanen (1992), restarted local search (RLS) by Pošík and Huyer (2012) and nearest-better clustering by Preuss (see Preuss (2010), Preuss (2012), Preuss (2015), and Wessing (2015)).

3.2.3 Diversity in Phenotypic Space

Section 3.1.2 gave evidence that it can make more sense to strive for diversity in phenotypic and behavioral space. A direct search is usually infeasible, due to the high dimensionality of the phenotypic space, or to the fact that it is not describable in closed form and can only be sampled through numerical or physical simulation. Instead, we can perform niching based on behaviorial or other phenotypic features. This is the idea behind quality diversity (QD) [5].

QD produces a set of solutions that are diverse in terms of their extended phenotype, yet still perform well. Instead of applying niching on the basis of genetic similarity, QD considers the behavior of a solution in its environment (see Fig. 3.6), like the average speed of a neural network controlled robot, the amount of turbulence caused by a car, or the amount of curvature of the surface of a complex water drain system, as the basis for niching. This enables QD to find unexpectedly good solutions, or make the optimization process more robust with respect to requirement changes. If one solution does not work well, engineers can take this new knowledge into account when selecting an alternative solution.

The search still usually takes place in genetic space. The high dimensionality of a closed form encoding, where we encode all points as separate parameters, would result in a very high dimensional search space which is not tractable. By intentionally confining the search into a lower dimensional representation, tractability is regained, while phenotypic niching ensures solution diversity.

The idea of measuring similarity on the phenotype level is not new. Pétrowski (1996) mentioned the possibility and the omni-optimizer by Deb and Tiwari (2008) took a more narrow view on phenotypic space, describing it in terms of objectives and constraint values. However, I argue that a fitness function should not be seen as an inherent property of the extended phenotype but rather a metric on the (implicit) goal of a creature or solution's ability to survive.

The first optimization paradigm that embraced behavior-based niching is novelty search (NS). In this new paradigm, Lehman and Stanley (2008) chose to ignore fitness, which can be deceptive and trap a search in a local optimum. Instead, they proposed to use behavioral similarity as an objective function and showed

[5] As in: diversity of qualities. Both quality diversity as well as *illumination* are used in the field, although one could view the deeper concept to be *phenotypic niching*.

that this can circumvent deceptive fitness functions, outperforming fitness-based optimization. They posed that many points in the search space collapse to a small set of simple behaviors, making novelty-based search feasible. NS introduces an archive of past, novel solutions, that serve as a basis for measuring novelty.

The approach was evaluated by evolving controllers for a biped robot. The controllers are able to use motor primitives of the robot's six degrees of freedom to assign poses, based on the inputs of two ground contact sensors only. The authors introduced a novelty metric based on a fixed-horizon sequence of changes in the horizontal coordinates of the robot's center of gravity. To compare two controllers, the sum of squared distances of both sequences was calculated. This way, robots that fall down are assigned the same niche (due to the fixed horizon). NS therefore ignores new solutions that fall down in the beginning and instead concentrates on finding new solutions that follow different trajectories. The results show that NS finds more biped controllers that do not fall down, and identifies better controllers than fitness-based search alone. This shows that NS is not just an exhaustive search method, but is able to ignore easy-to-reach local optima in a very hard search space.

Lehman and Stanley (2011a) posed that the problem of deceptiveness resides in the fitness function, based upon the fact that the underlying neuroevolution algorithm they used provides (genetic) diversity management and is not able to circumvent deceptiveness. The authors did acknowledge that removing the fitness function entirely is a problem in optimization, as it cannot be determined when solutions are fit or whether solutions are functional at all.

Phenotypic diversity allows us to judge different solutions in terms of how they behave in their environment. This can help ignoring solutions that show similar behavior and concentrate the search on novel behavior.

The introduction of NS led to further studying of the search for novel, non-optimal solutions. Inspired by the work of Mouret (2011b) on the reconciliation of NS with fitness, Lehman and Stanley (2011b) extended NS by adding a competitive factor. They introduced novelty search with local competition (NSLC). It formulates fitness and diversity as a bi-objective problem, treating both goals as equally important. NSLC finds a diverse set of high quality optimizers by performing niching in phenotypic space. Shown in applications for developing artificial creatures and robot controller morphologies, NSLC only allows solutions that belong to the same

phenotypic niche to compete. To this end it keeps track of an archive of niches. Solutions are added if their phenotype is novel enough or better than that of a similar solution.

For any new candidate solution, its novelty score is calculated by the average Euclidean distance to a fixed number of k nearest individuals in the archive, where k is determined experimentally, and distance is measured in phenotypic space. Then, the candidate receives a local competition score, represented by the sum of morphologically nearest individuals it outperforms. Both the novelty and local competition scores are then used in an adapted version of the MOO algorithm NSGA-II. The Pareto front that is produced consists of solutions that show the trade-off between novelty and local competition. Parents for a new generation are selected from the individuals in the front to produce new solutions that are different from those saved in the archive and high-performing when compared to similar solutions.

Although NSLC is shown to better exploit morphological niches than fitness- or novelty-based optimization, a number of problems exist with the approach. For one, the archive keeps growing, which makes the comparison of new candidates ever more expensive. The reliance on NSGA-II also makes it difficult to scale up to multiple diversity metrics. These problems were addressed by Mouret and Clune (2012). They formulated multi-objective landscape exploration, which works in a similar way, but emphasizes the aspect of exploring the landscape of phenotypic traits with respect to their fitness. Phenotypic niching is performed by collecting elites mapped by behaviorial traits. Combining this with the idea of building behavior repertoires (see Cully et al. (2013) and Cully and Mouret (2013)), a simplification of the multimodal QD algorithm, multidimensional archive of phenotypic elites (MAP-Elites), was introduced by Mouret and Clune (2015). It *illuminates* the fitness landscape through the lense of phenotypic features.

Algorithm Description QD searches in genetic space for solutions to fill phenotypic niches (Figure 3.7). A low-dimensional phenotype space is defined by selecting a number of phenotypic features, that can be based on behaviorial, morphological, or other features. Solutions' feature coordinates determine what niche they are placed into. They compete only with the solution in that niche. The archive represents the result of the QD algorithm.

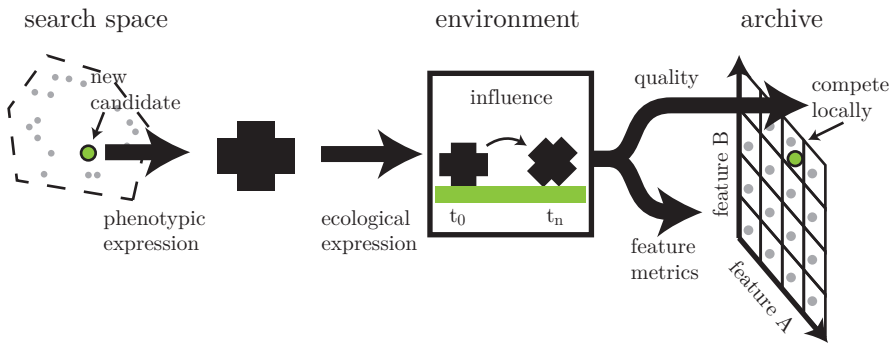

Figure 3.7: QD searches in genetic space but performs niching in a low-dimensional phenotypic archive. Candidate solutions are assigned to the niche based on their feature coordinates. These feature coordinates are calculated based on the expression of a solution, albeit phenotypic or ecologic (see Fig. 3.2). Candidates are only placed inside a niche if they improve its quality value.

QD algorithms follow the pattern described in Alg. 1. After initializing a population of solutions and setting an evaluation budget, parents are selected based on a scoring scheme. In NSLC, parents are selected from a separate population, whereas in MAP-Elites they are selected from the archive itself. Offspring is created, usually using mutation only. The performance of the children is evaluated and their feature coordinates are calculated. The individuals can be added to existing niches through local competition, or new niches can be created, depending on the archive structure that is used. Finally, the selection scores are updated.

Algorithm 1 Quality Diversity

Define and formalize phenotypic descriptors
Initialize genetic population $\mathcal{P}$
Initialize archive $\mathcal{A}$
for iter $= 1 \rightarrow$ generations budget **do**
  Select parents $\mathcal{P}$ to form new offspring based on scoring scheme
  Evaluate performance and phenotypic descriptors of offspring
  Add individuals (potentially) to niches in archive $\mathcal{A}$
  Update selection scores
end for

Cully and Demiris (2017) proposed a modular framework in which the main features of a QD algorithm are described: the type of container or archive and the

procedure for selecting the parents of the next generation.

Archive The archive is the central object in QD. It is constructed based on phenotypic features, and represents the current intuition of the diversity of high-quality solutions to a problem. The archives used in the first QD algorithms were either fixed (MAP-Elites) or unbounded (NSLC) (Figure 3.8). New archive designs allow them to be more efficient, effective, compact and capable of long term optimization.

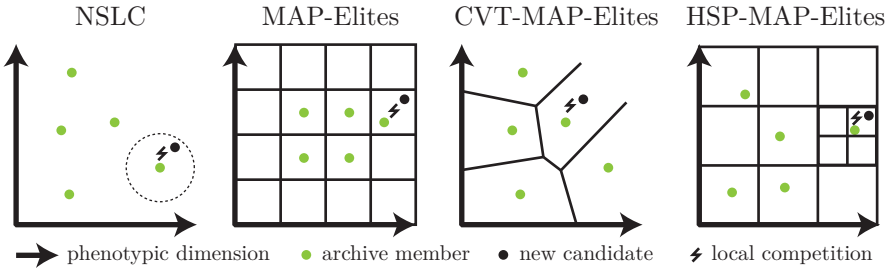

Figure 3.8: The first two QD algorithms introduced an unbounded (NSLC) and fixed-grid bounded (MAP-Elites) version of the phenotypic archive. The fixed grid in the latter can be replaced with a Voronoi or a hierarchical subdivision of the phenotype space. The feature space can also be subdivided in a hierarchical way.

The original definition of the grid, however, has a problem due to the growth of the number of niches, which is exponential in the number of feature dimensions. The ability of the MAP-Elites archive to deal with high-dimensional niching spaces was improved by Vassiliades et al. (2017b). The equidistant grid is replaced with a fixed number of predefined niches, independent of the dimensionality, using centroidal Voronoi tessellation (CVT) (see Figure 3.8). Smith et al. (2016) used hierarchical spatial partitioning (HSP), by which the speed of exploring the niching space was shown to be increased. HSP organizes optimization results in a hierarchical way, which would fulfill requirement B2 ("Knowledge should be organized hierarchically").

In work by Vassiliades et al. (2017b), the fixed extremes of the MAP-Elites archive were expanded during the QD process which leads to changing niche composition and possible removal of solutions. The now unbounded archive performs similarly to the bounded version, without having to manually predefine the extremes/bounds of the niching space.

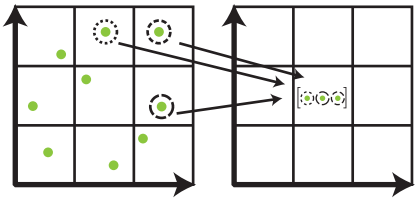

Figure 3.9: Hierarchical behavior repertoire. By building up a hierarchy of stacked archives, each archive can be filled using compositions of primitives from another layer.

The idea of using multiple archives was first introduced by Pugh et al. (2016), to give QD more options to sidestep local optima. Taking this idea one step further, Cully and Demiris (2018) introduced a hierarchical, stacked archive, where each layer uses the layer beneath because a set of primitives can decompose the problem of generating complex behaviors. This hierarchical organization of archives is another way to fulfill requirement B2 ("Knowledge should be organized hierarchically"). Finally, the archive can be decentralized and used in swarm robotics, as was done by Hart et al. (2018), which allows to simultaneously evolve multiple behaviors across multiple robots.

QD and its archive are similar to the idea of the morphological box by Zwicky (1969). They allow new creations by combining known solution configurations. Phenotypic features are an efficient method to represent the knowledge in the box, fulfilling requirements B1 and C5. The archive allows indirectly finding combinations of phenotypic feature, therefore it establishes an 'indirect recombination' operator, which makes the definition an explicit, encoding-dependent operation. obsolete.

Selection Procedure To form new offspring, parents are selected from the current set of solutions. Selection in MAP-Elites was originally introduced as an unbiased operator. Individuals are randomly selected from the archive, whereby each niche can be selected with equal probability. In NSLC the selection is based on both novelty as well as local quality and takes place in the accompanying population, not in the archive.

A number of alternative scoring schemes have been introduced as a basis for selection. Pugh et al. (2015) set the probability of selecting a parent in MAP-

Elites proportionate to the novelty of the solution within the niche of the parent. The authors analyzed when this selection method makes sense by changing the alignment (correlation) between the diversity and quality metrics. The higher this alignment, the more effective the selection procedure is to fill the archive. MAP-Elites can be adapted with this method to behave more like novelty search. The selection procedure automatically reverts back to random selection as soon as the archive is filled.

Another method to improve the speed at which MAP-Elites fills the archive is curiosity-based selection. Curiosity represents the propensity of an individual to generate offspring that gets successfully added to the archive. The metric is defined by counting how often a niche produces offspring that is accepted into the archive because of novelty or better performance. The niches that produce more offspring, or which are more novel, or higher performing, are selected with a higher probability. The method by Cully and Demiris (2017) is not only able to fill the archive more quickly than random selection, but it often finds better solutions in the niches. The selection method can have a significant impact on QD performance, although random selection as a default performs well.

One of the main reasons QD performs well at not only finding diverse solutions, but oftentimes even outperforms classical evolutionary optimization algorithms, can be ascribed to the archive. In the archive, *stepping stones*, solutions that are not high-performing themselves, can lead an evolutionary path to access high-performing regions of the search space, as was shown by Meyerson and Miikkulainen (2017).

Due to the decoupling of genetic search and phenotypic niching, QD supports addition, removal and/or variation of constraints, objectives and variable parameter bounds, fulfilling requirement C4, “Search should support addition, removal and/or variation of constraints, objectives and variable parameter bounds” (Table 2.3).

In QD, the archive becomes the central evolvable object. It can be seen as a lattice, whose points move through genetic space as elites are replaced over time. The QD algorithm used in this work is MAP-Elites, which is explained in detail below.

MAP-Elites MAP-Elites uses a fixed, discretized n-dimensional grid (Figure 3.7), with each dimension representing one phenotypic aspect. This prevents comparisons becoming more expensive as the algorithm progresses and emphasizes the phenotypic space. The number of features, which serve as diversity measures, is limited by the fact that the niching space becomes exponentially larger when adding dimensions. Contrary to NSLC, in MAP-Elites, parents are selected randomly from the archive. There is no distinction between the population and the archive, so it is updated in-place.

Algorithm 2 MAP-Elites (see Mouret and Clune (2015)). The population (X) in the input is usually initialized using a space-filling sequence or random distribution.

1: **function** MAP-ELITES($\mathcal{X}, d(), f(), \sigma$)
2: $\mathcal{D} \leftarrow d(\mathcal{X})$
3: $\mathbf{f} \leftarrow f(\mathcal{X})$
4: $\mathcal{A} \leftarrow (\mathcal{X}, \mathbf{f}, \mathcal{D})$
5: **while** $gens < maxGen$ **do**
6: $\mathcal{X} \leftarrow$ RANDOMSELECT($dim(\mathcal{X})$)
7: $\mathcal{X} \leftarrow$ MUTATE($\mathcal{X}, \sigma$)
8: $\mathcal{D} \leftarrow d(\mathcal{X})$
9: $\mathbf{F} \leftarrow f(\mathcal{X})$
10: $\mathcal{A} \leftarrow$ REPLACE($\mathcal{A}, \mathcal{X}, \mathbf{f}, \mathcal{D}$)
11: **end while**
12: **return** $\mathcal{A}$
13: **end function**

MAP-Elites is shown in Alg. 2. It receives the following inputs: the initial population's genomes $\mathcal{X}$, a descriptor function $d()$ that defines which niche an individual is assigned to, a fitness function $f()$ to determine the quality of an individual, and σ which determines the size of mutations. The archive (which is called 'map' in MAP-Elites) is filled with the initial solution set (line 4), after which an iterative loop is executed for a number $maxGen$ of generations. First, a random set of individuals is selected from the map (line 6), which can either be performed with a uniform random distribution or a space-filling sequence. Individuals are mutated (line 7, see Alg. 3), usually with a normal distribution with standard deviation σ.

The phenotypic descriptors and fitness values are updated for the perturbed individuals, which are then placed into the archive $\mathcal{A}$ using Alg. 4.

Algorithm 3 Mutation.

```
1: function MUTATE(𝒳, σ)
2:     for i = 1 to |𝒳| do
3:         for j = 1 to n do
4:             x_i^j ← x_i^j + 𝒩(0, σ)
5:         end for
6:     end for
7:     return 𝒳
8: end function
```

Algorithm 4 Replacement.

```
1: function REPLACE(𝒜, 𝒳, f, 𝒟)
2:     for i = 1 to |𝒟| do
3:         n ← DISCRETIZE(𝒟_i), 𝒟_i ∈ 𝒟          ▷ 𝒟_i is assigned to niche ID n.
4:         if exists(f(n)) && f(n) < f(i) then   ▷ Replace if i improves niche.
5:             𝒜(n) ← [x_i, f_i, 𝒟_i]
6:         end if
7:     end for
8:     return 𝒜
9: end function
```

Phenotypic descriptors are discretized to determine which niche in the grid-shaped archive they get assigned to. Then, the fitness value of a candidate is compared to the fitness value of the current member of the niche (if the niche is not empty). If the candidate improves the archive, it replaces (or gets assigned to) the niche. After *maxGen* generations, the archive is returned to the user.

3.2.4 Conclusions

The evidence on research question I ("Are solutions best compared using their genomes or their expressed phenotype or behavior?") that was provided in the previous section shows that the (extended) phenotype is a research object that is worthwhile to investigate. Research in evolutionary computation has indeed focused on the phenotype, specifically QD algorithms. QD algorithms fulfill requirements B1 ("Solutions should be categorized") and C5 ("Solution similarity should be based on their functional expression and presented along meaningful

feature dimensions").

Solutions are best compared using their expressed phenotype or behavior, instead of their genomes, due to the effects of neutrality and sensitivity in (evolutionary) representations. Requirement B2 ("Knowledge should be organized hierarchically") can be fulfilled by using hierarchical archives. Features can usually be easily switched, when their calculation can be done quickly, which partially fulfills requirement B3 ("User can filter knowledge base dynamically"). Requirement B3 can also be fulfilled by changing feature dimensions in QD.

In the next section, the main paradigms from multi-solution optimization are compared.

3.3 Comparing Divergent Search Methods

In this section, various techniques in multi-solution optimization are compared to answer research questions I ("Are solutions best compared using their genomes or their expressed phenotype or behavior?") and II ("What multi-solution optimization method produces the highest phenotypic diversity?"). The three main paradigms in multi-solution, divergent search methods, MOO, MMO and QD, are evaluated in terms of the diversity and performance of the solution sets they create. A new niching archive is introduced that allows comparing genetic and phenotypic diversity. State of the art diversity metrics are used in a new problem domain to evaluate all paradigms after which recommendations are made when to use which approach.

A number of survey and analysis articles have appeared in the last decade. In Basto-Fernandes et al. (2013) a taxonomy for diversity in optimization was introduced. Wessing and Preuss (2016) investigated how genetically diverse solution sets in MOO are found and showed that quality indicators used in MOO can be applied to MMO. Vassiliades et al. (2017a) compared two algorithms from MMO to two QD algorithms in a robotics task, showing that clearing performance can be comparable to that of QD. Finally, Li and Yao (2019) discussed 100 solution set quality indicators in MOO and Tian et al. (2019) discussed diversity indicators for MOO. A published scientific comparison between all three paradigms did not previously exist.

The algorithms that provide simple interpretations of diversity in objective space (MOO), genetic space (MMO) and phenotypic space (QD) are selected for this comparison, which answers the question which one of these interpretations of diversity produces a larger diversity of shapes. In this section, which is based on work by Hagg et al. (2020), a new, simple archive is introduced that allows comparing genetic and phenotypic niching with the same algorithmic mechanism.

3.3.1 Niching with Voronoi Tessellation

To remove variations in the search dynamics when comparing different algorithms, a simplified niching variant of NSLC and CVT-Elites is introduced. The Voronoi-Elites (VE) algorithm is illustrated in Fig. 3.10. Selection pressure is applied based on artifact similarity. In effect, VE tries to minimize the variation of distances between artifacts in the (unbounded) archive. The Voronoi niches are not predefined, niche generators do not have to coincide with the centroids and boundaries of the archive are not fixed, as opposed to the CVT-Elites algorithm by Vassiliades et al. (2017b). VE is explained in Alg. 5 and 6.

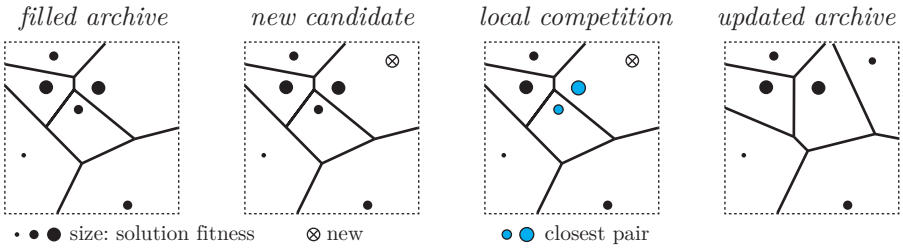

Figure 3.10: Updating the Voronoi archive. The VE formulation allows for a fixed number of archive members, independent of its dimensionality, which makes runtimes more predictable. In this example, the maximum number of artifacts in the archive is set to six. When a new candidate artifact is added, the pair of closest artifacts is compared. The worse of the two is rejected or removed from the archive and the artifact with higher fitness is kept or added to the archive. The borders between niches are drawn here to illustrate the range of influence of each item in the archive and how it changes after an update.

In the main procedure, the initial population of artifacts is created from a quasi-random, space-filling Sobol sequence (see Sobol (1967)) in line 2. The population is evaluated and its fitness values $\mathbf{f}$ and feature descriptors $\mathcal{D}$ determined based on

Algorithm 5 Voronoi-Elites.

```
1:  function VORONOI-ELITES(dim)
2:      𝒳 ← SOBOL(dim)
3:      f, 𝒟 ← EVALUATE(𝒳)
4:      𝒜 ← 𝒜 ∪ (𝒳, f, 𝒟)
5:      for 1 to numGen do
6:          𝒳 ← TOURNAMENT(𝒳, f)
7:          𝒳 ← MUTATE(𝒳, σ)
8:          f, 𝒟 ← EVALUATE(𝒳)
9:          𝒜 ← 𝒜 ∪ (𝒳, f, 𝒟)
10:         while |𝒜| > maxSize do
11:             𝒟 ← dist(𝒟, 𝒟) ▷ 𝒟 contains pair-wise distances between rows in 𝒟
12:             row, col ← argmin(𝒟)        ▷ Ignore diagonal in distance matrix 𝒟
13:             if f_row > f_col, with f_i ∈ f then
14:                 r ← col
15:             else
16:                 r ← row
17:             end if
18:             𝒜 ← 𝒜 \ {(x_r, f_r, 𝒟_r)}
19:         end while
20:     end for
21:     return 𝒜
22: end function
```

fitness and feature metrics, which have to be predefined by the user. The archive $\mathcal{A}$ is filled with tuples of genomes $\mathcal{X}$, their fitness values and feature descriptors. For *numGen* generations, the following pattern is applied. From the population, a set of parents is selected (with the same size as the population) using tournament selection. The function in Alg. 6 compares the fitness values of random parent pairs and selects the best performing parent. Then, the population is mutated by adding a normally distributed random value with a preset σ, shown in Alg. 3 (the same operator as used in MAP-Elites). The new population is evaluated to determine the new fitness and feature descriptor values. The archive $\mathcal{A}$ accepts all new solutions until the maximum number of archive niches is surpassed. The new population is assigned to the archive. The pair of elites that are phenotypically closest to each other are compared, removing the worst-performing from the archive, as shown in lines 10–19. By exerting selection pressure on the closest solutions, VE tries to

equalize the distances between individuals.

Algorithm 6 Tournament Selection.

```
1: function TOURNAMENT(𝒳, f)
2:     𝒲 = {}
3:     for 1 to |𝒳| do
4:         a ← random([1 : |𝒳|]), b ← random([1 : |𝒳|])
5:         if f_a > f_b, with f_i ∈ f then
6:             𝒲 ← 𝒲 ∪ f_a
7:         else
8:             𝒲 ← 𝒲 ∪ f_b
9:         end if
10:    end for
11:    return 𝒲
12: end function
```

Notice that, when genetic parameters are used as archive dimensions, VE behaves like an MMO algorithm by performing niching in genetic space. When phenotypic descriptors are used, VE behaves like a QD algorithm. With MAP-Elites this would theoretically be possible as well, yet the VE formulation allows for a fixed number of archive members, independent of the archive's dimensionality. This makes convergence times (time until an archive is filled) more predictable.

3.3.2 Diversity Metrics

A large number of metrics is available to compare the diversity of a set of solutions in optimization. Only those are considered that do not depend on precise domain knowledge, because no knowledge about actual local optima is available in real world applications. Three commonly used distance-based metrics are selected to evaluate the experiments.

The sum of distances to nearest neighbor (SDNN) measures the size of a solution set as well as the dispersion between members $\mathbf{s}$ of that set (see Wessing and Preuss (2016)). The metric is based on the total minimum dissimilarity between individual solutions, the nearest neighbors, in the set $\mathcal{X}$ (Eq. 3.1 and 3.2). Dissimilarity is measured using a distance metric, like the Euclidean norm.

$$d(\mathbf{s}, \mathcal{X}) = \min_{\mathbf{s}_i \in \mathcal{X}} (dissimilarity(\mathbf{s}, \mathbf{s}_i)) \tag{3.1}$$

$$SDNN(\mathcal{X}) = \sum_{\mathbf{s}_j \in X} (d(\mathbf{s}_j, \mathcal{X})) \tag{3.2}$$

Solow-Polasky Diversity (SPD), introduced by Solow and Polasky (1994), measures the effective number of species by using pairwise distances between the species in the set $\mathcal{X}$. To compute the metric, a matrix $\mathbf{R}$ has to be constructed, with entries $r_{ij} = exp(-\theta d(\mathbf{s}_i, \mathbf{s}_j))$. SPD is then calculated according to Eq. 3.3).

$$SPD(X) = \mathbf{1}^T \mathbf{R}^{-1} \mathbf{1} \tag{3.3}$$

If the solutions are similar with respect to each other, SPD tends to one. The sensitive parameter θ, which determines how fast a population tends to higher values of SPD with increasing distance, needs to be parameterized for every domain. It is set to one for genetic distances and to 100 for phenotypic distances in this section.

Pure Diversity (PD) is used in high-dimensional many-objective optimization (see Wang et al. (2016)). The metric is based on the minimum dissimilarity between individual solutions in the set X (Eq. 3.1). Eq. 3.4 shows that the metric is defined recursively. The PD value of a set X is equal to the maximum of the sum of its value on all but one of the members and the minimum distance of that member to the set. In the base case of the recursion, the set X contains only two members and the recursion stops.

$$PD(X) = \max_{s_i \in X} (PD(X \setminus \{s_i\}) + d(s_i, X \setminus \{s_i\})) \tag{3.4}$$

It does not have parameters, which makes it robust, and depends on a dissimilarity measure, which is usually the $L_{0.1}$-norm to deal with higher dimensional similarity spaces.

Publications in the field of QD have focused on a small number of metrics that do not seem to be very usable and certainly not independent of the used algorithms. The total fitness is used directly or through the *QD-score* by Pugh et al. (2015), which calculates the total fitness of all filled niches in a phenotypic archive. To

achieve this, the solutions from a non-QD algorithm are projected into a fixed phenotypic niching space. This score is domain-dependent and does not allow comparing QD algorithms that have different archiving methods. A comparison between archives created from different features introduces a bias towards one of the archives. The *collection size* indicates the proportion of the niching space that is covered by the collection, but again can only be used on a reference archive, as was done by Cully and Demiris (2017).

Archive-dependent metrics do not generalize well and introduce biases. Therefore, in this work only distance-based diversity metrics are used. The high dimensionality of phenotypic spaces is taken into account by using appropriate distance norms.

3.3.3 Evaluation

A simple polygon optimization domain is defined, in which the objective is to create symmetrical polygon shapes. Shape is an important basic component of design in art, architecture and engineering. On the one hand, shapes can serve as a medium for artistic expression which can also carry semantic meaning (e.g. the shape of a letter of a typeface) and on the other hand can help visualize an object during the design process before it is built or produced. Since the phenotypes can be expressed as binary bitmap images (Figs. 3.11b and 3.11c, resolution of 64x64 pixels), the Hamming (1950) distance is used in the diversity metrics to circumvent the problem of high dimensionality.

Three phenotypic features describing the polygons are defined that can be used either as criteria or as phenotypic features (Fig. 3.11d): the area of the polygon A, its circumference l and point symmetry P through the center. The polygon is sampled at $n = 1000$ equidistant locations on the polygon circumference. The sum of distances of all $n/2$ opposing sampling locations serves as a symmetry error E_s. The error on such a set of opposing samples $\mathbf{X}$ is calculated as shown in Eq. 3.5.

$$f_P(\mathbf{X}) = \frac{1}{1 + E_s(\mathbf{X})}, E_s(\mathbf{X}) = \sum_{j=1}^{n/2} ||\mathbf{x}_j, \mathbf{x}_{j+n/2}||, \text{ with } \mathbf{x}_i \in \mathbf{X} \tag{3.5}$$

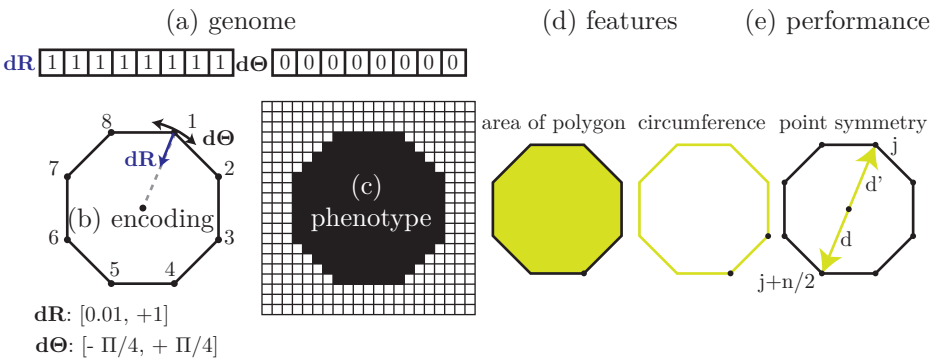

Figure 3.11: Free-form encoding of polygons. The genome (a) consists of 16 parameters that define angular and radial deformations (b). The range of possible gene values differs for the **dR** and **d**θ genes. The phenotype is considered to be the pixel representation of the polygon (c). Shown is a 20x20 phenotype, although 64x64 pixels are used throughout the experiments. Features and optimization criterion of the polygon domain are shown in (d) and (e).

