Carl von Ossietzky University of Oldenburg
Faculty II – School of Computing Science, Business Administration, Economics, and Law
Department of Computing Science

Prediction-Based Nature-Inspired Dynamic Optimization

A dissertation accepted by the School of Computing Science, Business Administration, Economics and Law of the Carl von Ossietzky University of Oldenburg (Germany)

to obtain the degree and title

Doctor of Natural Sciences (Dr. rer. nat)

Submitted by:

Almuth Meier

Reviewers:

Prof. Dr. Oliver Kramer
Prof. Dr.-Ing. Sanaz Mostaghim

Date of disputation:

April 20th, 2020

Abstract

Many real-world optimization problems are not static, but changing over time. This optimization scenario is known as dynamic optimization. Changing objective functions result in moving optima and may be due to changing conditions like environmental parameters in real-world. The task becomes to find and follow the optimum, while it is dynamically moving in the solution space. Nature-inspired optimization algorithms like evolution strategies and swarm algorithms are common choices for solving dynamic optimization problems. Nevertheless, their rather fast convergence impedes finding the new optimum. In real-world scenarios, the changes of objective functions often are not random, but certain time-depending characteristics exist. Algorithms exploiting such information often are superior to naive variants with random re-starts regarding convergence speed and tracking accuracy. Predicting the moving optimum based on information gained from the optimization and incorporating the predictions into the optimization process is a strategy that has attained attention in the recent past to circumvent premature convergence.

Prediction approaches in dynamic nature-inspired optimization employed so far mostly rely on statistical prediction methods. In this thesis, we show that neural network-based prediction methods are applicable as well. Furthermore, particle swarm optimization was rarely used with prediction until now. We propose extensions of particle swarm optimization that incorporate prediction and are superior to the base algorithm. Moreover, for prediction-based evolution strategies we suggest a strategy to adapt the population after an objective function change mitigating the negative effect of uncertain predictions. In addition, we propose a benchmark generator and a convergence measure that better than existing ones take into account the specific characteristics of prediction-based optimization algorithms.

Zusammenfassung

In realen Anwendungen sind Optimierungsprobleme oft nicht statisch, sondern über die Zeit veränderlich. Ein solches Optimierungsszenario wird als dynamische Optimierung bezeichnet. Die zu optimierende Zielfunktion verändert sich beispielsweise auf Grund veränderter Umgebungsbedingen und weist sich bewegende Optima auf. Dynamische Optimierung hat das Ziel, das globale Optimum der Zielfunktion zu finden und über die Zeit im Lösungsraum zu verfolgen. Naturinspirierte Optimierungsalgorithmen wie Evolutionsstrategien und Schwarmalgorithmen werden häufig zum Lösen dynamischer Optimierungsprobleme verwendet, allerdings erschwert ihre schnelle Konvergenz das Finden des Optimums der geänderten Zielfunktion. In realen Anwendungsszenarien sind die Änderungen der Zielfunktion selten vollständig zufällig, sondern weisen zeitabhängige Charakteristiken auf. Algorithmen, die diese Zusammenhänge ausnutzen, sind oft einfachen Algorithmen mit zufälligen Neustarts bezüglich Konvergenzgeschwindigkeit und Genauigkeit der Annäherung des Optimums überlegen. In den letzten Jahren haben Ansätze an Bedeutung zugenommen, die das Optimum vorhersagen, indem sie während des Optimierungsprozesses gewonnene Information ausnutzen, und die Vorhersage für die Optimierung der geänderten Zielfunktion verwenden.

Vorhersageansätze, die bislang in dynamischer naturinspirierter Optimierung verwendet werden, sind meist statistische Methoden. In dieser Arbeit wird gezeigt, dass auch neuronale Netze für die Vorhersage anwendbar sind. Da Partikelschwarmoptimierung bisher sehr selten mit Vorhersageansätzen kombiniert wird, werden in dieser Arbeit Erweiterungen der Partikelschwarmoptimierung vorgestellt, die Vorhersage verwenden und dem Basisalgorithmus überlegen sind. Außerdem wird für vorhersagebasierte Evolutionsstrategien ein Ansatz entwickelt, die Population nach einer Zielfunktionsänderung so anzupassen, dass die negativen Auswirkungen einer unsicheren Vorhersage vermindert werden. Darüber hinaus werden ein neues dynamisches Testproblem und ein Konvergenzmaß vorgestellt, welche die Charakteristiken vorhersagebasierter Optimierungsalgorithmen besser als existierende Testprobleme und Konvergenzmetriken berücksichtigen.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AR autoregressive model (p. 30)

ARR absolute recovery rate (p. 48)

BEBC best error before change (p. 48)

$\overline{\text{BOG}}$ best of generation (p. 47)

CMA-ES covariance matrix adaptation evolution strategy (p. 20)

DSB dynamic sine benchmark (p. 38)

ES evolution strategy (p. 19)

FPB free peaks benchmark (p. 38)

LSTM long short-term memory network (p. 32)

MC Monte Carlo (p. 87)

MPB moving peaks benchmark (p. 35)

NN neural network (p. 31)

PE prediction error (p. 91)

PSO particle swarm optimization (p. 21)

RCS relative convergence speed (p. 49)

RNN recurrent neural network (p. 32)

ROOT robust optimization over time (p. 13)

SRR group of experiments with Sphere, Rosenbrock, and Rastrigin (p. 59)

TCN temporal convolutional network (p. 33)

TMO tracking the moving optimum (p. 13)

Nomenclature

First we list symbols that occur in the whole thesis, then symbols follow that are only used for optimization, prediction or for benchmarking. In general, $\mathbf{v}$ is a vector, e.g., $\mathbf{v} \in \mathbb{R}^d$, and v_i is the ith entry of $\mathbf{v}$. A matrix is denoted by $\mathbf{V}$, e.g. $\mathbf{V} \in \mathbb{R}^{d \times d}$, and v_{ij} is the jth entry in the ith row.

General

α	significance level of Mann-Whitney U tests
$\hat{\mathbf{o}}_c$	predicted optimum in change period c
$\hat{\mathbf{u}}_c$	predictive uncertainty in change period c
$\lceil \cdot \rceil$	ceiling function, i.e., rounding to the next greater integer
$\lfloor \cdot \rfloor$	floor function, i.e., rounding to the next smaller integer
$\mathbf{I}$	identity matrix
$\mathbf{o}_c$	true global optimum in change period c
$\mathbf{p}_j$	jth pattern in training set
$\mathbf{x}^l$	local optimum
$\mathcal{N}$	normal distribution
$\mathcal{U}$	uniform distribution
$\boldsymbol{\varepsilon}$	multivariate noise
υ	change period length
ε	univariate noise
$\|\mathbf{x}\|_2$	Euclidean norm of vector $\mathbf{x}$ with $\|\mathbf{x}\|_2 = \sqrt{x_1^2 + \ldots + x_d^2}$
$\|\mathbf{x}\|_2^2$	squared Euclidean norm

b^l	lower bound of solution space in all dimensions
b^u	upper bound of solution space in all dimensions
c	change period counter, $1 \leq c \leq P$
d	problem dimensionality
E	number of constraint functions
e	constraint function
f	fitness function
g	overall number of generations
N	neighborhood relation to define a local optimum
P	overall number of change periods
r	overall number of runs
t	generation/iteration counter, $1 \leq t \leq g$

Optimization

$\mathbf{A}$	archive of solutions $\mathbf{x}_c^*$ found by the optimizer
$\mathbf{x}_c^*$	best solution found in change period c
$\mathbf{x}_{c_t}^*$	best solution found in generation t of change period c
σ	standard deviation of a normal distribution, σ^2 is the variance

Evolution Strategy

λ	number of offspring individuals
$\mathbf{P}'$	offspring population
$\mathbf{P}$	population
$\mathbf{p}^i$	ith parent individual selected for recombination
$\mathbf{x}'$	mutated offspring
μ	number of parents/immigrants

ψ	number of parents selected for recombination
ξ	adaptation factor for the step size in Rechenberg's 1/5th success rule
p_s	success rate in Rechenberg's 1/5th success rule
s	mutation strength
s'	mutation strength adapted by Rechenberg's 1/5th success rule

Particle Swarm Optimization

$\mathbf{r}$	randomness: $\mathbf{r}_1$ for cognitive part, $\mathbf{r}_2$ for social part, $\mathbf{r}_3$ for prediction
$\mathbf{S}$	swarm
$\mathbf{v}$	particle's velocity
$\mathbf{x}^p$	particle's best position
$\mathbf{x}^s$	swarm's best position
ω	inertia weight
θ	influence of attractor: θ_1 cognitive factor, θ_2 social factor, θ_3 influence of prediction

CMA-ES

$\mathbf{C}_t$	covariance matrix in generation t
$\mathbf{m}_t$	population mean in generation t

Re-Initialization

χ	constant used for computation of δ
δ	factor influencing how many individuals are placed around the prediction (re-initialization strategy `pKAL`)
$\hat{\mathbf{x}}_c$	predicted next position of individual $\mathbf{x}_{c-1}$
$\hat{u}_{\max}$	maximum entry of vector $\hat{\mathbf{u}}_c$
$\mathbf{x}_c$	immigrant in change period c
z	scaling factor for a normal distribution

Prediction

$\hat{\mathbf{y}}$	output of an NN (point prediction)
$\mathbf{x}_t$	observation of $\mathbf{x}$ at time step t
ϕ	function a neuron computes
Ψ	prediction model
$\tilde{\mathbf{p}}$	unknown new pattern the prediction model was not trained for
$\tilde{y}$	true label of unknown new pattern
M	training set size
w_s	window size
y_j	label of pattern $\mathbf{p}^j$ in the training set

Autoregression

$\boldsymbol{\Phi}_i$	determines influence of previous time steps
$\boldsymbol{\theta}$	vector of constants
$\mathbf{C}$	covariance of noise
p	order of the autoregressive model

Kalman Filter

$\hat{\mathbf{a}}_c$	a posteriori state estimation in change period c
$\hat{\mathbf{a}}_c^-$	a priori state estimation in change period c
$\hat{\mathbf{E}}_c$	a posteriori error covariance in change period c
$\hat{\mathbf{E}}_c^-$	a priori error covariance in change period c
$\mathbf{a}_c$	true sytem state in change period c

TCN

$\boldsymbol{\psi}$	filter
$\hat{y}_T$	output of TCN at index T in the time series output

$\mathbf{s}$	univariate input sequence with T time steps
ψ_i	filter weight at index i in $\boldsymbol{\psi}$
$\mathrm{E}\left[\hat{\mathbf{y}}\right]$	predictive mean
$\mathrm{Var}\left[\hat{\mathbf{y}}\right]$	predictive variance
b	block index, $0 \leq b < \#\text{blocks}$
D_b	dilation factor for block b
h	history captured by a TCN
k	filter size
m	number of Monte Carlo runs
n	network output function for point prediction
Q	convolutional operation
q	network output function for aleatoric uncertainty
s_j	value of jth time step in series $\mathbf{s}$
T	number of time steps in input sequence $\mathbf{s}$

Moving Peaks Benchmark

η	correlation factor, $\eta = 1 - \nu$
$\mathbf{S}^i$	shift vector for the position of the ith peak
$\mathbf{X}^i$	position of the ith peak
ν	noise, $\nu = 1 - \eta$
H	peak height
s_h	height severity
s_l	shift length
s_w	width severity
W^i	width of the ith peak

Dynamic Sine Benchmark

α	overall scaling of composite function ζ_w
$\beta_{\max}$	upper bound of frequencies β_i
β_i	frequency of component function i
γ_i	phase shift of component function i
$\iota_{\max}$	upper bound of amplitudes ι
ι_i	amplitude of component function i
κ	step size
$\mathbf{o}_{f_s}$	global optimum position of stationary fitness function f_s
ρ	number of component functions
$\rho_{\max}$	upper bound of ρ
τ	vertical shift
ζ_w	composite function for dimension w
C	desired curviness
$C_{\max}$	largest curviness function ζ_w can realize
C_{theo}	curviness theoretically realized by function ζ_w with respect to the specifications of the component functions
C_{ζ_w}	curviness realized by function ζ_w
f_s	stationary fitness function
L	period length of arbitrary sine function
V	desired velocity
V_{ζ_w}	velocity realized by ζ_w
w	a certain dimension

Metrics

$f\left(\mathbf{o}_c, c\right)$	global optimal fitness in change period c

$f\left(\mathbf{x}^*_{c_{\text{worst}}}, c\right)$ worst fitness achieved by any algorithm during change period c

$f\left(\mathbf{x}^*_{c_t}, c\right)$ best fitness achieved by the algorithm until generation t of change period c

$g\left(c\right)$ number of generations in change period c

1 Introduction

Dynamics is inherent to many real-world optimization problems. Immediate adaptation of process plans or continuously running systems to time-varying conditions with respect to arising expenses, lead time or other criteria is a frequent scenario in various industrial sectors [MLY17; HMR09]. Examples are cloud service providers that dynamically distribute and relocate virtual machines on physical machines trying to reduce the energy consumption [Cho+18], logistics companies that prepare delivery routes shortening the travel time, while targets may dynamically be added or canceled [Mei+15; Bos+19], or dynamic machine scheduling problems where jobs are distributed to different machines minimizing the lead time, and unexpected events as machine breakdown or new jobs involve replanning the schedule [OP09; And+18]. Dynamic rescheduling of railway timetables after delay is a similar application [EYG17]. Controlling the conditions inside a greenhouse to maximize produced biomass [PH99], influencing the throughput of chemical reactions [Mül+17], or production optimization in petrol industry [FKG18] are examples of dynamic optimization tasks as well.

In all these applications, the system state $\mathbf{x} \in \mathbb{R}^d$ is defined by a set of d parameters that have to be optimized subject to a certain objective. For example, in a greenhouse optimization task the current system configuration $\mathbf{x} \in \mathbb{R}^3$ may consist of values for the three control variables temperature, humidity and carbon dioxide content. Dynamic optimization problems have in common that the reward or cost $f(\mathbf{x}, t)$ of a system configuration is time-dependent. A frequent objective is to find for each time step t within the time interval for that optimization is conducted the optimal configuration $\mathbf{o}_t$. If the reward function does not change after each single time step, the task reduces to finding the optimal solution for each change period. But still the repeated optimization leads to large computational effort and therefore requires efficient algorithms.

In this chapter, we depict challenges of dynamic optimization problems and describe prediction as possibility to cope with them. Then, we motivate our contributions to tackle difficulties of prediction-based approaches in dynamic optimization and overview the structure of this thesis.

1.1 Challenges in Dynamic Optimization

Dynamic optimization is challenging since each change of the reward function involves a new optimization procedure which could be computationally expensive. Depending on how frequently changes occur, the time frame available to search for a solution could be very short. This requires the optimization to be fast and restricts the set of applicable algorithms. Among the many optimization algorithms that exist, most research on dynamic optimization focuses on nature-inspired methods [RY13; NYB12]. Nature-inspired optimization algorithms are stochastic optimization techniques that handle the objective function as black-box. In black-box optimization scenarios, no information is available but the evaluation $f(\mathbf{x})$ of configurations $\mathbf{x}$. Nature-inspired algorithms iteratively improve a set of candidate solutions until a sufficiently good solution is found. Candidate solutions are possible configurations of the system to be optimized.

A special kind of dynamic problems are optimal control problems [RY13]. In control problems, a dynamic system is influenced by an environment and observed by a controller that, in turn, influences the system by adapting the system's control variables so that a desired system state is achieved [JH75; CB07]. This forms a control loop running during a certain period of time without user interaction [Mor04]. Some of the applications mentioned above belong to this class. In control theory, the dynamic system is mostly described by differential equations, and the control strategy that varies the control variables to achieve the desired objective is computed either analytically or numerically [AP98]. Nature-inspired algorithms can tackle control problems in real-time as well, at least for slowly changing systems [UFK02]. Especially if the control problem cannot be solved analytically, e.g., for non-linear dynamics, nature-inspired methods are a possible alternative [RY13]. The publications [MLY17; HMR09] overview many dynamic optimization problems in which nature-inspired methods have found application. We restrict this thesis to nature-inspired methods, since they are problem-independent and also applicable to problems that cannot be solved analytically.

For nature-inspired methods, dynamic optimization is challenging as well. They tend to converge relatively fast so that it might be hard to find the new optimal solution after a change if it is dissimilar to the old one. Another disadvantage is that they need some time to find good solutions due to their iterative design. Applications with a short time frame for optimization prohibit optimization from scratch after each change and demand for sophisticated algorithms. Various ideas exist to circumvent theses shortcomings. One among them are prediction approaches that gain information about the optimum dynamics during the optimization process. This information is used to start optimization with better candidate solutions after a change to reduce search effort. We describe this approach more detailed in the next paragraph.

1.2 Prediction-Based Dynamic Optimization

Prediction-based dynamic optimization combines optimization and time series prediction. Time-series prediction methods are learning algorithms, often from the field of statistical or machine learning. For a given sequence of observations at succeeding points in time (time series) they predict which value would be observed next. Their internal parameters are adjusted by techniques specific for each prediction method so that the model represents a given time-series as good as possible. A concrete parameterized instance of a certain prediction method is called prediction model or predictor.

The overall procedure of prediction-based optimization is as follows. The optimizer conducts optimization as long as no change occurs. When a change is detected (the optimizer has to implement a detection mechanism for this), the best solution found during the past change period is stored and optimization is continued. Thus, the optimizer reports for each change period c the best found solution $\mathbf{x}_c^*$ so that a time series $[\mathbf{x}_1^*, \ldots, \mathbf{x}_{c-1}^*]$ of solutions is collected. If the optimizer is able to follow the optimum accurately, this series represents the optimum's movement. By means of time series prediction, it can be estimated which solution $\hat{\mathbf{o}}_c$ the optimizer will find in change period c. This is an estimate for the true optimum position $\mathbf{o}_c$. In the optimization procedure, prediction takes place at the beginning of change period c, i.e., after a change has been detected. Then, the set of candidate solutions could be placed around the prediction $\hat{\mathbf{o}}_c$. In case $\hat{\mathbf{o}}_c$ is near to the true optimum $\mathbf{o}_c$, the optimizer is likely to converge faster to the optimum than with a random set of candidate solutions.

Prediction-based optimization entails difficulties for both predictor and optimizer. Since the optimizer does not necessarily find the optimum, the time series the predictor relies on might be a very inaccurate approximation of the optimum dynamics hampering good predictions. Many time-series prediction methods provide more precise predictions with increasing number of time steps they can learn from. Therefore, prediction quality improves with increasing number of change periods for which optimization has been conducted. After each change period, a new solution is available so that the parameters of the prediction method have to be adapted to the extended time series. This operation is very time-demanding for some prediction methods so that the time available for prediction might be too short to use these prediction methods in case of high-frequent changes. Even if the time series perfectly represents the optimum movement and enough time is available for prediction, it might be hard to capture the relationship inherent in the dynamics since it could change during optimization time. For example, first the optimum could follow a linear direction and start oscillating later.

Even the predictor might hamper optimization. If the predicted optimum is far from the true one and hence also the new candidate solutions, the optimizer might need many iterations to find a good solution. The possibly inaccurate prediction and changing type of dynamics are reasons why the prediction model cannot replace the optimizer. Optimizer and predictor are required to support each other.

1.3 Contributions and Results

In nature-inspired optimization, various attempts already exist to incorporate prediction into the optimizer. Nevertheless, the prediction methods mostly employed are simple statistical ones. This rises the question, whether modern approaches from machine learning could cope with the challenges of dynamic optimization as well. Furthermore, most work concentrates on evolution strategies, one family among the many nature-inspired optimization algorithms, while for others like particle swarm optimization almost no approaches are available. In addition, literature lacks attempts to consider the inaccuracy of the predictor and to circumvent negative effects of uncertain predictions on the optimization process. Moreover, benchmark problems and convergence measures available to examine dynamic optimization algorithms are less suited for prediction-based algorithm. Existing benchmarks either consist of simple or non-predictable dynamics, while well-known convergence measures disadvantage algorithms starting with a better fitness after a change which often is the case for prediction-based algorithms. These points led us to the following five contributions of this thesis which are published in three conference articles and summarized in a book chapter [MK20].

Recurrent Neural Network Prediction for Evolution Strategies We propose to employ a recurrent neural network as prediction model and compare it with a statistical approach (autoregression). The results show that depending of the dynamics either of both achieves better results and hybridizing these prediction methods attains outstanding performance (Chapter 7). This work is published in [MK18a].

Prediction for Particle Swarm Optimization For particle swarm optimization, we propose algorithmic extensions to incorporate prediction. Using the predicted optimum as third attractor in the movement function is most promising (Chapter 8). This work is published in [MK18b].

Predictive Uncertainty for Evolution Strategies We propose a new re-initialization mechanism spreading the set of candidate solutions of an evolution strategy wider in dimensions where the prediction is likely to be inaccurate. This strategy supports finding better solutions at the beginning of change periods (Chapter 9). This work is published in [MK19].

Dynamic Sine Benchmark We construct the dynamic sine benchmark to generate problems with predictable dynamics that is adjustable regarding its difficulty for the optimizer and the predictor. By this means, also prediction methods can be employed that exhibit strengths in more complex dynamics. It is published in [MK19], whereas this thesis provides an improved version.

Relative Convergence Speed This convergence measure considers how fast an algorithm comes close to the optimum fitness instead of evaluating how fast it leaves worse solutions at the beginning of a change period. Thus, it represents the specifics of algorithms with and without prediction more fairly than existing measures. We proposed it in [MK18a].

1.4 Overview of the Thesis

In the first part of this thesis, we formalize dynamic optimization problems, describe their characteristics and depict the kind of dynamic optimization problems we investigate (Chapter 2). Then, we give an overview of optimization methods, present principles of nature-inspired optimization, and describe evolution strategies and particle swarm optimization in detail (Chapter 3). Chapter 4 covers approaches to enable nature-inspired optimization to cope with dynamic problems. Furthermore, time series prediction is introduced and prediction methods applied to dynamic optimization are described (Chapter 5). Afterwards, an overview of common benchmarks and measures to evaluate optimization performance in dynamic problems is given, and both an own benchmark and measure designed in this thesis are proposed (Chapter 6).

The second part comprises our contributions that are roughly depicted above. First, we investigate neural network prediction for evolution strategies (Chapter 7), then we propose three extensions of particle swarm optimization with prediction (Chapter 8). Finally, for evolution strategies we develop a new strategy to adapt the candidate solutions so that the uncertainty of the prediction affects the optimization less (Chapter 9).

In the third part, we summarize the results and discuss transferability to other optimization algorithms and kinds of dynamic problems (Chapter 10). The thesis ends with an perspective on follow up research (Chapter 11), while the appendix contains complete results of experiments and additional examinations.

Part I

Foundations

2 Dynamic Optimization Problems

Dynamic optimization problems can be seen as extension of static ones [Ahr+19]. Therefore, we formalize static optimization and introduce the definition of dynamic problems afterwards. A static optimization problem is defined as

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & e_i(\mathbf{x}) \leq 0 \quad \text{for} \quad i = 1, \ldots, E, \end{aligned} \tag{2.1}$$

where f is the objective or reward function to be optimized, in nature-inspired optimization also called fitness function. The objective function[1] $f : \mathbb{R}^d \to \mathbb{R}$ maps a decision vector $\mathbf{x} = [x_1, x_2, \ldots, x_d] \in \mathbb{R}^d$, also called candidate solution, from a d-dimensional solution space, also decision space, to a one-dimensional objective space. The solution space is restricted by E constraint functions e_i, and the number d of decision variables to be optimized determines problem dimensionality d. The domain of f is defined with b^l and b^u as lower and upper bounds, respectively, where $b^l < b^u$ holds. We use the same bounds for all dimensions, although the solution space could have different bounds in each dimension. Maximization problems can be converted to minimization problems by multiplying the fitness and constraint functions with -1.

An optimal solution (global optimum) $\mathbf{o} \in \mathbb{R}^d$ for a static optimization problem is a vector that complies with all constraints and never achieves a worse fitness value than any other vector in the solution space: $f(\mathbf{o}) \leq f(\mathbf{x}) \ \forall \mathbf{x} \in \mathbb{R}^d$ [BV11]. Vector $\mathbf{x}^l$ is a local optimum if it is best within a certain neighborhood N of vectors: $f(\mathbf{x}^l) \leq f(\mathbf{x}) \ \forall \mathbf{x} \in N(\mathbf{x}^l) \wedge \mathbf{x}^l, \mathbf{x} \in \mathbb{R}^d$ [MLY17]. In some applications, searching for the global optimum would be too expensive, e.g., in terms of runtime, so that a solution with a reasonable suboptimal fitness, e.g., a local optimum, is sufficient.

In dynamic optimization problems, objective and constraint functions, and solution and objective space are time-dependent. While time might be measured by variable t, the problem may remain unchanged during a change

[1] As we consider continuous problems in this thesis, we restrict the depiction to $\mathbb{R}^d$ and $\mathbb{R}$ as solution and objective space, respectively, whereas also other spaces are possible.

period c of a certain number of points in time t. Thus the optimization task has to be solved for all static problem instances $c \in \mathbb{N}_{>0} \wedge c \leq P$ within a time span of P change periods with length $\upsilon \in \mathbb{N}_{>0}$:

$$\begin{aligned}
\underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}, c) = f\left(\mathbf{x}, \left\lceil \frac{t}{\upsilon} \right\rceil\right) \\
\text{subject to} \quad & e_i(\mathbf{x}, c) = e_i\left(\mathbf{x}, \left\lceil \frac{t}{\upsilon} \right\rceil\right) \leq 0 \quad \text{for} \quad i = 1, \ldots, E \\
& \forall c \in \mathbb{N}_{>0} \wedge c \leq P
\end{aligned} \tag{2.2}$$

The upper bound P of change periods might be unspecified, e.g., in a real-world control problem where the optimization is continuously running. Analogously to static optimization, global and local optima can be defined for each change period.

In real-world scenarios, the dynamic objective function would be called without time parameter, since it would be realized by an external module whose behavior and output would depend on real-time. Nevertheless, in an implementation without external real-world component, the objective function requires an artificial time measure; in iterative optimization mostly the iteration counter $t \in \mathbb{N}_{>0}$. Depending on the specified change period length υ, which is unknown to the optimizer, the iteration counter is mapped to a change period with $\lceil \frac{t}{\upsilon} \rceil$, where $t < g$ and $g = P \cdot \upsilon$ is the overall number of iterations. With this mapping, changes take place only between iterations, though, in practice, it would be possible that changes occur within iterations. Since literature mostly ignores that case and we assume that the effect on the optimization performance is negligible, in this thesis we only consider changes between iterations.

Various attempts exist to define dynamic optimization problems [Fu+14], but most works agree that a sequence of static optimization problems only is a dynamic problem if relationships exist between the single static instances. Otherwise, restarting the optimization for each static instance would be sufficient, and there would be no need for sophisticated algorithms [Bra02; Ahr+19]. Often it is assumed that dynamic real-world applications underly rather small modifications such that it is reasonable to reuse appropriate information instead of starting optimization from scratch after each change [JB05].

In the remaining parts of this chapter, we describe characteristics of dynamic optimization problems, overview optimization goals and specify the type of problems considered in this thesis.

2.1 Problem Characteristics

The character of a dynamic optimization problem depends both on the topological properties of the static fitness landscape and the type of dynamics that connects the static instances. A fitness landscape is the multi-dimensional surface formed by the fitness values of all possible solutions [Ric10]. It consists of one or more peaks[2] constituting local and global optima. Different approaches to quantify properties of dynamic fitness landscapes have been developed [BSU05; Ric10; Fu+12]. In the following, we describe static and dynamic characteristics of optimization problems in general.

2.1.1 Characteristics of Static Optimization Problems

The properties of a static optimization problem, and thus also those of a dynamic one, depend on the nature of objective function, constraints and solution space. They influence how difficult the problem is to solve and which classes of optimization algorithms are suitable to solve it, see Chapter 3.

The main distinction is between convex and nonconvex problems. Convex problems consist of convex objective and constraint functions, and are therefore solvable in shorter time even with thousands of dimensions [BV11]. If constraint functions are available as in Equations (2.1) and (2.2), the problem is called constrained, otherwise unconstrained. In addition, problems are classified either as multimodal or unimodal. Problems with more than one local optimum are called multimodal [ES15], otherwise they are unimodal. Another distinction can be made according to whether the solution space is discrete or continuous.

Other properties solely concern the objective function. If the objective function is evaluated multiple times for the same input and returns diverging outputs, it is called noisy, otherwise deterministic. An objective function is separable (or decomposable [Rot11]) if a subset of decision variables can be optimized independently from the other ones. Separable problems are easier to solve than non-separable ones, as lower-dimensional sub-problems require less resources. This is referred to as "curse of dimensionality" [Bel15, p. 94]. The formalizations in Equations (2.1) and (2.2) depict single-objective problems. Another scenario are multi-objective problems with multiple objectives that have to be taken into account at the same time [ZLB04]. In multi-objective optimization, a Pareto set of solutions is found. Their fitness cannot be ranked further because no single solution can be best regarding

[2] With "peaks" we refer in maximization problems to the elevations and in minimization to the valleys such that the terms peak and (local) optimum can be used interchangeable.

all objectives if they are conflicting. The Pareto front comprises the fitness values of the solutions in the Pareto set.

Objective functions can also differ in the form in which they are available. They can be categorized as functions with given mathematical formulation and fitness functions that do not have a mathematical representation but only are observable by their input-output behavior. The first category applies for artificial standard benchmark problems, see Paragraph 6.1, while the latter one is much more frequent. It applies to benchmark test suites in online competitions, simulation models that simulate the real world and compute an approximate output for an input, and real-world systems that return a measurable value as reaction to an input, see also [Kra08; YK11]. The objective function's form determines which classes of algorithms are feasible to solve it.

Another property is the objective function's locality [Rot11]. A high locality indicates that the objective function maps similar input vectors to similar output values. This is a requirement for some optimization algorithms, especially nature-inspired ones. They perform worse on problems with low locality, since the objective values would not provide meaningful information about promising search directions.

2.1.2 Characteristics of Dynamics

Various attempts have been made to characterize the dynamics of changes [SC14; Wei03; Jon06; MLY17]. The main aspects are what, when, and how changes. We describe them in the following.

There are different components that can underly changes [NYB12]. The objective function can vary, but also the number of objective functions [CLY18]. The same applies for the constraint functions. Regarding the search space, changes in the number and domain of decision variables are possible. For example, in job shop scheduling problems the number of decision variables varies when jobs are canceled or new jobs arrive [JB05].

Simões and Costa [SC14] characterize dynamic optimization problems according to when and how they change. In some scenarios, the objective function remains static during a change period over a specific number of function evaluations, while changes occur at discrete points in time. The period length may follow a predictable pattern or may be completely random and unpredictable. In other cases, the objective function varies continuously so that each function evaluation leads to a different value. The same classification can be applied to the other modifiable components.

The way how the objective function changes can be classified regarding the function shape. In some problems the fitness landscapes moves, while their shape does not change, i.e., the fitness differences between solutions remains constant. Other problems show a dynamic change of the fitness landscape where new peaks can appear, become flatter, or disappear. Simões and Costa [SC14] introduce the predictability and the severity of the objective function change as categories for a dynamic optimization problem taxonomy.

Further characteristics of dynamic optimization problems, are the following. Some problems comprise time-linkage property [NYB12; MLY17] which means that the solution selected in a change period has influence on the fitness landscape in succeeding points in time. The detectability of changes with simple methods is another criterion [NYB12; MLY17]. Bosman [Bos07] distinguishes online and offline dynamic optimization problems. A dynamic problem is called online, if it is not possible to evaluate the objective function for future points in time. Otherwise, in offline dynamic optimization, it would be possible to optimize the objective function for various points in time at once before the optimized time span begins.

2.2 Optimization Goals

An obvious goal in dynamic optimization is to find for each change period the best combination of values for the decision variables [MLY17; Fu+14] as represented by Equation (2.2). In various publications, e.g., in [Fu+12; MLY17], this strategy is called tracking the moving optimum (TMO). But even for TMO, different performance measures exist (see Paragraph 6.2), that emphasize other aspects.

Robust optimization over time (ROOT) is an alternative goal in dynamic optimization and involves an adapted objective function. In ROOT, the goal is to prevent large differences in the decision variables between solutions of succeeding change periods, since in real-world applications the modification of a current solution would cause costs. Solutions are called robust if their fitness value is acceptable for a number of change periods and underlies only minor changes [Fu+12; Yu+10]. Work on ROOT can be found for example in [YNB19; Fu+12; Fu+13; CMP16; BS07].

In dynamic time-linkage problems it is assumed that chosen solutions influence the future fitness landscape. The goal is to optimize the accumulated reward over all change periods. For this it is helpful to take into account the effect a decision has on the problem. For this kind of problems, [NY13] suggests a more precise problem definition. Jin *et al.* [Jin+13] use autoregression and radial basis function models to predict the future fitness of solutions. Yu *et al.* [Yu+10] propose ROOT as means to tackle time-linkage problems.

A completely different task in dynamic optimization is surrogate modeling (or meta-modeling) that aims at creating a model of the dynamic objective function. If the surrogate model (or meta-model) is sufficiently good, some of the objective function calls can be evaluated with the surrogate model instead of the objective function. This is advantageous in applications where objective function calls are expensive, e.g., in terms of time. Recent work on this topic is done, e.g., by [Luo+19; MB15; MB13].

In general, all problem properties and optimization goals mentioned before can occur in combination. For example, [Yaz+18] deals with a dynamic multi-objective time-linkage problem with robust optimization as goal.

2.3 Dynamic Optimization Problems in the Thesis

In this thesis, we consider unconstrained deterministic optimization problems with one objective. Solution space $\mathbb{R}^d$ and objective space $\mathbb{R}$ are real-valued and do not change over time. We take into account problems with dependencies between problem instances at succeeding points in time. The objective functions mostly are nonconvex and multimodal, but we also apply convex unimodal benchmarks. All the problems are given as mathematical functions and some of them are separable. But we take advantage of neither the separability nor the mathematical formulation, since all optimizers in this thesis handle the objective function as black-box. The locality of the objective functions is sufficient for the employed optimizers, since they were explicitly designed for nature-inspired optimization.

In this thesis, only the objective functions underly modifications. Changes take place between change periods and occur with fixed frequency at discrete points in time. The change frequency is chosen so that the optimizer has some generations to start converging to the optimum. Too fast changes, e.g. after each generation, would require the optimizer to find the optimum in one step which contradicts the design of iterative meta-heuristics [Ric10].

We consider two kinds of objective function changes. In some settings, static objective functions move through the solution space leading to abrupt changes. In other scenarios, the objective function is subject to continuous modifications. In most cases, we consider predictable changes, otherwise no insight could be gained into the advantages of prediction approaches. No time-linkage problems are taken into account and changes are detectable, since prediction takes place change-triggered. Otherwise, prediction would never be conducted. We consider online optimization because this is more related to real-world scenarios like control problems. Our goal is accurate tracking of the optimum (TMO), and we ignore the robustness of solutions. With various performance measures, we examine different characters of the algorithms, see Paragraph 6.2.

3 Nature-Inspired Optimization Algorithms

Since decades, research on optimization has been driven in various domains like mathematics, operations research, or computational intelligence. The many domains led to diverse families of optimization algorithms. The algorithms can either be grouped by the kind of problems they solve, e.g., combinatorial or multimodal, or by the search strategy they pursue, e.g., local or stochastic. There exist various comprehensive publications on certain groups of methods, like convex [BV11], combinatorial [Hro04], numerical [NW99], or meta-heuristic [LMR09; ES15; RBK12] optimization. In the following, we describe the most important families of optimization algorithms according to the optimizers' properties. Afterwards, we introduce the main principles of nature-inspired optimization, and explain the nature-inspired algorithms evolution strategy and particle swarm optimization in detail because the contributions of this thesis rely on them.

3.1 Overview of Optimization Methods

In optimization, the goal is to find the global optimum, preferably with low computational effort. Optimization strategies differ in whether their focus is on optimality or effort, and can mainly be distinguished into exact and approximate approaches. Exact methods, among them are analytical and exploration-based ones, guarantee to find the global optimum. Analytical methods are not applicable if a mathematical description is not available or the problem is too complex, e.g., NP-hard [Kru+16; Rot11]. Complete or exhaustive exploration searches for the optimal solution by going through all possible solutions in the whole solution space [Kru+16]. This is only feasible for combinatorial problems, since they have a finite search space. Nevertheless, in practice even for most combinatorial problems this approach would be too time-consuming due to the curse of dimensionality. Since all methods mentioned so far mostly are not practical for real-world solution spaces with many dimensions and unknown mathematical description of the objective function, there exists a large variety of iterative methods that try to approximate the optimum, either by local or global search strategies.

Local optimization, most successful in unimodal problems, since they are likely to get stuck in local optima, can be distinguished into derivative-based and derivative-free ones. Derivative-based algorithms, like Newton's method or conjugate gradient methods [Yan11], require explicit derivatives and therefore only are applicable to objective functions with certain characteristics, e.g., twice differentiability. Local derivative-free methods do not have such requirements. They solely rely on objective function evaluations; examples are hill-climbing or pattern search [YK11]. Local optimization methods have in common that on multimodal problems the starting point has strong influence on the found solution. A local optimum near to the initial point is likely to be selected as final result.

Since many optimization problems are multimodal so that local methods often do not provide sufficient solution quality, the need for global optimization methods rose. Global optimization methods actively try to overcome local optima with random components, but nevertheless cannot guarantee finding the global optimum. Kruse *et al.* [Kru+16] differentiate between blind random search methods, that randomly evaluate solutions in the solution space, and guided random search methods, that gain information about promising regions during the optimization process and search more efficiently by utilizing it. Both types are derivative-free and can be applied to a broad range of problems.

The main class of algorithms belonging to guided random search are meta-heuristics. They are problem-independent, since they handle the objective function as black-box. Their parameters and operators have to be adapted to the respective problem. A key characteristic of meta-heuristics is their any-time property, which means that the algorithms provide a feasible solution for the problem at any point in time during the optimization run. This is a main advantage over analytical or exhaustive methods, that may have a long runtime and only return the solution at the end of the optimization process so that no intermediate, sub-optimal solutions are available.

Nature-inspired optimization methods are a class of meta-heuristics whose operators and algorithmic structure are inspired by principles that can be observed in nature. Examples are evolution strategies or swarm intelligence, and other stochastic optimization approaches like simulated annealing, tabu search [GL97], or deluge algorithm [Kru+16]. Many meta-heuristics are explained in [Yan14], and very recent ones are listed in [Dha+19].

3.2 Principles of Nature-Inspired Optimization

Nature-inspired methods start with an initial set (population) of random candidate solutions (individuals) and try to improve them iteratively. Information gained during the optimization process is utilized to lead the population to better solutions and to introduce randomness to overcome local optima. The algorithms differ in how they memorize good solutions and how they pull the population to attractive search regions. While nature-inspired methods are population-based, other meta-heuristics iteratively improve a single solution, i.e., are trajectory-based [BLA16; Yan14]. Populations are advantageous because they ease exploration of the solution space.

Exploration and exploitation describe different search strategies during the optimization process. Exploration should take place in the beginning, since the optimizer has to inspect the solution space to find regions with good fitness level. This phase is characterized by a large diversity in the population. Exploitation is the phase when the algorithm starts to converge to a solution. For this, decreasing diversity in the population is necessary so that it concentrates on the region surrounding the solution. A reasonable balance between exploration and exploitation is required in order to enable sufficient search in the whole solution space and to ensure final convergence to a solution. For nature-inspired optimizers, various techniques exist to influence the intensity of exploration and exploitation that mainly can be distinguished into static parameter tuning before the optimization and dynamic parameter control at runtime [ES15; Kra08].

3.3 Evolution Strategies

Evolution strategies (ES) belong to the family of evolutionary algorithms and have been introduced by Rechenberg and Schwefel [Rec73]. The search is based on a population $\mathbf{P} \in \mathbb{R}^{\mu \times d}$ of μ individuals $\mathbf{x} \in \mathbb{R}^d$, which are subject to recombination and mutation. Algorithm 1 shows the pseudocode of the prominent $(\mu + \lambda)$-ES. At the beginning, μ initial solutions are generated, e.g., randomly in the solution space (Line 1). In each generation, λ offspring solutions are created based on recombination and mutation (Line 3). At the end of each generation, the solutions with the best fitness are selected as parents for the next generation (Line 4). In the following, we explain the key concepts of ES in detail, which are recombination, mutation, selection, and step size adaptation. More information on ES can be found, e.g., in [BS02; HAA15; Kra17].

Algorithm 1 (μ+λ)-ES

```
P ← initialize_population()
for generations do
    P′ ← create_λ_offspring_individuals(P)        # recomb. & mutation
    P ← select_best_μ_individuals(P, P′)
```

Recombination Recombination, also called crossover [Kra08], creates λ offspring individuals from ψ randomly selected parent individuals $\mathbf{p}^1, \ldots, \mathbf{p}^\psi$. The goal of recombination is to retain and combine information that achieved good fitness during the last generations. For ES, mostly dominant or intermediate recombination is applied. In dominant recombination, entry x_j of the offspring is taken from the respective entry of parent $\mathbf{p}^i$ randomly chosen among the ψ parents:

$$x_j = p_j^i \text{ with } i \sim \mathcal{U}(1, \psi) \quad \forall\ j \in \{1, \ldots, d\}, \tag{3.1}$$

where i is independently drawn for all $j \in \{1, \ldots, d\}$. This procedure is repeated to generate all λ offspring individuals. In intermediate recombination, the offspring individuals are generated by averaging the chosen parents:

$$x_j = \frac{1}{\psi} \sum_{i=1}^{\psi} p_j^i \quad \forall\ j \in \{1, \ldots, d\} \tag{3.2}$$

Mutation Mutation takes place after recombination and adds randomness to offspring individuals in order to gain new information that might lead to better fitness. Often Gaussian mutation is applied [Kru+16]:

$$\mathbf{x}' = \mathbf{x} + \mathcal{N}\left(0, s^2\right) \tag{3.3}$$

The standard deviation of the normal distribution, in this context often called mutation strength or step size, is signified with s. There exist advanced strategies that sample a multivariate normal distribution with different spreads in the dimensions. The well known covariance matrix adaptation ES (CMA-ES) is one example, see e.g. [Rud12; HO01] for details.