For every opposing pair, the difference in distance to the center location is calculated. When the sum of these distance differences is zero, the shape is defined as fully symmetric.

Genetic or Phenotypic Diversity The Voronoi tessellation used in VE makes it easy to compare archives of different dimensionality by fixing the number of niches. VE is applied as an MMO algorithm, performing niching in 16-dimensional genetic space, and as a QD algorithm with a two-dimensional phenotypic space. The number of niches is increased to determine when differences between genetic and phenotypic VE appear (Fig. 3.12).

At 25 solutions, the approaches produce about the same diversity, but genetic VE finds higher quality solutions. As the number of niches is increased, based on where niching is performed (genetic or phenotypic space), the diversity in that space becomes higher. Phenotypic VE beats genetic VE in terms of phenotypic diversity, which gives us evidence that the answer to research question II should tend towards preferring phenotypic niching over genetic niching. At the same time, the average fitness values of genetic VE are higher than those of phenotypic VE, although the difference gets lower towards 400 solutions.

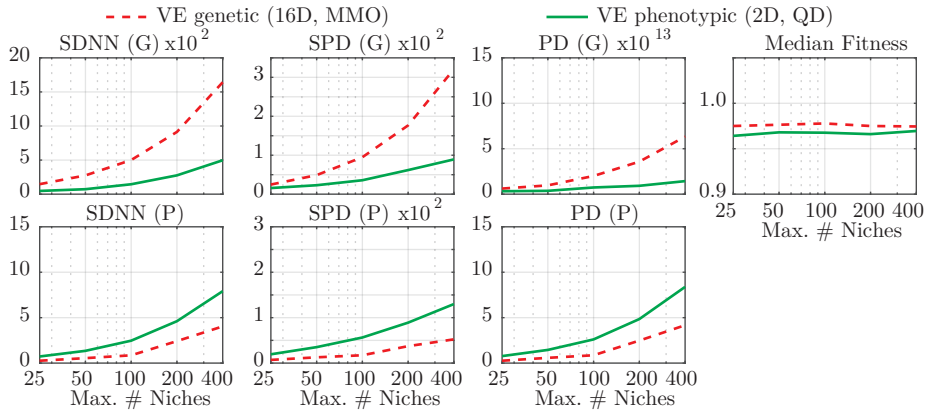

Figure 3.12: VE performed in 16D genetic and two-dimensional phenotypic space. Top: genetic diversity (SDNN, SPD, and PD) and median fitness, bottom: phenotypic diversity. The number of niches/solutions is increased (x-axis).

Comparison of Results The main question to answer is which paradigm provides the highest phenotypic diversity of shapes. VE, RLS (see Section 3.2.2) and NSGA-II (see Section 3.2.1) are compared in multiple experiments. Throughout these experiments the number of function evaluations and solutions is fixed. Five replicates per configuration are produced. In NSGA-II the features are used as optimization criteria, maximizing A and minimizing l. The true Pareto set consists of circles with varying sizes. The number of generations is set to 1024 and mutation strength to 10% of the parameter range. The probability of crossover for NSGA-II is 90% and the probability of mutation is $\frac{1}{dof} = 0.0625\%$, with $dof = 16$ degrees of freedom. VE's archive size is varied throughout the experiments. The number of children (64) and population size is set to the same value. RLS uses as many restarts as the size of the VE archive, and is restricted to the bounds of the genome's parameter ranges. Broyden-Fletcher-Goldfarb-Shanno (BFGS) (see Broyden (1970), Fletcher (1970), Goldfarb (1970), and Shanno (1970)) is used as a local search method. The initial solution set for VE and NSGA-II is created with a Sobol sequence – the initial RLS solution is in the center of the parameter range but RLS' space-filling character assures a good search space coverage.

Phenotypic VE is compared to NSGA-II and RLS. When the genes dr, which encode the radial deviation, are bounded between 0 and 1, and $d\theta$, which encode the angular deviation, are set between $+/-0.125 \times \pi$, genetic neutrality can be minimized. Neutrality is increased by expanding those bounds (Table 3.1). In

contrast to VE, the phenotypic diversity of RLS' solutions is expected to decrease as genetic neutrality increases. Since there is no mechanism to distinguish between similar shapes with different genomes, there is an increasing probability that RLS finds phenotypically similar solutions. It is expected that the solution set produced by RLS is more diverse than using NSGA-II, due to its space-filling character.

Table 3.1: Parameter settings in order of increasing genetic neutrality.

case	angular min.	angular max.	radial min.	radial max.	neutrality
A	0	1	-0.05	0.05	-
B	0	1	-0.125	0.125	+
C	-0.25	1	-0.25	0.25	++
D	-0.5	1	-0.5	0.5	+++
E	-1	1	-1	1	++++

It is expected that NSGA-II will easily find the Pareto set, which consists of circles of various sizes, maximizing the area while minimizing the length of the circumference, while QD should find a variety of shapes that can be any combination of large and small A and l. I postulate: allowing all criteria combinations, instead of using a Pareto approach, leads to higher diversity, while still approximating the Pareto set.

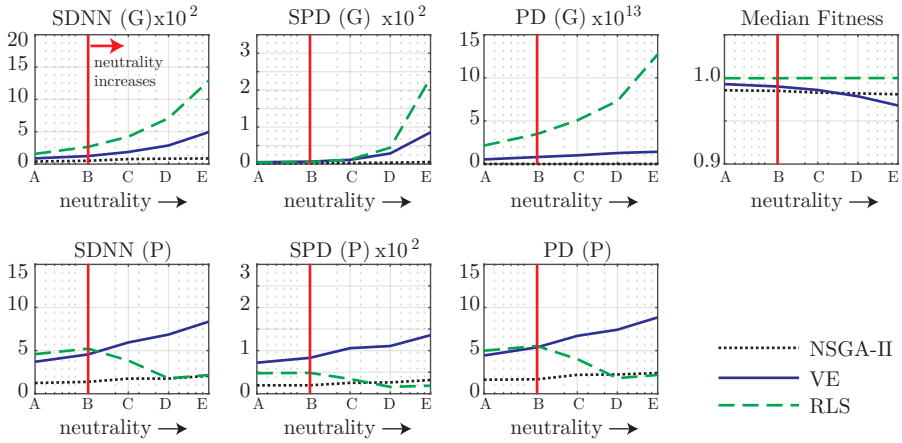

Figure 3.13: Genetic (top) and phenotypic (bottom) diversity, and median fitness. Right of red marker: neutrality increases, using parameter bounds shown in Table 3.1.

The number of solutions is set to 400. A result similar to Fig. 3.12 appears for the standard algorithms in Fig. 3.13. While phenotypic diversity of VE is highest, especially after the genetic neutrality threshold is crossed (at B), diversity of NSGA-II is lowest, as is expected for this setup. Although the diversity of VE is higher than that of RLS, the latter's solutions are all maximally symmetric (see fitness plots), for all settings of the encoding. This makes RLS more appropriate when quality is more important than diversity. The results give us more evidence to answer research question II. (Phenotypic) VE indeed provides a more diverse solution set than any of the other paradigms.

The ground truth Pareto set can be calculated a priori, as we know that circular shapes maximize area while minimizing circumference. The members of the Pareto set adhere to the following genome: $r_1 = r_2 = \cdots = r_8$ and $\theta_1 = \theta_2 = \cdots = \theta_8$. To create 100 shapes from the ground truth Pareto set, ten equidistant values for r and θ are combined.

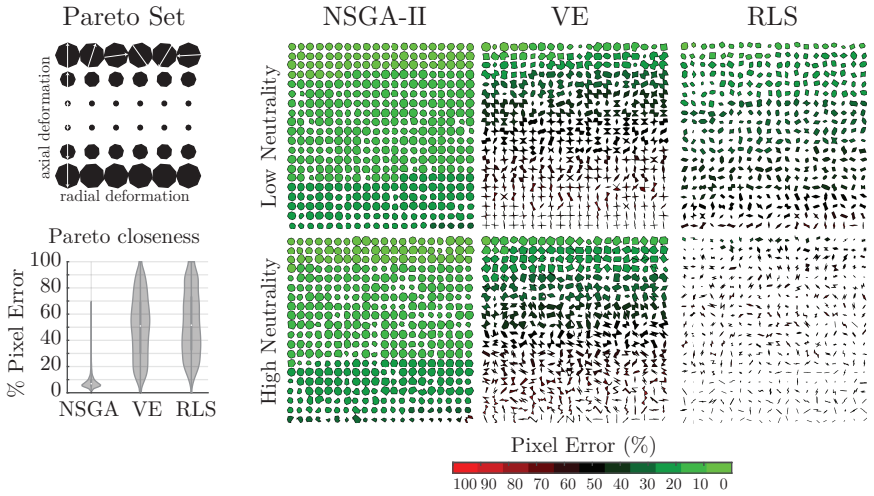

Figure 3.14: The ground truth Pareto set is shown over the entire parameter range, with negative as well as positive values for the radial deformation. Bottom left: closeness to Pareto set, measured as pixel errors. The six figures on the right show example solution sets for low and high neutrality.

Part of the ground truth Pareto set is shown in Fig. 3.14. The distance to the Pareto set is determined in phenotypic space, by measuring the smallest pixel error, the sum of pixel-wise differences, between a solution and the Pareto set. We see

that a number of solutions in VE and RLS are close to the Pareto set (Fig. 3.14 bottom left). Example results with low and high neutrality are shown on the right. Solutions that are close to the Pareto set are shown in the brightest green color. VE again seems to be more robust w.r.t. genetic neutrality, as it finds more solutions close to the Pareto set in high-neutrality domains (bottom row) than RLS. This provides us with even more evidence that QD should be the preferred over MOO and MMO methods because we aim to generate a phenotypically diverse solution set.

3.3.4 Conclusions

This section gave evidence for answers to research questions I ("Are solutions best compared using their genomes or their expressed phenotype or behavior?") and II ("What multi-solution optimization method produces the highest phenotypic diversity?"), fulfilling requirements B4 ("Process should provide diverse perspectives on existing knowledge"), C1 ("Search should efficiently generate a diverse set of novel, functional solutions"), C3 ("Search should identify regions of design feasibility and optimality"), and C4 ("Search should support addition, removal and/or variation of constraints, objectives and variable parameter bounds").

The highest phenotypic diversity is reached by comparing solutions based on their phenotypes, using a QD algorithm instead of MMO or MOO. QD is less sensitive to genetic neutrality than MMO. While the diversity of solution sets of QD and RLS is higher than that of MOO, they both find some solutions close to the ground truth Pareto set. The section included the introduction of the VE algorithm, which allows the comparison between genetic and phenotypic niching.

The next section discusses the phenotypic features that are used in QD and compares predefined features to those that are learned from data.

3.4 Phenotypic Features

Requirements A3 ("The initial solution set presented to user should need as little guidance from the user as possible") and B4 ("Process should provide diverse perspectives on existing knowledge") beg the question, whether we should use predefined features or learn them from data. There certainly is a tension between

explainable, predefined features and the need to maximize solution diversity. In this section I therefore investigate research question III ("Can we produce more diverse solution sets when learning phenotypic niching from data instead of using predefined features?").

3.4.1 Predefined Features

Phenotypic features can be viewed as the concretization of the idea of conceptual spaces, a term introduced by Gärdenfors (2004). The term formalizes concepts on the basis of a geometric structure that consists of a number of quality dimensions, basic features by which objects can be compared. The term is motivated by the notions of conceptual similarity and prototype theory.

The phenotypic features used for niching can be based on solutions' behavior, as was done by Cully et al. (2015), morphology, by Gaier et al. (2018), or other properties of the extended phenotype. For example, Cully et al. (2015) evolved an intuition of well-performing neural controllers for a hexapod robot. By expressing each robot controller, certain phenotypic traits can be measured, like the amount of time each leg touches the ground, along which we perform phenotypic niching. Each niche accepts only those controllers whose legs touch the ground in certain temporal patterns. Within that niche, the best solutions compete to survive. The resulting archive of locally well-performing behaviors provided the authors with many different walking gait strategies, which allows quick switching between strategies when necessary.

Selecting phenotypic features is very much a domain-dependent issue. Yet, we seek to minimize the amount of guidance by the user in the generative stage. Some work has been done on unsupervised and adaptive learning of features. A major breakthrough in unsupervised determination of features are *innovation engines*. The technique, introduced and applied on image generation by Nguyen et al. (2015, 2016), uses a generative model (GM) to learn phenotypic features that are 'interesting'. By training the network on 1000 image classes, its output confidence can be used to determine whether an image generated by a QD algorithm is a good representative of one of the classes. Similarly, the features used in the archive are found using a deep neural network or autoencoder in an unsupervised manner (see Cully and Demiris (2018) and Cully (2019)). Another idea is to measure how well a network is able to compress an image. The better it is able to do this, the

less interesting or novel an image is. Gaier et al. (2019) evolved indirectly encoded images. They periodically trained an autoencoder on all images in the archive thereby allowing themselves to measure a kind of novelty score represented by the image reconstruction error. Higher errors indicate more novel images. If an image is novel enough, it gets saved in the QD archive. The diversity metric is therefore not only unsupervised, but adaptive as well.

In some cases, unsupervised descriptors might help us to find phenotypic features that are hard to define in particular high-dimensional domains. This does reintroduce the issue of explainability, because there is no direct way to understand what the features actually represent. We can only perform a posteriori analysis on the deep neural networks, often only by using simple metrics, like a confidence level on the classification output of the network. Using deep neural networks to find features is not something that can be done in every domain, due to model's need of a large number of training examples. It is no coincidence that innovation engines were introduced for images, a domain in which a large amount of data generally is available.

It is important to take into account the cost of calculating the phenotypic descriptors, which has to be performed for each candidate solution. In this respect, an aspect that is based on e.g. morphology might be less expensive than one that is based on behavior in a simulation.

3.4.2 Learning Features

We might not always want or be able to define phenotypic features that are as influential as possible on diversity. Still, data-driven feature models can help us discover features in more complex domains, although issues around cost and explainability might prevent this.

Up to this point in this thesis, domain knowledge was used to construct a phenotypic niching space with VE. Intuitively, the area and circumference seem like good indicators for phenotypic differences in the used domain. But the comparison between QD and MMO is not completely fair, as the latter does not get any domain information. On the other hand, the features used in QD might not be the most diversifying.

Domain knowledge is now removed from QD and a phenotypic niching space is constructed by using a well-known dimensionality reduction (DR) technique to map the phenotypes to a latent space, as was done by Meyerson et al. (2016) and Cully (2019). This data-driven phenotypic niching approach, named Automatic Voronoi-Elites (AutoVE) and shown in Fig. 3.15, has never been applied to shape optimization.

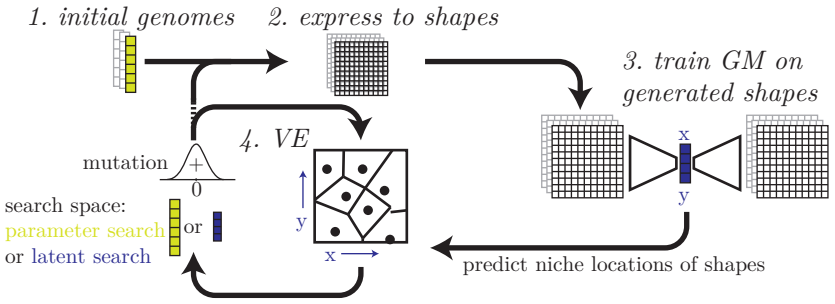

Figure 3.15: Generative model and VE combined into a generative system in two phases. First, initialization: (1) an initial random set of genomes is generated and (2) converted into shape bitmaps which are used to (3) train a GM. Second, optimization loop: (4) VE iteratively updates the archive of candidates. Two setups of this loop are compared: the VE performs search either in parameter space or in the GM's latent space.

An initial set of genomes is drawn from a space-filling Sobol sequence. The genomes are expressed into their phenotypes, which are then used to train a GM. A convolutional autoencoder (cAE), introduced by Hinton (1994), is used here (Fig. 3.16).

The bottleneck in the cAE is a compressed, latent space that assigns every phenotype to a coordinate tupel. The encoder predicts these coordinates of new shapes in the latent space, which are used as phenotypic features. VE is used to search through genetic space and to produce phenotypes that expand and improve the cAE archive.

The reasoning behind using VE as a QD algorithm is as follows. In the original MAP-Elites used by Mouret and Clune (2015), the archive consists of a fixed grid of niches, which leads to an exponential growth of niches with the number of phenotypic feature dimensions. The creators of CVT-Elites, Vassiliades, Chatzilygeroudis, Clune, and Mouret (Vassiliades et al.), dealt with this problem by predefining fixed

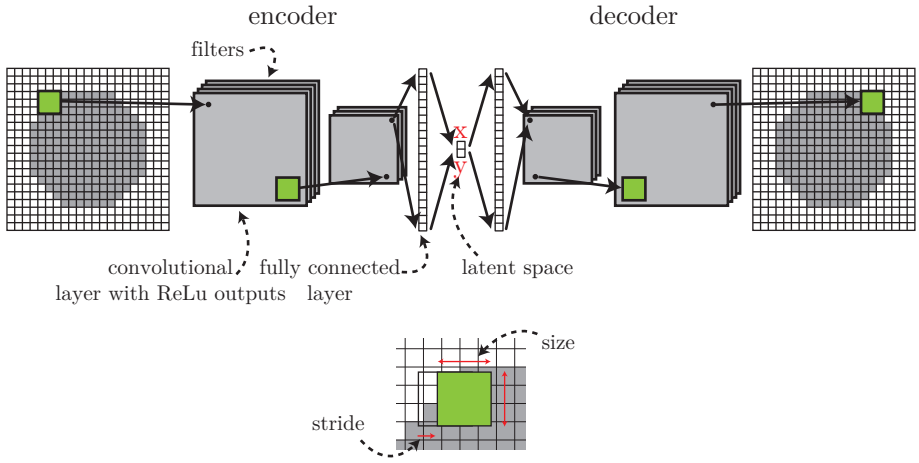

Figure 3.16: A convolutional autoencoder that compresses information, like an image of a polygon, by providing a bottle-neck in the center. The network is trained to reproduce the input at the output. The convolution operates on filters with a 3x3 size and a stride of one.

niches using a Voronoi tessellation of the phenotypic space. Due to their fixed archive, both methods tend to reduce the genetic variance of the solution set in the first iterations. Initial (random) samples tend to not cover the entire phenotypic space and thus competition is harsher, leading to the excluding of many solutions in the beginning. To maximize the number of available training samples for the cAE, the VE algorithm is therefore more appropriate. In contrast, VE, introduced in Hagg et al. (2020), does not precalculate the niches. It accepts all new artifacts until the maximum number of niches is surpassed.

The cAE is retrained with the new samples. It consists of two convolutional layers in the encoder and four transposed convolutional layers in the decoder. The filter size is set to three pixels, the stride to two pixels, and the number of filters to eight. The cAE is trained using ADAM, introduced by Kingma et al. (2015), with a learning rate of 0.001 and 350 training epochs and a mean square error loss function. Latent coordinates are normalized between zero and one.

3.4.3 Evaluation

AutoVE is compared with VE using manual features, continuing the experiment from Section 3.3.3. The number of generations (1024) is divided over two iterations for AutoVE and the number of latent dimensions is set to two (to compare with manual VE), five or ten.

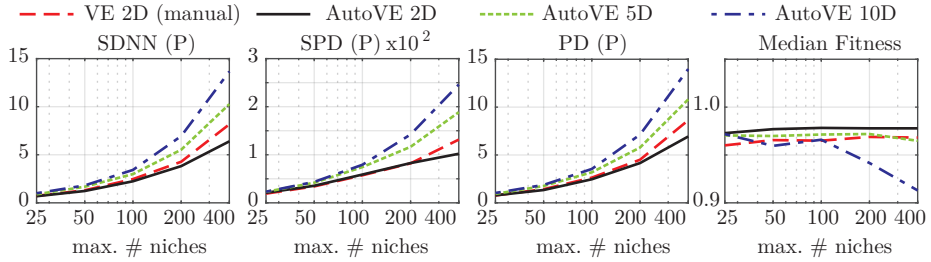

Figure 3.17: Phenotypic diversity and fitness of manually crafted features (VE) compared to using an autoencoder (AutoVE) with two, five or ten latent dimensions.

Fig. 3.17 shows that the two-dimensional manual and autoencoded phenotypic space (AutoVE 2D) produce similar diversity, whereby the quality of solutions from AutoVE 2D is higher. The higher-dimensional latent spaces increase the solution set diversity at the cost of fitness. This is to be expected, as lower-fitness optima are protected in their own niches. Finally, the diversity of higher-dimensional AutoVE is around 50% higher than that of any of the other tested methods in this thesis. Using an autoencoder produces higher diversity than manually defined features, making AutoVE a strong choice for high-diversity multi-solution optimization.

3.4.4 Conclusions

This section gave evidence for an answer to research question III ("Can we produce more diverse solution sets when learning phenotypic niching from data instead of using predefined features?"), fulfilling requirement A3 ("The initial solution set presented to user should need as little guidance from the user as possible") and indicating the approach's usefulness in fulfilling requirement B4 ("Process should provide diverse perspectives on existing knowledge").

More diverse solution sets can be produced when learning phenotypic niching from data instead of using predefined features. An autoencoder was used to discover phenotypic features in a shape optimization problem, showing that we do not need to manually predefine features to get a highly diverse solution set, which increases the evidence that QD is superior in terms of quality and diversity when compared to MOO and MMO.

It is often easy to manually define two or three phenotypic descriptors, but human imagination can run out of options quickly. Automatic discovery of phenotypic features is a more attractive option for increasing solution diversity. Real world multi-solution optimization and understanding solution diversity are important steps towards increasing the efficacy and efficiency at which engineers solve problems.

Requirement A5 ("The process should be adaptable along the resonance-dissonance dimension") is not fulfilled yet. The resonance-dissonance trade-off is not configurable, although this can be accomplished by using e.g. the reconstruction error of the cAE as a feature. This will be evaluated in the next section.

3.5 Limitations of Generative Models

The expectation of Miikkulainen (2020) that the combination of evolutionary computation with deep learning techniques can lead to the next big jump in AI development, machine creativity, begs the question how powerful GM actually are. In this section, research question IV is answered: "What are the limitations of generative models in terms of the possible diversity of the solutions they create?". In order to do this, a comparison is made between multi-solution evolutionary search in the parameter space of a generative system and the search in the latent space of a variational autoencoder (VAE), which were introduced by Kingma and Welling (2014), that was trained with samples from the same system. A generative system is described that produces two-dimensional shapes through the manipulation of eight control points. The system is evaluated in multiple scenarios, including one where it is forced to extrapolate away from solutions that are describable by its latent feature dimensions. This analysis is necessary to estimate, what effect requirement A5 (a resonance-dissonance trade-off should

be configurable) would have on such a system. This is based on work published in Hagg et al. (2021).

Deep generative models such as VAE find application in the context of optimization for their ability to extract patterns from raw data, learn meaningful representations of artifacts and accurately produce more samples with the same properties. Disentangled representation learning can furthermore equip a model's latent space with linearly separated factors of variation, revealing the underlying parameters of a generative process (see Burgess et al. (2017)).

The feature compression networks can be used to compute descriptors for QD algorithms. Defining similar descriptors by hand is a non-trivial task which requires expertise and intuition and, depending on the domain, often cannot compete with an automated solution. Evidence for this was already shown in Section 3.4. While the advantage of learning from data lies in the recognition of complex patterns, the expressiveness of the resulting GM is entirely dependent on the quality and representativeness of the data samples provided. This is especially critical when relying on such a model to produce novel examples and diverse sets of outputs. In fact, artists who employ a generative adversarial network (GAN) in their work often use a variety of strategies to actively diverge from the intended purpose of these models and to produce outputs significantly different from the original data (see Berns and Colton (2020)).

The evaluation is relevant in two scenarios: 1) when the generative process is manually defined but a VAE is used to compare artifacts (e.g. distance/similarity estimation), and 2) when only data is available and the underlying patterns are unknown or too difficult to extract manually and have to be learned by an appropriate model.

3.5.1 Variational Autoencoders

VAEs are a likelihood-based method for generative modeling in deep learning. They follow the standard architecture of an autoencoder: a compressing encoder network, mapping data samples to latent space, and a decoder network which is trained to generate the original samples from the corresponding latent codes (Fig. 3.18). A VAE can generate new samples by interpolating between training locations in the latent space. While common autoencoders draw from an unrestricted range of

latent code values, the latent space of a VAE is typically modelled to be a centered isotropic multi-variate Gaussian ($\mathcal{N}(0, I)$).

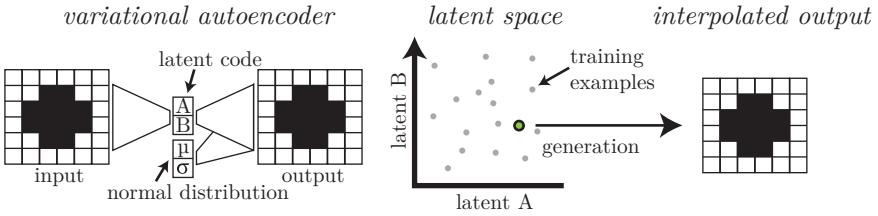

Figure 3.18: Left: variational autoencoder, center: sampling from latent space, right: interpolated output.

The VAE training objective is to optimize a lower bound on the log-likelihood of the data. A beta-annealing variant of the loss term is used to improve disentanglement with improved reconstruction (see Burgess et al. (2017)). This variant of the evidence lower bound (ELBO) calculates the loss function over the predicted output x and the ground truth $\hat{x}$ as follows:

$$ELBO(x, \hat{x}) = C(x, \hat{x}) + \beta \cdot (KL(x, \mathcal{N}(0, 1)) - \gamma) \tag{3.6}$$

Eq. 3.6 consists of a reconstruction loss term, in this case the binary cross-entropy $C(\cdot, \cdot)$, and a regularization term, which penalizes a latent distribution that is not similar to a normal distribution with $\mu = 0$ and $\sigma = 1$. The regularization term is calculated using Kullback-Leibler divergence (KL) and scaled by the parameter β. The annealing factor γ is increased from zero to five during training to focus on improving the distribution in latent space in the beginning of the training and then gradually improve the reconstruction error. The internal latent space of a converged model provides meaningful representations in which distances between data points correspond to their phenotypic similarity. The VAE's internal representation is used to estimate the similarity of artifacts.

Previous work employed autoencoders for dimensionality reduction and the encoding of behavioral descriptors in a control task (see Cully (2019)). In a robotics domain, the approach allows robots to autonomously discover the range of their capabilities, without prior knowledge. GM were used to distinguish parameterized representations in shape optimization (in Section 3.4). They have also been employed to automatically learn an encoding during optimization, using them

as a variational operator (see Gaier et al. (2020)). Other GMs, like GANs, have been used in latent variable evolution by Bontrager et al. (2018) to generate levels for the video games Super Mario Bros. (see Volz et al. (2018)) and Doom (see Giacomello et al. (2019)). A model's latent space is searched with an evolutionary algorithm for instances that optimize for desired properties such as the layout or difficulty of a level. While some authors view the generated levels as novel, none have studied exactly how novel or diverse of an output such a system can produce. The first attempt at such a comparison was made in Section 3.4, but here the comparison is made specifically between using a VAE as a generator and as a method to perform niching in a QD algorithm.

3.5.2 Study Setup

When a VAE is used to generate artifacts, the diversity of its output is bound by the expressiveness of its latent space. The objective of this section is to analyze the generative capabilities of a VAE's latent space and give empirical evidence for its limitations.

This section outlines the details of the study's subject matter, the generation of two-dimensional shapes. It lists the general configurations of the VAE and VE algorithm. Specific settings for experiments can be found in the experimental setups below. The section explains how the two methods are combined to build two versions of a generative system, one using VAE as a search space and one using VAE only to compare solutions and provide a niching space for VE.

Shape Generation For this study, a focus is put on the generation of two-dimensional shapes, an extension of the domain that was previously defined in Section 3.3.3. The shapes in the extended domain are similar to a data set which has been proposed for the evaluation for the quality of disentangled representations (see Higgins et al. (2016)). The setup of the shape generating system in the context of its later use with the VE algorithm is explained here. The shapes are again generated by connecting eight control points which can be freely placed in a two-dimensional space. Each control point is defined by two parameters, the radial (dr) and angular deviation ($d\theta$) from a central reference point, resulting in a total of 16 parameters (Fig. 3.19b). These 16 parameters serve as genomes (Fig. 3.19a),

encoding the properties of each individual and defining the parameter space of the VE algorithm.

Key differences to the last domain are the final phenotype and the symmetry calculation. To form a final smooth outline, the points are connected by locally interpolating splines, as introduced by Catmull and Rom (1974), and Fig. 3.19c). A discretization step renders this smooth shape onto a square grid resulting in a bitmap of 64×64 pixels (Fig. 3.19d).

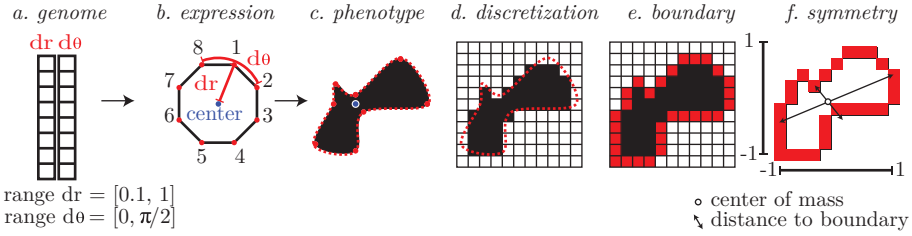

Figure 3.19: Generation of shapes: from (a) 16 genomes to (b) eight control points, freely placed in two-dimensional space, to (c) a smooth interpolated spline and (d) a final bitmap rendering. Quality evaluation: (e) the boundary of the shape is determined and (f) the shapeâĂŹs symmetry is measured from its center of mass.

Fitness As a simple objective and fitness criterion, an adaptation of the point symmetry metric that was used in Section 3.3.3 is used. To determine an artifact's quality, first, the boundary of the artifact is determined (Fig. 3.19e). Second, the coordinates of the boundary pixels are normalized to a range of minus one to one in order to remove any influence of the shape's size (Fig. 3.19f). Third, the center of mass of the boundary is determined to serve as the center of point symmetry. Fourth, the distances to the center of pixels opposite of each other w.r.t. the center of mass are compared. Finally, fitness is calculated as the symmetry error E_s, the sum of Euclidean distances of all $n/2$ opposing sampling locations to the center (see Eq. 3.5). A maximally symmetric shape is one for which this sum equals zero.