Selection Selection chooses μ individuals for the next generation in order to preserve information that probably will lead to promising solutions. For ES, plus and comma selection exist [Kru+16], denoted with $(\mu + \lambda)$-ES and (μ, λ)-ES, respectively. In plus selection, the best individuals among both the parent and offspring population are selected, while comma selection only considers the offspring population. ES with plus selection are more prone to get stuck in local optima, compared to comma selection. In contrast, comma selection can loose the best solution found so far.

Step Size Adaptation As motivated before, exploration and exploitation need to be balanced during the optimization process. An important strategy to achieve this is step size adaptation that modifies the mutation strength depending on the search progress. For exploration, larger step sizes are useful, as the population could be distributed wider, while exploitation is supported by smaller mutation steps. Step size adaptation is the main concept in ES to achieve faster convergence than random search [HAA15]. Rechenberg's 1/5th success rule [Rec73] is a famous way for step size adaptation that belongs to dynamic parameter control strategies [Kra08]. Repeatedly, after a specified number of iterations or mutations, the step size is adapted depending on a success factor. Success factor p_s is the ratio of successful mutations, that led to offspring individuals with better fitness, to the overall number of mutations [ES15]. The new step size s' is computed as

$$s' = \begin{cases} \frac{s}{\xi} & \text{if } p_s > \frac{1}{5} \\ s \cdot \xi & \text{if } p_s < \frac{1}{5} \\ s & \text{if } p_s = \frac{1}{5} \end{cases}, \tag{3.4}$$

where $0 < \xi < 1$. This update leads to increasing step size if many successful mutations occurred in the past.

3.4 Particle Swarm Optimization

Particle swarm optimization (PSO) belongs to the family of swarm intelligence algorithms [BL08; MLY17] that covers, among others, ant colony optimization [DS04], bacterial foraging optimization [Pas02] or bee colony optimization [Teo09]. PSO has been introduced by Kennedy and Eberhart [KE95], and is based on the idea of a swarm of flying particles (solutions) that are attracted by best found solutions. Algorithm 2 shows the pseudocode of the basic PSO algorithm.

Each particle in a swarm $\mathbf{S}$ has a position $\mathbf{x} \in \mathbb{R}^d$, that represents a candidate solution, and a velocity $\mathbf{v} \in \mathbb{R}^d$. Both parameters are iteratively

Algorithm 2 PSO

```
$\mathbf{S} \leftarrow$ initialize_swarm_particles()          # random position&velocity
for iterations do
    $\mathbf{S} \leftarrow$ move_particles($\mathbf{S}$)          # following Eq. (3.5), (3.6)
    $\mathbf{x}^p, \mathbf{x}^s \leftarrow$ update($\mathbf{x}^p, \mathbf{x}^s$)          # update $\mathbf{x}^p$ for each particle
```

updated with:

$$\mathbf{v}_t = \omega \mathbf{v}_{t-1} + \theta_1 \mathbf{r}_1 \circ (\mathbf{x}^p - \mathbf{x}_{t-1}) + \theta_2 \mathbf{r}_2 \circ (\mathbf{x}^s - \mathbf{x}_{t-1}) \tag{3.5}$$

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \mathbf{v}_t \tag{3.6}$$

Variable $\mathbf{x}^p$ stores the best solution the particle has found during lifetime, while $\mathbf{x}^s$ saves the best position ever found by the whole swarm. Parameter ω is the inertia weight controlling the influence of the previous velocity, and θ_1 and θ_2 are scalars balancing the particle's egoism and altruism. Sometimes they are called cognitive and social factors, respectively [Cho+18]. The random vectors $\mathbf{r}_1, \mathbf{r}_2$ are uniformly drawn from $[0,1]^d$ [BBL08] and allow random explorations, where $\circ$ is the Hadamard product. The optimization process is terminated when a criterion is met.

Many PSO variants have been proposed, see e.g. [CE15; PV10; PKB07]. Variations mainly concern the type of neighborhood model, and how the parameters are set. We depict these aspects in the following paragraphs.

Neighborhood Topologies The neighborhood topology determines which solution is used for $\mathbf{x}^s$. Algorithm 2 reflects the global neighborhood structure, also called global best model, where the best solution ever visited by the whole swarm is employed as $\mathbf{x}^s$ for all particles. In an alternative setting, called local best model, local neighborhood relations are defined, and $\mathbf{x}^s$ of a particle is the best particle in its local neighborhood [MLY17]. Thus, different particles may have distinct particles $\mathbf{x}^s$. The global best model is the traditional one, while the local best model is claimed to better prevent getting stuck in local optima. In addition, many other topologies, like random or ring, have been proposed trying to impede premature convergence [ZWJ15; CE15]. Engelbrecht [Eng13] compares global and local neighborhood models on many benchmark problems and shows that neither always outperforms the other topology. Thus, the topology has to be chosen regarding the specific problem at hand.

Choice of Parameters While the convergence properties of ES mainly depend on the mutation strength, in PSO the parameters ω, θ_1 and θ_2 influence exploration and exploitation, and by this means convergence. Large inertia values ($\omega > 1$) cause large steps of the particles, lead to exploration, and make it difficult for the particles to change their direction [LE13]. A small inertia weight ($\omega < 1$) leads to decreasing lengths of the particles' steps and supports exploitation. Besides, the parameters θ_1 and θ_2 influence the optimization progress. When θ_1 is large in contrast to θ_2, the particles are mostly attracted by their own best position so that the swarm diverges in different directions. The opposite case increases the particles altruism leading to convergence to the swarms' best position.

By dynamic modification of these three parameters, exploration or exploitation can be intensified in different phases during optimization [LE13]. Various strategies especially for dynamic adaptation of inertia weight have been proposed, see e.g. [HEOB16a] for an overview. Nevertheless, in some studies, e.g., [HEOB16b; HEOB16a; ZE14], dynamic choice of parameter values turn out to be likely to produce either divergent behavior or cause premature convergence. Harrison *et al.* [HEOB16a] conclude that a static or random inertia weight is the best option. There exist convergence proofs showing which parameter values lead to convergent behavior in PSO; we refer to some of them in Chapter 8.

4 Dynamic Optimization Approaches

The simplest approach to solve dynamic optimization problems is to treat each change period as a static problem and to run an optimization algorithm separately in each period. In contrast, dynamic optimization algorithms assume dependencies between change periods. They are executed once and adapt to changes during their runtime [NYB12]. By this means, they can utilize information from the past and might converge faster than an optimization started from scratch [Ahr+19]. The more similar the problem instances are at different points in time, the more reasonable it is to re-use information gained during the optimization process [JB05]. Meanwhile, many approaches have been proposed to solve dynamic optimization problems; surveys can be found in [BRAK13; CGP11]. Among nature-inspired approaches are, e.g., differential evolution [MM05; HDM13; Has+19], co-evolution [Liu+14; GT09], and particle swarm optimization [BBL08; Jor14; MLY17], while ES are most frequently used in dynamic optimization [BRAK13; CGP11; JB05]. Therefore, in this chapter we describe common extensions of ES for dynamic optimization problems; adaptations of PSO for dynamic optimization are depicted in Chapter 8.

4.1 General Approaches

Different extensions of ES have been developed in order to deal with dynamic problems. The general problem when ES dedicated to static optimization are applied to solve a dynamic problem is their fast convergence. When an ES was running during a time without objective function changes, it may be converged to a solution so that the whole population is located very close to this point. When the optimum slowly moves through the solution space, standard algorithms may be able to follow. But if optima tend to jump within the solution space, standard optimization algorithms might fail to follow or even to detect new basins of attraction and get stuck in the previously found local optimum. The basin of attraction of a local optimum is the set of all points from which this optimum can be reached by gradient descent [Shi12; Ase+13]. So, enforcing diversity in the population at the right point in time is important to enable flexibility to adapt to a modified fitness landscape [JB05]. Different approaches have been investigated to make ES suitable for dynamic problems. Based on the overviews [CGP11; NYB12], we describe some of them in the following.

Maintaining diversity A strategy to overcome getting stuck in a former local optimum is to maintain diversity. In each generation, some random solutions, in this context often called immigrants or gifts, are inserted into the population so that other regions of the solution space can be explored as well [Gre92; TY07; CY13]. There also exist approaches that modify the algorithms' operators in order to preserve diversity in the population [Ahr+19].

Change detection Another approach is to actively detect a change in order to adapt the algorithm behavior afterwards. A change can be detected for example by re-evaluating individuals [ATE17; Ric09]. If their fitness values in the current time step differ from those in the former time step, the fitness function has changed. After a change has been detected, introducing diversity, e.g., by adding random individuals into the population, could be a strategy to support the ES in discovering the new optimum [Gre92]. Another approach to achieve more diversity, called hyper-mutation [Cob90], increases the mutation rate in ES after a change.

Memory Furthermore, the ES can employ memory that stores good solutions found during previous generations. Individuals from the memory could replace individuals of the population in later generations if they have a better fitness. This might be helpful especially if the changes are cyclic. See [BLA16] for an overview.

Multi-population In multi-population approaches, multiple populations are simultaneously searching in various regions of the solution space. By this means, not only the global but also local peaks can be covered at the same time. If one of the local peaks becomes the global one, the optimizer can converge very fast as there already is a population located at that peak. A well-known multi-population ES is the self-organizing scouts algorithm [Bra+00], another recent work is [Li+16]. In [Li+15], a survey on multi-population strategies can be found. The multi-population approach can also be utilized for co-evolution or maintaining diversity [NYB12].

Self-adaptive In this strategy, parameters of the optimizer, e.g., mutation rate, are encoded as genes and underly the evolutionary process as well. By this means, the optimizer is supposed to adapt the parameters according to the current situation of the population. Examples for this branch of strategies are [Gre99; Urs00].

Prediction Additionally, approaches have been introduced that predict the next optimum after a change has been detected by the ES. Inserting this prediction as solution into the population, the ES starts with a population containing an individual that is located maybe somewhere near to the real optimum. This approach works best if the changes are not completely random but follow a defined pattern.

Often, combinations of these approaches are applied. For example, strategies that do not maintain diversity but react to changes, like prediction, frequently are combined with a change detection mechanism. In the next paragraph, we overview prediction strategies in detail.

4.2 Prediction-Based Approaches

If changes in the objective function are not solely based on randomness but obey a certain pattern, prediction approaches can be applied. Prediction approaches use an archive as a training set which is used for learning, i.e., for adapting parameters of the prediction method. These concepts are explained in Chapter 5. Following [NYB12; MK20], prediction approaches for dynamic optimization can be classified into the following concepts.

Predicted optimum as immigrant Predicted optima can be integrated into the population to enrich it with immigrants. An autoregressive approach is used by [HW06], a Kalman filter is applied by [RAD08; MTV16]. Similar approaches, sometimes called "population-based prediction" [Ron+19], adapt the population based on the predicted Pareto set [WJL15; FS17; RGZ16] or predicted characteristic points [ZJZ14; Jin+16; Zho+18].

Predicted optimum for operators Predicted optima can be used to bias operators, e.g., the mutation operator [RAD08] or the PSO equations [MK18b], to lead the population to the new optimum.

Prediction for fitness functions Predictions can also be used to influence the evaluation of solutions. For example, [RAD08] proposes to improve the fitness of solutions in the neighborhood of predicted optima. It is also possible to learn dynamic meta-models of the near future of the fitness function, e.g., one step ahead [Bu+17; Hem+01].

Prediction of individuals In this class of approaches, also named "individual-based prediction" [Ron+19], the next position for each individual is predicted based on the position of its predecessor [Zho+07; Liu+14]. Thus, for each individual a separate prediction model is employed.

Prediction of point in time Some approaches [SC08a; SC09] concentrate on predicting the time, at that the next change will take place. This information can be used to reset the algorithm, e.g., to initial settings like large step sizes. It can also be combined with approaches that predict the optima while using promising initial parameters.

Prediction of problem type For combinatorial problems, [SC14] proposes to predict the problem type that will occur next. Then, individuals from memory that had good fitness in the predicted kind of problem replace the worst ones in the population.

Prediction for time-linkage problems If the dynamics of the problem is influenced by solutions chosen by the optimization algorithm, e.g., in job scheduling algorithms, approaches can take into account predictions of probable changes of the objective function based on own solution choices [Bos05; BLP07; NY09b; Bu+17].

The choice of prediction approach is determined by the chosen optimization algorithm and the application context. For example, in this work we show that for PSO using predictions as immigrants is less successful than biasing the swarm to the predicted directions (Chapter 8). Moreover, predicting when the next change will occur is only helpful if the points in time at which changes take place follow a predictable pattern. For unpredictable points in time, the use of mechanisms which detect changes is recommended. In our work, we use predicted optima as immigrants and for biasing operators (Chapters 7, 8, and 9). In Chapter 5, we give an overview of time series prediction methods that can be employed for the mentioned approaches.

5 Time Series Prediction Methods

Prediction methods are statistical or machine learning techniques that fit models to observed data so that they are able to predict an output for new, unseen data. Oriented to the terminology in machine learning, we call the fitting process learning or training, and the data used for learning training data or training set. We use the term training item for a certain element of the training set. In a supervised learning scenario, training data comes in pattern-label pairs $(\mathbf{p}_j, y_j)$ with $j \in \{1, \dots, M\}$ and training set size M. Patterns are the inputs $\mathbf{p}_j$, while labels are the outputs y_j corresponding to the inputs. Both can also have other than the depicted formats. The training data is used to train a prediction model Ψ so that it outputs a reasonable prediction $\Psi(\tilde{\mathbf{p}})$ given a novel pattern $\tilde{\mathbf{p}}$. Training adapts the model parameters minimizing the difference $|\Psi(\tilde{\mathbf{p}}) - \tilde{y}|$ between the estimated and expected true output for the training data. This is an optimization problem solved in different ways by the prediction methods. For an introduction to machine learning and prediction methods, we recommend [HTF09; Bis07] and further introductory textbooks. The methods differ mainly in their computational complexity and in the types of relationships in the data they are intended to cover, e.g., linear or nonlinear.

Time series prediction is one branch of prediction methods where the task is to estimate the next step $\mathbf{x}_t$ of a series $[\mathbf{x}_0, \dots, \mathbf{x}_{t-1}]$ of multivariate observations $\mathbf{x}_i \in \mathbb{R}^d$ from succeeding time steps.[1] For this, techniques from different research domains exist. Various methods were developed in statistics, ranging from simple ones like autoregressive models to sophisticated approaches like ARIMA [SS17; MJK15; YM00]. There are also successful approaches from machine learning [BTB12; RS19], like decision trees, nearest neighbors regression, and neural networks [Zha12].

In time series prediction, training data consists of one time series.[2] For machine learning methods, this series has to be converted into supervised

[1] Another frequent task is to predict a scalar from a series of multivariate observations, e.g., predicting the output of a failing sensor based on measurements of other sensors.

[2] Also multiple (independent) time series would be possible. Since in this thesis the optimum movement is a single-series problem, we restrict the descriptions in this chapter to the single series case.

training data. Usually the sliding window approach is used, where w_s is the window size. The first w_s time steps of the series form the pattern, and the succeeding time step becomes the label. Then, the window is moved one step ahead so that it overlaps in $w_s - 1$ steps with the first window. The pattern of the second training item comprises the values underlying the window, while the next value is the label. This process is repeated until the whole series is split into pattern-label pairs. Overall, the training set would consist of $t - w_s$ items of the form:

$$(\underbrace{\mathbf{x}_i, \mathbf{x}_{i+1}, \dots, \mathbf{x}_{i+w_s-1}}_{\text{pattern}}, \underbrace{\mathbf{x}_{i+w_s}}_{\text{label}}) \tag{5.1}$$

The choice of w_s influences the learning ability, and depends on the prediction method and time series characteristics like length of periodicity. Time series prediction methods have different requirements on the data, such as stationarity or no missing data, that can heavily influence their performance. This makes data preprocessing necessary, see [MJK15; MSA18a] for different kinds of adaptations, and also restricts the possible applications.

In the following, we depict the relevant prediction methods that have found application in evolutionary dynamic optimization, e.g., in [RAD08; MK18b; HW06]. An overview of prediction methods in ES is given in Paragraph 7.1.1. Which of those methods should be applied, depends on the problem characteristics, e.g., kind of dynamics or number of change periods which determines the amount of training data.

5.1 Persistence Model

The persistence model, also called naive forecast, is the simplest prediction approach. The value for the next time step is assumed to be the same as for the last time step, thus no training procedure is required:

$$\hat{\mathbf{x}}_t = \mathbf{x}_{t-1} \tag{5.2}$$

5.2 Autoregressive Model

Statistical learning methods [Jam+13] are historically based on the area of statistics. Basic methods are linear models that assume a linear relationship between patterns and labels. Autoregressive models (AR) belong to this class and can be applied to various types of time series [HA13].

An AR model predicts $\mathbf{x} \in \mathbb{R}^d$ for time step t with

$$\hat{\mathbf{x}}_t = \boldsymbol{\theta} + \sum_{i=0}^{p-1} \boldsymbol{\Phi}_i \mathbf{x}_{t-1-i} + \boldsymbol{\varepsilon}_t, \tag{5.3}$$

where $\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\varepsilon} \in \mathbb{R}^d$, and $\boldsymbol{\Phi}_i \in \mathbb{R}^{d \times d}$. Order p defines how many previous time steps are considered [HA13], $\boldsymbol{\Phi}_i$ controls the influence of $\mathbf{x}_{t-1-i}$ on $\hat{\mathbf{x}}_t$, and vector $\boldsymbol{\theta}$ consists of constants. The noise parameters $\boldsymbol{\varepsilon}_t$ are uncorrelated [NS01], and contain values with zero mean and covariance matrix $\mathbf{C} \in \mathbb{R}^{d \times d}$. The least squares method is used to determine parameters $\boldsymbol{\theta}$, $\boldsymbol{\Phi}$ and $\mathbf{C}$. Autoregressive models base on some assumptions, so as the stationarity of time series and no missing data. There exist different approaches to transform data to ensure both criteria [YM00].

5.3 Kalman Filter

The Kalman filter [Kal60] is a linear time series model that is based on the assumption that the state of a system is observable by noisy measurements. It can be applied to both estimating the true state variables $\boldsymbol{a}_{c-1}$ underlying noisy observations and predicting the next state $\hat{\boldsymbol{a}}_c$ for that not yet observations are available. It is a recursive model. First, a priori estimations of the next system state $\hat{\boldsymbol{a}}_c^-$ and its error covariance $\hat{\boldsymbol{E}}_c^-$ are computed. After having obtained an observation for the next time step, the filter model is optimized so that the a posteriori covariance error is minimized. This is interpreted as a least-squares problem to minimize the deviation of the observation and the predicted value. With the updated model, the a posteriori state estimation $\hat{\boldsymbol{a}}_c$ and its error covariance $\hat{\boldsymbol{E}}_c$ are computed. The Kalman filter requires a linear dynamic and additive Gaussian noise. For further details on Kalman filter see, e.g., [Sor70; Wal02; BH12]. Although there exist sophisticated variants of the Kalman filter, e.g., the extended Kalman filter for nonlinear dynamics, so far only the linear Kalman filter has found application in nature-inspired dynamic optimization.

5.4 Recurrent Neural Network

Neural networks (NNs) have been introduced as nature-inspired learning methods in the 1940s [Cel91; GBC16]. Increasing amount of available data, hardware infrastructure and software tools caused the success of NNs

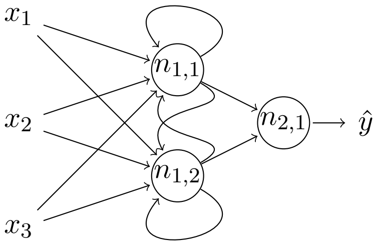


Figure 5.1: Recurrent neural network

since the 2000s and led to ongoing growth of powerful network architectures [GBC16].