VAE Configuration A convolutional variational autoencoder (cVAE) with beta-annealing loss term (see Burgess et al. (2017)) and its decoder is used as a mapping network from latent codes to phenotype bitmaps (see Fig. 3.20). The

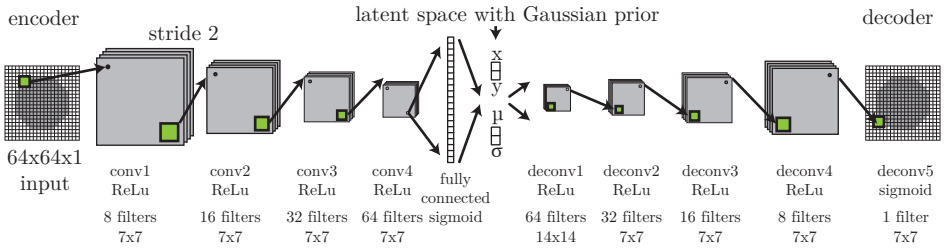

Figure 3.20: Architecture of convolutional variational autoencoder.

model's encoder network is made up of four downscaling blocks, each consisting of a convolution layer (8, 16, 32 and 64 filters respectively; kernel size 7×7; stride two) followed by a rectified linear unit (ReLU) activation function. The set of blocks is followed by a final fully-connected layer. The decoder network inversely maps from the latent space to bitmaps through five transposed convolution layers, which have 64, 32, 16, 8 and 1 filters respectively, kernel size 7×7 and stride two, except for the first layer which has a kernel size of 14×14. The last layer is responsible for outputting the correct size (64×64 pixels). The weights of both networks are initialized with the Glorot initialization scheme Glorot and Bengio (2010). The regularization term scaling factor β was set to four and the annealing factor γ was increased from zero to five over the course of the training, to have the training focus on improving the distribution in latent space in the beginning of the training, and improving the reconstruction error later in the process. Each model was optimized with the Adam optimizer, which was introduced by Kingma et al. (2015), with a learning rate $\mu = 0.001$ and a batch size of 128.

VE Configuration VE starts with an initial set of samples, generated from a Sobol sequence (see Sobol (1967)) in parameter space. Sobol sequences are quasi-random and space-filling. They decrease the variance in the experiments but ensure that the sampling is similar to a uniform random distribution and easily reproduced. In all experiments, VE runs for 1024 generations, producing 32 children per generation. Children are produced by adding a small mutation vector, drawn from a normal distribution centered around zero with $\sigma = 0.1$, to selected parent individuals. The selection is drawn at random from the archive. The number of artifacts in the archive remains constant, identical to the initial population size, over the entire experiment.

AutoVE with a Convolutional VAE The GM in AutoVE, which was introduced in Section 3.4.2, is simply replaced with a convolutional VAE to form a generative system with the objective to produce point symmetric two-dimensional shapes. The full generative process is illustrated in Fig. 3.15 and can be separated into two phases: 1) initialization and 2) an evolutionary optimization loop. At initialization time, a set of random genomes is drawn and translated into bitmaps, their phenotypic counterpart. The VAE is trained to convergence on this set of bitmap data. The learned latent space is then used in the following evolutionary process and the model's encoder and decoder networks serve as mapping functions between the phenotypic bitmap representations and the model's latent representations and vice versa.

In the evolutionary optimization loop, the VE algorithm iteratively updates the archive and tries to increase the diversity as well as the quality of the archive through local competition. To compare two candidates to each other, it relies on the VAE's lower-dimensional latent representations, which preserve semantically meaningful distances. This optimization process is performed in two different search spaces for the central comparison in this study: 1) parameter space (the explicit genome encoding) and 2) the VAE's latent space (the learned representation). This allows evaluating the expressiveness of a VAE's latent space and its capability to generate a diverse set of artifacts in comparison to the full space of possibilities which is reflected by the 16 predefined genetic parameters. The performance of the two approaches is measured in terms of the produced set's diversity. This setup permits studying the limitations of the latent space of a VAE and comparing it to the baseline diversity of searching for candidate solutions over the possible parameters.

This setup can easily be adapted beyond the binary case to gray-scale images represented by floating point values and to images with multiple channels. It further might be extendable to accommodate three-dimensional volumes. As is common in evolutionary computation, the difficulty may lie in mathematically expressing the appropriate fitness functions and measures.

3.5.3 Experiments

It is commonly assumed that a GM, like a VAE, has good interpolative and reasonable extrapolative capabilities, which makes itslatent space a potentially

appealing search space. But how well a search in this space performs in terms of generating a diverse output has not yet been adequately investigated. This section provides a first evaluation and hopes to contribute to a better understanding of GM latent spaces.

In the setup of the generative system the latent space of a VAE is used to search for and generate two-dimensional shapes in the form of square bitmaps. The output diversity of this process is compared to the baseline diversity of a search performed on the explicit genome encoding (parameter space). The pure diversity (PD) metric is used to measure diversity within a set of artifacts (see Section 3.3.2). By calculating PD on a set of bitmaps, diversity can be measured directly, independent of the representation in parameter space or the VAE's latent space.

In this context, insight is gained into two questions: 1) how accurately can a VAE represent a variety of shapes, that is to say how useful are its latent representations, and 2) how well can a VAE generate unknown shapes?

All data sets in the first experiment consist of samples which have been produced by varying two generating factors: scale and rotation (Fig. 3.21). In two experiments, a series of corresponding tasks is evaluated:

(a) With a complete set of samples as a baseline data set the standard reconstruction error of the model is evaluated in order to determine the general quality of latent representations.

(b) In the recombination task, a subset of artifacts in the center of the ranges of values of both generating factors is left out, leaving sufficient examples at either end of the ranges.

(c) In the interpolation task, the left-out subset of artifacts covers the complete range of one of the two generating factors, while some examples at the end of the range remain for the other.

(d) The extrapolation task consists of omitted samples at one end of values of one factor of variation, which affects the complete range of the other factor.

(e) The expansion task focuses on generating artifacts beyond the two given generating factors from the complete data set.

The VAE is expected to perform reasonably well in (b) recombining, (c) interpolating between and (d) extrapolating beyond the available variations to reproduce

the samples missing from the training data. In the expansion task (e), the VAE's latent space is expected to only produce artifacts of poor quality outside of the generating factors present in the training data.

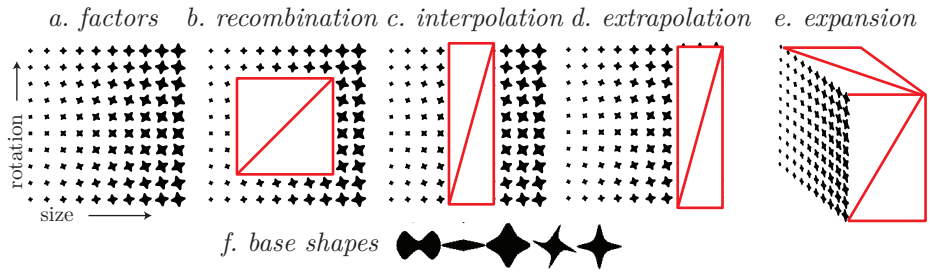

Figure 3.21: (a) generative factors used to create data sets; (b–e) four tasks on which to compare the performance of latent space search with parameter search, the red rectangles indicate artifacts that either have been left out of a data set (b, c, d) or are not available (e); (f) all base shapes used in this experiment.

Recombination, Interpolation and Extrapolation One baseline VAE is trained on the complete shape set (Fig. 3.21a) (256 shapes, scaled by factors of 0.1 to 1.0 and rotated by zero to $\frac{\pi}{2}$ in 16 steps each) and three additional models, each one on the data set of one special task (b–d) with held-out samples. The VAEs are trained for 3000 epochs, after which the models with the lowest validation error are chosen. The validation error is calculated on 10% of the input data.

To determine whether the VAE can correctly reproduce, and thus properly represent, the given shape, the models' reconstruction error is measured. For the baseline model this is done over the complete data set. For the task models (b–d) the reconstruction error is calculated only on the held-out examples. The error is defined as the Hamming distance between an input bitmap and a generated bitmap, normalized by the total number of pixels. A high reconstruction error would indicate that the model cannot properly generate the shapes and that its latent space does not provide an adequate search space for VE. While generating shapes to which there are no corresponding training examples, the reconstruction errors of unseen shapes that can be created with recombination and interpolation (b and c) are expected to be lower than for extrapolation (d).

To determine the resolution of the models, the distances in the latent space between the training examples for the baseline model and between the training and the unseen examples for the task models (b–d) are measured. If the latter are of a similar order of magnitude as the first, the models are able to distinguish unseen shapes from the training examples and from each other. This would indicate that the model's resolution is high enough to provide features for VE.

This experiment was performed on each of five base shapes (Fig. 3.21f) and for three different sizes of the VAE's latent space (4, 8, and 16 dimensions), as it is assumed that the model would not able to perfectly learn the two generating factors. Over the resulting 15 runs, average results are reported.

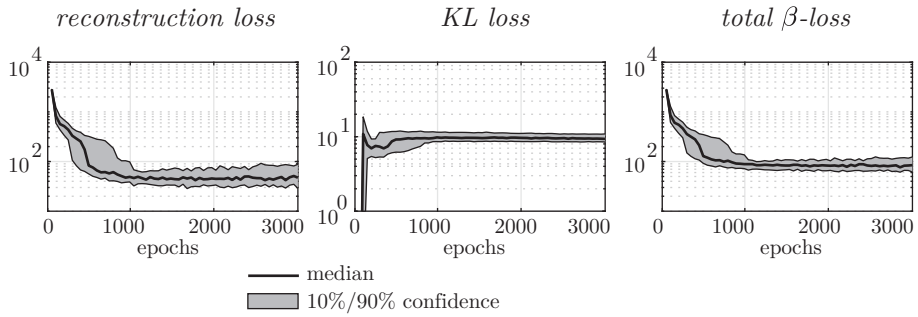

Figure 3.22: VAE validation losses during training.

Fig. 3.22 shows the reconstruction, KL and total β-loss on the validation data during training of the models. The training does not need much more than 1000 epochs to converge.

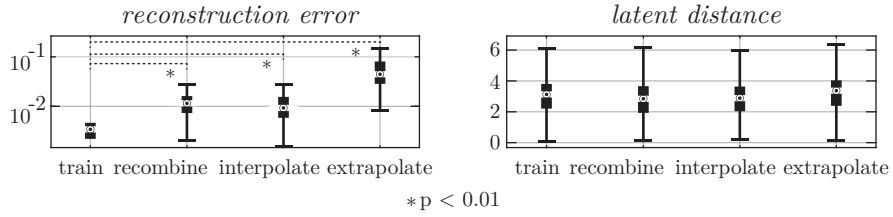

Figure 3.23: Reconstruction errors and latent distances for tasks a–d. Two-sample t-tests are performed comparing the reconstruction errors of the VAE's prediction of the training images to the recombination, interpolation and extrapolation data sets. Comparisons are marked with dotted lines.

Fig. 3.23 (left) shows the reconstruction errors for the training, recombination, interpolation and extrapolation sets. All significant results (two-sample t-test, $p < 0.01$) are marked with an asterisk. The error the models produce on the training samples is significantly lower than when reproducing the recombination and interpolation sets. As expected, the error on the extrapolated shapes is highest. The latent distances between the shapes in the four sets are shown in Fig. 3.23 (right). The distance distributions are similar.

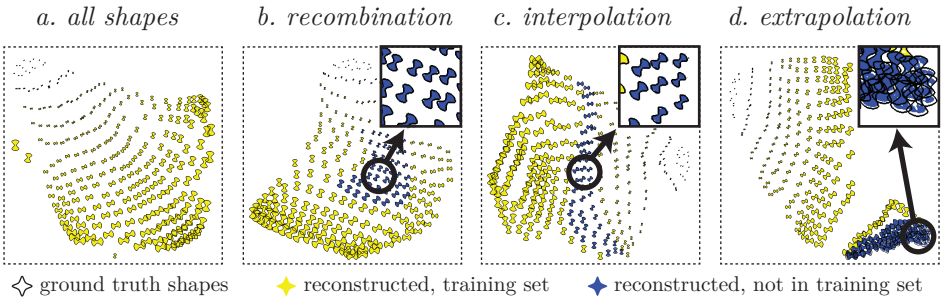

Figure 3.24: Examples of samples and latent spaces produced by the VAE with a latent dimensionality of eight (projected to two dimensions by t-SNE). Shapes in yellow represent samples that were present in the data set, while blue ones were not and have been generated by the model. Black outlines show the ground truth shapes, the difference between the outlines and shapes accounts for errors in reconstruction.

Four exemplary latent spaces are shown in Fig. 3.24 from models with a latent space dimensionality of eight, which has been projected to two dimensions with the dimensionality reduction method t-distributed stochastic neighbourhood embedding (t-SNE). This method was introduced by Maaten and Hinton (2008) and is further described in Appendix B. The first latent space (a, left) corresponds to the baseline model, trained on the complete training set. The other visualizations (b, c, and d) show the three tasks in which some shapes were omitted.

Expansion The last task, expansion (e), cannot be treated as per the previous experiment, because an a priori ground truth shape set 'outside of latent space' cannot be properly defined. Instead, the two search spaces (parameter search (PS) and latent search (LS)) are compared using AutoVE (see Fig. 3.15). It is measured which one of the two search spaces produces the most diverse set of artifacts using the PD metric. The experiment is split up into two configurations.

In the first configuration (**R**), both of the compared search approaches start from the same random initial set of genomes, which is common in many optimization problems. The size of the set is increased to 512, as this experiment poses a more difficult optimization problem. The genomes are translated into bitmaps, which serve as the training data for a VAE model. VE is then performed in both search spaces to fill two separate archives of 512 shapes each. The resulting shape sets are compared w.r.t. their diversity and average fitness, which are often in conflict with each other. As the translation from genome to bitmap always produces a contiguous shape, it is reasonable to expect that a VAE would learn to produce shapes, and not only random noise, even when starting with a randomly generated set of examples.

Often, a generative system does not start from a random set of data, but rather a set of examples that has been observed in the real world. A second configuration, continuation (**C**), is defined to reflect this. An experiment is performed to answer whether the diversity improves when training a VAE with a set of high-quality generated artifacts from a previous VE search. The archive of shapes produced by PS from the random initial set (R) is used as training data for a new VAE model. Both PS and LS are then performed again with this improved model.

It is expected that LS will interpolate between training samples, but will not be able to expand beyond the generative factors in the data, except through modeling errors. Since PS is performed on the explicit genome encoding, this search approach should be able to produce a much higher diversity of artifacts in both configurations **R** and **C**.

The number of latent dimensions of the VAE has been set to 8, 16 and 32 to analyze the influence of the degrees of freedom in latent space, when it is lower than, equal to, or higher than the number of parameters of the genome representation. A higher number of degrees of freedom gives an advantage to the latent model, a lower number would give it a disadvantage. When using 16 latent dimensions, VE deals with the same dimensionality in PS and LS. The number of filters in the VAE is quadrupled to give the model a better chance at learning the larger number of variations.

This experiment has been repeated ten times per configuration: 1) random initial set **R** in PS, 2) continuation **C** in PS, 3) **R** in LS and 4) **C** in LS. Average results are reported over the total of 10 experiment repetitions.

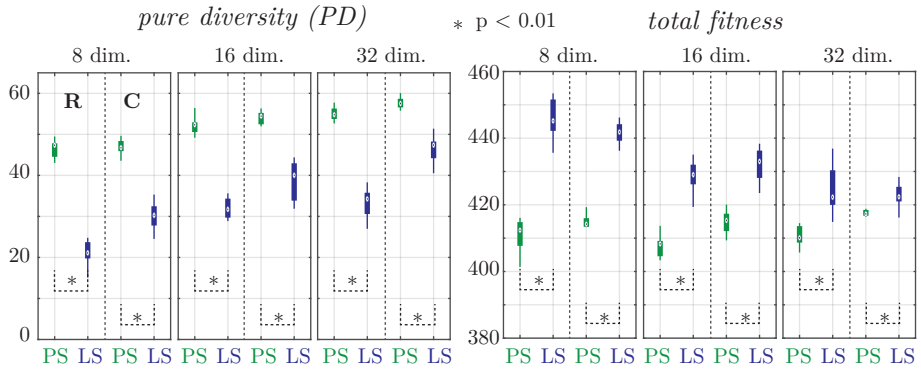

Figure 3.25: Diversity (left) and total sum of fitness (right) of artifact sets of both parameter (PS, green) as well as latent search (LS, blue). VAEs were trained with 8, 16 and 32 latent dimensions , respectively. Both the random initialization (**R**) and continuation (**C**) configurations of the experiments are shown (in every box the two left-hand bars correspond to **R** and the two right-hand to **C**). Significant differences (two-sample t-test, $p < 0.01$) are marked with an asterisk.

Fig. 3.25 compares the diversity and total fitness of the generated artifact sets. The diversity of PS is significantly higher than LS. In turn, LS produces artifacts with higher levels of fitness. Although the difference between PS and LS gets smaller when continuing search from an updated model (configuration **C**), it is still significant. All significant results (two-sample t-test, $p < 0.01$) are marked with an asterisk. Fig. 3.26 shows the expansion away from the latent surface achieved by PS, analogous to the hypothesis in Fig. 3.21e. The PS and LS artifacts' position in 16-dimensional latent space is reduced to two dimensions using t-SNE. The reconstruction error of the model's prediction of the PS artifacts is used as a distance measure to the latent surface.

An example of the resulting shape sets of PS and LS is shown in Fig. 3.27 to illustrate the effective difference in pure diversity.

3.5.4 Discussion

VAEs are able to reproduce recombined and interpolated artifacts quite well (Section 3.5.3). Extrapolation beyond the extremes of the generative factors is more difficult, which the higher reconstruction error in Fig. 3.23 gives evidence for.

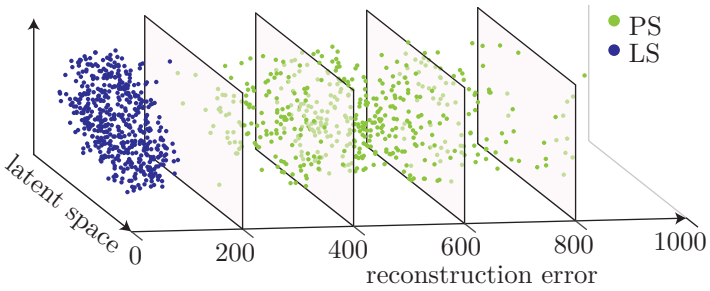

Figure 3.26: Expansion in a 16-dimensional latent model (projected to two dimensions with t-SNE). We interpret the reconstruction error of a shape as its distance from the latent surface. Samples from parameter search (PS, green) tend to extrapolate away from the latent distribution (LS, blue).

The distributions of latent distances between all four data set variants were similar. This provides some evidence that, even when VAEs are not able to reproduce the extrapolated shapes, they can still distinguish them from the training data and from each other. The position of examples in the latent space reflects the semantic relationship between shapes, as visualized in Fig. 3.24. Shapes are not properly reconstructed in the extrapolation task, but they are still positioned in a well-structured manner. The evidence leads to the hypothesis that expansion away from the latent surface will be even more difficult when searching the latent space directly. It also provides evidence that the generative system would be able to fulfill requirement A5 and allow a configurable resonance-dissonance trade-off, as long as the VAE is not used as a search space directly.

Section 3.5.3 attests to an answer to the last task, expansion. The ability of a VAE to find new shapes is indirectly measured by comparing the diversity of the artifact sets created by a parameter and a latent search (PS and LS). The diversity of PS is significantly higher than that of LS, as is shown in Fig. 3.25. This effect holds when the number of latent dimensions is increased beyond the number of degrees of freedom in the original explicit encoding or when the VAE is updated after a first VE run (**C**). Although a trade-off between diversity and fitness is expected, it becomes less pronounced in the 32-dimensional model. This provides evidence that parameter search actually finds both a more diverse as well as a higher performing set of artifacts than latent search. It can therefore be concluded that, when combining VAE and VE, it might make sense to use more powerful

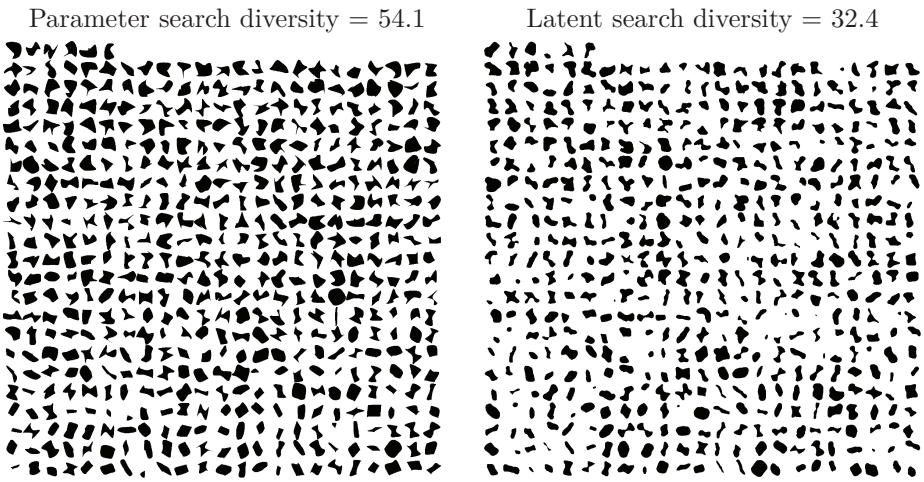

Figure 3.27: The artifact set generated by PS exhibits a higher diversity than the one generated by LS with a 16-dimensional latent model (projected to two dimensions with t-SNE).

encodings than those provided by a GM, but still take advantage of the VAE's ability to distinguish shapes and amplify diversity.

3.5.5 Conclusions

To answer research question IV ("What are the limitations of generative models in terms of the possible diversity of the solutions they create?"), a comparison was made between the use of latent spaces in GMs as a base for divergent search methods, specifically the AutoVE algorithm. The findings shown in this section quantify a VAE's ability to generate samples through recombination, interpolation and extrapolation within and expansion beyond the distribution of a given data set. The diversity of generated artifacts was compared when AutoVE is run either in latent space or parameter space. Based on these observations evidence was given that, in the context of divergent optimization, VAEs should be preferred to be used as a judge of similarity, instead or at least alongside their use as a creator of diverse artifact sets. Using GMs to distinguish solutions in evolutionary computation combines the advantages of automatically discovering the best way to describe similarity and diversity, while still allowing the use of powerful representations that produce more diverse sets of artifacts than data-driven encodings.

The usefulness of a GM's ability to interpolate, extrapolate or expand has to be discussed in the context of its application. In one setting, it might be ideal to perfectly reproduce a given domain and a model might be considered as working well if it can fit a distribution accordingly. In another context, however, and here artistic applications are included, a model's ability to surprise through its unexpected outputs can be of more value and desirable. Here we can use extrapolation to configure a resonance-dissonance effect in the mind of the user.

In the real world, the assumption that a GM can learn a 'perfect' representation does not hold. A perfect model might not produce anything unexpected, only creating high quality artifacts with low diversity. A broken model, on the other hand, might not produce anything useful. The use of modeling errors to find novel artifacts is certainly a mechanism that allows us to find novel solutions within the model. An important question for future work is whether early stopping can be used when training models to create novel yet useful artifacts: is there a correlation between training loss and diversity? This question is not answered in this thesis.

It is safe to assume that generative models are always limited by the training data, which biases models towards what they have learned. Of course, in very high-dimensional domains for which we can collect large data sets, like image and video data, a search in latent space already presents us with a vast amount of possible outcomes, which might be sufficient for some artistic contexts.

3.6 Chapter Summary

This chapter connected evolutionary computational encodings to ecology and showed that solutions are best compared using their (extended) phenotypes, due to neutrality and sensitivity effects. Now, different ways to solve a problem can be provided to users, which should increase the dissonance they experience. This answers research question I ("Are solutions best compared using their genomes or their expressed phenotype or behavior?").

In a critique by Whigham et al. (2017), the authors argue that biological evolution and its 'encoding' are not optimal and should not be blindly applied to evolutionary computation. They reason that genetic search operators must be aware of the consequences resulting from the genome-to-phenotype mapping. In support of

that criticism, I pose that, quite opposite to the emphasis on genetic search, search should take place in the phenotypic space. This space can oftentimes not be searched directly, for example in image generation, numerical or robotic simulations. Instead, a knowledge base needs to be built or learned that contains information about the common features of high-performing solutions.

Taking advantage of this with QD, performing niching in phenotypic space, enables an indirect search in those high-dimensional spaces. QD outperforms other multi-solution paradigms in generating diverse solution sets. A simple, self-expanding implementation of QD, Voronoi-Elites, was introduced, which makes comparisons between genetic and phenotypic search easier. This answers research question II ("What multi-solution optimization method produces the highest phenotypic diversity?").

QD's niching dimensions can be automatically extracted from data with GM, alleviating the user's input prior and increasing solution diversity even more than using predefined features. This answers research question II ("Can we produce more diverse solution sets when learning phenotypic niching from data instead of using predefined features?").

A further critical analysis was performed to highlight the limitations of GM and to answer research question IV ("What are the limitations of generative models in terms of the possible diversity of the solutions they create?"). Evidence was given that indicates that in some cases, GM might be better used as niching dimensions along which an EA can create a diverse set of solutions, instead of using the GM's latent dimensions for search directly. This is how we can enforce dissonance in the mind of the user and discover truly novel solutions.

CHAPTER 4

Efficiency

QD is able to produce many solutions with diverse behaviors. However, the algorithms in this class perform many evaluations, making them unsuitable for design problems that need computationally expensive or real world evaluation.

In this chapter, two of the problems QD algorithms have in computationally expensive domains are addressed. Although QD has been reformulated to use efficient surrogate models for the objective function by Gaier et al. (2017), expensive features are not calculated in an efficient way. The second issue arises in e.g. neural encodings that, due to structural dissimilarities and a high degree of sensitivity, cannot be modeled well.

In order to fulfill the efficiency requirements C1 ("Search should efficiently generate a diverse set of novel, functional solutions") and C2 ("Search should efficiently sample complex design spaces") from Table 2.3, the following research questions are answered in this chapter:

Research Questions

V Can we model behavioral features in a surrogate-assisted way by sampling based on optimality alone? (requirements C1 and C2)
VI Can we model neural encodings' behavior ex situ by sampling their outputs and using a behavior-kernel? (requirement C2)

The main insight highlighted in this chapter is that we can predict behavioral features from two domains, fluid dynamics and robotics, using statistical models. Using the models, the efficiency of QD can be increased.

What now follows is an extension of previous work on increasing the efficiency of QD applied to shape optimization in an expensive fluid dynamics domain.

4.1 Surrogate Modeling of Phenotypic Features

In order to create diverse solution sets in expensive domains, the feature dimensions along which QD's archive is defined have to be calculated in an efficient manner. Research question V ("Can we model behavioral features in a surrogate-assisted way by sampling based on optimality alone?") is answered in this section for a fluid dynamics optimization domain. Airflow features serve as behavioral features in this context, showing whether we can fulfill requirements C1 and C2, which were mentioned in the introduction to this chapter.

To decrease the number of expensive objective evaluations, approximative surrogate models can replace the evaluation of the objective function close to optimal solutions (see Jin (2011) for a review of this group of methods). To sample the design space effectively and efficiently for training samples, Bayesian optimization (BO) is used, which is explained in the next subsection. This section is based on earlier work by Hagg et al. (2020).

Surrogate models, oftentimes used in a Bayesian context (see Section 4.1.1) are based on similarity, the distance of candidate solutions to known examples. Similarity-based surrogate models have been used in such varied domains as: shape optimization in fluid dynamics (see Ong et al. (2003) and Daniels et al. (2018)), the discovery of new drugs (see De Grave et al. (2008)), the placement of hospital trauma centers (see Wang et al. (2016)), and even for the optimization of other machine learning methods (see Snoek et al. (2012) and Stork et al. (2017)).

4.1.1 Bayesian Optimization

BO is a strategy that is used to efficiently find optima of an expensive objective function (see Fig. 4.1). Given a prior[6] over the objective function, evidence from known samples is used to select the next best observation. The assumptions are encoded in a so-called covariance function. This decision is based on an acquisition function that balances exploration of the design space, sampling from unknown regions, and exploitation, choosing samples that are likely to perform well. The example shows an upper confidence bound (UCB) acquisition function. UCB was

[6] Priors usually consist of general assumptions on the mathematical description of the objective function. One such assumption is that a function is *smooth*: if points are close together, their function value is similar.

introduced by Auer (2002) and uses the uncertainty information from a random process to make decisions with an exploitation-versus-exploration trade-off. It is described by the function

$$\mathrm{UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}),$$

which is a balance between exploitation ($\mu(\mathbf{x})$, the predicted (mean) objective value of the model), and exploration ($\sigma(\mathbf{x})$, the model's uncertainty), weighted by κ. The surrogate model becomes more accurate in optimal regions when more samples are evaluated and added.

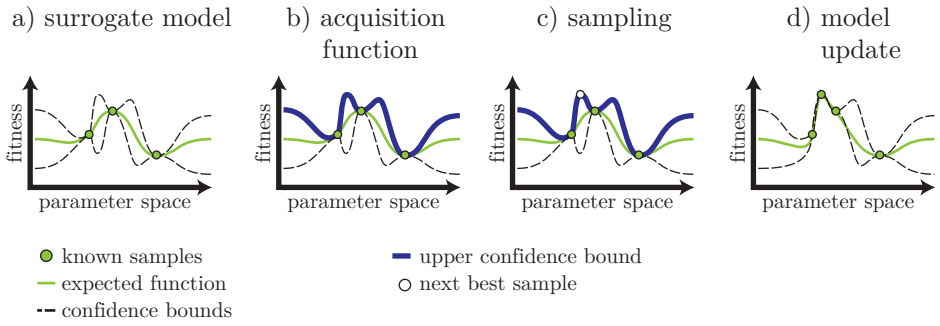

Figure 4.1: In Bayesian sampling, a statistical model serves as a surrogate for an expensive evaluation function. The model predicts function values as well as a confidence value on its prediction (a). Confidence is low depending on the distance to the next known sample in the model. The model's confidence value is included in the sampling process. An acquisition function (b), which takes into account both values, presents an 'optimistic' prediction about possible optimal locations. The sampling decision is based on a maximization of the acquisition function (c). After sampling, the model is updated, taking into account the new sampled location (d).

The most common surrogate models used are Gaussian process (GP) regression models, introduced by Rasmussen (2004). More information on GP regression is given in Appendix C.

4.1.2 Surrogate-Assisted Quality Diversity

QD became applicable to expensive optimization problems after the introduction of surrogate-assisted illumination (SAIL) by Gaier et al. (2017). In this Bayesian interpretation of QD, an archive is created based on UCB sampling. Surrogate

models, like GP models, cannot easily be based on phenotypes or behavior due to the high dimensionality of the artifacts. Furthermore, the models have to be able to deal with a large diversity of solutions, for example using a hierarchical decomposition of the parameter space (see Hagg (2017) and Stein et al. (2015)).

In SAIL, a GP regression model predicts the performance of new solutions based on the genetic distance to previous examples, serving as a surrogate (replacement) for expensive evaluation. The distance is modeled using the squared exponential (covariance) function, which has two hyperparameters: the length scale (or sphere of influence) and the signal variance, which are found by minimizing the negative log-likelihood of the process.