A kind of architectures called recurrent neural networks (RNNs) have shown strengths in time series prediction [Lev16; Wu+16] and are neural networks representing past observations with hidden states. In RNNs, neurons are connected with themselves so that the neurons' output is led back to their input [Roj96]. Figure 5.1 depicts a simple RNN architecture. Let $\mathbf{x}_t \in \mathbb{R}^d$ be the input of a neuron and ϕ the function the neuron outputs. Then the neuron computes $\phi\left(\mathbf{x}_t, \phi\left(\mathbf{x}_{t-1}, \phi\left(\mathbf{x}_{t-2}, \ldots\right)\right)\right)$ while taking into account values of $\mathbf{x}$ for previous time steps. Due to their backward connections, RNNs can model relationships between observations from multiple sequential time steps. How many time steps the RNN considers for learning, i.e., window size w_s, influences prediction performance and should be chosen depending on the problem. Due to the vanishing gradient problem, RNNs are trained with a modified backpropagation procedure known as backpropagation through time, see [Roj96; GBC16]. In backpropagation, the training set is divided into parts (batches). After a batch is passed through the network, the weights are adapted according to the computed error. The number of epochs, i.e., how often the complete training set is processed, influences prediction quality. Various extensions of RNNs have been proposed. A prominent one is the long short-term memory (LSTM) network [HS97] that allows learning insertions and deletions of patterns with mechanisms like attention and forget gates.

Unlike AR models or Kalman filters, neural network-based prediction does not come with certain restrictions to the time series. Nevertheless, appropriate data preprocessing, e.g., normalization, can strongly influence prediction performance [Smy19; Ahm+10].

5.5 Temporal Convolutional Network

Temporal convolutional networks (TCNs) are specialized to time series data. Among different TCN architectures, we describe the architecture proposed by [BKK18] (Figure 5.2). In contrast to fully-connected NNs, that consist of layers of neurons and weighted connections between them, TCNs are based on the concept of filters that are sliding over the input time series. For a univariate input time series $\mathbf{s} \in \mathbb{R}^T$ with T time steps, a filter is a vector $\boldsymbol{\psi} \in \mathbb{R}^k$; its elements are called weights. For each element s_j from input $\mathbf{s}$, the convolutional operation

$$Q(s_j) = \sum_{i=0}^{k-1} \psi_i \cdot s_{j-D \cdot i} \tag{5.4}$$

is computed so that the filter output is a scalar value. Here, ψ_i is the filter weight at index i in $\boldsymbol{\psi}$ and D denotes the so-called dilation factor. If $D = 1$, operation Q involves element s_j and a history of $k-1$ elements, i.e., the elements “left” to s_j in series $\mathbf{s}$. With an arbitrary dilation factor, the history captured by a filter is $D \cdot (k-1)$ which follows from Equation (5.4). Applying Q to all elements s_j of the input series $\mathbf{s}$ results in a sequence with the same length as $\mathbf{s}$ if additional elements with value zero are appended at the beginning of the sequence, see Figure 5.2. A filter covers a larger part of the input sequence, if the dilation factor is increased. For example, with $D = 2$ the filter is applied to every second input element so that it considers a two times longer history.

A complete TCN is constructed by stacking residual blocks. A residual block contains two layers of filters having the same dilation factor, and a residual connection so that the block input is added to the output of the second layer [BKK18]. A frequent setting is to increase the dilation factor exponentially among the blocks with $D_b = 2^b$, where b is the block index with $0 \leq b < \#\text{blocks}$. The history h captured by the whole network, i.e., the number of history time steps in the input series involved to compute an entry in the output series, equals twice the sum of histories the filters have in different blocks:

$$h = \sum_{b=0}^{\#\text{blocks-1}} 2 \cdot D_b \cdot (k-1) = 2 \cdot (k-1) \cdot \sum_{b=0}^{\#\text{blocks-1}} D_b \tag{5.5}$$

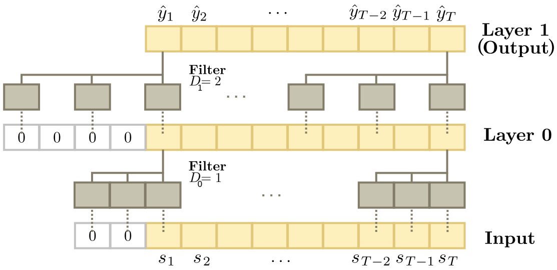


Figure 5.2: Two-layer TCN (no residual blocks) based on [BKK18], $k = 3$. Input and output sequences are univariate with T time steps

If $D_b = 2^b$, Equation (5.5) can be simplified with the geometric series to:

$$h = 2 \cdot (k-1) \cdot (2^{\#\text{blocks}} - 1) \tag{5.6}$$

With this relationship and chosen filter size k, the number of blocks necessary to cover an input series of a certain length w_s so that $w_s \leq h + 1$ is:

$$\#\text{necessary blocks} = \left\lceil \log_2 \left(\frac{w_s - 1}{2 \cdot (k-1)} + 1 \right) \right\rceil \tag{5.7}$$

Since history h is defined being one time step shorter than the whole network output series, the condition $w_s \leq h + 1$ is required. In our context of prediction in dynamic optimization, we use $\hat{y}_T$ in the network output sequence as predicted optimum, see Figure 5.2.

6 Performance Measurement

For evaluation and comparison of different approaches, benchmark problem sets and performance measures are essential. Various benchmarks and metrics are available in literature that examine different properties of algorithms. Which benchmarks and measures should be employed, depends on the optimization goal and the use case. In this chapter, we describe standard problems and measures frequently used in dynamic optimization. Furthermore, we propose a new benchmark and performance measure tailored to the specifics of prediction-based algorithms.

6.1 Benchmark Problem Sets

In nature-inspired dynamic optimization, various benchmark sets have been proposed [MD99; YY03; LY08; NY09a; GTA17; JY17; Yaz+19; PE19]; an overview is given in [NYB12]. In the following, we introduce a selection of well-known dynamic benchmark problems we find appropriate for testing prediction methods on problems of the kind considered in this thesis, see Paragraph 2.3.

6.1.1 Moving Peaks Benchmark

The moving peaks benchmark (MPB) proposed by [Bra99] is a standard test set in nature-inspired dynamic optimization. The fitness function is nonseparable [Yaz+19] and consists of multiple peaks with randomly changing positions, heights, and widths. The peaks move linearly controlled with noise that can be adjusted via a correlation factor, see [MC13]. The global optimum may jump if a previous local optimum becomes the new global one. Thus, the optimum fitness may change among change periods.

The fitness of solution $\mathbf{x}$ in change period c is computed with:

$$f(\mathbf{x}, c) = \max_{i=1,\dots,\#\text{peaks}} \frac{H_c^i}{1 + W_c^i \|\mathbf{x} - \mathbf{X}_c^i\|_2^2}, \tag{6.1}$$

where $\| \cdot \|_2^2$ is the squared Euclidean norm. Height H_c^i, width W_c^i, and position $\mathbf{X}_c^i$ of the ith peak underly random changes:

$$H_c^i = H_{c-1}^i + s_h \cdot \varepsilon \tag{6.2}$$

$$W_c^i = W_{c-1}^i + s_w \cdot \varepsilon \tag{6.3}$$

$$\mathbf{X}_c^i = \mathbf{X}_{c-1}^i + \mathbf{S}_c^i \tag{6.4}$$

with $\varepsilon \sim \mathcal{N}(0,1)$, $H_c^i, W_c^i \in \mathbb{R}$, and $\mathbf{X}_c^i \in \mathbb{R}^d$. Height severity s_h and width severity s_w are static factors determining the amount of disturbance in the height and width, respectively, while $\mathbf{S}_c^i$ causes the position movement. Its severity is determined by shift length $s_l = \|\mathbf{S}_c^i\|_2$, while $\mathbf{S}_c^i$ is computed with

$$\mathbf{S}_c^i = \frac{s_l}{\|\boldsymbol{\varepsilon} + \mathbf{S}_{c-1}^i\|_2} \left((1-\eta)\,\boldsymbol{\varepsilon} + \eta \mathbf{S}_{c-1}^i \right), \tag{6.5}$$

where $\boldsymbol{\varepsilon}$ is a vector of random values. Since in literature no information on the distribution is given, we sample the values from a standard normal distribution. Correlation factor $\eta \in [0,1]$ leads to a completely random movement for $\eta = 0$, and with $\eta = 1$ to a deterministic linear shift. If $\mathbf{S}_c^i$ would place $\mathbf{X}_c^i$ outside the solution space, it is multiplied with -1 so that the peak moves backwards again. Correlation factor η was introduced after the first publication [Bra99]. In the first variant, shift vector $\mathbf{S}_c^i$ was drawn completely randomly but also had specified length s_l.

According to literature on MPB, e.g., [Li+15; MC13; Yu+10], Equation (6.5) is constructed so that $\mathbf{S}_c^i$ has the specified length s_l. Factor $\frac{s_l}{\|\boldsymbol{\varepsilon}+\mathbf{S}_{c-1}^i\|_2}$ is claimed to normalize the new shift vector $(1-\eta)\,\boldsymbol{\varepsilon} + \eta\mathbf{S}_{c-1}^i$ such that it gets length s_l. Nevertheless, this is only the case if $\|\boldsymbol{\varepsilon} + \mathbf{S}_{c-1}^i\|_2$ and $\|\,(1-\eta)\,\boldsymbol{\varepsilon} + \eta\mathbf{S}_{c-1}^i\|$ are equal. This is not guaranteed, since factor η modifies the latter so that $\boldsymbol{\varepsilon}+\mathbf{S}_{c-1}^i$ and $(1-\eta)\,\boldsymbol{\varepsilon}+\eta\mathbf{S}_{c-1}^i$ are different vectors. To ensure proper length of $\mathbf{S}_c^i$, we modify Equation (6.5) to:

$$\mathbf{S}_c^i = \frac{s_l}{\|(1-\eta)\boldsymbol{\varepsilon} + \eta\mathbf{S}_{c-1}^i\|_2} \left((1-\eta)\,\boldsymbol{\varepsilon} + \eta \mathbf{S}_{c-1}^i \right) \tag{6.6}$$

Figure 6.1 visualizes an MPB instance as minimization problem with 10 peaks, $s_h = 7$, $s_w = 0.1$, and $s_l = 0.6$. The peaks' movement is shown in Figure 6.2. It is obvious that $\eta = 1$ leads to a deterministic linear movement, while a lower value disturbs the movement direction. In either case, the global optimum may jump among peaks.

In our experimental studies, we use most of the parameters as suggested in the original work [Bra99]: solution space $[0,100]^d$, 10 peaks, $s_l = 0.6$,

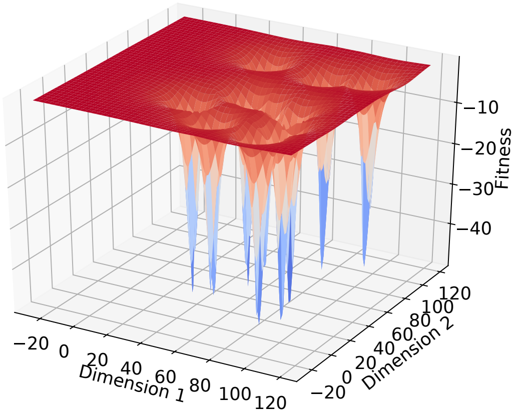


Figure 6.1: Static MPB fitness landscape as minimization problem

$s_h = 7$, $s_w = 0.1$, initial height 50, initial width 0.1, random initial position. Correlation factor η differs among the experiments. We multiply the fitness function (Equation (6.1)) by -1 because we consider minimization problems.

6.1.2 CEC Competition Benchmark

For the Congress on Evolutionary Computation (CEC), benchmark problem sets have been introduced as challenge for example in 2009, 2012, and 2014 [Li+08; LYP11; Li+13]. The latter contains a rotation peak function and composition of basic functions.

The dynamic rotation peak benchmark generator (DRPBG) is based on the concept of peaks like MPB. The fitness function in change period c is

$$f(\mathbf{x}, c) = \max_{i=1,\ldots,\#\text{peaks}} \frac{H_c^i}{1 + W_c^i \sqrt{\frac{1}{d}\|\mathbf{x} - \mathbf{X}_c^i\|_2^2}}, \tag{6.7}$$

with symbols as defined for MPB. In DRPBG, ten different types exist to change the peaks' heights, width, and position, for example with random, chaotic, recurrent, and noisy changes.

The other CEC 2014 benchmark is the dynamic composition benchmark generator (DCBG). It allows for composing a weighted selection of the basic functions Sphere, Rastrigin, Weierstrass, Griewank, and Ackley to one fitness function, see Appendix A for some of these functions. The dynamics for moving these functions are the same as for DRPBG.

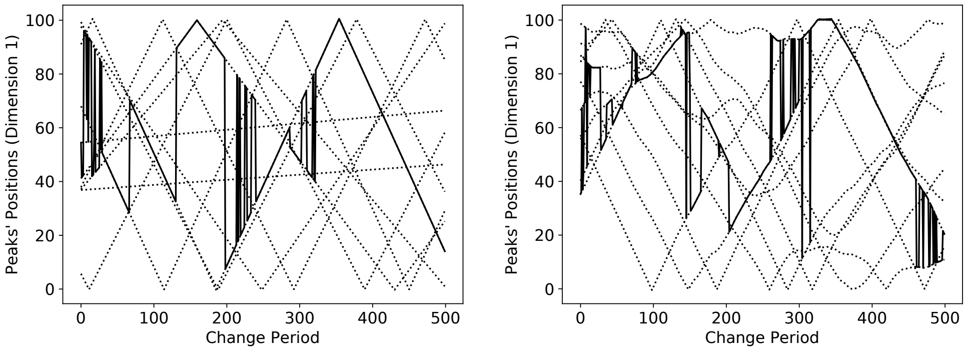


Figure 6.2: Movement of the peaks' positions (······) and the position of the global optimum (——) in dimension 1. Left: $\eta = 1$. Right: $\eta = 0.95$

6.1.3 Free Peaks Benchmark

A framework for constructing continuous optimization problems was introduced by [Li+18]. It is called free peaks benchmark (FPB) and can generate global, multi-modal, multi-objective, dynamic, and constrained optimization problems. In FPB, the solution space is divided into a defined number of non-overlapping partitions which each are equipped with one of eight base functions. The fitness of an individual is computed with the base function of the partition it is located in.

Among the dynamic optimization part, the free peaks benchmark offers equations to change an optimum's location, the shape of its neighborhood, the size of the optimum's basin of attraction, the height of the basin, and number of optima. Similar to MPB, the peaks' movement within their respective sub-space is linear with adjustable noise. To modify the landscape around a peak, the corresponding base function can be replaced by another one. By relocating the partition boundaries, a peak's basin of attraction can be enlarged or reduced. Also the basin's height is randomly changed like in MPB. Since the base functions are unimodal, the number of peaks can dynamically be changed by varying the numbers of partitions.

6.1.4 Dynamic Sine Benchmark

In [MK19] we proposed the dynamic sine benchmark (DSB) to address the problem that many benchmarks only allow simple movements of their optima. For example, the previously discussed and frequently applied MPB benchmark only provides noisy linear movement, as well as the FPB bench-

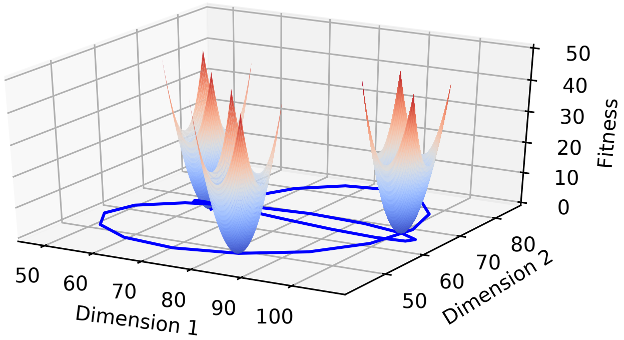


Figure 6.3: Optimum position of Sphere function in a two-dimensional solution space at three points in time

mark does. Although the CEC competition test sets provide more advanced dynamics, they are still rather simple. For example the recurrent dynamics is defined by a single sine function.

Our DSB problems are quantifiable in their dynamics and are based on moving classic stationary fitness functions, e.g., Sphere or Rosenbrock, with parameterized trigonometric functions, see Figure 6.3. A related benchmark, that has found application in time series prediction but so far not in dynamic nature-inspired optimization, is the multiple superimposed oscillators (MSO) benchmark [WGS05; Ste07; KB12; ORB19]. It consists of two or more additive sine functions to examine the performance of prediction methods on signals with large periodicity [RI10]. In contrast to this, DSB has multiplicative sine functions, which might lead to more complex dynamics, and offers quantifiable complexity

After the initial publication [MK19], we simplified the benchmark generator, solved some bugs and improved several details. This updated version is explained in the following.

Overall Approach

The underlying idea is that the optimum positions are generated by sampling trigonometric functions with a certain step size, see left part in Figure 6.4. In each dimension w of the solution space the optimum movement follows a separate trigonometric function

$$\zeta_w(c) = \tau + \alpha \cdot \prod_{i=1}^{\rho} \left(\iota_i \cdot \sin\left(\beta_i \cdot \kappa \cdot (c-1) + \gamma_i \right) \right) \tag{6.8}$$

which is composed by multiplying ρ sine functions, called component functions, with random amplitude ι_i, frequency β_i, phase shift γ_i, vertical shift τ, and overall scaling α. We call ζ_w composite function as well. In change period c the optimum is located at the sampled position

$$\mathbf{o}_c = [\zeta_1(c), \zeta_2(c), \dots, \zeta_d(c)] \,, \tag{6.9}$$

which is visualized in the right part of Figure 6.4. To generate optimum positions for P change periods, each function ζ_w is evaluated for $c \in [1, \dots, P]$. In ζ_w, step size κ determines the distance between sampling points. This can be illustrated with an alternative formulation where the optimum position is written as

$$\mathbf{o}_c = \left[\zeta'_1(\kappa \cdot (c-1)), \zeta'_2(\kappa \cdot (c-1)), \dots, \zeta'_d(\kappa \cdot (c-1))\right] , \tag{6.10}$$

where the sine functions that ζ'_w consists of are written in the basic form with x as point in time at which ζ'_w is sampled:

$$\zeta'_w(x) = \tau + \alpha \cdot \prod_{i=1}^{\rho} \left(\iota_i \;\cdot \sin\left(\beta_i \cdot x + \gamma_i\right)\right) \tag{6.11}$$

From Equation (6.10) it can easily be obtained that the distance between successive points at that ζ'_w is sampled equals step size κ in each dimension:

$$\kappa \cdot ((c+1)-1) - (\kappa \cdot (c-1)) = \kappa \cdot c - \kappa \cdot c + \kappa = \kappa$$

In Equation (6.8), $c-1$ is used instead of c to begin the sampling at point 0 because c is defined with $c \geq 1$, see Equation (2.2). Step size κ requires a careful choice to cover all important parts of function ζ_w. Therefore, in the updated benchmark we set it automatically as described later.

DSB can be combined with any stationary fitness function f_s. While shape and level of fitness function f_s remain unchanged, function ζ_w defines the location of the landscape in the solution space. It determines the position of the global optimum as the fitness landscape's anchor, see Figure 6.3. If f_s has multiple global optima, one of them has to be chosen as anchor. The fitness for individual $\mathbf{x}$ in change period c is

$$f(\mathbf{x}, c) = f_s(\mathbf{x} - (\mathbf{o}_c - \mathbf{o}_{f_s})) \tag{6.12}$$

where $\mathbf{o}_{f_s}$ denotes the global optimum of the unmoved function f_s.

In order to quantify the benchmark's complexity, we introduce the con-

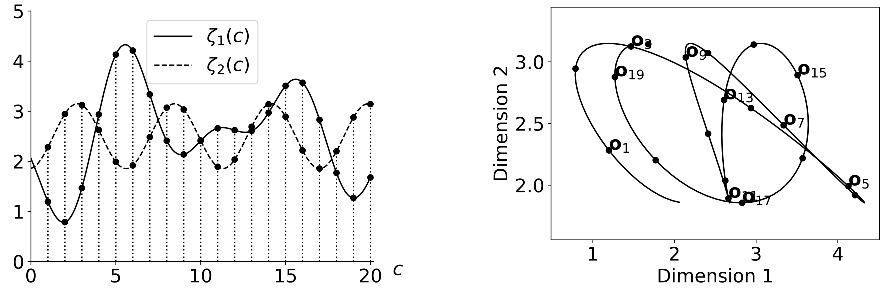


Figure 6.4: Left: sampling of ζ_w at 20 points in time. Right: corresponding optimum positions $\mathbf{o}_c$ in the solution space

cepts curviness and velocity. Curviness $C \in \mathbb{N}_{>0}$ specifies the number of extremes ζ_w has within 100 samples at successive equidistant points. Thus, curviness determines how many changes in the optimum's movement direction take place on the first 100 optimum positions. Since trigonometric functions are periodic, the overall number of extremes can be deduced based on the curviness. By this means, curviness indicates the difficulty for a prediction model to track the optimum. Velocity $V \in \mathbb{R}_{>0}$ is the median distance between successive optimum positions and represents the problem difficulty for the optimization algorithm. In case the velocity is much higher than the mutation strength, an ES without prediction might need many more generations to find the new optimum position. To generate a DSB instance with specified velocity and curviness, scaling factor α and frequencies β_i have to be chosen appropriately. The generation process and the mathematical relations are described in the next paragraphs.