Algorithm 7 SAIL and its extension SPHEN

1: **procedure** SAIL/SPHEN($f()$,$p()$)
2: Set $budget, \sigma$
3: Set $\mathcal{X}, \mathbf{p}, \mathcal{F} \leftarrow \emptyset$
4: $\mathcal{X}' \leftarrow$ SOBOL($dim(\mathcal{X})$)
5: **while** $|\mathcal{X}| < budget$ **do**
6: $(\mathcal{F}', \mathbf{p}') \leftarrow$ EVALUATE($\mathcal{X}', f(), p()$)
7: $(\mathcal{X}, \mathbf{p}, \mathcal{F}) \leftarrow (\mathcal{X} \cup \mathcal{X}', \mathbf{p} \cup \mathbf{p}', \mathcal{F} \cup \mathcal{F}')$
8: $(\mathbf{M}_p, \mathbf{M}_f) \leftarrow$ TRAIN($\mathcal{X}, \mathbf{p}, \mathcal{F}$)
9: $f_s() \leftarrow$ PREDICT($\mathcal{X}, \mathbf{M}_f$)
10: $p_s() \leftarrow$ UCB($\mathcal{X}, 20, \mathbf{M}_p$)
11: $\mathcal{A}_a \leftarrow$ MAP-ELITES($\mathcal{X}, f()/f_s(), p_s(), \sigma$)
12: $\mathcal{X}' \leftarrow$ SELECT($\mathcal{A}_a$)
13: **end while**
14: $p_s() \leftarrow$ UCB($\mathcal{X}, 0, \mathbf{M}_p$)
15: $\mathcal{A} \leftarrow$ MAP-ELITES($\mathcal{X}, f()/f_s(), p_s(), \sigma$)
16: **end procedure**

SAIL is explained in more detail in Alg. 7, as it is both used and extended in this work. The extension (red elements) is explained in the next section.

First, a space-filling sequence is used to fill the initial population (line 4). Then, as long as the evaluation budget is not fully consumed, the population is evaluated to extract its performance and feature values (line 6). The features are calculated based on the expressed phenotype, or behavior, of a solution. A surrogate model, usually a GP regression model, is trained to predict new solutions' performance

values (line 8). The surrogate-assisted performance function $p_s()$ consists of a UCB acquisition function (see Section 4.1.1) with a large exploration factor $\kappa = 20$. MAP-Elites (see Alg. 2) generates an acquisition archive based on the populations' genomes (the search space), the feature function $f()$, and the surrogate-predicted performance function $p_s()$ in line 10. After MAP-Elites fills the acquisition archive which contains 'optimistic' solution candidates, a random selection of those candidates (line 12) is analyzed in the expensive evaluation function to form additional training samples for the GP model.

This loop continues until the evaluation budget is exhausted. Then, the UCB exploration factor κ is set to 0 in a final MAP-Elites run to create an archive that now contains a diverse set of solutions that is predicted to be high-performing (line 15). SAIL needs a budget orders of magnitudes smaller than MAP-Elites because it can exploit the surrogate model without 'wasting' samples. SAIL, however, is constrained to features that are cheap to calculate, like shape features that can be determined without running the expensive evaluation.

With SAIL it became possible to use performance functions of expensive optimization domains. But the strength of QD, to perform niching based on behavior, cannot be applied when determining those *behaviors* is expensive. Phenotypic features describe phenomena that can be related to complex domains, like behavioral robotics, mechanical systems, or computational fluid dynamics (CFD). Only when we can predict those expensive features efficiently, the road to productive phenotypic niching is opened up. Only then can we generate phenotypically diverse solution sets in engineering.

4.1.3 Surrogate-Assisted Phenotypic Niching

To be able to handle expensive features, surrogate-assisted phenotypic niching (SPHEN) is introduced (see Fig. 4.2, Alg. 7 including red elements). By building on the insight that replacing the performance function with a surrogate model decreases the necessary evaluation budget, the exact features are replaced with surrogate models as well.

The initial training sample set $\mathcal{X}'$, used to form the first seeds of the acquisition map, is produced by a Sobol sequence in the design space (see Fig. 4.2a and line 4 in Alg. 7). Due to the lack of prior knowledge in black-box optimization, using space-filling sequences has become a standard method to ensure a good coverage

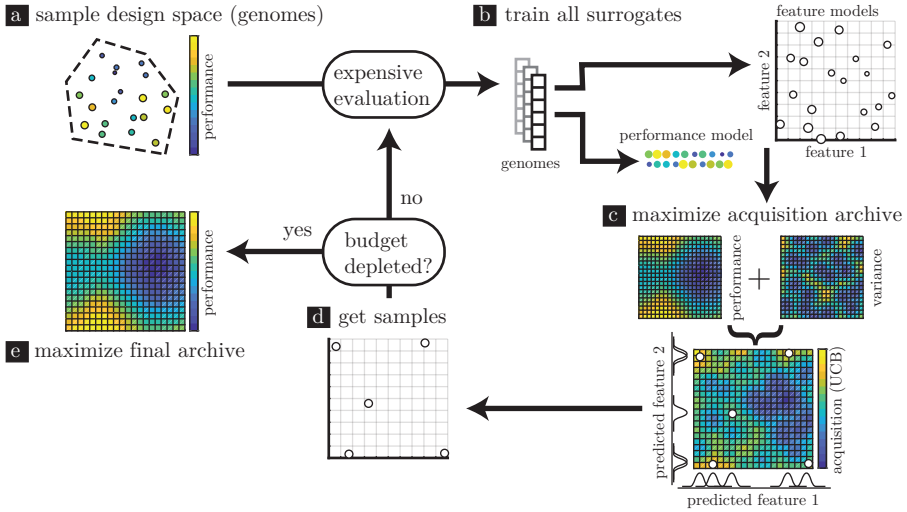

Figure 4.2: Surrogate-assisted Phenotypic Niching. An initial sample set (a) is evaluated. Surrogates are trained to predict performance and features (b). Surrogate-assisted MAP-Elites is used to produce an acquisition map, balancing exploitation and exploration with the UCB of the performance model. Feature models predict the niche of new individuals (c). New samples are selected from the acquisition archive (d). After the evaluation budget is depleted, the surrogate models are used to generate the final archive, ignoring model confidence (e).

of the search domain. The initial set is evaluated, for example in a computational fluid dynamics simulator (line 6).

Performance values ($\mathbf{p}$) and phenotypic features ($\mathcal{F}$) of those samples are derived from the results, or, in the case of simpler non-behavioral features, from the solutions' expression or shape themselves. The key issue here is to check the range of the initial set's features. Since it is unknown what part of the phenotypic space will be discovered in the process, the initial set's feature values only give us a first hint of the reachable space. A space-filling sampling technique used in the design space is not necessarily space-filling in feature space.

After collecting performance and feature values, the surrogate models are trained (see Fig. 4.2b and line 8). The GP models limit the number of samples to around 1000, as the training and prediction becomes quite expensive. A squared exponential covariance function is used and the (constant) mean function is set to the training samples' mean value. The covariance function's hyperparameters, length scale and

signal variance, are deduced using the GP toolbox GPML's (see Rasmussen and Nickisch (2010)) conjugate gradients based minimization method, which is run for 1000 iterations.

Importantly, MAP-Elites (Alg. 2) does not receive feature and performance functions directly, but instead, the feature model $\mathbf{M}_f$ predicts feature locations through the predicted feature function $f_s()$ and the performance model $\mathbf{M}_p$ is used as part of a predicted performance function $p_s()$ using UCB with $\kappa = 20$ (lines 9 and 10). MAP-Elites creates the acquisition map, a version of the archive that contains an 'optimistic' set of proposed samples. It does so by optimizing the UCB of the performance model only. Feature models assign samples to their niches (see Fig. 4.2c).

Notably, in the surrogate-assisted version of MAP-Elites the confidence of feature models is not taken into account. The reasoning behind this is that, although the search takes place in a high-dimensional space, QD only has to find the *elite hypervolume* (see Vassiliades and Mouret (2018)), or *prototypes* (see Hagg et al. (2018)), the regions consisting of high-performing solutions. Only the performance function can guide the search towards the hypervolume. Taking into account the feature models' confidence intervals adds unnecessary complexity to the modeling problem. SPHEN's goal is to be able to only predict features for high-performing solutions, so we let feature learning 'piggyback' on this search.

After having received back the acquisition archive, a selection of samples from the acquisition archive is taken using a space-filling algorithm like Sobol (see Fig. 4.2d). The samples are then evaluated to continue training the surrogate models. This process iterates as long as the evaluation budget is not depleted. Finally, MAP-Elites is used to create a prediction map, ignoring the models' confidence altogether (see Fig. 4.2e) by setting $\kappa = 0$, which is filled with diverse, high-performing solutions (Alg. 7, line 15).

SPHEN extends SAIL by replacing the direct calculation of phenotypic features with the predictions of a surrogate model. Before SPHEN is applied to an expensive CFD domain, its performance is compared to MAP-Elites and SAIL in a simpler, inexpensive domain.

4.1.4 Quantitative Comparison

SPHEN is compared to SAIL and MAP-Elites in terms of how many precise function and feature values have to be calculated. It is important to evaluate how accurate the feature models are when trained with a performance-based acquisition function. Only then is SPHEN applied to an expensive domain. To be able to calculate all performance and feature values, we optimize the same free-form deformed, eight-sided polygons as were shown in Fig. 3.11.

QD optimization is run without (MAP-Elites) and with surrogate model(s) (SAIL, SPHEN) on the polygon domain. This way all ground truth performance and feature values can be calculated in a feasible amount of time. The shape features are expected to be easier to learn than the flow features of the airflow domain. SPHEN is expected to perform somewhere between SAIL and MAP-Elites, as it has the advantage of using a surrogate model but also has to predict two phenotypic features. However, since the ultimate goal is to be able to use QD on expensive features, SPHEN will be the only possibility to achieve this.

Budgets Comparing MAP-Elites, SAIL and SPHEN in a fair manner is not straightforward, although the surrogate-assisted algorithms have a very similar structure. The budget, the number of precise function evaluations, can be defined in two manners. Because SAIL was introduced to reduce the number of function evaluations for expensive performance functions, it was compared with MAP-Elites based on the number of precise performance evaluations (PE) by Gaier et al. (2017). However, due to the introduction of expensive feature evaluations, the number of precise phenotypic feature evaluations (PFE) has to be taken into account. Here, SAIL will quickly consume the budget.

The (maximum) PE and PFE budgets of all algorithms used in the following comparison are listed in Table 4.1. The PE budget is the same for all algorithms, except MAP-Elites, as it was already shown that it needs many more PE than SAIL (see Gaier et al. (2017)). MAP-Elites receives a budget of 65,536 generations multiplied with the number of children per generation. The choice to restrict the PE instead of the PFE budget makes it easier to compare results to previous work.

Table 4.1: Maximum budget for MAP-Elites, SAIL, budget-reduced SAIL^{r} and SPHEN in experiments.

Parameter	MAP-Elites	SAIL	SAIL^{r}	SPHEN
# MAP-Elites generations	4096	1024	64	1024
# MAP-Elites children	16	32	16	32
# acquisition iterations	-	64	64	64
budget per iteration	-	16	16	16
total PE	65,536	1024	1024	1024
total PFE	65,536	2,098,192	65,536	1024

The fixed maximum PE budgets yield the PFE budgets shown in the last row of Table 4.1. It is easy to see how SAIL is intractible when using expensive features, as it needs 2,098,192 PFE to reach the GP model's maximum number of 1024 samples. By reducing the budget (SAIL^{r}) to at most the number of PFE used by MAP-Elites, we can both reestablish a more feasible number of PFE as well as maintain some kind of comparability. This is accomplished by reducing the number of generations and children in the internal (MAP-Elites) loop in SAIL^{r}. It is not useful to reduce the budgets even more because for a 'fair' comparison, the number of PFE within the inner loop would have to be reduced to $1024/64 = 16$, which would amount to a single MAP-Elites generation.

A comparison is now possible by showing the results for all PFE budgets (16, 1024, 65,536, and 2,098,192), as far as they are practically attainable.

Parameterization The initial sample set of 16 examples as well as the selection of new samples (16 in every iteration) is created using a pseudo-random Sobol sequence. In QD, the phenotypic archive allows us to find many diverse solutions but in order to fill the archive, it needs to be ensured that we can sweep through the searched genetic space multiple times. The σ value is therefore quite high. The mutation operator adds a value drawn from a Gaussian distribution with $\sigma = 10\%$ of the parameter range.

Misclassification in SPHEN Due to the expected inaccuracy of the feature models, solutions will be misclassified. Misclassification will decrease the accuracy of the niching mechanism. Fig. 4.3 shows a niching archive at a resolution of 32x32 and the true performance and feature archive. Holes appear due to misclassification,

which is why SPHEN is trained on a higher resolution map. The archive is a reduction to a resolution of 16x16. Most niches are now filled. In this experiment all archives have a resolution of 16x16 solutions.

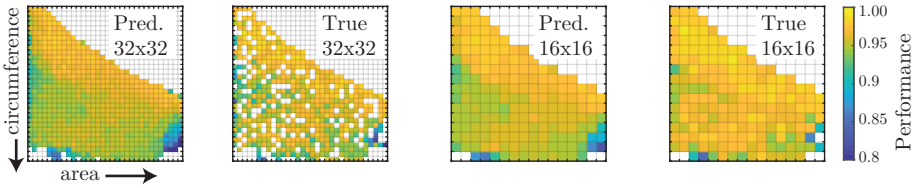

Figure 4.3: Predicted and true SPHEN maps on symmetry domain, trained in 32x32 resolution (left), then reduced to 16x16 resolution to remove holes (right).

Results The mean number of filled archive niches and performance values for five replicates are shown in Fig. 4.4. SAIL and SPHEN find about the same number of solutions using the same number of PE. Notably, the mean performance of SPHEN's solutions is higher than that of SAIL.

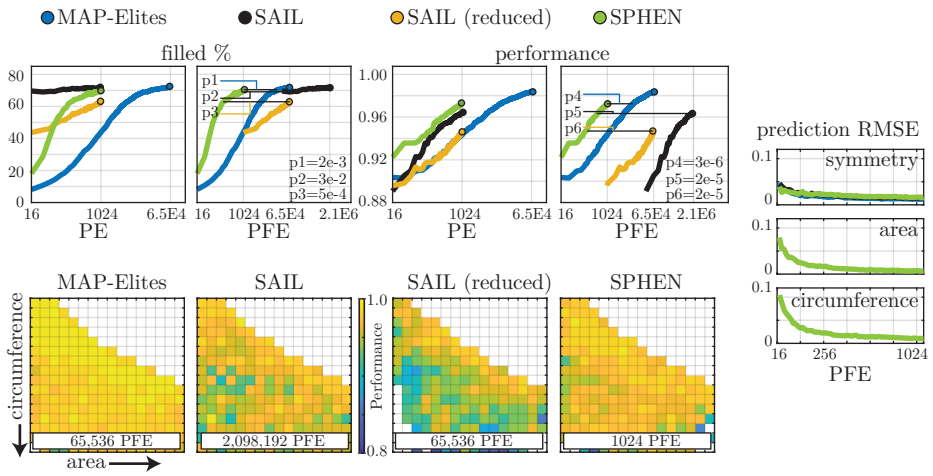

Figure 4.4: Comparison of MAP-Elites, SAIL and SPHEN based on performance evaluations (PE) and performance/feature evaluations (PFE). Experiments were repeated five times to produce the mean percentage of the archive's filled and mean performance values. Prediction errors are included on the right and example archives at the bottom. The experiments include SAIL with a budget reduced to the number of PFE used in MAP-Elites.

In domains with expensive feature evaluations, the performance and PFE need to be taken into account. Due to the number of necessary feature evaluations, SAIL needs more than two million PFE to perform almost as well as SPHEN, which only needs 1024 – over three orders of magnitude less and still more than an order of magnitude less than MAP-Elites. Since in expensive real world optimization problems we cannot expect to run more than about 1000 function evaluations, due to the infeasibly large computational investment, the efficiency gain of SPHEN is substantial. If the number of PFE of SAIL is lowered to the same budget as MAP-Elites and given more time to search the iteratively improving surrogate model before running out of the budget of 65,536 PFE (see Table 4.1), SAIL still takes a big hit. It is not able to balance out quality and diversity. The example prediction maps are labeled with the number of PFE necessary to achieve those maps. Although new training examples are not sampled to improve the feature models specifically, their root mean square error (RMSE) ended up at 0.012 and 0.016 respectively. Finally, SPHEN is compared to the three alternative configurations on the null hypothesis that they need the same number of PFE to reach an equally filled archive or equal performance. Significance levels, calculated using a two-sample t-test, are shown in Fig. 4.4. In all cases, the null hypothesis is improbable ($p < 0.05$), although for the comparison of filled levels to SAIL it is rejected with less certainty.

The acquisition function of SAIL needs no adjustments. SPHEN and SAIL search for the same elite hypervolume, which is only determined by the performance function.

4.1.5 Use Case: Wind Nuisance in Architecture

The previous section showed that both performance as well as feature models can be predicted using an optimality-based acquisition function. The implementation of an efficient divergent search method, SPHEN, produces a diverse and high-performing set of solutions with the help of simple statistical models. To put SPHEN to a real test, however, at least one of the features should describe the behavior of a solution in an expensive domain. Fluid dynamics is commonly used as a use case in optimization studies. The problem of wind nuisance in the built environment combines this expensive domain with a typical setting for human-computer co-creation, where not all preferences can be formalized. Co-creating

diverse solutions in this domain can uncover a large potential of solutions that, in real world design processes, lead to better alternatives to current design.

The polygon domain that was described in Section 3.3.3 is used again (see Fig. 3.11). One of the features is replaced by wind nuisance. It is defined in building norms in NEN 8100 (2006), described by Janssen et al. (2013), and uses the wind amplification factor measured in standardized environments, with respect to the hourly mean wind speed. In a simplified two-dimensional setup, we translate this problem to that of minimizing the maximum airflow speed (u_{max}). The performance metric is defined as the inverse over the normalized maximum velocity in the flow: $p(x) = \frac{2}{(1+u_{max}(x))} - 1$ and is calculated with a fixed flow input speed in the simulation. The value for u_{max} only needs to be kept within a *nuisance threshold*, which is set to $u_{max} \leq 0.12$.

The closed form encoding from the polygon domain is used to produce two-dimensional shapes that are then placed into a CFD simulation. To put emphasis on the architectural nature of the domain, two features are used: area and airflow turbulence. The chaotic behavior of turbulence provokes oscillations around a mean flow velocity, which influences the maximum flow velocity.

Both features are not optimization goals. Rather, it is to be analyzed how the size of the area and turbulence are related to each other, under the condition of keeping the flow velocity low. Shapes should be produced that are combinations between their appearance (small to large) and their effect on the flow (low to high turbulence). Concretely, at the lowest and highest values of area and turbulence, regular intuitive shapes should be generated by the algorithm such as slim arrow-like shapes for low turbulence and area, or regular polygons for high turbulence and area. However, for area/turbulence combinations in between, the design of the shape is not unique and will possibly differ from intuition.

Lattice Boltzmann Method Viscous fluid dynamics systems are described by the Navier-Stokes equations, which is a set of partial differential equations. The Lattice Boltzmann method (LBM) is an established tool for the simulation of fluid dynamics (see Krüger et al. (2017)). Instead of directly solving the Navier-Stokes equations, the method operates a stream and collide algorithm of particle distributions derived from the Boltzmann equation. In this contribution, LBM is used on a two-dimensional grid with the usual lattice of nine discrete particle velocities. At the inlets and outlets, the distribution function values are set to

equilibrium according to the flow velocity. The full bounce-back boundary condition is used at the solid grid points corresponding to the polygon. Although there are more sophisticated approaches for the boundaries, this configuration is stable throughout all simulations. In addition, the bounce-back boundary condition is flexible, as the boundary algorithm is purely local with respect to the grid points.

As an extension of the Bhatnagar-Gross-Krook (BGK) collision model by Bhatnagar et al. (1954), a Smagorinsky subgrid model (see Gaedtke et al. (2018)) is used to account for the under-resolved flow in the present configuration. A more detailed description of the underlying mechanisms can be found in Krüger et al. (2017). The results of the two-dimensional domain do not entirely coincide with results that will be found in three dimensions, caused by the difference in turbulent energy transport, which was shown by Tennekes (1978).

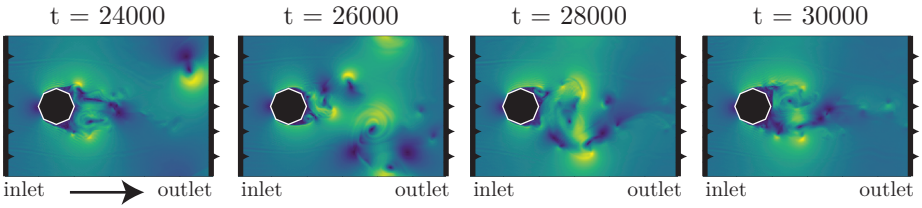

Figure 4.5: Airflow around a circular polygon shape at four different time steps.

The simulation domain consists of 300×200 grid points. A bitmap representation of the polygon is placed into this domain, occupying up to 64×64 grid points. As the Lattice Boltzmann method is a solver of weakly compressible flows, it is necessary to specify a Mach number (0.075), a compromise between computation time and accuracy. The Reynolds number is $Re = 10,000$ with respect to the largest possible extent of the polygon. For the actual computation, the software package *Lettuce* by Krämer et al. (2020) is used, which is based on the PyTorch framework (see Paszke et al. (2019)), allowing easy access to graphics processing unit (GPU) functionality. The fluid dynamics experiment was run on a cluster with four GPU nodes, each simulation taking ten minutes. Fig. 4.5 shows the airflow around a circular polygon at four different, consecutive time steps. Brighter colors represent higher magnitudes of airflow velocity. Throughout the 100,000 time steps of the simulation, maximum velocity and enstrophy are measured. The enstrophy, a measure for the turbulent energy dissipation in the system with respect to the

resolved flow quantities (see Gassner and Beck (2013) and Krämer et al. (2019)), increases as turbulence intensity increases in the regarded volume.

Validation and Prediction of Flow Features The maximum velocity u_{max} and enstrophy E are measured every 50 steps. A running average is employed over the last 50,000 time steps. To test whether the simulations indeed converge to a stable value, they are tested with different shapes (nine varied-size circles and nine deformed star shapes) and calculate the moving average of the enstrophy values, which is plotted in Fig. 4.6. The value converges to the final feature value (red).

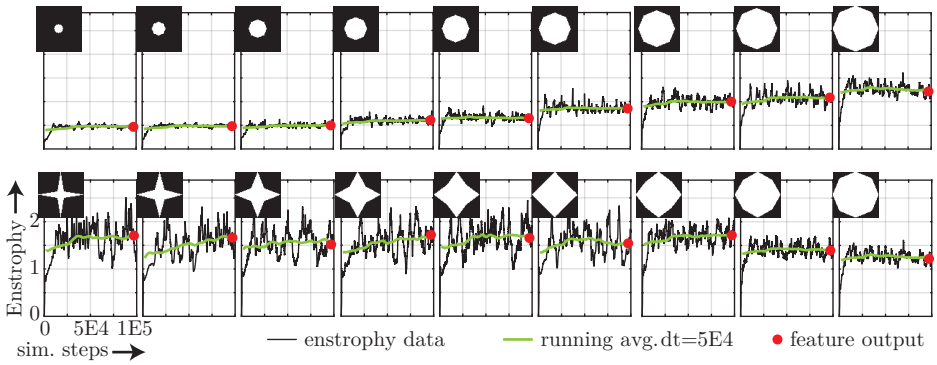

Figure 4.6: Enstrophy values during simulation of circles and stars. The running average of the last 50,000 time steps converges to the final feature output.

The two small shape sets are used to validate the two measures. Increasing the radius of the circles set should lead to higher u_{max} and E, as more air is displaced by the larger shapes. The stars set is expected to have larger u_{max} and E for the more irregular shapes. This is confirmed in Fig. 4.7.

Next, the prediction accuracy for the flow feature values is evaluated using a GP model. Although GP models are often called 'parameter free', this is not entirely accurate. The initial guess for the hyperparameter's values, before minimization of the negative log-likelihood of the model takes place, can have large effects on the accuracy of the model. The log-likelihood landscape can contain local optima. A grid search is performed on the initial guesses for length scale and signal variance. Using leave-one-out cross validation, GP models are trained on all but one shape, after which the accuracy is measured using the mean absolute percentage error

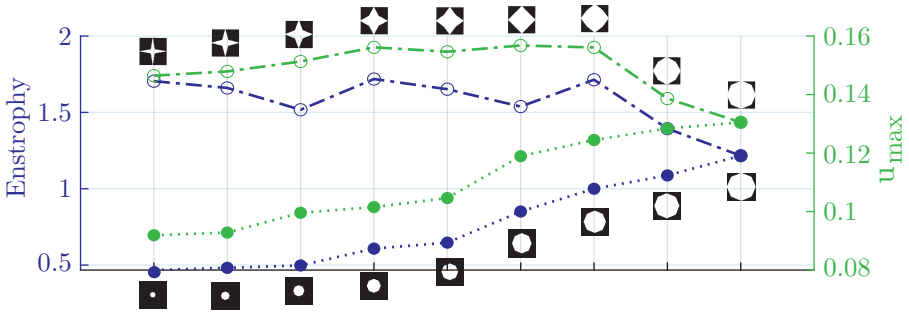

Figure 4.7: Enstrophy and maximum velocity of circles and stars.

(MAPE), giving a good idea about the magnitude of the prediction error. The process is repeated until all examples were part of the test set once. The MAPE on u_{max} was 2.4% for both sets. The enstrophy was harder to model, at 4.9% and 10.3% for the respective sets, but still giving us confidence that these two small hypervolumes can be modeled.

Resulting Shapes The objective is to find a diverse set of airflows using a behavioral feature, turbulence, and one shape feature, the surface area of the polygon. This should answer the question how the size of the area and turbulence are related to each other and which shapes do not pass the wind nuisance threshold. The same budget for SPHEN is used as listed in Table 4.1, with the exception of allowing 4096 generations in the prediction phase. The enstrophy and velocity are normalized between 0 and 1 using a predetermined value range of $E \in [0.15, 1.1]$ and $u_{max} \in [0.05, 0.20]$.

The resulting archive in Fig. 4.8 shows that turbulence and surface area tend to increase the mean maximum airflow velocity, as expected. A small selection of airflows is shown in detail. Due to the chaotic evolution of turbulent and transient flows, a static snapshot of the velocity field provides only limited information about the flow structures. Therefore, dynamic mode decomposition (DMD) is used to extract and visualize coherent structures and patterns over time from the flow field (see Demo et al. (2018) and Schmid (2010)).

Especially those shapes at the extrema of area and turbulence align with the aerodynamic expectations. At low turbulence intensity, the shapes tend to be slim and long with respect to the flow direction (shapes A and B). High turbulence

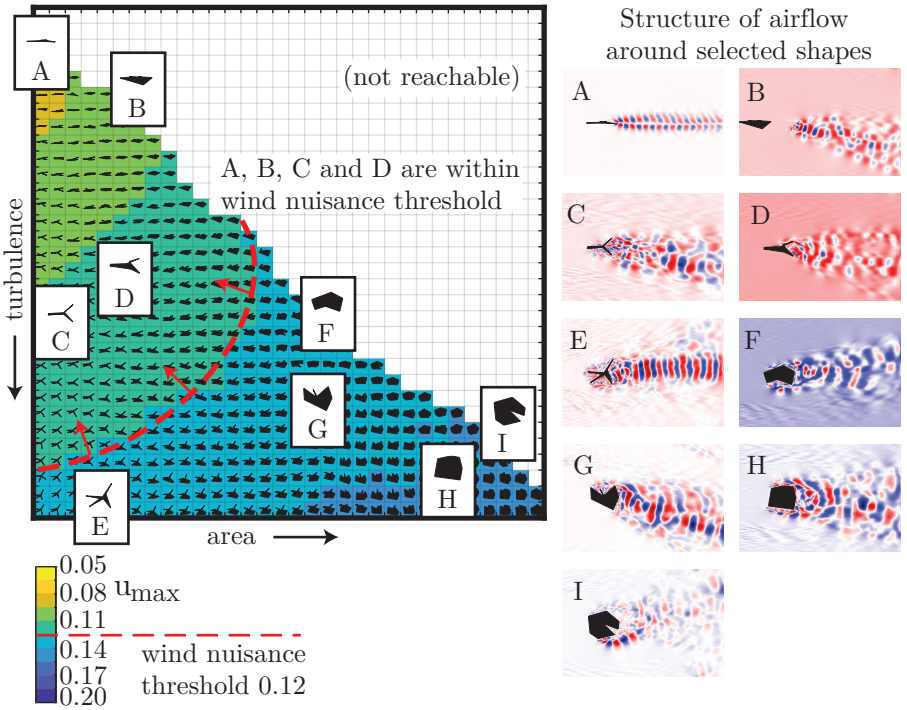

Figure 4.8: A diversity of shapes and airflows that shows which designs conform to the wind nuisance threshold. The dominant DMD mode shows the structure of the airflow around nine selected shapes. A, B, C and D are within the wind nuisance threshold.

levels at small shape areas are achieved if the shapes are oriented perpendicularly to the flow (shape E). Pentagons or hexagons evoke high turbulence levels at large areas (shapes H and I). However, impressively, there is an enormous variety of nuances in between these extrema with non-intuitive shapes, enabling the designer to determine a shape for given flow parameters down to a lower turbulence bound for each area value. Furthermore, the algorithm also suggests known tricks to manipulate the flow. Side arms are an appropriate measure to vary the turbulence intensity in the wake (shapes C, D, E, and G). Indentations or curved shapes redirect the flow and extract kinetic energy similar to turbine blades (see Dorschner et al. (2017)), which can be observed in shape D. Conclusively, for the highest and lowest area and turbulence values, SPHEN matches the expectations, while for the shapes in between, SPHEN exceeds expectations by introducing unusual shape nuances, which encourage further investigation.

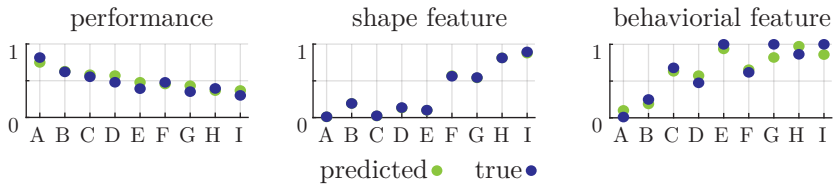

Figure 4.9: RMSE of performance and feature models.

The accuracy of the surrogate models is shown in Fig. 4.9. The RMSE of the models is 0.06, 0.01 and 0.10, respectively. A non-parametric hypothesis test is performed to determine the rank correlation between pair-wise comparisons using either the predicted or true values. In evolutionary computation, only the rank correlation is important, because values are compared only pair-wise within the evolutionary process.

The rank correlation coefficient by Kendall and Gibbons (1990) is used (τ). In contrast to Pearson correlation, this measure only considers ordinal correlation, i.e., the ranks of two compared sets of samples. Kendall correlation is therefore a good measure to estimate the accuracy of a model when used in rank-based evolutionary optimization methods. The coefficient τ amounts to one when all comparisons lead to the same outcome and to minus one when all comparisons lead to the exact opposite outcome. If in this experiment, τ equals one, the surrogate models produce the same outcome as using the true values. In this case, τ amounts to 0.78, 1.00 and 0.73. The models are therefore accurate enough for rank-based selection.

In this real world optimization case, SPHEN is able to produce a diverse set of solutions. The surrogate models are able to both predict performance as well as diversity.

4.1.6 Conclusions

This section gave evidence for a positive answer to research question V (“Can we model behavioral features in a surrogate-assisted way by sampling based on optimality alone”), fulfilling requirements C1 (“Search should efficiently generate a diverse set of novel, functional solutions”) and C2 (“Search should efficiently sample complex design spaces”). Behavioral surrogate models can be trained by

piggybacking upon an acquisition function that selects samples based on optimality alone.

In the polygon domain, both surrogate-assisted algorithms are able to find a large variety of solutions. When features do not have to be modeled, they show similar performance, although SAIL converges much sooner. However, when taking into account the number of feature evaluations, SPHEN clearly outperforms SAIL as well as MAP-Elites. Modeling features does not lower the performance of a prediction map. In terms of solution performance, both surrogate-assisted algorithms are outperformed by MAP-Elites in the simple domain, but SPHEN clearly beats MAP-Elites by requiring less evaluations. The feature models become more accurate even when sampling only to improve the performance model.

When designing diverse airflows, one SPHEN run took 23 hours, producing 494 different flow profiles. With SAIL, obtaining the same result would have taken over five years, which is not feasible in practice. Although MAP-Elites outperformed SAIL in the simple polygon domain, and might have outperformed it in the airflow domain as well, it still would have taken two months to calculate with uncertain results. Fig. 4.8 shows the structure in the airflows that can appear in this problem domain. Variations (area) of the object we want to design as well as their effect on the environment (turbulence) can be efficiently created. Even when only using two phenotypic features, the nuances between the variations give us an idea which shapes do not pass the wind nuisance threshold and which ones do. With this we could continue the design process based on our new intuition.

Expensive features can be efficiently predicted. As such, phenotypically diverse solution sets can be efficiently generated, even in expensive domains such as CFD.

4.2 Surrogate Modeling of Neural Behaviors

Evolutionary encodings can have a non-closed form or even be indirect, including e.g. neural representations. Research question VI ("Can we model neural encodings' behavior ex situ by sampling their outputs and using a behavior-kernel?") is answered to fulfill requirement C2 ("Search should efficiently sample complex design spaces") for neural representations.

Besides having to deal with expensive numerical fitness evaluations, neural representations, commonly used in the robotics domain, have the need to run physics-enabled simulations or real world experiments. Iterative optimization requires many of these evaluations to reach a satisfactory solution.

A prerequisite for similarity-based surrogate models is that a distance metric is defined for the encoding of a solution. Surrogate models are therefore usually applied to solution representations that encode a fixed number of parameters. Recently, more complex non-closed form encodings have been developed that do not have a constant input space. A prime example of such encodings are compositional pattern producing network (CPPN) (see Stanley (2006)), that encodes complex shapes or behaviors indirectly. In neuroevolution (see Stanley and Miikkulainen (2002)), the topology of neural networks can be evolved. Genetic programming (see Koza (1994)), evolves the topology of graphs, trees representing computer code, or mathematical equations. The non-uniform input space of these encodings frustrates typical ways of measuring distance as the dimensionality and even the meaning of these dimensions varies from one individual to the next (see Figure 4.10).

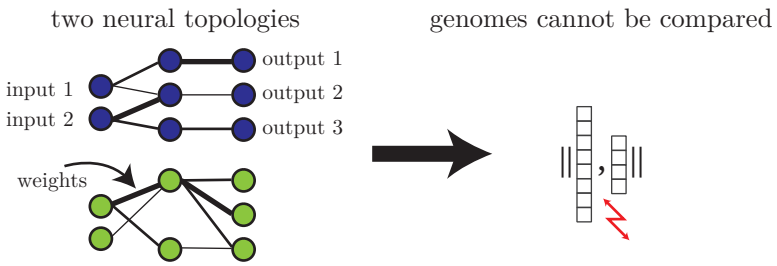

Figure 4.10: Two networks with different topologies cannot be compared based on their genomes.

A second problem arises when the quality of a solution depends on interaction with its environment. This behavior might vary greatly even if the parameterization of the encoding is only minimally changed. If a similarity-based model would be trained to predict the quality of such an encoding, a parameterization that is close to a training example would be assigned a similar fitness, although its actual fitness might be very different.

To enable surrogate-assisted optimization of neural encodings, the idea of measuring distances not of the encoding, the genome, but rather of their expression, the

phenotype, is investigated in this section (based on Hagg et al. (2019)). The phenotype may include morphological as well as behavioral aspects and can therefore give us more information about how similar two individual solutions are than the genome alone (see Stork et al. (2019)). The main insights are that (1) regardless of a network's internal composition, the size of the output in relation to the input is constant, and (2) the relation between input and output describes the behavior, and is thus a useful proxy for similarity between networks. To measure the difference in the behavior of two networks, the same input sequence is used on the networks. The difference in the output sequence can then be compared using a standard metric, like Euclidean distance.

By using randomly selected, but fixed input sequences, actual simulations are not necessary to get the output sequence. Instead, the input/output relation is sampled and uses the ad hoc difference in the output sequences of two individuals to measure their distance. This distance measure can now be used to build a similarity-based surrogate model.

4.2.1 Related Distance Kernels

Similar to phenotypic distances, semantic distances are used in genetic programming by Moraglio et al. (2012). These semantic distances can be defined as a distance of the outputs of population members, determined with the same measure that is used in the fitness function. Semantic distances are applicable where the fitness function can be computed as a distance between the optimal target vector and the candidate outputs, such as in supervised classification or symbolic regression. In these cases, the semantic distance has a fitness distance correlation of exactly one and can be utilized to construct specific mutation and crossover operators, rendering the problem uni-modal.

Phenotypic distances have also been employed in a surrogate modeling context. Hildebrandt and Branke (2015) suggested a phenotypic distance for dynamic job shop scheduling problems. Their definition of phenotypic distance compares individual solutions according to the results of evolved dispatching rules on a small set of test situations. Unlike semantic distances, their phenotypic distance is not identical to the measure used in the actual fitness function. This is necessary in the context of surrogate modeling for expensive fitness functions: if they are expensive to compute, it would also be expensive to use the same evaluation to compute a

distance between candidates. Such an approach would render the construction of the surrogate model itself expensive, which defeats its very purpose.

Zaefferer et al. (2018) compared different genetic and phenotypic distances for surrogate models in symbolic regression. Here, the underlying measure is not identical to that used in the fitness function. Specifically, the fitness function considers fixed coefficients in the symbolic expression. These coefficients are otherwise optimized during an actual fitness evaluation, which may become costly. In both of these cases, the phenotypic distance was reported to yield better results than genetic distances (see Hildebrandt and Branke (2015); Zaefferer et al. (2018)).

Doncieux and Mouret (2010) discussed the use of behavioral similarity in evolutionary robotics to employ a diversity measure for a multiobjective optimization approach. They compared different distances based on the states, outputs and trajectories given concrete robot tasks. The authors outlined that using these behavioral distances as a second objective in multi-objective optimization is able to enhance the overall performance.

A first approach utilizing a surrogate model for evolving neural networks given complex control tasks was discussed by Gaier et al. (2018). An evolutionary algorithm was combined with a surrogate model based on a hereditary distance, which is defined in the context of neuroevolution of augmenting topologies (NEAT) as *compatibility distance*. The approach is able to significantly improve the evaluation efficiency. Stork et al. (2019) also investigated surrogate models for neuroevolution. They examined simple classification tasks and compared a phenotypic distance measure to genetic distances in surrogate-assisted Cartesian genetic programming (CGP) (see Miller et al. (1997)). The use of a phenotypic distance was shown to be very promising in terms of evaluation efficiency.

4.2.2 Sampled Phenotypic Distance

The networks that are investigated here are results of optimization runs with fixed network topologies. This allows a comparison of the efficiency of models based on both genetic and phenotypic distance measures. To define a genetic distance the vector of weights of the neural networks is considered. Let $\mathbf{w} = [w_1, w_2, ..., w_j]$ be a weight vector of length j, then the genetic distance is calculated by the

related weights of two samples: $d(\mathbf{w}, \mathbf{w}')$, with d being an appropriate distance metric.

The disadvantage of genetic distance measures is their lack of applicability when changing topologies are considered. If in these cases no clear concept to compare genetic changes exists (as applied by Gaier et al. (2018)), the genetic distance comparison is difficult, misleading and even destructive (see Stork et al. (2019); Doncieux and Mouret (2010)). The ability to compare non-uniform topologies makes phenotypic distances a valuable technique, especially in cases when typical distances are not a viable option.

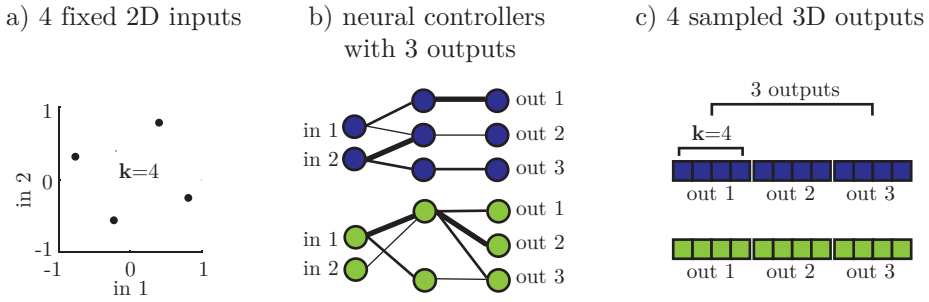

Figure 4.11: Sampling the phenotype with fixed inputs (a) to compare two individual networks (b). The fixed input strings are inserted into the networks to calculate the phd outputs (c).

The phenotype displays the behavior of a neural network given a certain set of inputs. For example, in the case of neural networks used as controllers for robots the phenotype can be defined as the control commands that are issued in response to different sensor inputs. Phenotypic distance is defined as follows: Let $\mathbf{s} = [s_1, s_2, ..., s_k]$ be the vector of inputs with length k, then $\mathbf{o} = [o_1, o_2, ..., o_{k \times z}]$ is the associated processed output vector, or phenotype, for a neural network with length $k \times z$, where z is the number of neural network output neurons. The phenotypic distance is employed by calculating the difference in the outputs of two samples: $d(\mathbf{o}, \mathbf{o}')$. Fig. 4.11 illustrates the sampling of phenotypes and Fig. 4.12 shows a comparison of both distances.

The phenotypic distance is always task sensitive, i.e., a comparison of two samples requires the definition of an adequate input vector $\mathbf{s}$. It needs to fulfill two requirements in the context of model-based optimization:

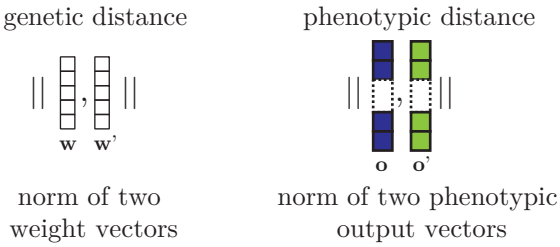

Figure 4.12: Weight models are based on weight vectors for fixed-topology networks. Phenotypic distance models are based on fixed-length sampled phenotypic output vectors for any-topology networks. The L1 norm (Manhattan distance) is used here for interpolative modeling.

a) The input should be representative for the underlying task, i.e., in case of robot control it should follow the given sensor ranges and/or depict a trajectory of states present in the task.

b) The dimensionality of the phenotype needs to be considered, the length of the input vector for generating the phenotypes might significantly affect the modeling performance as well as the computation time for querying the networks.

Given a carefully selected input vector, the phenotypic distance should be able to provide an impression of how the behaviors of two candidate networks compare to each other. A possible disadvantage of this definition of a phenotypic distance is that depending on the underlying task, the real behavior cannot be defined by the output of the neural network controller alone. This is in line with the discussion in Section 3.1.2. A full description of a behavior can only be obtained through its ecologic expression, by inserting a solution into its environment and measuring its entire influence and interaction with the environment. For example, a robot is influenced by the structure of the environment and its own body. Two robots with different controllers and phenotypes, one that uses four legs for movement and another that uses three legs, might behave the same if the fourth leg is disabled due to damage. This use case was introduced by Doncieux and Mouret (2010), utilizing an effect of neutrality.

The ultimate task is to obtain a representative set of samples of the input/output relationship which is descriptive enough to capture the behavioral differences and so allow the construction of surrogate models.

4.2.3 Evaluation

It needs to be determined, whether reasonable surrogate models can be trained based on a diverse set of phenotypic vectors, and whether the models have a comparable performance to genetic models. The main question answered in this evaluation is, whether we can correctly predict the pair-wise ranks of the performance of neural controllers. Due to the use of surrogates in an evolutionary context, it is only interesting to be able to correctly select the better-performing neural controller.

For this, model-free optimization algorithms are used that optimize the weights of fixed topology neural networks for robot control. Then, a QD archive of several hundred diverse neural networks is created based on these starting points. Different genetic and phenotypic surrogate models are then produced based on a subset of these networks, and their performance tested by predicting the performance of the remainder of the networks.

Robot controllers are designed for the multi-modal maze problem depicted in Figure 4.13. The environment consists of multiple rings and openings (Figure 4.13a). The robot begins in the center of the maze and needs to find a path to exit the maze. In the context of QD, it is not interesting to find the best solution to escape it. Instead, it needs to be established to what degree the behavior of neural network controllers can be sampled, and how accurate the derived performance model is. This problem is much more fundamental and difficult than predicting fitness alone. To produce this data set of as many different high-performing behaviors as possible, an archive consisting of robot controllers that reach every point in the maze in the shortest path possible (b) is created. To force a diversity of ending positions, a grid-like diversity measure is defined (c). At the end of the optimization, every niche should contain a robot that was able to reach it using a short path. This way the distance measures can be evaluated over a diverse set of optimal behaviors.

Simple feed forward controllers (see Figure 4.11) consisting of either two or five hidden neurons are sought that traverse the maze. Evaluation is performed using the simulation that was created by Mouret (2011a). The robot is equipped with three laser sensors that are able to detect the distance to the nearest walls, and are set at 45 degree angles around the front (d). In addition, each robot has a home beacon that detects the direction of its start position.

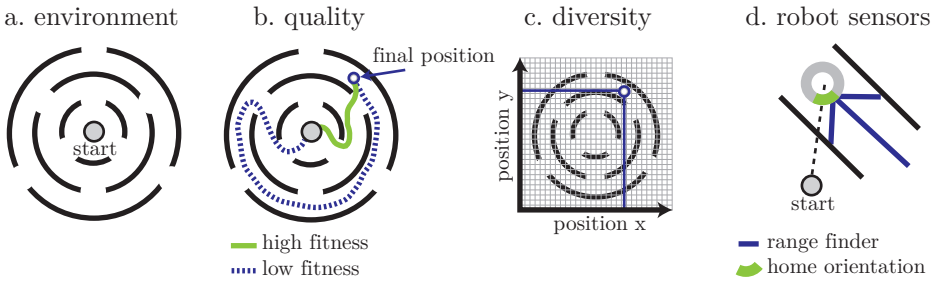

Figure 4.13: Evaluation takes place in a maze environment (a) with a robot starting in the center. The distance of the path of a robot to its final position defines its quality (b), whereby a diversity measure allows us to train robots to reach all niches in the archive (c). Robots can sense the orientation quadrant of the start position and use three range finders to perceive the distance to the nearest wall (d).

Data Generation Data sets are generated to test the quality of the surrogate models. To that end, the data of model-free optimization experiments is recorded. Here, optimization is performed with the MAP-Elites algorithm. It is not only used to find good solutions, but it is also intended to find as many diverse and high-performing solutions as possible. Here, niches are defined as bins in the grid shown in Figure 4.13.

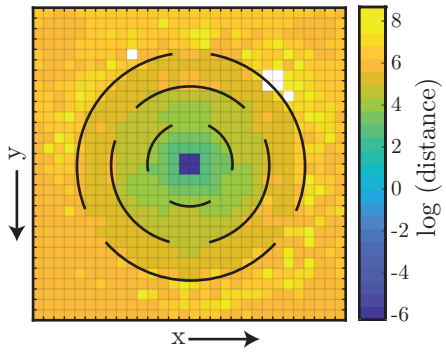

Figure 4.14: Distance archive generated by MAP-Elites (lower distance equals higher fitness). Each niche in the archive contains a robot controller that is optimized towards reaching that niche via the shortest path possible.

Figure 4.14 shows an example distance archive after 5000 generations, with almost every niche filled with a high-performing controller. The distance values naturally

grow the further they are from the center. A number of controllers end up driving around the maze in circles, which explains the high distance values in some niches.

For this experiment, 20 different MAP-Elites runs were produced for each experiment configuration, each with a different random number generator seed. This leads to 40 data sets (20 for each number of hidden neurons). Each data set is produced by roughly 900 neural network controllers. For each of those, nine different data sets were created: one with the weights, and eight with phenotypes of different sizes (4, 8, 16, 32, 64, 128, 256, 512). Note that the phenotypes are derived from the two outputs of the networks, that is, if the network is fed with four input samples, eight phenotype values are observed. Each of the 900 controllers can now be described either by its weights, or by a phenotype output vector.

The nine different data sets are now used to model the (ranked) performance of neural controllers. Again, it is only of interest to correctly predict the pair-wise ranks of the performance in this evolutionary context. During modeling, the data sets are split as follows: 400 controllers are used to train a model, the remainder is used to test the model quality. It must be considered that the observed values y will be log-scaled before modeling, as the data contains strong outliers which might deteriorate the models.

GP Model The GP model is created using the R-package CEGO (which can be found at Zaefferer (2019)) as follows. For MLE, the optimization of the likelihood is performed via the locally biased variant of the dividing rectangles (DIRECT) algorithm, which was proposed by Gablonsky and Kelley (2001). It is configured to stop after 2000 likelihood evaluations, or when the relative decrease in function values between iterations drops below 10^{-16}. Regularization of the model is turned on, to potentially account for noise in the data or ill-conditioned kernel matrices. The model uses the Manhattan distance.

Comparison Baseline: Linear Model A linear regression model is included as a benchmark for the GP model. Like the GP model, the linear model is trained with the genetic or phenotypic data. Since the generated data is potentially very high dimensional, some form of variable selection is needed to generate reasonable models. A forward selection approach via the Aikake information criterion (AIC) (see Venables and Ripley (2002)) fulfills this need, starting from a model that only

consists of an intercept. The most complex linear model may include main effects for all variables, but no interactions or higher order terms are considered.

Results and Discussion To judge the quality of the models, the rank correlation coefficient by Kendall and Gibbons (1990) is used. Figure 4.15 shows the Kendall correlation achieved by each of the models. Firstly, it can be observed that the GP model outperforms the linear model in most cases, as expected. Secondly, the variants based on phenotypic data are able to perform at least as well as the weight models, if the number of elements in the phenotype vector is at least 32 or more. The larger phenotype vectors do not seem to yield much further improvement.

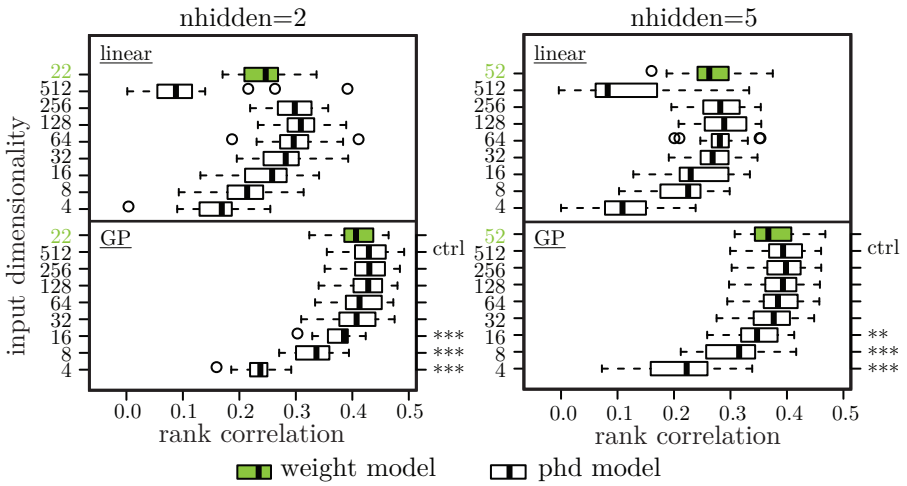

Figure 4.15: The model quality in terms of correlation (x-axis), for linear and GP models and different input spaces, and different numbers of hidden neurons (nhidden). Here, the numbers at the start of each y-axis label denote the dimensionality of the input vector for the corresponding model (number of weights or length of the phd sampling string). Green fill color indicates a model based on the weights or genome, the white fill color indicates phenotypic models. The y-axis labels on the right-hand side indicate p-values from a statistical test that compares each of the GP models against the model marked with ctrl (*: $p < 0.05$,**: $p < 0.01$,***: $p < 0.001$).

The observations are confirmed by applying statistical tests for each number of hidden neurons. A first experiment is run to evaluate the global presence of significant differences via the non-parametric rank-sum test by Kruskal and Wallis

(1952), which yields p-values of less than 10^{-8} in both cases, indicating that differences are present. Afterwards, the non-parametric many-to-one comparison test by Conover and Iman (1979) is performed, comparing each of the GP models against a single model, the control group. The chosen control group is the most complex model with phenotype data of dimensionality 512. The implementations of the employed tests are taken from the `stats` and the `PMCMRplus` `R` packages (see R Core Team (2018); Pohlert (2018)): `kruskal.test` and `kwManyOneConoverTest`. The respective cases with indications for significant differences are marked on the right-hand side of each plot in Figure 4.15. The statistical test largely confirms the visual evaluation. No evidence for differences is found between the control group and the model with the genetic weight data. Only models with phenotypic data of a dimensionality of 16 or less is deemed to be different from the control group.

Importantly, the results suggest that phenotypic surrogate models can be used instead of those based on the genome. The phenotypic data is largely unaffected by the number of hidden neurons, and, hence, the number of weights. Where standard models would struggle to compare the weights of differently structure networks, a phenotypic comparison would still be possible.

The baseline linear model shows some peculiar behavior. The model's performance drops off for models with phenotype vectors of more than 256 elements. This behavior can be largely explained with the number of coefficients selected by the AIC forward selection procedure, as shown in Figure 4.16. Clearly, the selection procedure will not select more than n variables. The required number of variables seems to increase non-linearly with the dimensionality of the data.

Notably, the GP model does not show such a performance drop, and in fact performs quite well even for the very high dimensional phenotype vectors. This may be counter-intuitive at first: GP regression is usually not suggested for high-dimensional data. But there are two possible reasons that could explain this behavior. Firstly, an isotropic model was used, which avoids the complex optimization of fitting numerous kernel parameters (θ). Secondly, there may be a correlation in the observed phenotypes. Increasing the number of samples used to generate the phenotype vector will not only increase the dimension, but also the density in the sampled space. In that sense, a new phenotype observation is likely to be quite similar to the large set of existing observations it is added

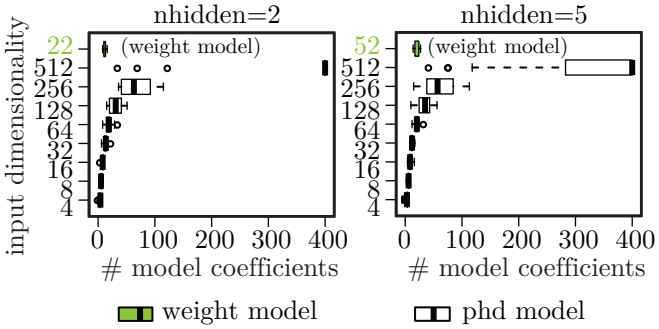

Figure 4.16: The number of linear model coefficients selected via forward selection based on AIC. Green fill color indicates a model based on the weights or genome, the remainder are based on phenotype data.

to. Essentially, it is assumed that the latent dimensionality of the data is much lower.

To verify this, a principal component analysis (PCA), introduced by Pearson (1901), of the input data is considered (that is, excluding the dependent variable). For each of the data sets, a PCA was performed on the weight data, as well as on the phenotype data. In each case, the number of principal components required to explain 90% of the variation in the data set was recorded. This number is shown in Figure 4.17.

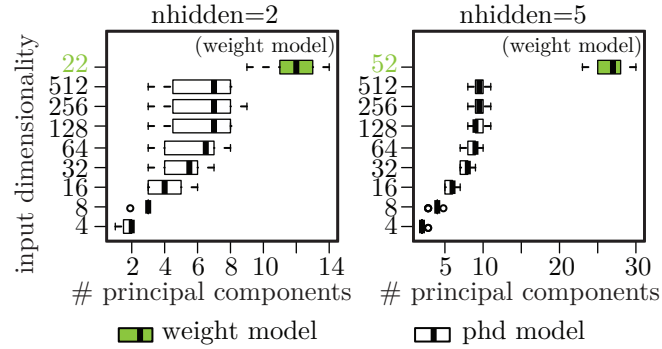

Figure 4.17: For each data set, the number of principal components required to explain 90% of the variation in the data. This only concerns the respective input data of the surrogate models, the observed output (i.e., quality of the controller) is not considered here. Green fill color indicates weight or genome data, the remainder is based on phenotypic data.

There are two interesting observations here. Firstly, the number of components levels off for the largest phenotype vectors. The median stays at seven (nhidden = 2) and nine (nhidden = 5), despite data sets with several hundreds of variables. It seems that this confirms the assumption that the additional columns due to higher-dimensional phenotype vectors actually describe a much lower-dimensional, latent space. Secondly, the number of principal components for the weights are much larger, yet, this does not coincide with better models based on the weight data.

4.2.4 Conclusions

This section gave evidence for an answer to research question VI ("Can we model neural encodings' behavior ex situ by sampling their outputs and using a behavior-kernel?"), fulfilling requirement C2 ("Search should efficiently sample complex design spaces") for neural encodings that are highly non-linear, sensitive and neutral. Phenotypic data, sampled from neural encodings, can serve as a kernel for surrogate modeling. Models based on phenotypic data can perform at least as well as those based on genetic data. This holds both for a baseline, linear model, and a non-linear GP model. The analysis further indicates that even high dimensional phenotypes with several hundreds of observations can yield sound models. A principal component analysis reveals that these high-dimensional data sets can be very well reproduced with only very few components. A much larger number of components is required for the genetic data.

This success of a phenotypic model is promising. A model based on genomes becomes infeasible if the compared networks have different structures or topologies. This can happen in the context of evolutionary algorithms that change the structure and size of the solution encoding, e.g. in surrogate-assisted neuroevolution. Measuring behavior of neural networks without using actual simulations not only seems to be possible, but also a practical way to compare networks.

Phenotypic distances can be used successfully as kernels to build surrogate models that predict the fitness of networks with varying sizes and topologies. Whereas previous approaches to construct surrogate models of neural networks with non-uniform structure rely on the peculiarities of the evolutionary algorithm (see Gaier et al. (2018)), the presented work is independent of the optimization approach. In fact, a phenotypic distance approach to modeling is independent even of encoding:

a neural network grown with NEAT, a fixed topology network optimized with particle swarm optimization, and a controller evolved with genetic programming could all share the same surrogate model.

As the PCA showed, as well as the diminishing returns for models with more phenotype samples, a lower-dimensional data set may suffice to produce good models. Creating better, more condensed phenotype samples with less redundant information is hence of interest for future work, to reduce the load of distance calculations.

Being able to successfully model the performance of a neural encoding by observing its behavior provides a computationally efficient and effective approach for surrogate modeling of varying-length representations. Modeling the behavior of networks avoids some complexities that are caused by genetic comparisons. Surrogate-assisted optimization of non-uniform representations allows a much more diverse set of solutions to be calculated with a limited number of real evaluations.

4.3 Chapter Summary

This chapter introduced a method that increase the efficiency of QD algorithms when using expensive phenotypic features for niching. It was shown that behavioral airflow features can be learned by piggybacking on Bayesian optimization, acquiring samples only according to uncertainties in the prediction of their performance. The resulting surrogate-assisted QD method, SPHEN, needs orders of magnitudes less exact evaluations than methods that do not or only partially use surrogate-assistance. This answers research question V ("Can we model behavioral features in a surrogate-assisted way by sampling based on optimality alone?"). Efficiently generating phenotypically diverse solution sets is necessary to the application of divergent search in problem domains common to engineering.

Furthermore, neural representations were modeled through their behavior using phenotypic sampling. This method allows building kernels for GP models based on behavioral distance. Although the application of phenotypic distance-based kernels has to be analyzed in future work, evidence was given for a positive answer to research question VI ("Can we model neural encodings' behavior ex situ by sampling their outputs and using a behavior-kernel?").

Both methods enable phenotype-based multi-solution optimization methods to be used as an efficient divergent component in co-creative processes.

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.

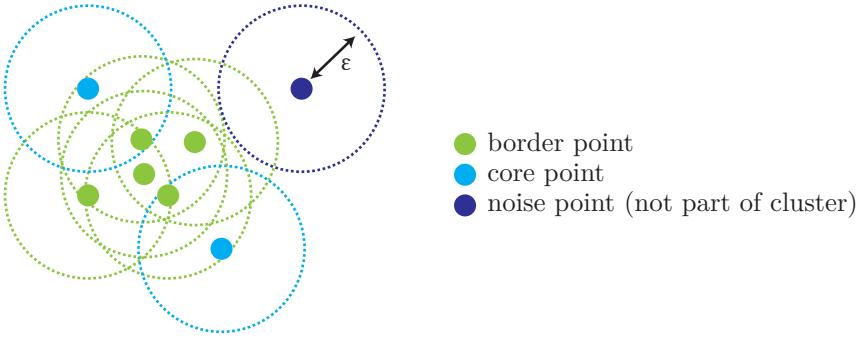

Figure 5.1: Density-based spatial clustering of applications with noise, introduced by Ester et al. (1996). The minimum number of neighbors n for a core point is four.

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

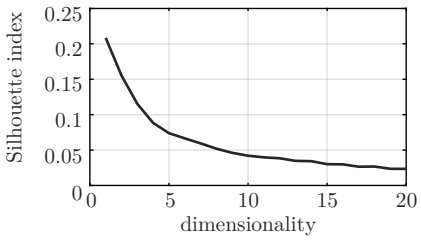

Figure 5.2: Silhouette index for a random cloud of samples in 1D to 20D. Two fixed Gaussian clusters sampled with diagonal covariance, one unit (in one dimension) apart from each other were used.