In a d-dimensional DSB instance, for dimensions $w \in [1, \ldots, d]$ the functions ζ_w are generated with the same curviness and velocity in order to ensure that the complexity of DSB instances with different dimensionality but equal curviness and velocity only depends on the number of dimensions. Figure 6.5 visualizes DSB instances with different parameterizations for velocity and curviness. Apparently, curviness influences the shape of ζ_w, while velocity influences the vertical scaling. For example, the range of DSB instances with $V = 1.0$ is twice as large as the range of instances with $V = 0.5$.

Algorithmic Implementation

The DSB generator provides parameters that have to be set to construct a DSB instance with desired properties. They are listed in the following.

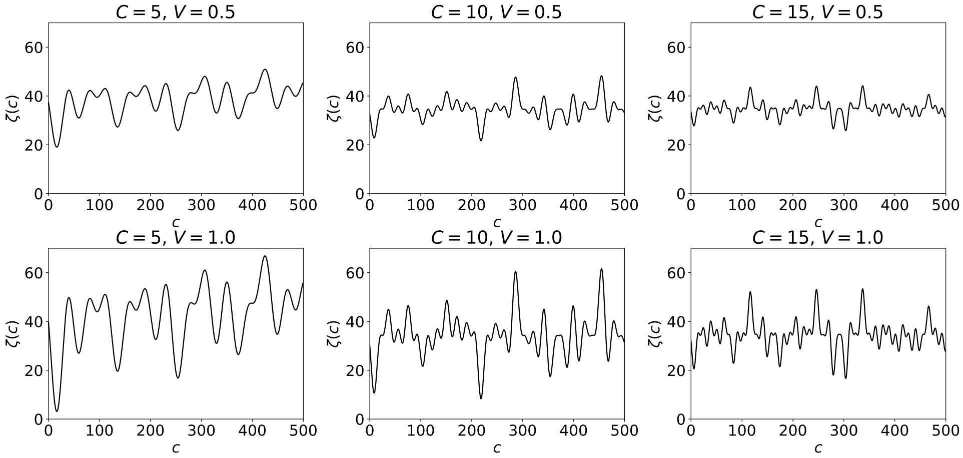


Figure 6.5: 500 optimum positions of 1-dimensional DSB instances ($\rho_{\max} = 4$) with varying curviness and velocity. First row: $V = 0.5$, second row: $V = 1.0$. From left to right: $C \in \{5, 10, 15\}$

$d \in \mathbb{N}_{>0}$	dimensionality
$b^l, b^u \in \mathbb{R}$	lower and upper bounds for range of ζ_w; correspond to bounds of solution space
$P \in \mathbb{N}_{>0}$	number of samples; equal to number of change periods P and optimum positions
$C \in \mathbb{N}_{>0} \wedge C < 100$	curviness
$V \in \mathbb{R}_{>0}$	velocity
$\rho_{\max} \in \mathbb{N}_{>0}$	maximum number of functions that are composed to ζ_w

Furthermore, the seed of the random number generator can be set to allow for reproducibility.

In our definition, curviness counts the extremes of ζ_w per 100 samples. The minimum curviness realizable is one, since the generator can only count discrete numbers of extremes on a given set of samples. Theoretically, it might be interesting to examine benchmarks with lower curviness values, e.g., $C = 0.6$. To construct such an instance, the number of samples which curviness refers to, must be increased in the generator's implementation. For the given example, $100 \cdot 5 = 500$ samples would be appropriate since $0.6 \cdot 5 = 3$ would result in a discrete number of extremes. However, if the number of reference samples is changed, it should be mentioned in the description of the experiments to enable comparability with other studies.

The benchmark generation process is described in the following; pseudocode is provided in Algorithm 3. Based on the user-defined specification, parameters are set and upper bounds $\iota_{\max}$ and $\beta_{\max}$ for parameters ι and β of the functions ζ_w are determined (Lines 4–7). Then, functions ζ_w are preliminarily parameterized (Lines 9–12). The chosen parameter values are corrected after all functions have been initialized in order to comply with the specification (Lines 14–16.) Justification for the upper bounds and explanation of the parameter correction is given in the next paragraph.

Algorithm 3 DSB generator

```
 1: given $d, b^l, b^u, \rho_{\max}, P, C, V$
 2: for $w = 1, \dots, d$ do
 3:     # set constants and upper bounds for parameters
 4:     $\rho \sim \mathcal{U}\left(1, \rho_{\max}\right)$
 5:     $\iota_{\max} = \left\lfloor \sqrt[\rho]{\frac{b^u - b^l}{2}} \right\rfloor$
 6:     $\kappa = 1$
 7:     $\beta_{\max} = \frac{1}{2 \cdot \kappa}$
 8:     # parameterize function $\zeta_w$ randomly
 9:     for $i = 1, \dots, \rho$ do
10:         $\iota_i \sim \mathcal{U}\left(-\iota_{\max}, \iota_{\max}\right) \wedge \iota_i \neq 0$
11:         $\beta_i \sim \mathcal{U}\left(-\beta_{\max}, \beta_{\max}\right) \wedge \beta_i \neq 0$
12:         $\gamma_i \sim \mathcal{U}\left(0, \frac{2\pi}{\beta_i}\right)$
13:     # adapt parameterization
14:     $\beta_i \leftarrow$ correct curviness
15:     $\alpha \leftarrow \frac{V}{V_{\zeta_w}}$                    # $V_{\zeta_w}$: velocity realized by $\zeta_w$
16:     $\tau \leftarrow$ correct range
```

Mathematical Justification

Roughly following the order of operations in Algorithm 3, we describe how the upper bounds for parameters are determined and how the generator fulfills the specifications regarding curviness, velocity and range.

Number of Component Functions The maximum number $\rho_{\max}$ of component functions forming ζ_w can be chosen arbitrarily. However, our studies

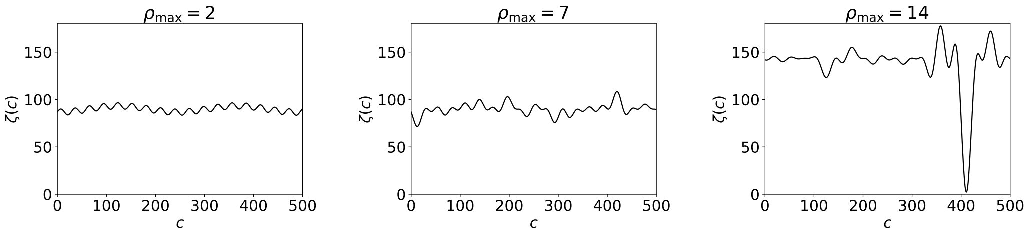


Figure 6.6: 500 optimum positions of 1-dimensional DSB instances ($C = 7, V = 0.5$) with varying $\rho_{\max}$. From left to right: $\rho_{\max} \in [2, 7, 14]$

showed that $\rho_{\max} < C$ is sufficient. Otherwise, interference becomes likely so that ζ_w exhibits extreme peaks or flat regions, see Figure 6.6 for $\rho_{\max} = 14$, which might be unusual in real-world applications. In our applications, $\rho_{\max} = 4$ was a reasonable choice independent of C. In each function ζ_w, we choose ρ randomly (Line 4) to allow for more diverse function shapes.

Amplitudes The upper bound $\iota_{\max}$ for amplitudes ι_i of component functions (Line 5) depends on range[1] $[b^l, b^u]$ allowed for composite function ζ_w. In order to meet it, the amplitude of ζ_w must be less or equal to $\frac{b^u - b^l}{2}$ since the maximum range of a trigonometric function is twice its amplitude. The amplitude of a function consisting of ρ multiplied trigonometric functions is the product of the component amplitudes ι_i which leads to the condition $\prod_{i=1}^{\rho} \iota_i \leq \frac{b^u - b^l}{2}$. Applying the ρ-th root results in the upper bound for amplitudes ι_i. For simplification, the result is rounded to an integer value:

$$\iota_{\max} = \left\lfloor \sqrt[\rho]{\frac{b^u - b^l}{2}} \right\rfloor \tag{6.13}$$

Step Size As described earlier, the optimum positions are sampled from functions ζ_w. A proper choice for step size κ is necessary to sample all important parts of the function ζ_w. The Nyquist-Shannon sampling theorem [Sha49] defines the relation $\kappa < \frac{1}{2 \cdot \beta_i}$. We choose $\kappa = 1$ (Line 6) as fixed step size and parameterize the functions ζ_w with frequencies compatible with κ, see next paragraph. Any other value would be possible for κ as well, the possible specifications, e.g., for curviness, would be the same.

[1] Without loss of generality, we depict the benchmark generator with equal bounds b^l and b^u for all ζ_w, while different bounds are applicable as well.

Frequency (Curviness) Frequency and curviness are directly related. Relying on the largest possible frequency $\beta_{\max}$ that follows from the Nyquist-Shannon theorem (Line 7), the maximum curviness $C_{\max}$ that ζ_w theoretically can realize is

$$C_{\max} = \sum_{i=1}^{\rho} |50 \cdot \beta_{\max}|. \tag{6.14}$$

Value $C_{\max}$ is the sum of the largest possible curviness values that the ρ component functions can achieve. The maximum curviness of a component function is deduced as follows. Any sine function consists of 2 extremes within each period. If the function is sampled with the upper bound for the step size according to the Nyquist-Shannon theorem, one period is covered by four sampling points:

$$\kappa < \frac{1}{2 \cdot \beta_i} = \frac{1}{2 \cdot \frac{2\pi}{L}} = \frac{L}{4\pi} \tag{6.15}$$

with period length L as multiple of π and the known relationship for trigonometric functions $\beta_i = \frac{2\pi}{L}$. Thus, not more than two extremes can lie within four sampling points, which is visualized in Figure 6.7. Our generator specifies the curviness in relation to 100 samples. Hence, any ideally sampled function has $\frac{100}{4}$ periods per 100 samples and $\frac{100}{4} \cdot 2 = 50$ extremes within 100 samples. Therefore, $C = 50$ is the largest realizable curviness for this and any other function. From this follows that sine functions with $\beta_i \leq \beta_{\max}$ provide $50 \cdot \beta_i$ extremes per 100 samples. Summing up the component curviness values results in the theoretical curviness of ζ_w:

$$C_{\text{theo}} = \sum_{i=1}^{\rho} |50 \cdot \beta_i| \tag{6.16}$$

This leads to the upper bound $C_{\max}$ in Equation (6.14) if every β_i equals $\beta_{\max}$. In practice, C_{theo} is not necessarily equal to the curviness C_{ζ_w} realized by ζ_w due to two reasons. First, the component functions might not be ideally sampled because of the phase shift. Second, we compute curviness C_{ζ_w} empirically. It is determined by computing the number of sign changes of the differences $\mathbf{o}_c - \mathbf{o}_{c-1}$ for all $c \in \{1, \ldots, P\}$. Therefore peaks located at a border cannot be detected, for example the extreme at point 4π in Figure 6.7.

In the implementation, we restrict C to be less than 100, since 100 is the

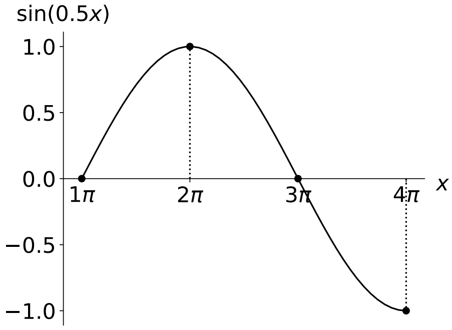


Figure 6.7: Ideal sampling of a period of an arbitrary sine function. Black dots mark sampling points

number of samples the definition of C refers to, and it is not possible to have more extremes than samples. However, we allow $C > C_{\max}$, even if it is only realizable by violating the Nyquist-Shannon condition for β_i, since in some applications the resulting benchmark might be useful although the functions are not ideally sampled.

During the generation process, first all frequencies are chosen from interval $[-\beta_{\max}, \beta_{\max}]$ regardless of the realized curviness, see Line 11 in Algorithm 3. After the component functions' parameterization was completed by setting the phase shift, the frequencies are adapted so that ζ_w complies with the specified curviness (Line 14). This works as follows.

All frequencies that do not become too large by this operation are multiplied with factor $\frac{C}{C_{\zeta_w}}$. In case the specified curviness can only be realized by violating the bounds of all β_i, all frequencies are multiplied with $\frac{C}{C_{\zeta_w}}$. The correction is done iteratively since multiple steps might be necessary to achieve C. If the correction was not successful for a specified number of iterations, e.g., when the frequencies are increased and decreased by the same factor, only a random subset of frequencies is updated. By this means, it becomes more probable to find a proper parameterization. After the correction procedure, some β_i might exceed $\beta_{\max}$. Thus, the sampling would not be mathematically exact, but for dynamic optimization as application context it should be sufficient.

Phase Shift Phase shift γ_i is chosen from $[0, \frac{2\pi}{\beta_i})$, where $\frac{2\pi}{\beta_i}$ is the period length of the corresponding component function (Line 12). This interval is sufficient because sine functions are periodic so that larger or smaller phase shifts can be expressed by values within this interval as well. It is reasonable to parameterize the component functions with different phase shifts in order to lower the risk of interference and to increase the number of possible functions.

Scaling (Velocity) The overall scaling factor α influences the velocity. It is set to $\frac{V}{V_{\zeta_w}}$ to achieve the specified velocity (Line 15). Here, V_{ζ_w} is the velocity realized by ζ_w and is equal to the median distance of the differences $\mathbf{o}_c - \mathbf{o}_{c-1}$ for all $c \in \{1, \ldots, P\}$. The scaling factor is computed after the curviness has been corrected, since changes in the frequencies might change the velocity again.

Vertical Shift (Range) The term τ shifts ζ_w vertically such that the range covered by ζ_w matches the specified interval $[b^l, b^u]$ (Line 16). It might be the case, that no such shift can be found as the range of ζ_w is too wide. In this case ζ_w cannot be corrected because adapting the scaling α would in turn violate the velocity specification. The only solution would be that the user selects a larger range.

6.2 Quality Measures

To evaluate the efficiency of dynamic optimization problems, various quality measures have been introduced in the past. Surveys are given in [BRAK13; NYB12; CGP11]. The authors of [NYB12] propose to differentiate between optimality-based measures, which evaluate the ability to track the optimum, and behavior-based measures taking into account properties like convergence speed or diversity within the population. In real-world scenarios, the application context indicates which characteristics are of interest and hence which measures to use. In the following, we present four measures we employ for evaluating our prediction methods in the next chapters.

6.2.1 Best of Generation

Best of generation ($\overline{\text{BOG}}$) is a known fitness-based performance measure averaging the fitness of the best found solutions over a defined number of g generations and r runs of repeated experiments. It evaluates the algorithm's behavior over the whole run. The best value of $\overline{\text{BOG}}$ is problem-dependent.

$$\overline{\text{BOG}} = \frac{1}{g \cdot r} \sum_{t=1}^{g} \sum_{j=1}^{r} \text{BOG}_{tj}, \tag{6.17}$$

where BOG_{tj} is the best fitness found during generation t in run j. Low values are desirable in case of minimization problems.

6.2.2 Best Error before Change

Best error before change (BEBC) [TM99] is a frequently used fitness-based measure considering the fitness of the best solution found during a change period. It averages the fitness differences between the best found solution $\mathbf{x}_c^*$ and the optimum $\mathbf{o}_c$ over all change periods c:

$$\mathrm{BEBC} = \frac{1}{P} \sum_{c=1}^{P} |f(\mathbf{x}_c^*, c) - f(\mathbf{o}_c, c)| \tag{6.18}$$

In contrast to $\overline{\mathrm{BOG}}$, BEBC takes into account only the lowest error per change period. By this means, BEBC ignores fitness peaks during early generations of a change period. An optimal algorithm finds the global optimum for each change period leading to a BEBC of zero. BEBC has no upper bound for the worst case. It can only be applied if the optimum $\mathbf{o}_c$ is known.

6.2.3 Absolute Recovery Rate

Absolute recovery rate (ARR) is a behavior-based measure for the speed an optimization algorithm starts converging to the global optimum before the next change happens [Ngu11; NY12]. It is computed as follows:

$$\mathrm{ARR} = \frac{1}{P} \sum_{c=1}^{P} \frac{\sum_{t=1}^{g(c)} \left(f(\mathbf{x}_{ct}^*, c) - f(\mathbf{x}_{c1}^*, c)\right)}{g(c) \cdot \left(f(\mathbf{o}_c, c) - f(\mathbf{x}_{c1}^*, c)\right)} \tag{6.19}$$

Variable $g(c)$ denotes the number of generations during change period c, and P is the overall number of change periods. Variable $f(\mathbf{o}_c, c)$ signifies the global optimum fitness for change period c, while $f(\mathbf{x}_{ct}^*, c)$ represents the best fitness the algorithm has found in change period c until generation t. Thus, $f(\mathbf{x}_{c1}^*, c)$ is the best fitness the algorithm has found in the first generation of change period c. Following [Ngu11; NY12], the best value achievable for ARR is one, the worst is zero. However, in Appendix B we show that also negative values can occur. Therefore, in our implementation we use absolute differences to circumvent negative values.

6.2.4 Relative Convergence Speed

In [MK18a], we introduced the measure relative convergence speed (RCS) because in dynamic optimization it is not only important to quickly *leave* poor solutions, considered by ARR, but also to *find* the global optimum with as few generations as possible. RCS measures how fast the ES approaches the global optimum relatively to the other algorithms included in the comparison:

$$\mathrm{RCS} = \frac{1}{P} \sum_{c=1}^{P} \frac{\sum_{t=1}^{g(c)} t \cdot |f(\mathbf{x}_{ct}^{*}, c) - f(\mathbf{o}_c, c)|}{\sum_{t=1}^{g(c)} t \cdot |f(\mathbf{x}_{c_{\text{worst}}}^{*}, c) - f(\mathbf{o}_c, c)|} \tag{6.20}$$

The semantic of the symbols is the same as defined for ARR. The worst fitness any algorithm has achieved during change period c is denoted by $f(\mathbf{x}_{c_{\text{worst}}}^{*}, c)$. RCS ranges from zero as best to one as worst value.

ARR is designed so that it measures how fast an algorithm can get *far away* from its *first solution*, i. e., the best fitness found in the first generation after a change. Due to this, algorithms that start with a very poor solution may achieve a larger ARR than those starting with a better solution. We illustrate this with an example. Assuming the best fitness values found by algorithm A during one change period are $[9, 6, 6, 0]$, those of algorithm B are $[4, 3, 3, 0]$, and zero is the global optimum fitness. Then, A has ARR $= 0.42$ and B has ARR $= 0.36$. Thus, A outperforms B although B in every but the last generation achieves a better fitness than A.

To overcome this unexpected behavior, we proposed RCS that takes into account how fast an algorithm finds solutions *near to* the *global optimum*. There exist four main differences to ARR. 1) As motivated above, we employ the difference $f(\mathbf{x}_{ct}^{*}, c) - f(\mathbf{o}_c, c)$ instead of the difference $f(\mathbf{x}_{ct}^{*}, c) - f(\mathbf{x}_{c1}^{*}, c)$ to measure how near a solution is to the optimum. 2) In order to scale this difference into the interval $[0, 1]$ it is divided by the difference between the global optimum fitness and the worst fitness of any algorithm. By this means, RCS measures the convergence speed of the algorithms relative to the worst fitness. If instead the difference $f(\mathbf{x}_{c1}^{*}, c) - f(\mathbf{o}_c, c)$ would be incorporated like in ARR, the RCS would exhibit behavior similar to ARR in cases like the example presented above, but with inverted range. 3) The difference terms are weighted with the generation number in order to reward early good fitness values and penalize high fitness values at the end of change periods. 4) In contrast to ARR, the best value for RCS is zero and the worst is one. In Appendix B, we scrutinize ARR and RCS deeper and show further disadvantages of ARR as division by zero and negative ARR values.

Part II

Prediction-Based Dynamic Optimization

7 Prediction for Evolution Strategies

Several studies attest that prediction approaches in ES are successful. So far, mostly statistical prediction methods have found application in dynamic ES. Motivated by the success of NNs in domains like object recognition and time-series prediction, we investigate NN-based prediction. The approach is published in [MK18a], and was the first combining NN-based prediction and dynamic optimization. Because of the no-free-lunch theorems [WM97] and results of competitions and comparing surveys, we are aware that neither NN-based nor statistical prediction methods are best for all problems. In this work, we show that it is possible to integrate RNN prediction into ES and that there exist cases in which RNN prediction better supports optimization compared to autoregression. We also show that hybrid prediction can further improve the optimizer's performance.

7.1 Related Work

As described in Chapter 5, time series prediction methods with different properties exist. First, we review which prediction methods have been employed in dynamic evolutionary optimization showing a lack of sophisticated methods. Then, we list literature that compares prediction methods.

7.1.1 Time Series Prediction Methods in Dynamic ES

Literature on prediction approaches in ES investigates multi-objective optimization more frequently than single-objective optimization. We include literature from both domains, although the focus of this thesis lies on single-objective optimization. In ES, the prediction methods autoregression, Kalman filter, and the persistence model are used most often.

The earliest work employing autoregression is [HW06]. Based on solutions discovered for previous change periods, the optimum position is predicted and inserted into the population. In multi-objective scenarios, autoregression is applied to characteristic points of the Pareto set [Li+19b; ZJZ14] or Pareto front [Zou+17], respectively. The first contribution with a linear

Kalman filter is [RAD08]. The authors employ the predicted optimum position to influence the optimizer's operators, penalize individuals far from the predicted optimum with a poor fitness, and place some individuals around the prediction. Another strategy with ES and linear Kalman filter was suggested by [MTV16]. An approach that is different to the ones mentioned before is pursued in [Jia+18a; Hu+19]. Instead of predicting positions as in the other approaches, they classify randomly generated points whether they will be part of the next Pareto set. The points with a positive label are included into the population. A support vector machine [SC08b] is the classifier.

The most prediction-based publications do not use an external prediction model, like Kalman filter or autoregression, but rely on calculating differences between two observations. For example, [Zho+07] predicts for each individual its new position by adding the difference to its predecessor. This resembles the persistence model (Chapter 5), since the movement direction is assumed to be the same. Liu *et al.* [Liu+14] additionally add some noise to the new position. Most difference-based strategies predict the movement of the center of the Pareto set by computing the deviation of the present to the previous center point [WJL15; Li+19a; Li+19b; Zho+18; Zou+17; Cao+17; Ahr+19; Pen+15; Rua+17]. The predicted movement is added to each individual leading to the new population. The work of [Cao+18] does not only take into account the movement but also the acceleration of the center point. An alternative approach is to cluster the population after each change into various groups having one center point each and to predict the movement of the center point [Jin+16; Ron+19; RGZ16; LD19]. The new individuals are generated by adding the difference between the current and the previous cluster center it belongs to.

Other approaches make use of the orthogonal design strategy to generate better populations [Zen+06; Liu+15; Wan+16; Rua+17], apply Newton's law for prediction [FS17], or employ a quadratic model [Bos05]. Furthermore, works exist on predictive gradient strategies [KGT10] and transfer learning to build a prediction model [Jia+18b].

This literature overview shows that prediction in evolutionary dynamic optimization so far mostly relies on the persistence model or, more seldom, is conducted with traditional time-series models (autoregression, Kalman filter). In the next paragraph, we give an overview on comparative studies for time-series prediction methods.

7.1.2 Comparative Studies on Time Series Prediction Methods

Research on time series prediction was dominated by statistical approaches for a long time, see Chapter 5. In recent years, the advent of new, neural network-based time series prediction methods became popular by applications like Google's Neural Machine Translation [Wu+16] or Apple's speech recognition system Siri [Lev16]. Often it is discussed whether simpler traditional or more sophisticated machine learning models perform better. Performance comparison of time series prediction methods is done both descriptive, e.g., by [YM00; BTB12], and empirically in survey studies, e.g., by [MSA18a; PSB19; Zha01]. Also international competitions on time series prediction tasks, e.g., Makridakis or Kaggle competitions, give insights into the performance of a large variety of methods [FO07]. While survey articles frequently compare either machine learning or statistical methods, e.g., [BKK18; Ahm+10; Geo+17], competitions are advantageous in that there often are contributions with prediction methods from diverse research fields.

One of the most famous competitions on time series prediction are the Makridakis competitions (M-Competitions). They were founded in 1982 and have been conducted overall four times. They are based on univariate real-world time-series data from various domains like finance or industry. Mainly pushed from the statistics community, in the first three competitions almost no contributions with machine learning approaches participated [CHN11]. The results of the three competitions are very similar [MH00; MSA18b]. Often simple methods perform at least equally well as complex ones, and, in the M4-competition, no evidence is found that machine learning methods generally have better results than traditional methods. Furthermore, hybrid methods relying on different statistical or machine learning methods often are better than raw ones.

Also the NN3-competition [CHN11], that is an extension of the M3-competition with more submissions from machine learning, shows that machine learning methods perform equally to statistical ones and that hybrid methods are good. The authors of [CHN11] emphasize, that studies on data sets that are different to those employed in the M-competitions are required to show whether neural networks in time series prediction are powerful or not.

The study of [MSA18a] extends the work of [Ahm+10] by comparing not only machine learning methods but also statistical ones. The authors find worse performance of machine learning models than of statistical methods but state that machine learning models could have advantages if the time series would exhibit nonlinear components. Moreover, they claim that more diverse data sets would be beneficial which led to the M4-competition. In the

M4-competition, a hybrid method of machine learning and statistical methods, called “Exponential Smoothing-Recurrent Neural Networks” [Smy19], clearly outperforms all other methods. The author claims, that for NN-based approaches it is difficult to learn in classical prediction tasks because of limited data and inappropriate preprocessing. Zhang [Zha12] summarizes that neither raw machine learning nor pure statistical models perform best on all tasks and propose that hybrid models should be the method of choice. In general, much work has been spent on hybridizations of statistical and machine learning methods, e.g., [OJOMN19; KB11], as well as on ensembles of machine learning methods [KLT13; Bis07; HTF09; KBC14].

The study [Wol19] examined a TCN, an LSTM, a Kalman filter and a persistence model on multivariate time series where each dimension follows a simple second-order differential equation. In this study, the Kalman filter and the persistence model outperformed LSTM and TCN. However, comparing these methods on the dynamic sine benchmark led to better results of the neural network approaches emphasizing the influence of the chosen benchmark problem on the results, see Appendix C.

Furthermore, a recent Kaggle competition on time series is the “Web Traffic Time Series Forecasting” [Kag17] where the number of views on certain Wikipedia pages had to be forecast. Also there the contributions are very diverse. The best solution is an RNN-based method, while other well performing approaches are, e.g., polynomial autoregression (5th place), a convolutional NN (6th place), or Kalman filter (8th place).

It can be concluded, that simpler models often turn out to be superior to more complex ones as well as machine learning models do not necessarily outperform statistical ones. There is evidence, that much attention has to be spent on preprocessing matching the requirements of the machine learning model otherwise the performance probably is poor [Ahm+10]. Moreover, the chosen benchmark series have strong influence on the performance, and hybrid approaches seem to be promising.

7.2 Recurrent Neural Network Prediction for Evolution Strategies

The fact that so far mostly the persistence model or statistical methods are employed for prediction in dynamic optimization motivates us to examine whether and how more enhanced methods, i.e., NN-based prediction, can be utilized. Despite the disappointing performance of machine learning methods in some studies and competitions (Paragraph 7.1.2), we assume that dynamic optimization can profit from capabilities of machine learning methods if the model is reasonably instantiated, combined with suitable preprocessing and applied to an appropriate problem. This is justified by the success of machine learning methods in specialized cases, see Paragraph 7.1.2. We explain our approach employing a raw RNN architecture as prediction model for dynamic ES in the following.

The original ES, described in Chapter 3, needs only few modifications to integrate a prediction model. Algorithm 4 shows pseudocode for our prediction-based ES. At the beginning of every generation it has to check whether the objective function has changed (Line 7). We realize it by re-evaluating half of the population and additional random solutions somewhere in the solution space. This strategy does not necessarily recognize all changes. There are different approaches trying to attain high detection quality [ATE17; Ric09].

If a change has been detected, the best solution $\mathbf{x}^*_{c-1}$ found during the last change period is appended to archive $\mathbf{A}$ (Line 10). Thus, at the beginning of change period c archive $\mathbf{A} \in \mathbb{R}^{c-1 \times d}$ consists of a time series of $c-1$ found solutions $[\mathbf{x}^*_1, \mathbf{x}^*_2, \ldots, \mathbf{x}^*_{c-1}]$ that are used to train the prediction model Ψ and to estimate the next optimum position $\hat{\mathbf{o}}_c$ (Line 11):

$$\hat{\mathbf{o}}_c \leftarrow \Psi\left(\mathbf{x}^*_1, \mathbf{x}^*_2, \ldots, \mathbf{x}^*_{c-1}\right) \tag{7.1}$$

Since prediction models comprise parameters that have to be fit, at the first change periods prediction cannot be conducted. After a reasonable number of change periods, i.e., when enough training data has been collected in $\mathbf{A}$, the prediction model is trained the first time. In further course of optimization, the model is trained at the beginning of every new change period with the new found optimum. This re-training is necessary because the data the model was trained on the first time, does not necessarily capture all properties of the dynamics. Also it would be possible that the kind of dynamics varies over time.

After change detection and prediction, the population is reinitialized by using the prediction (Line 12). This can be done by introducing solutions (immigrants) that are initialized near the predicted optimum. In our setting, one third of the immigrants is randomly placed in the solution space. The other immigrants are the predicted optimum and neighbors $\mathbf{x}_c$ obtained by adding noise to the prediction: $\mathbf{x}_c \sim z \cdot \mathcal{N}\left(\hat{\mathbf{o}}_c, \mathbf{I}\right)$ with $z \in \{0.01, 0.1, 1, 10\}$. Afterwards, the ES is continued by producing offspring individuals (Line 13), and selecting the best ones for the next generation (Line 14). After the selection step, the population has its original size again.

Furthermore, the mutation strength of the ES is controlled by Rechenberg's 1/5th success rule [BS02; Rec73] (Line 6). In static optimization this is a well-known approach to influence the amount of exploration and exploitation depending on the success of search, see Paragraph 3.3. After a change is detected, we set the mutation strength to its initial value so that the ES can better explore the solution space (Line 9). This is a means to overcome the general problem of convergence in dynamic optimization. Without modification, the mutation strength would be too small to find a new optimum because in the previous change period the ES might already have left the exploration phase and started to converge to the optimum, leading to a decreasing mutation strength.

Algorithm 4 Dynamic (μ+λ)-ES with prediction

```
 1: $\mathbf{P} \leftarrow$ initialize_population()
 2: $s \leftarrow$ initialize_mutation_strength()
 3: $\mathbf{A} \leftarrow [\,]$                                          # found solutions
 4: $c \leftarrow 1$                                             # change period counter
 5: for generations do
 6:     $s \leftarrow$ adapt_mutation_strength($s$)        # Rechenberg's 1/5th rule
 7:     if change_detected() then
 8:         $c \leftarrow c + 1$
 9:         $s \leftarrow$ reset_mutation_strength()
10:         $\mathbf{A} \leftarrow \mathbf{A}$.append($\mathbf{x}^*_{c-1}$)                     # store best solution
11:         $\hat{\mathbf{o}}_c \leftarrow$ train_and_predict_optimum($\mathbf{A}$)
12:         $\mathbf{P} \leftarrow$ reinitialize_population($\mathbf{P}, \hat{\mathbf{o}}_c$)
13:     $\mathbf{P}' \leftarrow$ create_$\lambda$_offspring_individuals($\mathbf{P}, s$)   # recomb.&mutation
14:     $\mathbf{P} \leftarrow$ select_best_$\mu$_individuals($\mathbf{P}, \mathbf{P}'$)
```

7.3 Experimental Setup

We compare three different algorithms: an ES without prediction, denoted by `noPred` in the following, an ES with an autoregressive prediction model (`autoPred`), and an ES with our new RNN-based prediction (`rnnPred`). For all three approaches we employ the ES framework presented in Algorithm 4.

The ES framework employs a (50+100)-ES and is run for $g = 6000$ generations. We use dominant recombination ($\psi = 2$), plus selection, and Gaussian mutation. The mutation strength is initially set to $s = 1.0$, and adapted every five mutations with Rechenberg's 1/5th success rule ($\xi = 0.5$). The fitness function changes every $\upsilon = 20$ generations. After a change occurred, μ random immigrants are incorporated into the population. Prediction is conducted from the 1000th generation, i.e., when 50 training items have been collected. As preprocessing, training data is scaled into interval $[-1, 1]$. The prediction models use window size $w_s = 7$, i.e., seven previous optima are used to estimate the next one, and are re-trained after each change with the new found optimum. The RNN is trained for 30 epochs. These parameter settings are the same for all experiments unless otherwise stated.

We employ a simple RNN approach instead of an LSTM because in our application we are faced with very few training data (dozens till hundreds) whereas LSTMs require due to their larger complexity more training data to fit the parameters well. Because of this, we employ a rather small network with only two layers. The first one comprises 20 neurons each with the tanh activation function. The second layer is the output layer and therefore consists of d neurons with linear activation function. It returns the d-dimensional estimation of the optimum for the next change period.

The experimental study comprises four groups of experiments that investigate different characteristics of optimization problems. Two further groups deal with improving prediction quality by optimizing the predictor's parameters or employing a hybrid a prediction approach, respectively. Each group comprises several experiments that are repeated 20 times. In the following, we motivate the groups of experiments and present their parameter settings.

7.3.1 Group SRR

The experiments of this group concern dynamic variants of the Sphere, Rastrigin and Rosenbrock (SRR) benchmark functions. The goal is to examine whether linear or sine movement has different effects on `noPred`, `rnnPred` and `autoPred`, and whether the dimensionality of the functions plays a role. Therefore, we perform one experiment for each combination of dimensionality $d \in \{2, 5, 10, 20, 50, 100\}$ and the movement types linear and sine.

The Sphere function (Figure A.1) is a simple convex function with one global and no local optimum. Rosenbrock (Figure A.2) is a nonconvex function with one global optimum and, in high dimensions, one local optimum [Kra16]. Also Rastrigin is nonconvex (Figure A.3); its shape resembles the Sphere function but has many local optima. We make these functions dynamic by moving the whole function landscape linearly or sine-like within the solution space. The linear movement is realized by adding an offset, that is increased every change, to all dimensions. The sine-like movement is applied to the first two dimensions. The overall fitness level is not modified, only the position of the function changes within the space. Sphere, Rosenbrock and Rastrigin are minimization problems, the optimum fitness for every change is zero.

7.3.2 Group MPB-Random

In this experiments, we investigate the case where the optimum movement does not follow a pattern but is random. We apply the original moving peaks benchmark (MPB) with completely random movement [Bra99], see Paragraph 6.1.1, and instantiate it for dimension $d \in \{2, 5, 10, 20, 50, 100\}$.

7.3.3 Group MPB-Noisy

Again the MPB benchmark is applied, but now we restrict the movement of the peaks' positions to be linear with noise. For this, we instantiate shift vector $\mathbf{S}_c^i$ with all ones and add Gaussian noise. In the experiments, we vary the strength of noise among the levels 0.0, 0.1, 1.0, and 10 in order to examine to which extent randomization does affect the prediction. We use MPB instances with dimensionality $d \in \{2, 20\}$.

7.3.4 Group Ros-Length

In the forth group of experiments, we modify the change period length, i. e., the number of generations lying between two changes. We apply the Rosenbrock function with sine-like optimum movement and investigate different settings for dimensionality $d \in \{2, 20, 50\}$ and period length $\upsilon \in \{5, 10, 20, 40, 60\}$. The longer the change periods, the more generations are required to collect enough training data. Therefore, we conduct optimization for $g = \upsilon \cdot 300$ generations so that in each experiment the final number of training data is 300.

7.3.5 Group SRR-Neurons

The goal of this group is to test whether the usage of more neurons for the RNN of our `rnnPred` approach has an effect. We employ the Sphere, Rosenbrock and Rastrigin functions with sine-like optimum movement and $d \in \{2, 5, 10, 20, 50, 100\}$. In general, the higher the dimensionality, the more neurons are required to cover characteristics inherent in the data. We set the number of neurons depending on the problem dimensionality with $\#\text{neurons} = \lceil 1.3 \cdot d \rceil$ as a compromise between network capability and computational effort.

7.3.6 Group SRR-Hybrid

Since in comparative studies often hybrid approaches outperformed raw ones as described in Paragraph 7.1.2, in this group of experiments we compare the prediction methods employed in the previous experiments to an ES with hybrid prediction (`hybPred`). In `hybPred`, both an autoregressive model and an RNN are trained in parallel and both predict the optimum independently. The predicted optimum that achieves a better fitness is used in the ES. The intention behind this is that the optimum dynamics might exhibit parts that are better explainable with an autoregressive model and others that can easier be fitted with an RNN. A hybrid predictor could combine the advantages of both prediction methods. The algorithms are evaluated on the Sphere function with linear and sine movement parameterized as in group SRR. The RNN is instantiated with #neurons $= \lceil 1.3 \cdot d \rceil$ and each training phase is conducted for 20 epochs.

7.4 Experimental Results

In the following paragraphs, we examine the different groups of experiments separately. The result tables, e. g., Table 7.1, depict in column "Mov." the movement type, while "Dim." is the dimensionality. The best value within a row is highlighted bold.

7.4.1 Group SRR

All experiments of this group demonstrate the complexity of problems with a high dimensionality. The algorithms show deteriorating $\overline{\text{BOG}}$ and BEBC values the larger the dimensionality gets (Tables 7.1, 7.2, 7.5, 7.6, 7.7, and 7.8). The best possible $\overline{\text{BOG}}$ is zero in all except MPB-based experiments. In addition, the differences among the algorithms decrease with increasing dimensionality. These findings are caused by the curse of dimensionality: the more dimensions, the more difficult becomes the exploration of the search space.

Furthermore, common to all experiments is that ES with prediction outperforms `noPred` regarding $\overline{\text{BOG}}$ and BEBC. Another observation regarding $\overline{\text{BOG}}$ and BEBC results for the Sphere function is that `autoPred` always outperforms `rnnPred` when the optimum is moved linearly. But for the more complex Rosenbrock and Rastrigin functions, `rnnPred` often is better than `autoPred` in case of a 5-, 10- or 20-dimensional linearly moved problem. When the optimum is moved sine-like, `rnnPred` has always better $\overline{\text{BOG}}$ and

Table 7.1: SRR (Sphere): $\overline{\text{BOG}}$

Mov.	Dim.	`noPred`	`rnnPred`	`autoPred`
linear	2	2.14E-1	9.31E-2	**7.67E-2**
	5	3.12E+0	2.04E+0	**1.71E+0**
	10	1.52E+1	1.07E+1	**9.35E+0**
	20	6.20E+1	4.45E+1	**3.99E+1**
	50	3.85E+2	2.70E+2	**2.26E+2**
	100	1.64E+3	1.19E+3	**7.91E+2**
sine	2	1.80E+0	**4.43E-1**	6.73E-1
	5	9.75E+0	**4.93E+0**	6.01E+0
	10	2.53E+1	**1.64E+1**	2.12E+1
	20	7.25E+1	**5.57E+1**	6.44E+1
	50	3.46E+2	**2.94E+2**	3.08E+2
	100	1.00E+3	9.41E+2	**9.29E+2**

Table 7.2: SRR (Sphere): BEBC

Mov.	Dim.	`noPred`	`rnnPred`	`autoPred`
linear	2	7.31E-7	3.88E-7	**2.93E-7**
	5	9.23E-3	6.22E-3	**5.09E-3**
	10	4.99E-1	3.56E-1	**3.10E-1**
	20	7.51E+0	5.25E+0	**4.66E+0**
	50	1.25E+2	8.59E+1	**6.90E+1**
	100	8.44E+2	6.18E+2	**3.77E+2**
sine	2	6.60E-6	**2.26E-6**	3.34E-6
	5	2.80E-2	**1.68E-2**	1.88E-2
	10	7.23E-1	**4.77E-1**	6.17E-1
	20	8.62E+0	**6.31E+0**	7.50E+0
	50	1.26E+2	**1.01E+2**	1.08E+2
	100	5.34E+2	4.89E+2	**4.83E+2**

BEBC results than `autoPred`, except for 100-dimensional functions. These observations emphasize the strengths of RNNs for optimization problems where the optimum movement follows a periodic pattern.

For the Sphere function, we include the results for the convergence measures ARR (Table 7.3) and RCS (Table 7.4); the convergence values for the other functions can be found in Appendix D.1. Regarding ARR, `noPred` always is best except for three cases with sine-like movement, whereas for the RCS measure `noPred` is outperformed by `rnnPred` and `autoPred`: `autoPred` is always best in case of linear movement and `noPred` converges best in sine-like dynamics, except for 100 dimensions. These contrary results are due to the measures' design (see discussion in Paragraph 6.2.4). Since it has no information about the next optimum, `noPred` probably starts after a change with a worse population than `rnnPred` and `autoPred`. Therefore, `noPred` may have more potential to quickly achieve large fitness improvements than `rnnPred` and `autoPred` leading to its good ARR values. The results for RCS are consistent to the observations made for $\overline{\text{BOG}}$ and BEBC: `rnnPred` converges faster for sine-like and `autoPred` for linearly moved functions.