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/`

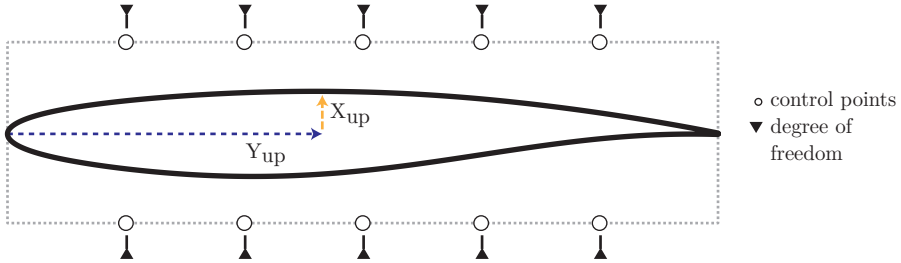

Figure 5.3: Two-dimensional airfoil encoding and features (X_{up}, Y_{up}).

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.

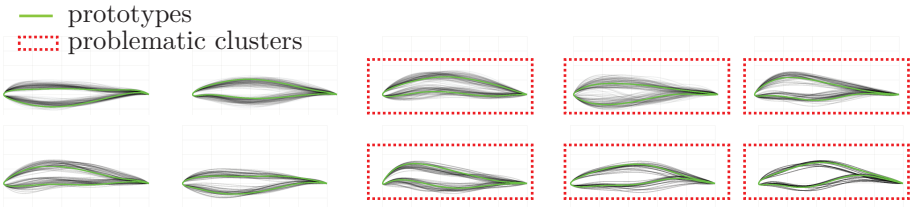

Figure 5.4: Example set of clusters found using t-SNE and DBSCAN on the airfoil domain. Prototypes are shown in green. Red boxes show problematic clusters, due to the wide variety of shapes contained within the cluster.

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

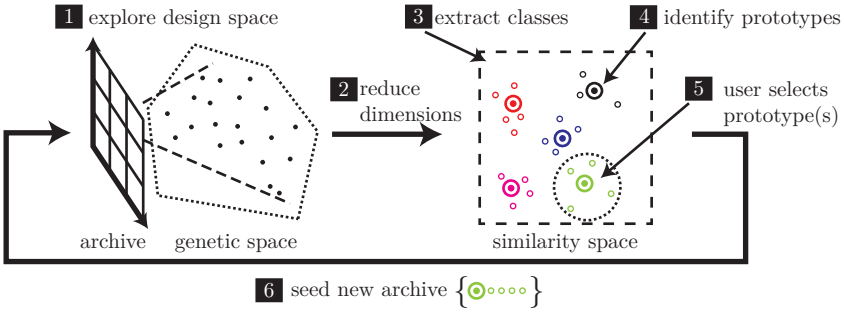

Figure 5.5: PRODUQD cycle. The design space is explored with QD (1). The archive's members' genomes are projected into a low-dimensional similarity space (2). In this space, classes are extracted (3) and one member of each class is chosen as a representative prototype (4). The user then selects one or 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: $\mathcal{X} \leftarrow \text{SOBOL}(dim(\mathcal{X}))$
3: $\mathbf{f} \leftarrow \text{EVALUATE}(\mathcal{X})$
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: $\mathcal{X}_{red} \leftarrow \text{DIMENSIONALITY-REDUCTION}(\mathcal{X})$
8: $\mathbf{c} \leftarrow \text{CLUSTERING}(\mathcal{X}_{red})$
9: $\mathbf{p} \leftarrow \text{PROTOTYPE}(\mathcal{X}, \mathbf{c})$
10: $\mathbf{p}_{sel} \leftarrow \text{SELECT}(\mathcal{X}(\mathbf{p}), \mathbf{f}_{pred}(\mathbf{p}))$ ▷ 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.

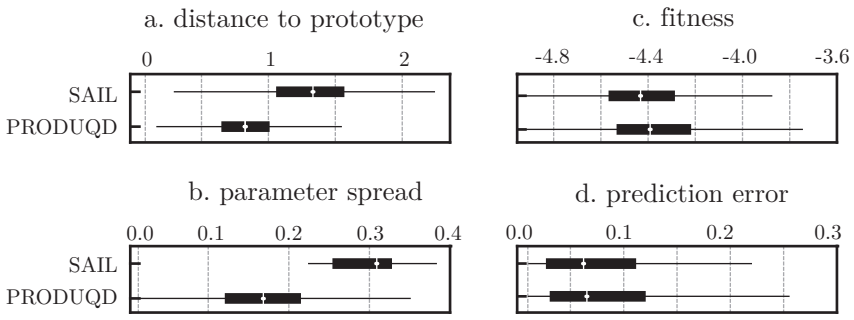

Figure 5.6: PRODUQD produces designs that are more similar to the selected prototype than using SAIL (the distance to prototypes is smaller). This is also visible in the smaller parameter spread. The produced designs have similar performance compared to SAIL's and the surrogate model is equally accurate.

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

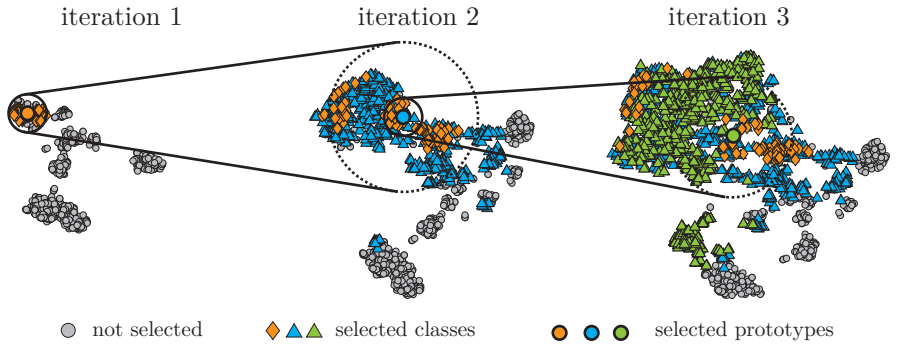

Figure 5.7: The region around the selected class is enlarged in similarity space and structure is discovered as more designs are added in later iterations. In each iteration the archive is filled with solutions from the selected class.

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.

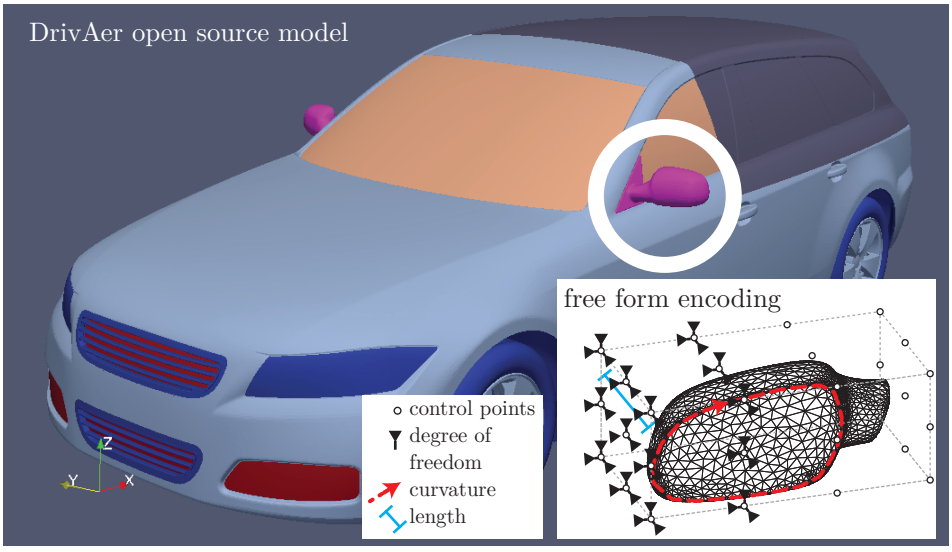

Figure 5.8: Three-dimensional mirror encoding and features (curvature and length).

[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.

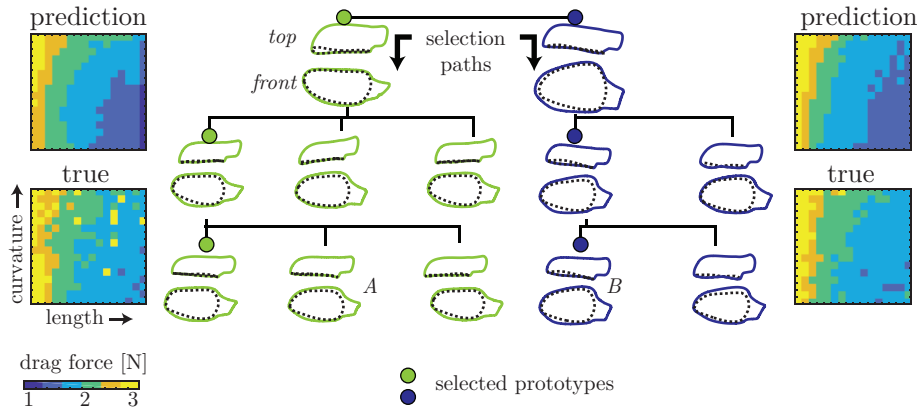

Figure 5.9: Phylogenetic tree of two PRODUQD runs diverging after first iteration, and predicted drag force maps (ground truth values are shown underneath).

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.

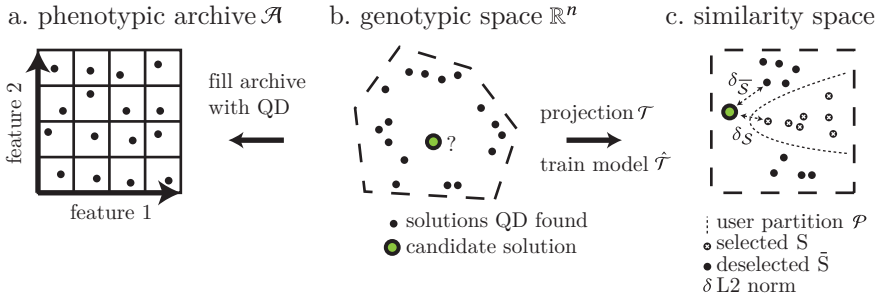

Figure 5.10: QD searches through genetic space $\mathbb{R}^n$ (b) to fill an archive $\mathcal{A}$ of diverse, high-performing phenotypes (a) in a low-dimensional phenotypic (or behavior) space. The genetic dimensionality n can be very high. By projecting the archive's members onto a low-dimensional 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.

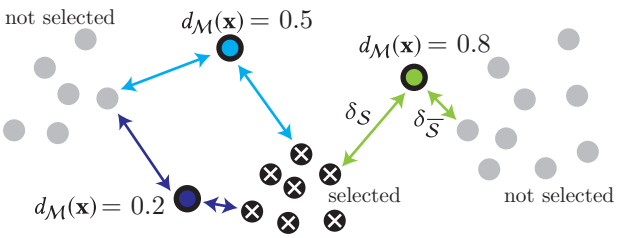

Figure 5.11: User selection drift $d_{\mathcal{M}}$ is based on the distance to the closest selected point $\delta_{\mathcal{S}}$ and the distance to the closest deselected point $\delta_{\overline{\mathcal{S}}}$.

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: $\mathcal{X} \leftarrow$ SOBOL($dim(\mathcal{X})$)
3: $\mathbf{f} \leftarrow$ EVALUATE($\mathcal{X}$)
4: $budget \leftarrow budget/iterations$
5: **for** 1 to $iterations$ **do**
6: $(\mathcal{X}, \mathbf{f}_{pred}) \leftarrow$ QD($\mathcal{X}, \mathbf{f}, budget$)
7: $\mathcal{X}_{red} \leftarrow$ DIMENSIONALITY-REDUCTION($\mathcal{X}$)
8: $\mathbf{c} \leftarrow$ CLUSTERING($\mathcal{X}_{red}$)
9: $\mathbf{p} \leftarrow$ PROTOTYPE($\mathcal{X}, \mathbf{c}$)
10: $\mathbf{p}_{sel} \leftarrow$ SELECT($\mathcal{X}(\mathbf{p}), \mathbf{f}_{pred}(\mathbf{p})$) ▷ Performed by the user.
11: $\widehat{\mathcal{T}} \leftarrow$ 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

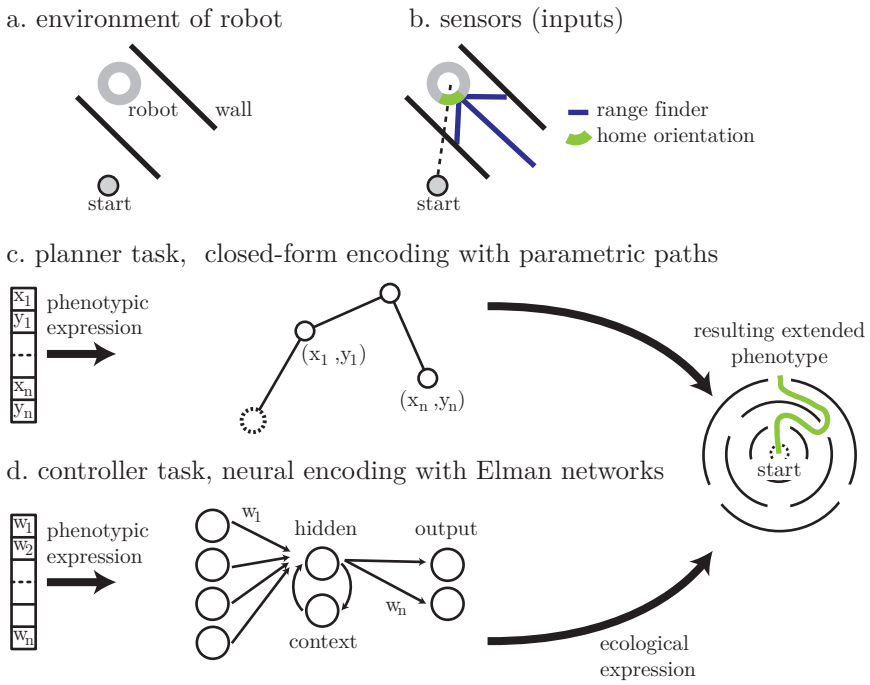

Figure 5.12: Closed form (c) and neural encoding using an Elman network (d) of maze problem (a, b) with the same (extended) phenotypic space.

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.

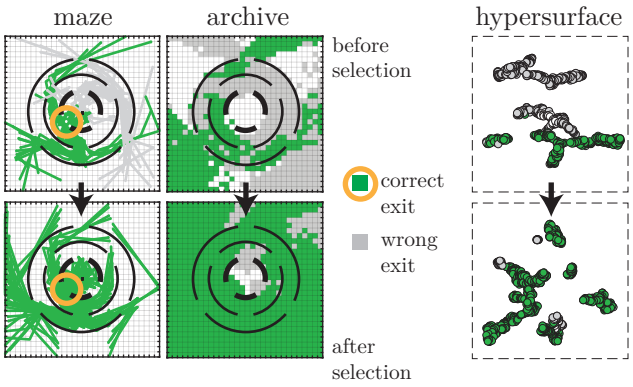

Figure 5.13: Path planning task with closed form encoding before (top) and after (bottom) selection of the 3rd exit. Left: example paths (every 5th). Center: end points of paths in QD archive with color assigned depending on whether exit was selected. Right: points projected onto the decision hypersurface.

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.

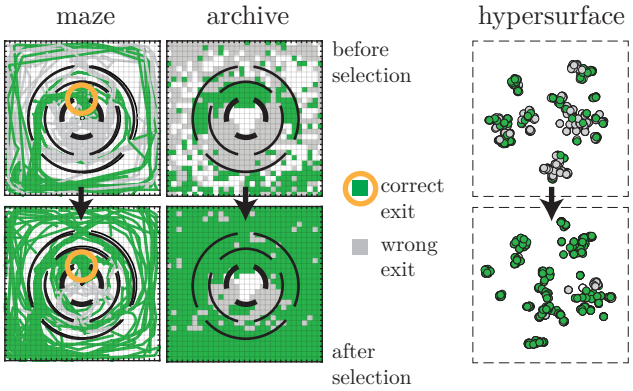

Figure 5.14: Neurocontrol task with neural encoding before (top) and after (bottom) selection of the 1st exit. Left: example paths (every 5th). Center: end points of paths in QD archive with color assigned depending on whether exit was selected. Right: points projected onto the decision hypersurface.

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.

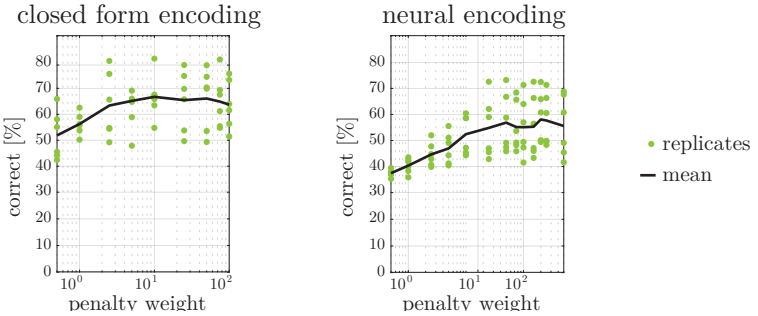

Figure 5.15: Influence of drift penalty weight on percentage of correct exits for the closed form and neural encodings. Three exit selections were replicated six times, totaling 18 runs.

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.

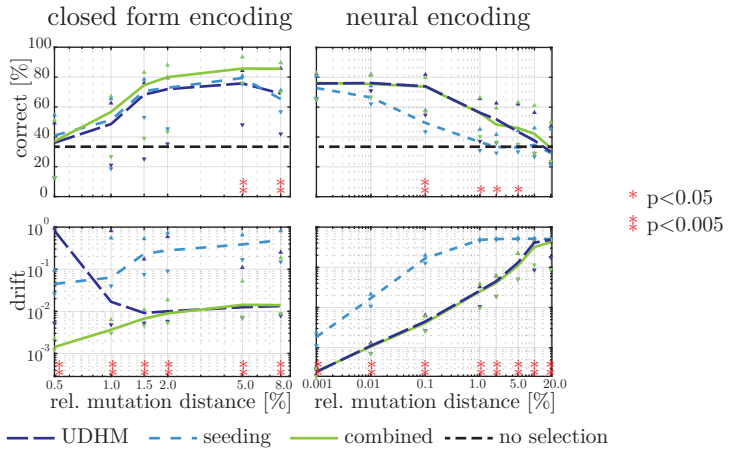

Figure 5.16: Median percentage of correct solutions and median user selection drift in both tasks using UDHM, the seeding method and a combination of the two. The baseline (dotted gray lines) uses no selection. The triangles show the 25%/75% percentiles. Significance results from a two-sample Kolmogorov-Smirnov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

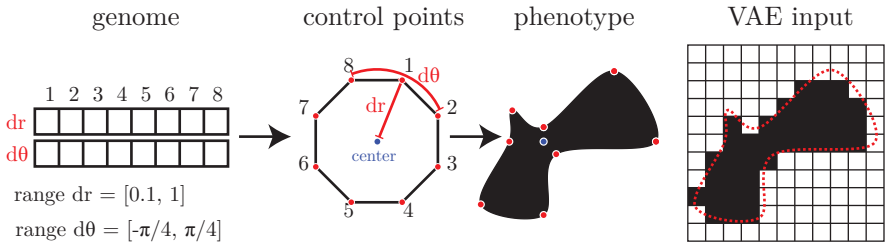

Figure 5.17: Shapes are encoded with 16 polar control points that get transformed into the final shape using locally interpolating splines. The shape is then discretized to serve as the input to the latent model.

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

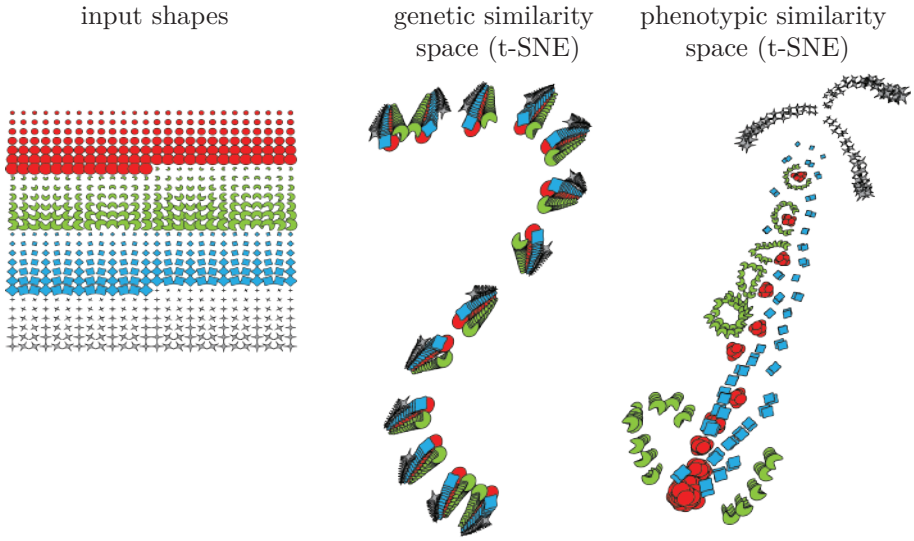

Figure 5.18: Genetic (16D) and phenotypic space (4096D) show the effect of neutral and sensitive, non-linear mappings on a simple polygon encoding domain.

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

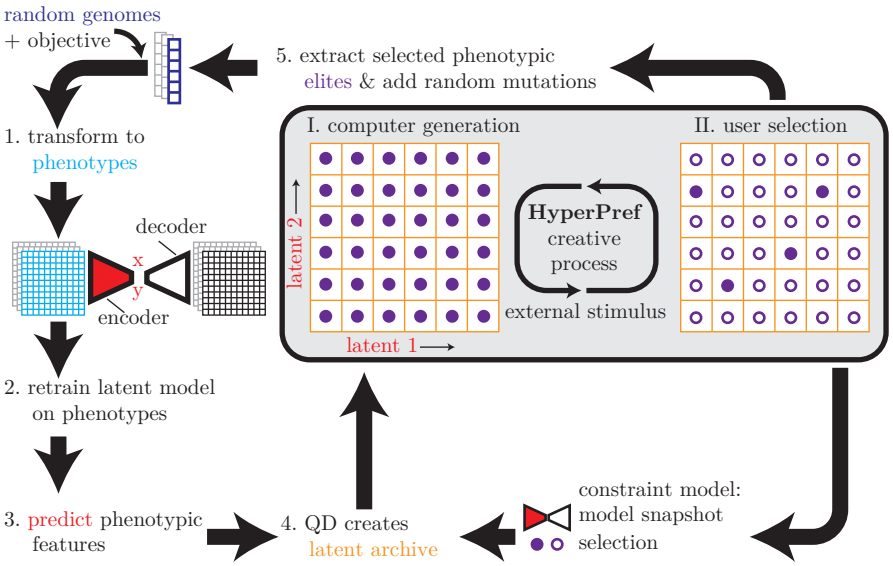

Figure 5.19: Discovering the preference hypervolume with HyperPref. The co-creative process (gray box) consists of computer-generated solution sets (step I) that are influenced by the users' selection (step II). A latent model is trained on the phenotypes (1, 2) of a random set of solutions. The model 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.

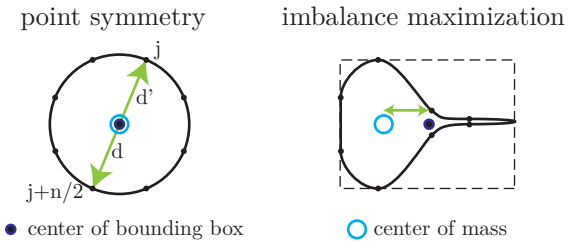

Figure 5.20

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.

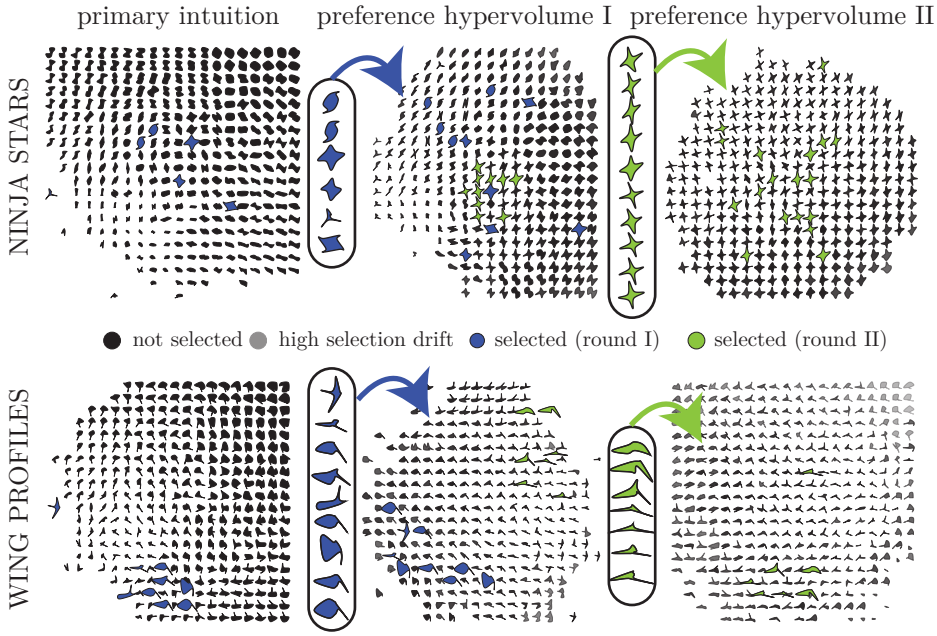

Figure 5.21: Top: starting with centrally symmetric shapes to design ninja stars. Bottom: starting with unbalanced shapes to design wing profiles.

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.

CHAPTER 6

In Conclusion

Creative processes, especially in engineering, suffer from the problem that user preferences are not easily formalizable. Creativity is suppressed when engineers are very experienced, causing them to become less inclined to try novel solutions. In contrast, a lack of experience makes it nearly impossible to find solutions at all but leaving the task of creation to machines takes away responsibility from us humans, which is not practical and raises many ethical questions.

Co-creative processes between humans and artificial intelligence can help to inspire beyond the user's own intuition. This work realized co-creative processes between humans and an artificial agent based on state of the art methods of evolutionary optimization and machine learning. By measuring similarity and diversity based on a solution's expression or behavior, multi-solution optimization creates a highly diverse set of solutions, both in their morphology as well as their interaction with the environment. The latter, whether it is a robotics or fluid dynamics simulation, determines not only whether a solution has a high quality but also offers alternatives when a quality metric is hard to specify. The co-creative system is able to create diverse solution sets in an efficient manner and capture human preferences.

Chapter 1 discussed that humans need to keep control of decisions made by AI systems. It grounded this thesis in the tension field between the Hegelian idea of creativity and engineering design. The communication paths of Jung's extraverted intuition, as opposed to introverted intuition, served as an inspiration. Increasing transparency by externalizing the creative process is one way to increase the explainability of decisions where ultimately AI systems are involved.

Chapter 2 analyzed formalisms of human creativity. The components of co-creative stage theories were connected to cognitive psychological traits, affecting human creativity. This analysis determined, which components of the process could be turned into an artificial system. The considerations, merged with those from literature, were converted to a list of requirements for the process. State of the art co-creative processes were then described. Finally, a formal model for a co-creative system based on divergent search methods was defined. The model supports humans with varying degrees of experience.

Chapter 3 connected ecologic effects in natural evolution to evolutionary multi-solution optimization. With the first research question (I) in this chapter, it was determined that solutions are best compared using their phenotype or behavior, not their genome. Niching methods that are common in evolutionary computation were discussed. Only by maintaining a collection of solutions can we differentiate between solutions' similarity. A description of genome, phenotype and behavior in natural evolution and their effects on diversity were discussed. These concepts were then projected back into the encodings used in evolutionary computation. Specifically, the terms phenotypic and ecologic expression were distinguished. The effects of neutrality and sensitivity of such multi-level encodings provided an answer to the research question. It is the phenotype, specifically morphology and behavior, that hold the key to a concept diversity that is easily understood by humans. QD algorithms fill a gap in evolutionary computation, by providing a mechanism for 'indirect, high-level recombination'. Values for phenotypic features, found in known solutions, are recombined in QD's archive to produce new solutions, but without the necessity of an explicit recombination operation.

Answering the second research question (II), it was determined that QD algorithms produce the highest phenotypic diversity compared to other multi-solution optimization methods. Three main multi-solution paradigms, multiobjective optimization, multimodal optimization and phenotypic niching (the main contribution of QD algorithms), were compared. A simplified archive was introduced that made it possible to compare genetic and phenotypic niching. QD was shown to create a much more diverse set of solutions. By not explicitly defining phenotypic features, and instead learning them from the data set generated by QD, solution diversity was increased even more. The treatment of the first research question already

provided insight into why QD is such a powerful method and the answer to the second question confirmed this.

The next part in this chapter treated the question (III) whether we can learn more diverse solution sets when learning phenotypic niching from data instead of using predefined features. By taking advantage of convolutional autoencoders, phenotypic space could be compressed into two dimensions, increasing QD's solution set diversity. This removed the necessity for users to predefine phenotypic features, which can be difficult to determine as well as be prone to be influenced by a human creator's biases. This data-driven phenotypic niching approach, named Auto-Voronoi-Elites (AutoVE, see Fig. 3.15), has never been applied to shape optimization before. Indeed, data driven techniques lead to more diverse sets of artifacts.

The data-driven approach was further analyzed by asking what the limitations of generative models are in terms of the possible diversity of the solutions they create (research question IV). It turns out that generative models are better used as similarity models, providing niching spaces for multi-solution algorithms, rather than be used as an encoding. In this work only bootstrapped techniques were evaluated, to prevent initial biases and to simulate the problem of initial creativity. It was shown that generative models are usable to create a configurable resonance-dissonance trade-off, by evaluating their expressiveness when extrapolating away from known solutions. This extrapolation is only possible when using the models as niching space, not as search space.

Chapter 4 introduced two methods to help increase the data efficiency of QD algorithms. First, the question (V) was answered, whether behavioral features can be modeled in a surrogate-assisted way by sampling based on optimality alone. By connecting Bayesian optimization applied to QD with a model for phenotypic diversity, it was shown that the optimal hypersurface can be modeled, both in terms of its coordinates in phenotypic similarity as well as solutions' quality. Only now a full surrogate-assisted version of QD exists.

The second question (VI) treated a special case of neural encodings. An evaluation was performed to determine, whether the behavior of artifacts, robots in this case, with neural encodings can be modeled. It was shown that simulation-free behavioral prediction was possible by sampling outputs of neural networks and using this in a behavioral kernel for statistical models. This method removes the need for

most evaluations of such an encoding and allow comparisons between solutions with different structures. The hope is that this helps to more easily choose such an encoding, which can create very complex morphologies and behaviors.

Chapter 5 discussed various ways to integrate QD into the co-creative process, using the design by shopping paradigm, where the user selects solutions presented to them. Results need to be presented to the user in a concise way, as QD can create very large solution sets.

First, the question (VII) whether QD results can be summarized using representatives was answered. Evidence that QD solution sets often consist of genetic clusters allowed compressing them into a smaller set of representative prototypes. An unsupervised clustering method combined with dimensionality reduction allowed such a compression of the set into a small number of dimensions. The set could now be presented to the user as prototypes.

Second, the prototypes were used to influence QD after humans select their preferred ones. This influence was analyzed for two domains in research question VIII. After the user selects their preferred solutions, QD is restarted from the set of selected prototypes. QD runs based on these 'seeds' produce solution sets that are more similar to the user's preferred prototypes. QD tends to drift away from the preferred region in genetic space.

Therefore, an evaluation was performed to determine, whether selected prototypical genomes can be modeled with a statistical model (IX), and whether we can constrain the search space by penalizing the objective function (X). Gaussian process models were created to predict the coordinates of new solution in the similarity space. This allowed defining a selection metric, which compares how far a solution has drifted from preferred solutions as opposed to non-preferred solutions. The metric was turned into a penalty and applied to QD's objective function. This biased QD to move towards the user's preference hypervolume, the space that contains preferred solutions. Two different encodings in the same domain were used to show that the drift penalty was able to constrain the solutions, genetically. However, due to the fact that the constraints are only applied to parameters, or genes, not to the phenotype, QD still produced unexpected solutions, especially when an encoding contains a high amount of neutrality and sensitivity.

Therefore, finally, the constraints were applied on a phenotypical level (XI). By modeling the high-dimensional space of pictures of geometrical shapes with a convolutional variational autoencoder, a phenotypic similarity space was created. The phenotypic drift penalty was applied to QD, creating the final co-creative process called HyperPref, in which the phenotypic preference hypervolume can be discovered. This was demonstrated on a 2D spline domain.

Introduced Methods. The following table shows an overview of all methods introduced in this work.

Table 6.1: Method Overview.

Method	Description
VE	Produces diverse set of high performing solutions using archive defined by phenotypic features, fixed number of niches, independent of archive dimensionality and without fixed niche boundaries.
AutoVE	VE adaptation that automatically extracts archive descriptors by bootstrapping and updating a generative phenotypic model.
SPHEN	Bayesian interpretation of MAP-Elites that uses GP models and a sampling strategy to fill an archive with only a small number of real objective evaluations. Predicts archive descriptors while piggybacking on sampling strategy.
PHD	Distance kernel/metric based on phenotypes.
UDHM	Preference model.
PRODUQD	Interactive version of QD. User influences QD by selection of archive members. Influence is imposed using a penalty applied to the objective based on genetic drift.
HyperPref	Version of PRODUQD that influences QD based on phenotypic drift.

Final Words This work gave insights into the matter of creativity in engineering, the importance of phenotypic diversity, reflecting on the formulation of optimization algorithms for the real world, bringing the user back into the loop, learning what the user wants, but providing as much creativity as possible within the user's selection. An informed and challenged engineer can make better design decisions and in this context we can and should develop algorithms.

Creators can use the Hegelian co-creative methods introduced in this thesis to discover their preferences, supported by a 'creative' machine. The machine helps humans interactively reflecting upon themselves as an objective participant, an artificial Other, a concept that is necessary, according to Sartre and Elkaïm-Sartre (1946). It helps to shape a process that is in line with the idea on creativity that was described by Hegel (1842) and uses communication paths used in extraverted intuition, the term coined by Jung (1923).

The formal model and the implementations of this model provide input for generative learning as well as the active learning community, combining concepts from both. The model can potentially support all levels of user expertise. The contributions of this work consist not only of answers to the research questions, but also of a body of methods that was shown on various peer reviewed conferences, in the community of evolutionary computation as well as computational creativity.

Surely, this work can be continued ad infinitum. The combination of QD and generative models is actively being explored in the scientific community. The aspects of interpolation and extrapolation w. r. t. known examples and features of those examples might provide us with insights on the true strength and limits of generative models. Although they produce convincing results, the question remains whether in the end they limit our possibilities. Only using data-driven solution generators might in the end hinder, not advance engineering solutions.

Cooperation between humans and artificial intelligence begs the question who or what bears the responsibility to take decisions that impact our society. Computational co-creativity allows us to evaluate these questions that can have a large impact on a society that depends more and more on AI, and less on human responsibilities. Yet, in the end, humans are the agents responsible for decisions (see Sartre and Elkaïm-Sartre (1946)).

To answer the question posed in Section 2.3.2, in contrast to the 'parameters tell the design story' hypothesis by Bradner et al. (2014), the results from this thesis quantitatively and qualitatively show that design stories, in terms of diversity, are told by examples (see Sartre and Elkaïm-Sartre (1946)) and their phenotypical features, not parameters. The objective of divergent search methods should be to provide new insights to the engineers. Parameters provide a computational search space but also contain implicit biases. Only a continuous analysis of phenotypes and behavior of solutions can provide new insights in an accessible manner.

I believe that in AI and optimization research, a larger emphasis should be put on the *process* of creativity and optimization. The optimization community oftentimes relies on the assumption that we can fully formalize our objectives and preferences. I do not believe this to be true. Instead, we might loosen this assumption and design methods, algorithms, and processes, that instead of giving us solutions, give us insight into the problems we try to solve and our preferences that grow with those insights.

APPENDIX A

Evolutionary Algorithms

This appendix is based on previous published work by Asteroth and Hagg (2015). In EA, an approximative solution to an optimization problem is iteratively found. In each iteration, a set of solution candidates, called a *population* is maintained in a stochastic procedure. This procedure consists of 4 basic steps (see Figure A.1):

1. *evaluation* – assignment of a real valued *fitness* to the candidates
2. *selection* – survival of the fittest candidates
3. *crossover* – recombination to produce *offspring*
4. *mutation* – randomized changes to the individual offspring

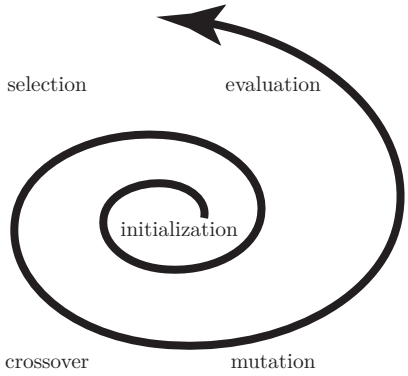

Figure A.1: Four basic steps of an EA.

In this repeated procedure the balance between *exploitation*, the refinement of

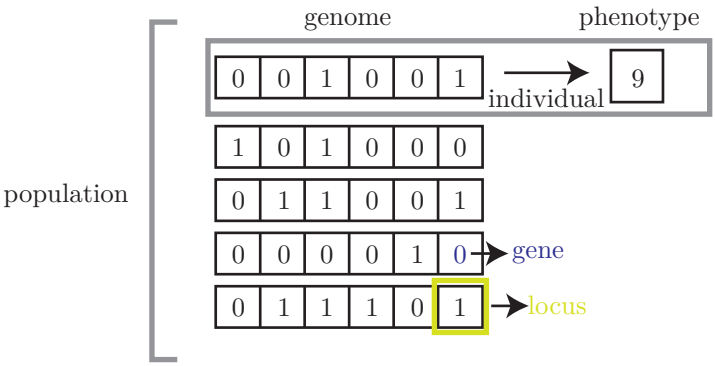

Figure A.2: Structure of the population

individuals with high fitness, and *exploration*, the search for new solutions, possibly through a part of the search space that has lower fitness, plays a crucial role.

Members of the population are called *individuals* (see Figure A.2). Each is represented by its *genotype*, which here can be considered a one-dimensional array. The array's cells are called *loci* and their value a *gene*. All genes of the individual make up its *genome*, an instance of the genotype. Before the individual is assigned a fitness the genome is first mapped to its *phenotype* (1:1 or n:1 mappings are possible).

Individuals from the current population survive and breed based on their fitness and therefore the fitness function must provide sufficient information to direct the search towards an optimum. Formally, the fitness function is a function from phenotype space $\mathcal{P}$ to the positive real numbers $\mathbb{R}^+$.

$$f : \mathcal{P} \to \mathbb{R}^+$$

While in classical optimization algorithms, search is driven by the gradient of an error function, in EA a gradient is not necessary but fitness must still represent some distance to the solution.

The exploration rate plays a crucial role for the *convergence* speed and approximation quality. During the selection process, exploration happens if individuals with submaximal fitness survive. If on the other hand exploitation is high, the search will converge very fast but will get stuck in local extrema. This problem does not occur when exploration is high, but the convergence speed will then be much lower.

A.1 Selection

A balance between refining good individuals and nurturing promising ones has to be found. Good solutions are often combinations of *building blocks* found in less fit individuals, therefore diversity plays a central role. On the other hand, selection must assure the survival of the fittest individuals, otherwise, the search will be more or less undirected. Common search strategies include:

- *fitness proportionate* selection, such as roulette wheel selection or stochastic universal sampling
- *rank-based* selection, e.g. linear ranking or tournament selection

A.2 Crossover

A new population of children is created by recombining the selected individuals. This breeding process is done by *crossover* of genomes. The process is highly dependent on the genotype, the representation of the individuals. Simple strategies for crossover include:

- *N-point crossover*: two genomes form two children. Between crossover-points, genes are taken from alternating parents, as shown in Figure A.3
- *uniform crossover*: for all loci, choose a gene from either parent with a given probability (usually dependent on the fitness)
- *arithmetic crossover*: calculate the offspring's genes by arithmetic combination of the parents genes

Usually two parents are chosen for crossover, though it is possible to create offspring from more parents. A *crossover rate* (CR) defines the probability of an individual to undergo crossover, otherwise it is copied to the next generation.

A.3 Mutation

After individuals are selected as parents and recombined by crossover, there is a certain probability for *mutation* of the offspring. For binary genes it is an obvious

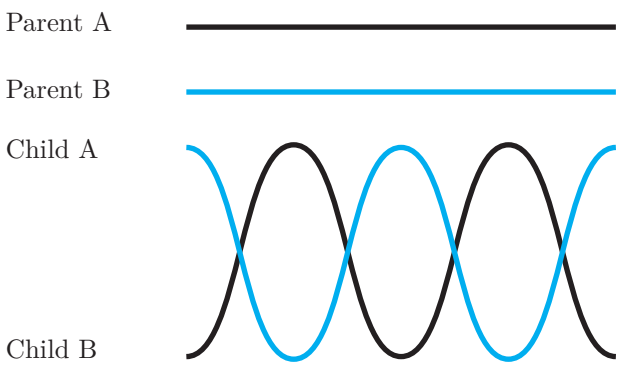

Figure A.3: Four-point crossover

option to flip values with a certain probability. For real valued genes a random value can be drawn from some distribution (e.g. Gaussian) and added to the gene. In many cases it is better to permutate a genome. For example, if the genome encodes a TSP tour it is better to permutate two cities than to randomly mutate one city because by random mutation the vast majority of variants will become invalid.

The idea behind mutation is to introduce new information into the population. Thus a high *mutation rate* corresponds to a high exploration rate. A balance between exploration and exploitation has to be found.

Another important secondary parameter is the mutation distance s. This distance represents the strength (and therefore also amount of disruption) a single mutation can provoke. In case of a Gaussian mutation on a real number, this distance would be controlled by changing the σ of the underlying distribution.

A.4 Diversity Management

Precautions must be taken to prevent premature convergence and ensure exploration to take place even after convergence to local suboptimal extrema. In particular if the search space contains disconnected regions, it is necessary to sustain diversity in the population. Techniques to implement this include

- *niching*, in crowded regions the probability to reproduce is lower

- *speciation*, only "similar" solutions are recombined

Niching prevents all individuals from populating only the region around one particular extremum by fitness sharing.

$$fitness_{shared} = \frac{fitness}{|close\ individuals|}$$

Niching allows for dissimilar solutions to be recombined. While this is desired since it ensures exploration it usually results in less fit individuals if two solutions from dissimilar regions/optima are combined. A solution to this problem is *speciation* which allows only similar individuals to be recombined. This can be implemented by dividing the population into species (using some metric). Each species is assigned a number of children according to their fitness and these are created solely from individuals of the corresponding species.

A.5 Representation

The representation of a possible solution is probably the most important deciding factor, whether an EA will find a solution quickly or at all. The fitness of an individual must represent the distance to a solution in some way or other. For mutation to work best, minor changes in the genome should result in minor changes to the fitness value, otherwise the search conducted by an EA will be rather erratic and convergence will be slow.

APPENDIX B

T-Distributed Stochastic Neighborhood Embedding

Commonly used for visualization, the dimensionality reduction method t-SNE has been shown to be capable of retaining the local structure of the data, as well as revealing clusters at several scales. It does so by finding a lower-dimensional distribution of points $\mathbb{Q}$ that is similar to the original high-dimensional distribution $\mathbb{P}$. The similarity of data point $\boldsymbol{x}_j$ to datapoint $\boldsymbol{x}_i$ is the conditional probability ($p_{j|i}$ for $\mathbb{P}$ and $q_{j|i}$ for $\mathbb{Q}$, Eq. B.1), that $\boldsymbol{x}_i$ would pick $\boldsymbol{x}_j$ as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian distribution centered at $\boldsymbol{x}_i$. The Student-t distribution is used to measure similarities between low-dimensional points $\boldsymbol{y}_i \in \mathbb{Q}$ in order to allow dissimilar objects to be modeled far apart in the archive (Eq. B.1).

$$p_{j|i} = \frac{e^{\frac{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma_i^2}}}{\sum_{k \neq i}\left(e^{\frac{-\|\boldsymbol{x}_i - \boldsymbol{x}_k\|^2}{2\sigma_i^2}}\right)}, \; q_{j|i} = \frac{1 + \|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2)^{-1}}{\sum_{k \neq i}(1 + \|\boldsymbol{y}_i - \boldsymbol{y}_k\|^2)^{-1})} \tag{B.1}$$

The local scale σ_i is adapted to the density of the data (smaller in denser parts). The parameter σ_i is set such that *perplexity* of the conditional distribution equals a predefined value. The perplexity of a distribution defines how many neighbors for each data point have a significant $p_{j|i}$ and can be calculated using the Shannon entropy $H(P_i)$ of the distribution P_i around x_i (Eq. B.2).

$$\text{Perp}(P_i) = 2^{-\sum_j p_{j|i} log_2 p_{j|i}} \tag{B.2}$$

$$KL(\mathbb{P}\|\mathbb{Q}) = \sum_{i \neq j} p_{ij} log(\frac{p_{ij}}{q_{ij}}) \tag{B.3}$$

Using the bisection method, σ_i are changed such that $\text{Perp}(P_i)$ approximates the preset value (commonly 5–50). The similarity of $\boldsymbol{x}_j$ to $\boldsymbol{x}_i$ and $\boldsymbol{x}_i$ to $\boldsymbol{x}_j$ is absorbed with the joint probability p_{ij}. A low-dimensional archive is learned that reflects all similarities p_{ij} as well as possible. Locations $\boldsymbol{y}_i$ are determined by iteratively minimizing the Kullback-Leibler divergence of the distribution $\mathbb{Q}$ from the distribution $\mathbb{P}$ (Eq. B.3) with gradient descent.

APPENDIX C

Gaussian Process Regression

To perform interpolation or regression on a given data set, GP models (introduced by Rasmussen (2004)) assume that the underlying data is sampled from a Gaussian process – a process that generates points that are distributed in a Gaussian fashion, and are correlated in a local and smooth fashion. For an in-depth introduction to GP regression and its application in model-based optimization, refer to Forrester et al. (2008). What follows is a summarized explanation.

The models assume that the objective function is smooth: the closer a candidate is to a known example, the closer their function values will be to each other. Here, the training data of the model is denoted as a set of n solutions $\boldsymbol{X} = \{\mathbf{x}^{(i)}\}_{i=1\ldots n}$ in a k-dimensional search space. The corresponding n observations are denoted with $\mathbf{y} = \{y^{(i)}\}_{i=1\ldots n}$. For an unknown point in our search space, $\mathbf{x}^*$, Gaussian process regression intends to estimate the unknown function value $\hat{y}(\mathbf{x}^*)$. In its core, the model assumes that the observations at each location $\mathbf{x}$ are correlated via a kernel function. Kernel functions of the following type are considered here:

$$\mathrm{k}(\mathbf{x}, \mathbf{x}') = \exp\left(-\theta \mathrm{d}(\mathbf{x}, \mathbf{x}')\right). \tag{C.1}$$

This essentially expresses the correlation of two samples $\mathbf{x}$ a $\mathbf{x}$', based on their distance $\mathrm{d}(\mathbf{x}, \mathbf{x}')$, and a kernel parameter $\theta \in \mathbb{R}^+$. Kernel parameters are usually determined by Maximum Likelihood Estimation (MLE), that is, they are chosen such that the data has the maximum likelihood under the resulting model. MLE usually involves a numerical optimization procedure. The distance measure $\mathrm{d}(\mathbf{x}, \mathbf{x}')$ can potentially be any measure, though not all ensure that the kernel is positive semi-definite, a common requirement. By using the Manhattan distance, the

distance measure is less affected by issues related to high-dimensional data, see Aggarwal et al. (2001). This distance is defined as:

$$\mathrm{d}_{\mathrm{Man}}(\mathbf{x}, \mathbf{x}') = \sum |x_i - x_i'| \tag{C.2}$$

Rather than a single parameter θ, a different θ can be used for each dimension i of the input samples, enabling the model to estimate the influence of each individual dimension on the observed values. However, in the interest of simplicity and computational efficiency we opt for an isotropic kernel with a single θ.

Once the pairwise correlations between all training samples are collected in a matrix $\mathbf{K}$, the GP predictor can be specified with

$$\hat{y}(\mathbf{x}^*) = \hat{\mu} + \mathbf{k}^{\mathrm{T}}\mathbf{K}^{-1}(\mathbf{y} - \mathbf{1}\hat{\mu}), \tag{C.3}$$

where $\hat{\mu}$ is another model parameter (estimated by MLE), $\mathbf{k}$ is the vector of correlations between training samples $\boldsymbol{X}$ and the new sample $\mathbf{x}^*$, and $\mathbf{1}$ is a vector of ones. The error or uncertainty of the prediction can be estimated with

$$\hat{s}^2(\mathbf{x}) = \hat{\sigma}^2(1 - \mathbf{k}^T\mathbf{K}^{-1}\mathbf{k}^T), \tag{C.4}$$

where $\hat{\sigma}^2$ is a further model parameter to be estimated by MLE.

Bibliography

Aggarwal, C. C., A. Hinneburg, and D. A. Keim (2001). On the surprising behavior of distance metrics in high dimensional space. In *Database Theory — ICDT 2001: 8th International Conference*, LNCS, London, UK.

Aldous, C. R. (2017). Modelling the creative process and cycles of feedback. *Creative Education 8*(12), 1860–1877.

Amabile, T. M. (1988). A model of creativity and innovation in organizations. *Research in organizational behavior 10*(1), 123–167.

Amabile, T. M. (1996). Creativity in context: The social psychology of creativity. *Boulder, CO: Westview*.

Arieti, S. (1976). Creativity: The magic synthesis.

Association for Computational Creativity ("2019 (accessed October 2, 2020)"). *https://computationalcreativity.net/home/about/computational-creativity*.

Asteroth, A. and A. Hagg (2015). How to successfully apply genetic algorithms in practice: Representation and parametrization. In *2015 International Symposium on Innovations in Intelligent SysTems and Applications (INISTA)*, pp. 1–6. IEEE.

Auer, P. (2002). Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research 3*(Nov), 397–422.

Balling, R. (1999). Design by shopping: A new paradigm? In *Proceedings of the Third World Congress of structural and multidisciplinary optimization (WCSMO-3)*, Volume 1, pp. 295–297. International Soc. for Structural and Multidisciplinary Optimization Berlin.

Basto-Fernandes, V., I. Yevseyeva, A. Deutz, and M. Emmerich (2017). A survey of diversity oriented optimization: Problems, indicators, and algorithms. In *EVOLVE–A Bridge between Probability, Set Oriented Numerics and Evolutionary Computation*, Volume 7, pp. 3–23. Springer.

Basto-Fernandes, V., I. Yevseyeva, and M. Emmerich (2013). A survey of diversity-oriented optimization. *EVOLVE 2013-A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computing 1*(2013), 101–109.

Berns, S. and S. Colton (2020). Bridging Generative Deep Learning and Computational Creativity. In *Proceedings of the 11th International Conference on Computational Creativity*.

Beyer, K., J. Goldstein, R. Ramakrishnan, and U. Shaft (1999a). When is "nearest neighbor" meaningful? In *International conference on database theory*, pp. 217–235. Springer.

Beyer, K., J. Goldstein, R. Ramakrishnan, and U. Shaft (1999b). When is "nearest neighbor" meaningful? In *International conference on database theory*, pp. 217–235. Springer.

Bhatnagar, P. L., E. P. Gross, and M. Krook (1954). A model for collision processes in gases. i. small amplitude processes in charged and neutral one-component systems. *Physical review 94*(3), 511.

Bonnardel, N. and F. Zenasni (2010). The impact of technology on creativity in design: An enhancement? *Creativity and Innovation Management 19*(2), 180–191.

Bontrager, P., W. Lin, J. Togelius, and S. Risi (2018). Deep interactive evolution. In *International Conference on Computational Intelligence in Music, Sound, Art and Design*, pp. 267–282. Springer.

Bontrager, P., A. Roy, J. Togelius, N. Memon, and A. Ross (2018). Deepmasterprints: Generating masterprints for dictionary attacks via latent variable evolution. In *Proceedings of the IEEE International Conference on Biometrics Theory, Applications and Systems*, pp. 1–9. IEEE.

Bradner, E., F. Iorio, and M. Davis (2014). Parameters tell the design story: Ideation and abstraction in design optimization. *Simulation Series 46*(7), 172–197.

Broyden, C. G. (1970). The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics 6*(1), 76–90.

Burgess, C. P., I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner (2017). Understanding disentangling in Beta-VAE. In *NIPS Workshop on Learning Disentangled Representations*.

Campbell, D. T. (1960). Blind variation and selective retentions in creative thought as in other knowledge processes. *Psychological review 67*(6), 380.

Carroll, J. V. and R. K. Mehra (1982). Bifurcation analysis of nonlinear aircraft dynamics. *Journal of Guidance, Control, and Dynamics 5*(5), 529–536.

Catmull, E. and R. Rom (1974). A class of local interpolating splines. In *Computer aided geometric design*, pp. 317–326. Elsevier.

Christiano, P. F., J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei (2017). Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pp. 4299–4307.

Clune, J. and H. Lipson (2004). Evolving Three-Dimensional Objects with a Generative Encoding Inspired by Developmental Biology. *Methods*.

Colton, S., G. A. Wiggins, et al. (2012). Computational creativity: The final frontier? In *Ecai*, Volume 12, pp. 21–26. Montpelier.

Conover, W. J. and R. L. Iman (1979). On multiple-comparisons procedures. Technical Report LA-7677-MS, Los Alamos Sci. Lab.

Cropley, D. and A. Cropley (2010). Functional creativity. *Camb. Handb. Creat*, 301–318.

Cully, A. (2019). Autonomous skill discovery with quality-diversity and unsupervised descriptors. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 81–89.

Cully, A., J. Clune, D. Tarapore, and J.-B. Mouret (2015). Robots that can adapt like animals. *Nature 521*(7553).

Cully, A. and Y. Demiris (2017). Quality and Diversity Optimization: A Unifying Modular Framework. *IEEE Transactions on Evolutionary Computation*, 1–15.

Cully, A. and Y. Demiris (2018). Hierarchical behavioral repertoires with unsupervised descriptors. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 69–76.

Cully, A. and J.-B. Mouret (2013). Learning to Walk in Every Direction. *Evolutionary Computation 24*(1), 59–88.

Cully, A., U. Pierre, and J.-B. Mouret (2013). Behavioral Repertoire Learning in Robotics. *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference - GECCO '13*, 175–182.

Daniels, S. J., A. A. M. Rahat, R. M. Everson, G. R. Tabor, and J. E. Fieldsend (2018). A suite of computationally expensive shape optimisation problems using computational fluid dynamics. In *International Conference on Parallel Problem Solving from Nature*.

Dasgupta, S. (2004). Is creativity a darwinian process? *Creativity research journal 16*(4), 403–413.

Dawkins, R. (1982). *The Extended Phenotype*. Oxford University Press Oxford.

De Grave, K., J. Ramon, and L. De Raedt (2008). Active learning for high throughput screening. In *International Conference on Discovery Science*.

De Jong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems.

Deb, K., A. Pratap, S. Agarwal, and T. Meyarivan (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation 6*(2), 182–197.

Deb, K. and A. Srinivasan (2006). Innovization: Innovating design principles through optimization. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pp. 1629–1636.

Deb, K. and S. Tiwari (2008). Omni-optimizer: A generic evolutionary algorithm for single and multi-objective optimization. *European Journal of Operational Research 185*(3), 1062–1087.

Demo, N., M. Tezzele, and G. Rozza (2018). Pydmd: Python dynamic mode decomposition. *Journal of Open Source Software 3*(22), 530.

Doncieux, S. and J.-B. Mouret (2010). Behavioral diversity measures for evolutionary robotics. In *IEEE Congress on Evolutionary Computation*.

Dorschner, B., S. S. Chikatamarla, and I. V. Karlin (2017). Transitional flows with the entropic lattice Boltzmann method. *J. Fluid Mech. 824*, 388–412.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science 14*(1 990), 179–211.

Ester, M., H.-P. Kriegel, J. Sander, X. Xu, et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, Volume 96, pp. 226–231.

Finke, R. A., T. B. Ward, and S. M. Smith (1996). Creative cognition: Theory, research, and applications.

Fletcher, R. (1970). A new approach to variable metric algorithms. *The computer journal 13*(3), 317–322.

Forrester, A., A. Sobester, and A. Keane (2008). *Engineering Design via Surrogate Modelling*. John Wiley & Sons.

Friedman, R. S., A. Fishbach, J. Förster, and L. Werth (2003). Attentional priming effects on creativity. *Creativity Research Journal 15*(2-3), 277–286.

Gablonsky, J. M. and C. T. Kelley (2001). A locally-biased form of the DIRECT algorithm.

Gaedtke, M., S. Wachter, M. Rädle, H. Nirschl, and M. J. Krause (2018). Application of a lattice boltzmann method combined with a smagorinsky turbulence model to spatially resolved heat flux inside a refrigerated vehicle. *Computers & Mathematics with Applications 76*(10), 2315–2329.

Gaier, A., A. Asteroth, and J.-b. Mouret (2017). Data-Efficient Exploration, Optimization, and Modeling of Diverse Designs through Surrogate-Assisted Illumination.

Gaier, A., A. Asteroth, and J.-B. Mouret (2018). Data-efficient design exploration through surrogate-assisted illumination. *Evolutionary computation*, 1–30.

Gaier, A., A. Asteroth, and J.-B. Mouret (2019). Are quality diversity algorithms better at generating stepping stones than objective-based search? In *Proceedings*

of the Genetic and Evolutionary Computation Conference Companion, pp. 115–116.

Gaier, A., A. Asteroth, and J.-B. Mouret (2020). Discovering representations for black-box optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Volume 11.

Gärdenfors, P. (2004). *Conceptual spaces: The geometry of thought*. MIT press.

Gardner, H. (1993). Creating minds: An anatomy of creativity seen through the lives of freud. *Einstein, Picasso, Stravinsky, Eliot, Graham, and GandhZ Basic Books, New York, NY*.

Gassner, G. J. and A. D. Beck (2013). On the accuracy of high-order discretizations for underresolved turbulence simulations. *Theoretical and Computational Fluid Dynamics 27*(3-4), 221–237.

Gerber, D. J., S.-H. Lin, B. Pan, and A. S. Solmaz (2012). Design optioneering: multi-disciplinary design optimization through parameterization, domain integration and automation of a genetic algorithm. In *Proceedings of the 2012 Symposium on Simulation for Architecture and Urban Design*, pp. 1–8.

Giacomello, E., P. L. Lanzi, and D. Loiacono (2019). Searching the Latent Space of a Generative Adversarial Network to Generate DOOM Levels. In *Proceedings of the IEEE Conference on Games (CoG)*.

Glorot, X. and Y. Bengio (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*.

Goldfarb, D. (1970). A family of variable-metric methods derived by variational means. *Mathematics of Computation 24*(109), 23–26.

Goldschmidt, G. (1991). The dialectics of sketching. *Creativity research journal 4*(2), 123–143.

Guilford, J. P. (1967). The nature of human intelligence.

Hagg, A. (2017). Hierarchical surrogate modeling for illumination algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO 2017*.

Hagg, A. (2021). *Phenotypic Niching Using Quality Diversity Algorithms*, pp. 287–315. Cham: Springer International Publishing.

Hagg, A., A. Asteroth, and T. Bäck (2018). Prototype discovery using quality-diversity. In *Proceedings of the 16th International Conference on Parallel Problem Solving from Nature - PPSN 2018*, pp. 500–511. Springer.

Hagg, A., A. Asteroth, and T. Bäck (2019). Modeling User Selection in Quality Diversity. In *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO 2019*.

Hagg, A., A. Asteroth, and T. Bäck (2020). A Deep Dive Into Exploring the Preference Hypervolume. In *Proceedings of the International Conference on Computational Creativity - ICCC 2020*.

Hagg, A., S. Berns, A. Asteroth, S. Colton, and T. Bäck (2021). Expressivity of Parameterized and Data-driven Representations in Quality Diversity Search). In *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO 2021*.

Hagg, A., M. Mensing, and A. Asteroth (2017). Evolving parsimonious networks by mixing activation functions. In *GECCO 2017 - Proceedings of the 2017 Genetic and Evolutionary Computation Conference*.

Hagg, A., M. Preuss, A. Asteroth, and T. Bäck (2020). An Analysis of Phenotypic Diversity in Multi-Solution Optimization. In *Proceedings of the 9th International Conference on Bioinspired Optimisation Methods and Their Applications - BIOMA 2020*.

Hagg, A., D. Wilde, A. Asteroth, and T. Bäck (2020). Designing air flow with surrogate-assisted phenotypic niching. In *Proceedings of the 16th International Conference on Parallel Problem Solving from Nature - PPSN 2020*.

Hagg, A., M. Zaefferer, J. Stork, and A. Gaier (2019). Prediction of neural network performance by phenotypic modeling. In *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO 2019*.

Hamming, R. W. (1950). Error detecting and error correcting codes. *The Bell system technical journal 29*(2), 147–160.

Harik, G. R. (1995). Finding multimodal solutions using restricted tournament selection. In *ICGA*, pp. 24–31.

Hart, E., A. S. Steyven, and B. Paechter (2018). Evolution of a functionally diverse swarm via a novel decentralised quality-diversity algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 101–108.

Heft, A. I., T. Indinger, and N. A. Adams (2012). Introduction of a new realistic generic car model for aerodynamic investigations. Technical report, SAE Technical Paper.

Hegel, G. W. F. (1842). *Vorlesungen über die Ästhetik*, Volume 1. Duncker und Humblot.

Herring, S. R., C.-C. Chang, J. Krantzler, and B. P. Bailey (2009). Getting inspired!: understanding how and why examples are used in creative design practice. *Proceedings of the 27th international conference on Human factors in computing systems*, 87–96.

Higgins, I., L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner (2016). Beta-vae: Learning basic visual concepts with a constrained variational framework. In *Proceedings of the International Conference on Learning Representations*.

Hildebrandt, T. and J. Branke (2015). On using surrogates with genetic programming.

Hinton, G. E. (1994). Autoencoders, minimum description length and helmholtz free energy. *Advances in NIPS 6*, 3–10.