7.4.2 Group MPB-Random

The MPB-Random experiments are characterized by randomly changing peak heights, widths and positions. Thus, the optimal $\overline{\text{BOG}}$ varies among the settings and therefore is listed in the result tables. The results show that `noPred` mostly outperforms the prediction-based ES in the 2-, 5-, and 10-dimensional case regarding all four quality measures (Tables 7.9, 7.10, 7.11, and 7.12). For the other dimensions, `rnnPred` or `autoPred` are best but the differences between the three algorithms are very marginal. The prediction-based approaches perform poor in these settings, since the changes are ran-

Table 7.3: SRR (Sphere): ARR

Mov.	Dim.	`noPred`	`rnnPred`	`autoPred`
linear	2	**9.17E-1**	8.88E-1	1.94E-1
	5	**8.79E-1**	8.47E-1	2.98E-1
	10	**7.89E-1**	7.62E-1	2.51E-1
	20	**6.45E-1**	6.34E-1	1.90E-1
	50	**4.70E-1**	4.33E-1	1.33E-1
	100	**3.51E-1**	2.88E-1	1.21E-1
sine	2	**9.15E-1**	8.90E-1	8.98E-1
	5	**8.86E-1**	8.49E-1	8.66E-1
	10	**8.08E-1**	7.87E-1	8.03E-1
	20	6.67E-1	**6.85E-1**	6.80E-1
	50	4.39E-1	**4.75E-1**	4.64E-1
	100	2.89E-1	**3.15E-1**	3.11E-1

Table 7.4: SRR (Sphere): RCS

Mov.	Dim.	`noPred`	`rnnPred`	`autoPred`
linear	2	1.36E-2	5.33E-3	**2.49E-3**
	5	3.18E-2	1.25E-2	**5.32E-3**
	10	8.58E-2	3.19E-2	**1.44E-2**
	20	2.01E-1	6.87E-2	**3.33E-2**
	50	3.90E-1	1.59E-1	**6.57E-2**
	100	5.40E-1	3.09E-1	**9.04E-2**
sine	2	1.23E-2	**5.33E-3**	7.69E-3
	5	2.75E-2	**1.60E-2**	1.88E-2
	10	7.16E-2	**4.48E-2**	5.89E-2
	20	1.79E-1	**1.12E-1**	1.45E-1
	50	4.19E-1	**2.90E-1**	3.24E-1
	100	5.98E-1	4.98E-1	**4.76E-1**

Table 7.5: SRR (Rosenbr.): $\overline{\text{BOG}}$

Mov.	Dim.	`noPred`	`rnnPred`	`autoPred`
linear	2	4.14E+0	2.34E+0	**2.21E+0**
	5	3.49E+4	3.46E+4	**3.45E+4**
	10	9.24E+5	**9.22E+5**	9.28E+5
	20	7.40E+6	7.39E+6	**7.39E+6**
	50	5.89E+7	5.83E+7	**5.83E+7**
	100	2.36E+8	2.26E+8	**2.20E+8**
sine	2	9.05E+1	**1.66E+1**	1.93E+1
	5	5.54E+4	**2.92E+4**	3.25E+4
	10	1.00E+6	**9.63E+5**	9.79E+5
	20	8.40E+6	**8.34E+6**	8.36E+6
	50	6.13E+7	**6.12E+7**	6.12E+7
	100	2.28E+8	2.28E+8	**2.28E+8**

Table 7.6: SRR (Rosenbr.): BEBC

Mov.	Dim.	`noPred`	`rnnPred`	`autoPred`
linear	2	1.60E-2	1.22E-2	**6.63E-3**
	5	7.84E+0	7.20E+0	**6.16E+0**
	10	9.75E+2	**9.54E+2**	3.24E+3
	20	1.11E+5	**1.11E+5**	1.11E+5
	50	7.60E+6	7.42E+6	**7.41E+6**
	100	7.57E+7	7.03E+7	**6.61E+7**
sine	2	8.57E-2	**3.11E-2**	4.25E-2
	5	1.67E+1	**1.13E+1**	1.26E+1
	10	1.04E+3	**9.70E+2**	9.97E+2
	20	1.44E+5	**1.43E+5**	1.43E+5
	50	8.07E+6	**8.06E+6**	8.06E+6
	100	6.67E+7	6.66E+7	**6.66E+7**

Table 7.7: SRR (Rastrigin): $\overline{\text{BOG}}$

Mov.	Dim.	`noPred`	`rnnPred`	`autoPred`
linear	2	5.92E+0	2.89E+0	**1.31E+0**
	5	5.46E+1	**3.81E+1**	4.74E+1
	10	2.17E+2	**1.63E+2**	1.84E+2
	20	7.23E+2	**6.09E+2**	6.25E+2
	50	3.54E+3	3.45E+3	**3.44E+3**
	100	1.20E+4	1.01E+4	**7.24E+3**
sine	2	5.29E+0	**2.47E+0**	3.46E+0
	5	4.35E+1	**2.34E+1**	3.60E+1
	10	1.91E+2	**1.07E+2**	1.66E+2
	20	8.88E+2	**6.00E+2**	6.28E+2
	50	4.87E+3	**2.69E+3**	2.71E+3
	100	1.69E+4	1.30E+4	**8.64E+3**

Table 7.8: SRR (Rastrigin): BEBC

Mov.	Dim.	`noPred`	`rnnPred`	`autoPred`
linear	2	4.42E+0	2.06E+0	**9.35E-1**
	5	4.05E+1	**2.72E+1**	3.55E+1
	10	1.62E+2	**1.17E+2**	1.39E+2
	20	5.49E+2	**4.64E+2**	4.75E+2
	50	2.85E+3	2.86E+3	**2.79E+3**
	100	1.02E+4	8.64E+3	**6.00E+3**
sine	2	4.80E-1	**2.18E-1**	4.12E-1
	5	2.15E+1	**1.08E+1**	1.74E+1
	10	1.38E+2	**7.54E+1**	1.24E+2
	20	8.16E+2	**5.37E+2**	5.55E+2
	50	4.67E+3	**2.52E+3**	2.54E+3
	100	1.65E+4	1.26E+4	**8.25E+3**

Table 7.9: MPB-Random: $\overline{\text{BOG}}$

Dim.	`noPred`	`rnnPred`	`autoPred`	Expected
2	**-1.44E+2**	-1.23E+2	-1.26E+2	-1.70E+2
5	**-1.07E+2**	-1.02E+2	-1.02E+2	-2.05E+2
10	-5.60E+1	**-5.65E+1**	-5.63E+1	-1.34E+2
20	-5.19E+1	**-5.21E+1**	-5.21E+1	-1.61E+2
50	-5.83E+1	-5.85E+1	**-5.86E+1**	-2.05E+2
100	-6.31E+1	-6.33E+1	**-6.35E+1**	-1.50E+2

Table 7.10: MPB-Random: BEBC

Dim.	`noPred`	`rnnPred`	`autoPred`
2	**2.66E+1**	4.73E+1	4.44E+1
5	**9.25E+1**	9.85E+1	9.85E+1
10	7.15E+1	**7.15E+1**	7.15E+1
20	1.01E+2	**1.01E+2**	1.01E+2
50	1.36E+2	1.36E+2	**1.36E+2**
100	7.70E+1	7.68E+1	**7.67E+1**

Table 7.11: MPB-Random: ARR

Dim.	`noPred`	`rnnPred`	`autoPred`
2	**8.46E-1**	7.23E-1	6.54E-1
5	**3.54E-1**	3.08E-1	3.17E-1
10	**2.66E-1**	2.63E-1	2.65E-1
20	1.43E-1	1.45E-1	**1.45E-1**
50	9.70E-2	9.83E-2	**9.86E-2**
100	1.19E-1	1.21E-1	**1.22E-1**

Table 7.12: MPB-Random: RCS

Dim.	`noPred`	`rnnPred`	`autoPred`
2	**5.10E-1**	6.94E-1	6.67E-1
5	**6.17E-1**	6.44E-1	6.45E-1
10	6.92E-1	**6.90E-1**	6.90E-1
20	8.16E-1	**8.14E-1**	8.14E-1
50	8.64E-1	8.63E-1	**8.63E-1**
100	8.30E-1	8.27E-1	**8.25E-1**

dom and prediction especially makes sense when the changes follow a pattern that can be learned.

An explanation for the better performance of `noPred` for small dimensions may be that after a change the prediction-based approaches initialize a part of their population near to the predicted optimum. Due to random changes, this prediction is probably very poor. Instead, `noPred` initializes that population part randomly somewhere in the solution space. Therefore, it has more diversity within the population and can more easily find a better solution. A reason why this difference between the approaches cannot be observed in higher dimensions may be the curse of dimensionality. Many more, but not only few more, random individuals are required to adequately explore the solution space.

7.4.3 Group MPB-Noisy

Considering BEBC (Table 7.14), it can be observed that `noPred` outperforms the prediction-based approaches (`autoPred` is better than `rnnPred`) in the 2-dimensional setting, but for 20 dimensions both predictors are better than `noPred`. The results for $\overline{\text{BOG}}$ (Table 7.13) are similar. These findings are akin to the observations made in the MPB-Random experiments and can be justified by the same explanation.

In addition, it can be observed that in the MPB-Noisy experiments the results differ much more among the approaches than in the MPB-Random settings. Especially regarding BEBC the performance differences between

Table 7.13: MPB-Noisy: $\overline{\text{BOG}}$

Dim.	Noise	`noPred`	`rnnPred`	`autoPred`	Expected
2	0.0	-1.38E+2	-1.38E+2	**-1.41E+2**	-1.77E+2
	0.1	**-1.43E+2**	-1.39E+2	-1.42E+2	
	1.0	**-1.58E+2**	-1.57E+2	-1.56E+2	
	10.0	-1.25E+2	**-1.27E+2**	-1.27E+2	
20	0.0	-1.64E+1	-4.22E+1	**-9.20E+1**	-2.25E+2
	0.1	-2.01E+1	-4.89E+1	**-9.55E+1**	
	1.0	-1.08E+1	-1.30E+1	**-1.46E+1**	
	10.0	-6.17E-1	-6.66E-1	**-6.76E-1**	

Table 7.14: MPB-Noisy: BEBC

Dim.	Noise	`noPred`	`rnnPred`	`autoPred`
2	0.0	**3.45E+1**	3.72E+1	3.57E+1
	0.1	**2.88E+1**	3.65E+1	3.44E+1
	1.0	**1.10E+1**	1.58E+1	1.70E+1
	10.0	**2.13E+1**	2.46E+1	2.54E+1
20	0.0	1.83E+2	1.50E+2	**1.31E+2**
	0.1	1.73E+2	1.34E+2	**1.11E+2**
	1.0	1.95E+2	1.90E+2	**1.87E+2**
	10.0	2.23E+2	2.23E+2	**2.23E+2**

`noPred` and the prediction approaches are larger than in MPB-Random. A reason for this is that in MPB-Noisy the peaks basically follow a linear movement. Though the relationship is disturbed to some extent by noise, the predictors seem to be able to learn the pattern.

Furthermore, it is interesting that in the 2-dimensional case $\overline{\text{BOG}}$ and BEBC become better with increasing noise but then become worse for noise 10. In the 20-dimensional, case both measures improve only for very small noise (0.1) and deteriorate for higher noise. The measures of convergence speed show no clear relationship to the strength of noise and are included in Appendix D.1.

7.4.4 Group Ros-Length

In general, both $\overline{\text{BOG}}$ (Table 7.15) and BEBC (Table 7.16) are improving with increasing period length υ. This is because the ES has more generations per change period to explore the solution space and may converge to a better solution. Another observation is that `rnnPred` is outperformed only one time. This shows that `rnnPred` is able to cope with fast and slow changes.

It is interesting that the prediction-based approaches in these settings always outperform `noPred` although, especially in case of very frequent changes, the training data for the prediction models probably are rather poor, since the ES has very few generations to converge. The convergence results are consistent with these findings; they are listed in Appendix D.1.

7.4.5 Group SRR-Neurons

Tables 7.17 and 7.18 present the results for these experiments; remaining results are available in Appendix D.1. The bold highlighting signifies results that are better than the corresponding ones in group SRR (Paragraph 7.4.1). In the settings with 2, 5 and 10 dimensions, `rnnPred` achieves worse results compared to the corresponding experiments of the SRR settings, since now

Table 7.15: Ros-Length: $\overline{\text{BOG}}$

Dim.	v	`noPred`	`rnnPred`	`autoPred`
2	5	1.35E+2	**5.28E+1**	5.39E+1
	10	9.81E+1	**2.30E+1**	2.63E+1
	20	9.50E+1	**1.74E+1**	1.94E+1
	40	5.45E+1	**1.14E+1**	1.26E+1
	60	3.52E+1	**6.61E+0**	7.61E+0
20	5	3.56E+7	**3.43E+7**	3.46E+7
	10	1.64E+7	**1.62E+7**	1.62E+7
	20	8.23E+6	**8.17E+6**	8.19E+6
	40	3.82E+6	**3.79E+6**	3.80E+6
	60	2.68E+6	**2.66E+6**	2.66E+6
50	5	2.68E+8	2.65E+8	**2.65E+8**
	10	1.23E+8	**1.23E+8**	1.23E+8
	20	6.03E+7	**6.02E+7**	6.02E+7
	40	3.01E+7	**3.01E+7**	3.01E+7
	60	1.99E+7	**1.98E+7**	1.98E+7

Table 7.16: Ros-Length: BEBC

Dim.	v	`noPred`	`rnnPred`	`autoPred`
2	5	1.44E+1	**7.47E+0**	7.67E+0
	10	4.17E+0	**1.13E+0**	1.44E+0
	20	5.44E-2	**3.14E-2**	3.87E-2
	40	1.99E-3	**1.24E-3**	1.63E-3
	60	1.29E-4	**1.13E-4**	5.92E-4
20	5	1.86E+7	**1.79E+7**	1.80E+7
	10	2.93E+6	**2.89E+6**	2.91E+6
	20	1.46E+5	**1.45E+5**	1.45E+5
	40	6.50E+2	**6.25E+2**	6.41E+2
	60	6.58E+1	**6.08E+1**	6.26E+1
50	5	1.89E+8	1.87E+8	**1.87E+8**
	10	5.30E+7	**5.28E+7**	5.28E+7
	20	7.92E+6	**7.91E+6**	7.91E+6
	40	2.62E+5	**2.61E+5**	2.61E+5
	60	1.11E+4	**1.11E+4**	1.11E+4

Table 7.17: SRR-Neurons: $\overline{\text{BOG}}$

Dim.	Sphere	Rosenbrock	Rastrigin
2	6.61E-1	2.56E+1	3.31E+0
5	5.33E+0	4.39E+4	2.43E+1
10	1.67E+1	1.16E+6	1.15E+2
20	**5.49E+1**	**8.02E+6**	**5.71E+2**
50	**2.90E+2**	**5.79E+7**	**2.29E+3**
100	**9.11E+2**	**2.16E+8**	**1.12E+4**

Table 7.18: SRR-Neurons: BEBC

Dim.	Sphere	Rosenbrock	Rastrigin
2	3.41E-6	6.11E-2	3.09E-1
5	1.78E-2	1.31E+1	1.11E+1
10	5.72E-1	1.08E+3	8.21E+1
20	**6.77E+0**	**1.37E+5**	**5.08E+2**
50	**9.85E+1**	**7.93E+6**	**2.12E+3**
100	**4.74E+2**	**6.05E+7**	**1.08E+4**

fewer neurons are employed. However, in case of 20, 50 and 100 dimensions, more neurons are used than in the SRR-Neurons settings. Here, `rnnPred` becomes better and even outperforms `autoPred`. The reason for this is that RNNs with a more complex architecture, i. e., many neurons and layers, can often better learn difficult relationships than RNNs with simpler architectures. These experiments show that there is some potential to optimize our prediction model in order to improve the results.

7.4.6 Group SRR-Hybrid

Here, we do not list the absolute results, but whether pairwise Mann-Whitney U tests are significant (significance level $\alpha = 0.05$). Tables 7.19 and 7.20 contain '▼' ('△'), if the algorithm of the respective row achieves a significantly lower (larger) value than the algorithm in the respective column regarding the specific metric on the given benchmark. The symbol '$-$' signifies a non-significant test result. For each algorithm, the column contains the metrics $\overline{\text{BOG}}$, BEBC, ARR, and RCS (from left to right).

First, it has to be mentioned that the experiments for `noPred`, `rnnPred`,

Table 7.19: SRR-Hybrid: linear

Alg.	Dim.	autoPred	rnnPred	hybPred
noPred	2	△ △ △ △	△ △ △ △	△ △ △ △
	5	△ △ △ △	△ △ △ △	△ △ △ △
	10	△ △ △ △	△ △ △ △	△ △ △ △
	20	△ △ △ △	△ △ △ △	△ △ △ △
	50	△ △ – △	△ △ △ △	△ △ △ △
	100	△ △ △ △	△ △ △ △	△ △ △ △
autoPred	2		▼ ▼ – ▼	△ △ – △
	5		▼ ▼ ▼ ▼	△ △ – △
	10		▼ ▼ ▼ ▼	△ △ – △
	20		▼ ▼ ▼ ▼	△ △ △ △
	50		△ △ △ △	△ △ △ △
	100		▼ ▼ ▼ ▼	△ △ △ △
rnnPred	2			△ △ – △
	5			△ △ △ △
	10			△ △ △ △
	20			△ △ △ △
	50			– – – –
	100			△ △ △ △

Table 7.20: SRR-Hybrid: sine

Alg.	Dim.	autoPred	rnnPred	hybPred
noPred	2	△ △ ▼ △	△ △ △ △	△ △ △ △
	5	△ △ △ △	△ △ △ △	△ △ △ △
	10	△ △ △ △	△ △ △ △	△ △ △ △
	20	△ △ △ △	△ △ △ △	△ △ △ △
	50	△ △ △ △	△ △ △ △	△ △ △ △
	100	△ △ △ △	△ △ △ △	△ △ △ △
autoPred	2		△ △ △ △	△ △ △ △
	5		△ △ △ △	△ △ △ △
	10		△ – △ △	△ △ △ △
	20		▼ ▼ ▼ ▼	△ △ △ △
	50		△ △ △ △	△ △ △ △
	100		▼ ▼ ▼ ▼	△ △ △ △
rnnPred	2			△ – △ –
	5			△ △ △ –
	10			△ △ △ △
	20			△ △ △ △
	50			– – – –
	100			△ △ △ △

and autoPred reproduce the results of group SRR (Tables 7.1–7.4), except for marginal differences. As in group SRR, noPred outperforms the other approaches regarding ARR, autoPred is better than rnnPred on linear movement, except for ARR, and rnnPred outperforms autoPred on the sine dynamics up to medium-sized dimensionality. Minor deviations in the results are possible, since we train the RNN of the hybrid predictor only with 20 epochs instead of 30 to reduce the runtime, and it is instantiated with #neurons = $\lceil 1.3 \cdot d \rceil$ because in group SRR-Neurons this turned out to be a good choice.

Considering the results for hybPred, its success is obvious. It is never outperformed by any other approach neither for linear nor sine dynamics. This emphasizes the advantage of hybrid prediction that consists of prediction methods which are able to learn different types of time series dynamics.

7.5 Summary

In this chapter, we presented a new approach to predict the position of the next optimum of a dynamically changing fitness function. We designed a recurrent neural network to serve as prediction model and trained it during the optimization with new optima found by the ES. In the experimental study, our approach showed competitiveness to an existing prediction approach (autoPred) that employs an autoregressive model.

Based on the results of the experimental study we can conclude that our new approach (rnnPred) works well for objective functions where the optimum movement follows a recurrent pattern but also yields good results for other movement patterns. It could be observed that rnnPred is able to cope with different change frequencies as well as for different problem dimensionalities. In case of noisy optimum movements it did not perform best, but nevertheless yields results competitive to those of `autoPred`. Furthermore, it turned out that our prediction model can be improved by incorporating more neurons into the architecture. Then it would be able to yield even better results for high-dimensional problems. Further improvement of the prediction could be achieved by hybrid predictors utilizing the advantages of prediction methods suited to different types of dynamics. In general, prediction-based ES performed better than ES without predictor. Only in low-dimensional experiments with random or noisy optimum movement `noPred` was able to achieve better results.

8 Prediction for Particle Swarm Optimization

In order to enable particle swarm optimization (PSO) to cope with dynamic optimization problems, various strategies have been introduced, e. g., random restart, memory, and multi-swarm approaches. However, in contrast to ES, literature for PSO lacks approaches based on prediction. In this chapter, we propose three different PSO variants employing a prediction approach based on recurrent neural networks to adapt the swarm behavior after a change of the objective function. This contribution was published in [MK18b].