Hinton, G. E. and R. R. Salakhutdinov (2006). Reducing the dimensionality of data with neural networks. *science 313*(5786), 504–507.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. MIT press.

Hu, T., J. L. Payne, W. Banzhaf, and J. H. Moore (2011). Robustness, evolvability, and accessibility in linear genetic programming. In *European Conference on Genetic Programming*, pp. 13–24. Springer.

Janssen, W., B. Blocken, and T. van Hooff (2013). Pedestrian wind comfort around buildings: Comparison of wind comfort criteria based on whole-flow field data for a complex case study. *Building and Environment 59*, 547–562.

Jin, Y. (2011). Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation 1*(2).

Jung, C. (1923). Psychological types: or the psychology of individuation.

Kamentsky, L. A. and C.-N. Liu (1963). Computer-automated design of multifont print recognition logic. *IBM Journal of Research and Development 7*(1), 2–13.

Kendall, M. G. and J. D. Gibbons (1990). *Rank Correlation Methods*. Charles Griffin Book Series. London: Oxford University Press.

Kingma, D., L. Ba, et al. (2015). Adam: A method for stochastic optimization.

Kingma, D. P. and M. Welling (2014). Auto-Encoding Variational Bayes. In *Proceedings of the 2nd International Conference on Learning Representations*.

Koos, S., J.-B. Mouret, and S. Doncieux (2012). The Transferability Approach: Crossing the Reality Gap in Evolutionary Robotics. *IEEE Transactions on Evolutionary Computation*, 1–25.

Koza, J. R. (1994). Genetic programming. MIT Press.

Krämer, A., D. Wilde, and M. Bedrunka (2020). Lettuce: PyTorch-based Lattice Boltzmann Solver.

Krämer, A., D. Wilde, K. Küllmer, D. Reith, and H. Foysi (2019). Pseudoentropic derivation of the regularized lattice Boltzmann method. *Phys. Rev. E 100*(2), 1–16.

Krüger, T., H. Kusumaatmaja, A. Kuzmin, O. Shardt, G. Silva, and E. M. Viggen (2017). *The lattice Boltzmann method: Principles and practice*.

Kruskal, W. H. and W. A. Wallis (1952). Use of ranks in one-criterion variance analysis.

Laland, K. N. (2004). Extending the extended phenotype. *Biology and Philosophy 19*(3), 313–325.

Lehman, J. (2012). Evolution Through the Search for Novelty. pp. 223.

Lehman, J., J. Clune, D. Misevic, C. Adami, L. Altenberg, J. Beaulieu, P. J. Bentley, S. Bernard, G. Beslon, D. M. Bryson, et al. (2020). The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities. *Artificial life 26*(2), 274–306.

Lehman, J. and K. O. Stanley (2008). Exploiting Open-Endedness to Solve Problems Through the Search for Novelty. *Alife*, 329–336.

Lehman, J. and K. O. Stanley (2011a). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation 19*(2), 189–222.

Lehman, J. and K. O. Stanley (2011b). Evolving a diversity of virtual creatures through novelty search and local competition. *Proceedings of the 13th annual conference on Genetic and evolutionary computation - GECCO '11* (Gecco), 211.

Li, M. and X. Yao (2019). Quality evaluation of solution sets in multiobjective optimisation: A survey. *ACM Computing Surveys (CSUR) 52*(2), 1–38.

Liapis, A., H. P. Martínez, J. Togelius, and G. N. Yannakakis (2013). Transforming exploratory creativity with delenox.

Liapis, A., G. N. Yannakakis, and J. Togelius (2013a). Designer modeling for personalized game content creation tools. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Liapis, A., G. N. Yannakakis, and J. Togelius (2013b). Sentient sketchbook: computer-assisted game level authoring.

Liapis, A., G. N. Yannakakis, and J. Togelius (2014). Designer modeling for sentient sketchbook. In *2014 IEEE Conference on Computational Intelligence and Games*, pp. 1–8. IEEE.

Lubart, T. (2005). How can computers be partners in the creative process: classification and commentary on the special issue. *International Journal of Human-Computer Studies 63*(4-5), 365–369.

Lubart, T. I. (2001). Models of the creative process: Past, present and future. *Creativity research journal 13*(3-4), 295–308.

Maaten, L. v. d. (2014). Accelerating t-SNE using Tree-Based Algorithms. *Journal of Machine Learning Research 15*, 3221–3245.

Maaten, L. v. d. and G. Hinton (2008). Visualizing data using t-sne. *Journal of machine learning research 9*(Nov), 2579–2605.

Maher, M. L. (2012). Computational and collective creativity: Who's being creative? In *ICCC*, pp. 67–71. Citeseer.

Martinsen, Ø. (1995). Cognitive styles and experience in solving insight problems: Replication and extension. *Creativity Research Journal 8*(3), 291–298.

Mednick, S. (1962). The associative basis of the creative process. *Psychological review 69*(3), 220.

Meyerson, E., J. Lehman, and R. Miikkulainen (2016). Learning behavior characterizations for novelty search. *GECCO 2016 - Proceedings of the 2016 Genetic and Evolutionary Computation Conference*, 149–156.

Meyerson, E. and R. Miikkulainen (2017). Discovering evolutionary stepping stones through behavior domination. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 139–146.

Miikkulainen, R. (2020). Creative ai through evolutionary computation. In *Evolution in Action: Past, Present and Future*, pp. 265–269. Springer.

Miller, J. F., P. Thomson, and T. Fogarty (1997). Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study.

Mlot, N. J., C. A. Tovey, and D. L. Hu (2011). Fire ants self-assemble into waterproof rafts to survive floods. *Proceedings of the National Academy of Sciences 108*(19), 7669–7673.

Moraglio, A., K. Krawiec, and C. G. Johnson (2012). Geometric semantic genetic programming. In *International Conference on Parallel Problem Solving from Nature*.

Mouret, J.-B. (2011a). Encouraging Behavioral Diversity in Evolutionary Robotics: An Empirical Study. *Evolutionary computation* (x).

Mouret, J.-B. (2011b). Novelty-based multiobjectivization. In *New horizons in evolutionary robotics*, pp. 139–154. Springer.

Mouret, J.-B. and J. Clune (2012). An algorithm to create phenotype-fitness maps. *Proc. of the Artificial Life Conf. 375*(2012), 593–594.

Mouret, J.-B. and J. Clune (2015). Illuminating search spaces by mapping elites. pp. 107–108.

Müller-Wienbergen, F., O. Müller, S. Seidel, and J. Becker (2011). Leaving the beaten tracks in creative work - A design theory for systems that support convergent and divergent thinking. *Journal of the Association for Information Systems 12*(11), 714–740.

Mumford, M. D. and S. B. Gustafson (1988). Creativity syndrome: Integration, application, and innovation. *Psychological bulletin 103*(1), 27.

Mumford, M. D., M. I. Mobley, R. Reiter-Palmon, C. E. Uhlman, and L. M. Doares (1991). Process Analytic Models of Creative Capacities. *Creativity Research Journal 4*(2), 91–122.

NEN 8100 (2006). Wind comfort and wind danger in the built environment (in dutch). Norm NEN 8100.

Nguyen, A., J. Yosinski, and J. Clune (2015). Innovation Engines: Automated Creativity and Improved Stochastic Optimization via Deep Learning. *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference - GECCO '15*, 959–966.

Nguyen, A., J. Yosinski, and J. Clune (2016). Understanding innovation engines: Automated creativity and improved stochastic optimization via deep learning. *Evolutionary Computation 24*(3), 545–572.

Nijstad, B. A. and W. Stroebe (2006). How the group affects the mind: A cognitive model of idea generation in groups. *Personality and social psychology review 10*(3), 186–213.

Ong, Y. S., P. B. Nair, and A. J. Keane (2003). Evolutionary optimization of computationally expensive problems via surrogate modeling.

Parmee, I. (2002). Towards interactive evolutionary search and exploration systems. In *Bird-of-a-feather workshop: Genetic and Evolutionary Conference*.

Parmee, I. C. and C. R. Bonham (2000). Towards the support of innovative conceptual design through interactive designer/evolutionary computing strategies. *Ai Edam 14*(1), 3–16.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. D'Alché-Buc, E. Fox, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc.

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 2*(11), 559–572.

Pétrowski, A. (1996). A clearing procedure as a niching method for genetic algorithms. In *Proceedings of IEEE international conference on evolutionary computation*, pp. 798–803. IEEE.

Pohlert, T. (2018). PMCMRplus: calculate pairwise multiple comparisons of mean rank sums extended - R package, version 1.4.1.

Pošík, P. and W. Huyer (2012). Restarted local search algorithms for continuous black box optimization. *Evolutionary Computation 20*(4), 575–607.

Preuss, M. (2006). Niching Prospects. *In Proceedings of Bioinspired Optimization Methods and their Applications (BIOMA 2006)*, 25–34.

Preuss, M. (2010). Niching the cma-es via nearest-better clustering. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, pp. 1711–1718.

Preuss, M. (2012). Improved topological niching for real-valued global optimization. In *European Conference on the Applications of Evolutionary Computation*, pp. 386–395. Springer.

Preuss, M. (2015). *Multimodal Optimization by Means of Evolutionary Algorithms*.

Pugh, J. K., L. B. Soros, and K. O. Stanley (2016). Searching for quality diversity when diversity is unaligned with quality. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 9921 LNCS*(Ppsn), 880–889.

Pugh, J. K., L. B. Soros, P. A. Szerlip, and K. O. Stanley (2015). Confronting the challenge of quality diversity. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 967–974.

R Core Team (2018). R: A language and environment for statistical computing.

Raidl, M.-H. and T. I. Lubart (2001). An empirical study of intuition and creativity. *Imagination, Cognition and Personality 20*(3), 217–230.

Rasmussen, C. E. (2004). Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pp. 63–71. Springer.

Rasmussen, C. E. and H. Nickisch (2010). Gaussian processes for machine learning (GPML) toolbox. *Journal of Machine Learning Research 11*, 3011–3015.

Rosch, E. (1975). Cognitive reference points. *Cognitive psychology 7*(4), 532–547.

Runco, M. A. (2010). Divergent thinking, creativity, and ideation. *The Cambridge handbook of creativity 413*, 446.

Runco, M. A. and G. J. Jaeger (2012). The standard definition of creativity. *Creativity research journal 24*(1), 92–96.

Runco, M. A. and S. R. Pritzker (2020). *Encyclopedia of creativity*. Academic press.

Salvador, S. and P. Chan (2003). Determining the Number of Clusters / Segments in Hierarchical Clustering / Segmentation Algorithms. *Work*, 20.

Santanen, E. L., R. O. Briggs, and G.-J. D. Vreede (2004). Causal relationships in creative problem solving: Comparing facilitation interventions for ideation. *Journal of Management Information Systems 20*(4), 167–198.

Sartre, J.-P. and A. Elkaïm-Sartre (1946). L'existentialisme est un humanisme.

Schaaf, L. J., F. J. Odling-Smee, K. N. Laland, and M. W. Feldman (2003). *Niche construction: the neglected process in evolution*. Princeton University Press.

Schmid, P. J. (2010). Dynamic mode decomposition of numerical and experimental data. *Journal of fluid mechanics 656*, 5–28.

Schmidhuber, J. (2007). Simple algorithmic principles of discovery, subjective beauty, selective attention, curiosity & creativity. In *International Conference on Discovery Science*, pp. 26–38. Springer.

Schölkopf, B., A. Smola, and K.-R. Müller (1997). Kernel principal component analysis. In *International conference on artificial neural networks*, pp. 583–588. Springer.

Secretan, J., N. Beato, D. B. D'Ambrosio, A. Rodriguez, A. Campbell, J. T. Folsom-Kovarik, and K. O. Stanley (2011). Picbreeder: A case study in collaborative evolutionary exploration of design space. *Evolutionary computation 19*(3), 373–403.

Shaham, U. and S. Steinerberger (2017). Stochastic Neighbor Embedding separates well-separated clusters. pp. 1–8.

Shanno, D. F. (1970). Conditioning of quasi-newton methods for function minimization. *Mathematics of computation 24*(111), 647–656.

Shaw, M. P. (1989). The eureka process: A structure for the creative experience in science and engineering. *Creativity Research Journal 2*(4), 286–298.

Shir, O. M., M. Preuss, B. Naujoks, and M. Emmerich (2009). Enhancing decision space diversity in evolutionary multiobjective algorithms. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 5467 LNCS*, 95–109.

Simonton, D. K. (2007). The Creative Process in Picasso's Guernica Sketches: Monotonic Improvements versus Nonmonotonic Variants. *Creativity Research Journal 19*(4), 329–344.

Smith, D., L. Tokarchuk, and G. Wiggins (2016). Rapid phenotypic landscape exploration through hierarchical spatial partitioning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 9921 LNCS*, 911–920.

Snoek, J., H. Larochelle, and R. P. Adams (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*.

Sobol, I. M. (1967). On the distribution of points in a cube and the approximate evaluation of integrals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki 7*(4).

Solow, A. R. and S. Polasky (1994). Measuring biological diversity. *Environmental and Ecological Statistics 1*(2), 95–103.

Sørensen, P. D., J. M. Olsen, and S. Risi (2016). Breeding a diversity of super mario behaviors through interactive evolution. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–7. IEEE.

Stanley, K. O. (2006). Exploiting regularity without development. In *Proceedings of the AAAI Fall Symposium on Developmental Systems*. AAAI Press.

Stanley, K. O. and R. Miikkulainen (2002). Evolving neural networks through augmenting topologies.

Stein, B. v., H. Wang, W. Kowalczyk, T. Bäck, and M. Emmerich (2015). Optimally weighted cluster kriging for big data regression. In *International Symposium on Intelligent Data Analysis*, pp. 310–321. Springer.

Stein, M. I. (1953). Creativity and culture. *The journal of psychology 36*(2), 311–322.

Sterelny, K. et al. (2001). Niche construction, developmental systems, and the extended replicator. *Cycles of contingency: Developmental systems and evolution*, 333–350.

Stork, J., M. Zaefferer, and T. Bartz-Beielstein (2019). Improving neuroevolution efficiency by surrogate model-based optimization with phenotypic distance kernels. In *Applications of Evolutionary Computation*.

Stork, J., M. Zaefferer, A. Fischbach, and T. Bartz-Beielstein (2017). Surrogate-assisted learning of neural networks. In *Proceedings 27. Workshop Computational Intelligence*.

Stump, G. M., M. Yukish, T. W. Simpson, and E. N. Harris (2003). Design space visualization and its application to a design by shopping paradigm. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Volume 37009, pp. 795–804.

Tarapore, D., J. Clune, A. Cully, and J.-B. Mouret (2016). How do different encodings influence the performance of the map-elites algorithm? In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pp. 173–180.

Tenenbaum, J., V. de Silvia, and J. Langford (2000). A global framework for nonlinear dimensionality reduction. *Science. v290*, 2319–2323.

Tennekes, H. (1978). Turbulent flow in two and three dimensions. *Bulletin of the American Meteorological Society 59*(1), 22–28.

Tian, Y., R. Cheng, X. Zhang, M. Li, and Y. Jin (2019). Diversity assessment of multi-objective evolutionary algorithms: Performance metric and benchmark problems [research frontier]. *IEEE Computational Intelligence Magazine 14*(3), 61–74.

Toffolo, A. and E. Benini (2003). Genetic diversity as an objective in multi-objective evolutionary algorithms. *Evolutionary Computation 11*(2), 151–167.

Tomašev, N. and M. Radovanović (2016). Clustering evaluation in high-dimensional data. In *Unsupervised Learning Algorithms*, pp. 71–107. Springer.

Törn, A. and S. Viitanen (1992). Topographical global optimization. *Recent advances in global optimization*, 384–398.

Ulrich, T. (2010). Integrating Decision Space Diversity into Hypervolume-based Multiobjective Search Categories and Subject Descriptors. *GECCO 2010*, 455–462.

Ursem, R. K. (1999). Multinational evolutionary algorithms. In *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, Volume 3, pp. 1633–1640. IEEE.

Valiant, L. G. (2009). Evolvability. *Journal of the ACM (JACM) 56*(1), 1–21.

Vassiliades, V., K. Chatzilygeroudis, J. Clune, and J.-B. Mouret. Scaling-up MAP-Elites Using Centroidal Voronoi Tesselations. *999*(999), 1–6.

Vassiliades, V., K. Chatzilygeroudis, and J.-B. Mouret (2017a). Comparing multimodal optimization and illumination. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 97–98.

Vassiliades, V., K. Chatzilygeroudis, and J.-B. Mouret (2017b). A comparison of illumination algorithms in unbounded spaces. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1578–1581.

Vassiliades, V. and J.-B. Mouret (2018). Discovering the elite hypervolume by leveraging interspecies correlation. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 149–156.

Venables, W. N. and B. D. Ripley (2002). Modern applied statistics with s. Springer.

Volle, E. (2018). Associative and controlled cognition in divergent thinking: Theoretical, experimental, neuroimaging evidence, and new directions.

Volz, V., J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi (2018). Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the Genetic and Evolutionary Computation Conference*.

Wales, D. J. and J. P. Doye (1997). Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms. *The Journal of Physical Chemistry A 101*(28), 5111–5116.

Wallas, G. (1926). The art of thought.

Wang, H., Y. Jin, and J. O. Jansen (2016). Data-driven surrogate-assisted multi-objective evolutionary optimization of a trauma system.

Wang, H., Y. Jin, and X. Yao (2016). Diversity assessment in many-objective optimization. *IEEE transactions on cybernetics 47*(6), 1510–1522.

Wang, K. and J. V. Nickerson (2017). A literature review on individual creativity support systems. *Computers in Human Behavior 74*, 139–151.

Ward, T. B., S. M. Smith, and R. A. Finke (1999). Creative cognition. *Handbook of creativity 189*, 212.

Weisberg, R. W. and R. Hass (2007). Commentaries: We are all partly right: Comment on Simonton. *Creativity Research Journal 19*(4), 345–360.

Wessing, S. (2015). *Two-stage methods for multimodal optimization*. Ph. D. thesis, Universitätsbibliothek Dortmund.

Wessing, S. and M. Preuss (2016). On multiobjective selection for multimodal optimization. *Computational Optimization and Applications 63*(3), 875–902.

Whigham, P. A., G. Dick, and J. Maclaurin (2017). On the mapping of genotype to phenotype in evolutionary algorithms. *Genetic Programming and Evolvable Machines 18*(3), 353–361.

Wittgenstein, L. (1953). *Philosophische Untersuchungen*. Basil Blackwell.

Woolley, B. G. and K. O. Stanley (2014). A Novel Human-Computer Collaboration: Combining Novelty Search with Interactive Evolution. *Proceedings of the 16th annual conference on Genetic and evolutionary computation, GECCO '14* , 233–240.

Wright, N. A., D. W. Steadman, and C. C. Witt (2016). Predictable evolution toward flightlessness in volant island birds. *Proceedings of the National Academy of Sciences 113*(17), 4765–4770.

Yannakakis, G. N., A. Liapis, and C. Alexopoulos (2014). Mixed-initiative co-creativity.

Zaefferer, M. (2019). Combinatorial efficient global optimization in R - CEGO v2.3.0. accessed: 2019-03-19.

Zaefferer, M., J. Stork, O. Flasch, and T. Bartz-Beielstein (2018). Linear combination of distance measures for surrogate models in genetic programming. In *Parallel Problem Solving from Nature – PPSN XV*, Coimbra, Portugal.

Zwicky, F. (1969). Discovery, invention, research through the morphological approach.

Acronyms

AI artificial intelligence.

AIC Aikake information criterion.

AutoVE Automatic Voronoi-Elites.

BFGS Broyden-Fletcher-Goldfarb-Shanno.

BGK Bhatnagar-Gross-Krook.

BO Bayesian optimization.

cAE convolutional autoencoder.

CFD computational fluid dynamics.

CGP Cartesian genetic programming.

CPPN compositional pattern producing network.

cVAE convolutional variational autoencoder.

CVT centroidal Voronoi tessellation.

DBSCAN density-based spatial clustering of applications with noise.

DIRECT dividing rectangles.

DL deep learning.

DMD dynamic mode decomposition.

DR dimensionality reduction.

EA evolutionary algorithm.

ELBO evidence lower bound.

GAN generative adversarial network.

GM generative model.

GP Gaussian process.

GPU graphics processing unit.

HSP hierarchical spatial partitioning.

HyperPref interactive, co-creative process, determining the preference hypervolume.

KL Kullback-Leibler divergence.

kPCA kernel principal component analysis.

LBM Lattice Boltzmann method.

LS latent search.

MAP-Elites multidimensional archive of phenotypic elites.

MAPE mean absolute percentage error.

MMO multi-solution, multi-local or multimodal optimization.

MOO multi-objective (or multicriteria) optimization.

NEAT neuroevolution of augmenting topologies.

NS novelty search.

NSGA-II non-dominated sorting genetic algorithm II.

NSLC novelty search with local competition.

PCA principal component analysis.

PD Pure Diversity.

PE precise performance evaluations.

PFE precise phenotypic feature evaluations.

PRODUQD prototype discovery using quality diversity.

PS parameter search.

QD quality diversity.

ReLU rectified linear unit.

RLS restarted local search.

RMSE root mean square error.

SAIL surrogate-assisted illumination.

SDNN sum of distances to nearest neighbor.

SPD Solow-Polasky Diversity.

SPHEN surrogate-assisted phenotypic niching.

t-SNE t-distributed stochastic neighbourhood embedding.

UCB upper confidence bound.

UDHM user decision hypersurface model.

VAE variational autoencoder.

VE Voronoi-Elites.

Summary

Creative processes in engineering are time-consuming. For quite some time, we have striven to develop algorithms that help us alleviate this task and extend our ability to think creatively. Automated generative systems are creeping into this domain. In this thesis, questions are answered about how to design fruitful interactions between generative systems and human decision makers. Instead of replacing the human, this thesis embraces human-computer interaction in creative design, putting the human back into the loop of algorithmic design in generative systems and optimization. This thesis mostly revolves around shape optimization in expensive settings, for example when creating shapes that are used in fluid dynamics. It is here where a large potential exists to increase the efficiency and efficacy of creative engineering.

In Chapter 2, common theories on creativity and the creative process are described. Aspects from creative cognition that can hinder or facilitate the ability to find innovate solutions are taken as an inspiration to design a computational co-creative process model. Requirements are developed which allow this thesis to then explore efficient methods to enhance engineers' capabilities independent of their experience level.

The ability to generate diverse solutions in an automated manner depends on how we measure similarity and how diversity can be maintained. Chapter 3 discusses a key concept of evolutionary computation, that of the separation of search space (the genome) and the actual solution space (the phenotype). By measuring similarity and diversity in high-dimensional phenotypic space, more diverse solution sets can be developed. When comparing multisolution optimization paradigms, it is shown that quality diversity (QD) optimization, specifically the Voronoi-Elites (VE) algorithm, produces the most diverse solution sets. QD performs niching in a low-dimensional space that describes factors of phenotypic variation. By making use

of a generative model (GM), e.g. variational autoencoders, the AutoVE algorithm combines VE with such an autoencoder. It is shown that we can automatically learn phenotypic features in an on-line fashion instead of using a manual a priori definition. Limitations of GM are shown as well. Evidence is gathered that show that GM should be used as a niching, not as a search space in divergent search.

Efficiency is a key ingredient to co-creative systems in fluid dynamics and other computationally expensive problems. In Chapter 4, surrogate-assisted methods are developed that can predict phenotypic features and behavior. The ability to model behavioral features through surrogate-assisted phenotypic niching (SPHEN) by sampling training examples based on optimality alone, allows us to not only create optimal solution sets efficiently but also to predict their diversity in an on-line fashion. The ability to model neural encodings' behavior using phenotypic distance (PHD) allows the use of more complex encodings in optimization. Both are key to developing flexible, efficient QD algorithms.

Finally, the preference hypervolume, which contains all preferred (phenotypic) solutions is discussed in Chapter 5. This hypervolume is what a co-creative process aims to find and describe and is the central object of this research, in which we use the design by shopping paradigm to allow a user to select their preferences from generated examples. The ability to efficiently generate diverse solution sets is helpful but the vast number of solutions and complex aspects of diversity needs to be presented to a user. This is accomplished by compressing the solution set into representative genetic prototypes. Users then are able to effectively select what examples they prefer and which ones they dislike. This selection is turned into the user selection drift metric that can be used as a penalty to any objective function. The underlying selection is modeled using the user decision hypersurface model (UDHM) preference model. This forces algorithms like QD to start producing artifacts that are similar to the user's preferred solutions. The UDHM is integrated with QD in the prototype discovery using quality diversity (PRODUQD) algorithm. The chapter then combines the idea of focussing on phenotypes, not genomes (Chapter 3), to produce the phenotypic drift metric and penalty. The final algorithm, HyperPref, implements the requirements of a co-creative process as defined in Chapter 2.

Samenvatting

Creatieve processen in engineering zijn tijdrovend. Reeds geruime tijd streven wij ernaar algoritmen te ontwikkelen die ons helpen deze taak te verlichten en ons vermogen om creatief te denken uit te breiden. Geautomatiseerde generatieve systemen sluipen dit domein steeds meer binnen. In deze dissertatie worden vragen beantwoord over hoe vruchtbare interacties kunnen worden ontworpen tussen generatieve systemen en menselijke deelnemers. In plaats van de mens te vervangen, omarmt deze dissertatie mens-computer interactie in het creatief ontwerpsproces, waarbij de mens de controle houdt in de lus van algoritmisch ontwerp in generatieve systemen en optimalisatie. Deze thesis draait vooral rond vormoptimalisatie in dure omgevingen, bijvoorbeeld bij het creëren van vormen die gebruikt worden in vloeistofdynamica. Het is hier waar een groot potentieel bestaat om de efficiëntie en doeltreffendheid van creatieve engineering te verhogen.

In Hoofdstuk 2 worden gangbare theorieën over creativiteit en het creatieve proces beschreven. Aspecten van creatieve cognitie die het vermogen om vernieuwende oplossingen te vinden kunnen belemmeren of vergemakkelijken worden als inspiratie genomen om een computationeel co-creatief procesmodel te ontwerpen. Er worden vereisten ontwikkeld die deze dissertatie toelaten om vervolgens efficiënte methodes te verkennen die de capaciteiten van ingenieurs vergroten, onafhankelijk van hun ervaringsniveau.

De mogelijkheid om op een geautomatiseerde manier diverse oplossingen te genereren hangt af van hoe we gelijkenis meten en hoe diversiteit kan worden behouden. Hoofdstuk 3 bespreekt een sleutelconcept van evolutionary computation, namelijk de scheiding tussen de zoekruimte (het genoom) en de eigenlijke oplossingsruimte (het fenotype). Door gelijkenis en diversiteit te meten in de hoog-dimensionale fenotypische ruimte, kunnen diversere oplossingen worden ontwikkeld. Bij vergelijking van multisolution optimalisatie paradigma's is gebleken dat quality diversity

(QD) optimalisatie, bijvoorbeeld het Voronoi-Elites (VE) algoritme, de meest diverse oplossingsreeksen oplevert. QD voert niching uit in een laag-dimensionale ruimte die factoren van fenotypische variatie beschrijft. Het AutoVE algoritme combineert VE met een generatief model (GM), b.v. variationele autoencoders. Er wordt aangetoond dat we automatisch fenotypische kenmerken kunnen leren op een on-line manier in plaats van een handmatige a priori definitie te gebruiken. De beperkingen van GM worden ook aangetoond. Er worden bewijzen verzameld die aantonen dat GM moet worden gebruikt als een nichingmechanisme, niet als een zoekruimte in divergente optimalisatie.

Efficiëntie is een belangrijk ingrediënt voor co-creatieve systemen in vloeistofdynamica en andere computationeel dure problemen. In hoofdstuk 4 worden surrogaat-ondersteunde methoden ontwikkeld die fenotypische kenmerken en gedrag kunnen voorspellen. De mogelijkheid om gedragskenmerken te modelleren via surrogaat-ondersteunde fenotypische niching (SPHEN) door het samplen van trainingsvoorbeelden op basis van optimaliteit alleen, stelt ons in staat om niet alleen efficiënt optimale oplossingsreeksen te creëren, maar ook om hun diversiteit op een on-line manier te voorspellen. De mogelijkheid om het gedrag van neurale coderingen te modelleren met behulp van fenotypische afstand (PHD) maakt het gebruik van meer complexe coderingen in optimalisatie mogelijk. Beide zijn essentieel voor de ontwikkeling van flexibele, efficiënte QD-algoritmen.

Tenslotte wordt het voorkeurshypervolume, dat alle (fenotypische) voorkeursoplossingen bevat, besproken in hoofdstuk 5. Dit hypervolume is wat een co-creatief proces beoogt te vinden en te beschrijven en is het centrale object van dit onderzoek, waarin we het design by shopping paradigma gebruiken om een gebruiker zijn voorkeuren te laten selecteren uit gegenereerde voorbeelden. De mogelijkheid om op efficiënte wijze diverse sets van oplossingen te genereren is nuttig, maar het grote aantal oplossingen en de complexe aspecten van diversiteit moeten aan een gebruiker worden gepresenteerd. Dit wordt bereikt door de verzameling oplossingen te comprimeren tot representatieve genetische prototypen. Gebruikers zijn dan in staat om effectief te selecteren aan welke voorbeelden zij de voorkeur geven en aan welke niet. Deze selectie wordt omgezet in de gebruikersselectie-drift metriek die kan worden gebruikt als een straf op een objectieve functie. De onderliggende selectie wordt gemodelleerd met behulp van het voorkeursmodel UDHM (User Decision Hypersurface Model). Dit dwingt algoritmen zoals QD om artefacten te gaan produceren die lijken op de voorkeursoplossingen van de gebruiker. Het

UDHM is geïntegreerd met QD in het prototype discovery using quality diversity (PRODUQD) algoritme. Het hoofdstuk combineert vervolgens het idee om te focussen op fenotypen, niet op genomen (Hoofdstuk 3), om de fenotypische drift metriek en penalty te produceren. Het uiteindelijke algoritme, HyperPref, implementeert de vereisten van een co-creatief proces zoals gedefinieerd in hoofdstuk 2.

About the Author

Alexander Hagg, born 1979 in Delft, The Netherlands, received his Master degree of Autonomous Systems at Bonn-Rhein-Sieg University of Applied Sciences (BRSU), Sankt Augustin, Germany in 2016. He has received scholarships from the German Academic Scholarship Foundation as well as from the Institute of Technology, Resource and Energy-Efficient Engineering (TREE) in Sankt Augustin, Germany. He worked at BRSU in the AErOmAt project, which was funded by the German Federal Ministry of Education and Research. During this time he developed most of his doctoral work, while he was supervised by Prof. Dr. Thomas Bäck from the Leiden Institute of Advanced Computer Science (LIACS) and Prof. Dr. Alexander Asteroth from the TREE institute in Sankt Augustin. His research interests are evolutionary computation, especially those algorithms that promote solution diversity, as well as efficient statistical machine learning methods and generative models. He aims to combine state-of-the-art algorithms to create artificial intelligence systems that extend, not replace, human capabilities to understand complex problems.