8.1 Related Work

Like ES, PSO faces the challenges diversity loss and premature convergence. Thus, after a fitness function change it is hard for the optimizer to escape the previously found optimum in favor of the new one probably located further away. Approaches allowing PSO to solve dynamic optimization problems are similar to those developed for ES (see Chapter 4). A broad overview of PSO in dynamic problems is given by [MLY17; Jor14; Bla07]. First, we give an example of a dynamic PSO that is equipped with some techniques frequently applied in dynamic optimization. Afterwards, we overview prediction-related literature for dynamic PSO.

8.1.1 Dynamic Particle Swarm Optimization

To apply basic PSO, introduced in Chapter 3, to dynamic optimization only few adaptations are required. Nevertheless, a wide range of more sophisticated PSO derivatives exists, like quantum particles [BB04] or heterogeneous PSO [Eng10], that try to overcome certain limitations. Algorithm 5 shows a simple dynamic PSO that is the performance reference for our PSO variants; `dynPSO` refers to it in the following. It requires mechanisms to detect changes of the fitness function (Line 6) and to adapt the swarm after a change so that it becomes more diverse in order to explore new regions of the solution space (Line 9). Changes can be detected by re-evaluating particles and comparing their current and previous fitness values. The diversity of the swarm can be increased by a partial restart (re-randomization) as proposed by Hu and Eberhart [HE02]. For example, the worst individuals could be replaced by new random ones so that probably new regions of the solution space are covered by the swarm.

An optional setting for PSO is an adaptive inertia weight ω in the velocity update (Equation (3.5)) that decreases with increasing iteration number (Line 5). Extensive work on how to set the inertia weight for static optimization problems exists, see e. g. [HEOB16a] for an overview. If an adaptive inertia weight is employed for dynamic optimization, it could be reset to its initial value after a change (Line 7).

After a fitness function change, the particles' best positions $\mathbf{x}^p$ should be reset, since the old ones could lead them into directions that are not promising anymore. A possible choice is to set $\mathbf{x}^p$ to the current position of the respective particle. Hence, the swarm's best position has to be recomputed as well (Line 10).

Algorithm 5 Dynamic PSO with partial restart

```
1:  $\mathbf{S} \leftarrow$ initialize_swarm()                    # random position&velocity
2:  $\omega$, $\theta_1$, $\theta_2 \leftarrow$ initialize_values()
3:  $\mathbf{A} \leftarrow [\,]$                                  # found solutions
4:  for iterations do
5:      $\omega \leftarrow$ adapt_inertia($\omega$)
6:      if change_detected() then
7:          $\omega \leftarrow$ reset_inertia()
8:          $\mathbf{A} \leftarrow \mathbf{A}$.append($\mathbf{x}^s$)        # store best solution
9:          $\mathbf{S} \leftarrow$ adapt_swarm($\mathbf{S}$)
10:         $\mathbf{x}^p, \mathbf{x}^s \leftarrow$ reset($\mathbf{x}^p, \mathbf{x}^s$)    # reset $\mathbf{x}^p$ for each particle
11:     $\mathbf{S} \leftarrow$ move_particles($\mathbf{S}$)          # following Eq. (3.5), (3.6)
12:     $\mathbf{x}^p, \mathbf{x}^s \leftarrow$ update($\mathbf{x}^p, \mathbf{x}^s$)     # update $\mathbf{x}^p$ for each particle
```

8.1.2 Prediction Approaches in Dynamic PSO

As far as we know, two publications exist for dynamic PSO with prediction. When we published the work presented in this chapter [MK18b], only the work by [BLY17] was present. It investigates constrained optimization problems and assumes that feasible regions in the solution space move linearly. The authors employ a linear prediction model to forecast the locations of the feasible regions. They track the movement of subpopulation centers, i.e., the individual with the best fitness in the respective subpopulation, and predict the new center positions by adding the difference to the last position as done in many publications for ES (see Paragraph 7.1.1). The authors of [BLY17] claim that the predicted points represent the feasible regions or are near to them, and employ them to build the new population. Besides, they introduce various other extensions like memory. The second work considering prediction for dynamic PSO is [Liu+18] and was published after our work. It is a multi-population approach for multi-objective optimization. The authors employ a population-based prediction mechanism (Paragraph 4.2) adopted from [ZJZ14], and predict the population center with autoregression.

In contrast to these works, we employ a more sophisticated prediction method (RNN) and propose PSO variants that extend the PSO movement functions. A further minor difference is that we only use one population while the related approaches are based on multiple populations. However, our PSO variants could easily be extended to a multi-population setting as well. After our work [MK18b] was published, we found the work [Li+14] proposing a prediction approach for static optimization. It applies a polynomial model to forecast the static optimum position during the optimization process and employs it as third attractor like we independently suggested in our work to handle the predicted dynamic optimum.

8.2 Prediction-Based Particle Swarm Optimization

We propose three different ways to integrate a prediction strategy into PSO for dynamic optimization. After a fitness function change, they predict the optimum based on the best solutions found for the previous change periods, like done for ES in Chapter 7. The PSO variants differ in how they use the prediction in guiding the optimizer.

8.2.1 Prediction as Particle (`pred2p`)

This approach integrates the predicted optimum position as separate particle into the swarm. This mechanism is similar to approaches for ES, where the predicted optimum is often employed to create individuals that are included into the population [NYB12]. It also resembles the related approach of [BLY17] that integrates some of the predicted subpopulation centers as particles into the population. Such a new particle is treated like the other ones. Its position and velocity are updated every iteration following Equations (3.5) and (3.6). If the predicted optimum has a promising fitness, the corresponding particle will become the swarm's best particle and lead the swarm in direction of the predicted global optimum. If the prediction is not promising, the prediction particle will probably not influence the movement of the swarm.

8.2.2 Prediction as Third Attractor (`pred3`)

The idea of this approach is to move the particles not only in direction of the swarm's and particle's best positions, but also to the location of the predicted optimum. In order to implement this, we extend the velocity update function (Equation (3.5)) by a further term:

$$\mathbf{v}_t = \omega \mathbf{v}_{t-1} + \theta_1 \mathbf{r}_1 \circ (\mathbf{x}^p - \mathbf{x}_{t-1}) + \theta_2 \mathbf{r}_2 \circ (\mathbf{x}^s - \mathbf{x}_{t-1}) + \theta_3 \mathbf{r}_3 \circ (\hat{\mathbf{o}}_c - \mathbf{x}_{t-1}) \quad (8.1)$$

Here, the predicted optimum $\hat{\mathbf{o}}_c \in \mathbb{R}^d$ of the current change period serves as third attractor. The influence of $\hat{\mathbf{o}}_c - \mathbf{x}_{t-1}$ is controlled by θ_3, while $\mathbf{r}_3$ is a vector of random values like $\mathbf{r}_1$ and $\mathbf{r}_2$. This approach differs from the first one (`pred2p`) because the particles are always moved in direction of the predicted optimum, independently of whether the prediction has good or poor fitness.

8.2.3 Prediction as Particle and Attractor (`pred3p`)

In our third approach we combine the first two ones. The predicted optimum is both a particle in the swarm and a third attractor within the velocity function (Equation (8.1)). This strategy is similar to the second one (`pred3`) in that the particles are moved in direction of the predicted optimum whether it has a good fitness or not. If the prediction has a good fitness, it may become the swarm's best particle resulting in two of three attractors moving the particles to the predicted optimum. In this case, the prediction influences the particles' movement stronger than in `pred2p` and `pred3`.

8.2.4 PSO Framework

The PSO that serves as basis for our prediction-based PSO variants is depicted in Algorithm 6. It applies a re-randomization strategy to increase swarm diversity after a change (Line 12). The worst 1/5th particles are replaced by random ones and, in case of `pred2p` and `pred3p`, an arbitrary one of them is replaced by the predicted optimum. Hu and Eberhart [HE02] examine in their work different re-randomization rates and report a rate of ten percent as reasonable value. However, on our benchmark problems a rate of 20 percent seems to be better.

Changes are detected by re-evaluating half of the swarm and some random points in the solution space (Line 7). In addition, we adapt the inertia weight at the beginning of each iteration with Rechenberg's 1/5th success rule (Line 6). If more than 1/5th of the particles have found in the previous iteration a solution that is better than their particle's best solution, the inertia weight is doubled, in case less than 1/5th have been successful, the inertia weight is halved. Although some studies report poor performance of dynamic parameter adaptation in PSO, see Paragraph 3.4, we apply it because it seemed to have positive influence regarding our benchmarks.

Overall, our PSO framework with prediction (Algorithm 6) is very similar to basic dynamic PSO without prediction (Algorithm 5). The only differences are that a prediction is made (Line 11), the prediction becomes a swarm particle in case of `pred2p` and `pred3p` (Line 12), the velocity movement function contains a third attractor for `pred3` and `pred3p` (Line 14), and in some experiments θ_3 is adapted besides the inertia weight for `pred3` and `pred3p` (Lines 6 and 9).

Algorithm 6 Dynamic PSO with partial restart and prediction

```
 1: $\mathbf{S} \leftarrow$ initialize_swarm()                      # random position&velocity
 2: $\omega, \theta_1, \theta_2, \theta_3, \leftarrow$ initialize_values()
 3: $\mathbf{A} \leftarrow [\,]$                                            # found solutions
 4: $c \leftarrow 1$                                             # change period counter
 5: for iterations do
 6:     $\omega, \theta_3 \leftarrow$ adapt_parameters($\omega, \theta_3$)
 7:     if change_detected() then
 8:         $c \leftarrow c + 1$
 9:         $\omega, \theta_3 \leftarrow$ reset_parameters()
10:         $\mathbf{A} \leftarrow \mathbf{A}$.append($\mathbf{x}^s$)                        # store best solution
11:         $\hat{\mathbf{o}}_c \leftarrow$ train_and_predict($\mathbf{A}$)
12:         $\mathbf{S} \leftarrow$ adapt_swarm($\mathbf{S}, \hat{\mathbf{o}}_c$)
13:         $\mathbf{x}^p, \mathbf{x}^s \leftarrow$ reset($\mathbf{x}^p, \mathbf{x}^s$)            # reset $\mathbf{x}^p$ for each particle
14:     $\mathbf{S} \leftarrow$ move_particles($\mathbf{S}$)               # following Eq. (3.5), (3.6)
15:     $\mathbf{x}^p, \mathbf{x}^s \leftarrow$ update($\mathbf{x}^p, \mathbf{x}^s$)         # update $\mathbf{x}^p$ for each particle
```

8.2.5 Convergent Parameter Settings

As shown in previous studies, e. g., [BE06; Pol09; HEOB16b], the performance of PSO heavily depends on its parameter values such that inappropriate values can lead to divergent behavior, see Paragraph 3.4. A particle converges if its position varies only within a specified radius around a point in the solution space which is not necessarily the optimum [CE14b]. As basis for the experimental study, we select values for w, θ_1, θ_2, and θ_3 that will theoretically lead to convergence.

Convergent Settings for `pred2p` For ordinary PSO with two attractors in the movement equation, different convergence proofs exist [CE15; CE14b; BE06; Pol09]. For example, [Pol09] proposes

$$\theta_1 + \theta_2 < \frac{24(\omega^2 - 1)}{5\omega - 7}, \quad \theta_1, \theta_2 > 0,\ \omega \in [-1, 1] \tag{8.2}$$

theoretically leading to convergent behavior. Which of the proposed convergence proofs should be employed is challenging to decide. Since each theoretical study is based on specific assumptions that do not hold in practice, e.g., that the swarm's and particle's best position do not change anymore (stagnation), empirical justification is required [CE15]. In an empirical analysis, [CE14a] show that Equation (8.2) is best among the examined variants.

For approach `pred2p`, we follow the common approach to assign θ_1 and θ_2 the same value. We set $\theta_1 = \theta_2 = 1.49618$ and $\omega = 0.7298$, since these values comply with Equation (8.2) and have often been applied in previous works. Parameter θ_3 is set to 0 as the predicted optimum should not serve as attractor.

Convergent Settings for `pred3` and `pred3p` For a so called fitness-distance-ratio PSO with a third attractor, [CE17] derived the relationship:

$$\theta_1 + \theta_2 + \theta_3 < \frac{18(1 - \omega^2)}{5 - 4\omega}, \quad |\omega| < 1,\ \theta_1 + \theta_2 + \theta_3 > 0 \tag{8.3}$$

We set $\omega = 0.7298$ like in the setting for `pred2p` and also apply equal values for θ_1, θ_2, and θ_3. Thus, we get $\theta_1 = \theta_2 = \theta_3 < 1.3477$ as maximum bound. Since in previous experiments larger parameter values seemed to yield better results, we set $\theta_1 = \theta_2 = \theta_3 = 1.0532$, which is an arbitrary value slightly below the maximum bound.

8.3 Experimental Setup

We empirically compare four different algorithms: the proposed three variants of PSO with prediction (`pred2p`, `pred3`, `pred3p`), and the standard PSO without prediction (`dynPSO`, Algorithm 5). We do not include the related approach [BLY17] (see Paragraph 8.1.2) into the experimental study because in a single-population setting the approach resembles our strategy `pred2p`. The only difference is the employed prediction method: their prediction is the persistence model, while we employ an RNN. In Chapter 7, we could show, that NNs can be superior to simpler models and vice versa. Therefore, we evaluate our strategies only against each other and the raw PSO.

We execute each algorithm with both its theoretically convergent parameter values, see Paragraph 8.2.5, and the settings of the other algorithms. In addition, we test further parameter settings so that we analyze five different settings overall; they are listed in the following. In all experiments, inertia weight ω is set to 0.7298, while θ_3 is always set to 0 for `dynPSO` and `pred2p`.

- $\theta_1 = \theta_2 = \theta_3 = 1.0532$
  This is a convergent setting for PSO variants with three attractors (`pred3` and `pred3p`). It is also convergent for `dynPSO` and `pred2p` following Equation (8.2).

- $\theta_1 = \theta_2 = \theta_3 = 1.49618$
  This is a convergent setting for PSO with two attractors (`dynPSO` and `pred2p`). It is not convergent for `pred3` and `pred3p` according to Equation (8.3).

- $\theta_1 = \theta_2 = \theta_3 = 2.0$
  With this setting, we examine whether larger parameter values lead to better results. It is convergent neither for the variants with two nor with three attractors.

- $\theta_1 = \theta_2 = 1.49618$, $\theta_3 = \theta_1 + \theta_2 = 2.99236$
  If the predicted optimum has a good fitness, it might be advantageous that it gets more influence than the swarm's and particle's best solutions. In order to analyze this, we employ a value for θ_3 that is the sum of θ_1 and θ_2. Only `pred3` and `pred3p` are executed with this setting, it is theoretically non-convergent.

- $\theta_1 = \theta_2 = 1.49618$, θ_3: adaptive
  In case of `pred3` and `pred3p` it may happen that the swarm's best

particle $\mathbf{x}^s$ and the predicted optimum particle $\hat{\mathbf{o}}_c$ are contradictory when $\hat{\mathbf{o}}_c$ is located in a completely different region than $\mathbf{x}^s$. Therefore, we examine whether an adaptive θ_3 shows better results. If the fitness of $\hat{\mathbf{o}}_c$ is better or only slightly (≤ 1.5 times) worse than the fitness of $\mathbf{x}^s$, θ_3 is doubled, otherwise it is halved. This setting is not convergent.

The experimental design is similar to that in Chapter 7. Each benchmark function undergoes a change every 20 iterations. In total, 6000 iterations are performed for each experiment and the experiments are repeated 20 times. The swarms of all PSO variants comprise 200 particles, employ the global neighborhood model, and adapt the inertia weight similar to Rechenberg's 1/5th success rule.

As prediction model we use a recurrent neural network parameterized according to Chapter 7. The window size is seven and the number of neurons depends with $\#\text{neurons} = \lceil 1.3 \cdot d \rceil$ on the problem dimensionality. The RNN is utilized after 50 training data have been collected, i.e., from the 1000th iteration, and is re-trained with 30 epochs at each change with the new training item. The training set is scaled into the interval $[-1, 1]$.

Our benchmark problems are the MPB variant MPB-Noisy which is explained in Chapter 7, and dynamic variants of the Sphere, Rosenbrock, and Rastrigin functions (together denoted by SRR). We employ MPB-Noisy in order to examine the influence of noise on prediction approaches, since prediction is known to perform best when data provides a learnable pattern. We transform the static functions Sphere, Rosenbrock, and Rastrigin into dynamic ones by moving them linearly or sine-like. The linear movement is implemented by adding an offset to all dimensions, the sine-like movement by applying a separate sine function with random amplitude and frequency to each dimension. We instantiate all benchmarks with dimensions $d \in \{2, 20, 50, 100\}$.

8.4 Experimental Results

First, we identify the best parameter setting for each algorithm. Then, we compare the PSO variants employing these settings.

8.4.1 Identification of Best Parameter Settings

To ease the depiction, we show results only for the Sphere function and MPB-Noisy with noise level 1.0, which are representative for the results of the other SRR and MPB-Noisy experiments. The complete results are given in Appendix D.2. The parameter values are abbreviated as follows: 1.05 for 1.0532 and 1.49 for 1.49618. In Table 8.1, the column headings “additive” and “adaptive” denote the fourth and fifth settings, respectively, described in Paragraph 8.3. A bold value signifies the best parameter setting regarding the respective benchmark and metric. In the following, we use $\theta = \theta_1 = \theta_2 = \theta_3$.

On the SRR benchmark (Table 8.1), `dynPSO` with $\theta = 1.49$ clearly outperforms the other parameter settings as it achieves most often the best results. Variant `pred2p` performs with $\theta = 1.05$ and $c = 1.49$ rather equally regarding the measures $\overline{\text{BOG}}$ and BEBC, but the convergence speed (RCS) is much better with 1.05. Therefore, we employ setting $\theta = 1.05$ for the further evaluations, although this actually is a non-convergent setting for `pred2p`. Approaches `pred3` and `pred3p` achieve best results with $c = 1.05$ regarding all metrics. The settings with an additive or adaptive θ_3 seem not to be advantageous; they achieve similar performance to $c = 1.49$. For the Rastrigin and Rosenbrock functions, the same settings as for the Sphere function appear to be the best, see Tables D.10, D.11, D.12, and D.13 in the appendix.

In the MPB-Noisy experiments (Table 8.2), for `dynPSO` $c = 1.49$ is slightly better than $\theta = 2.0$. The variants `pred2p`, `pred3` and `pred3p` perform best with $\theta = 1.05$. For the other noise levels, the same settings are reasonable choices, see Tables D.14, D.15, D.16, and D.17.

8.4.2 Comparison of PSO Variants

We compare the PSO variants separately for SRR and MPB-Noisy experiments. The parameter settings are those identified as best in Paragraph 8.4.1.

SRR

Table 8.3 shows the experiments with respect to the three metrics $\overline{\text{BOG}}$, BEBC and RCS for each of the four examined PSO variants averaged over all runs. The abbreviations “sph.”, “ros.”, and “rast.” stand for Sphere, Rosenbrock, and the Rastrigin function, respectively, the abbreviations “lin.”, and “sin.” denote a linear and sine movement, respectively, of the objective function. The last row of the table counts how often the respective algorithm is the best over all experiments. An algorithm has a bold value, if it outperforms the other three PSO variants in that experiment regarding the respective metric.

Table 8.1: SRR (Sphere, sine movement): comparison of parameter settings

Metric		BOG					BEBC					RCS				
Dim		1.05	1.49	2.00	additive	adaptive	1.05	1.49	2.00	additive	adaptive	1.05	1.49	2.00	additive	adaptive
2	dynPSO	8.1E+0	1.6E-1	**1.3E-1**	-	-	1.3E-4	**1.0E-4**	1.4E-4	-	-	**0.01**	0.02	0.02	-	-
5		1.2E+1	**2.7E+0**	7.2E+0	-	-	1.9E-1	**3.6E-2**	1.8E-1	-	-	**0.03**	0.03	0.05	-	-
10		4.5E+2	**9.4E+1**	1.7E+2	-	-	2.9E+2	**5.7E+0**	1.8E+1	-	-	0.15	**0.06**	0.10	-	-
20		1.3E+3	**4.6E+2**	8.3E+2	-	-	9.3E+2	**1.0E+2**	2.5E+2	-	-	0.31	**0.18**	0.27	-	-
50		6.6E+4	**9.2E+3**	1.4E+4	-	-	6.3E+4	**5.4E+3**	9.1E+3	-	-	0.82	**0.47**	0.55	-	-
100		4.9E+5	**4.1E+4**	5.2E+4	-	-	4.8E+5	**3.3E+4**	4.2E+4	-	-	0.94	**0.71**	0.73	-	-
2	pred2p	7.3E+0	1.5E-1	**1.4E-1**	-	-	**9.5E-5**	1.1E-4	1.5E-4	-	-	**0.01**	0.02	0.02	-	-
5		1.2E+1	**2.8E+0**	7.0E+0	-	-	7.1E-2	**3.8E-2**	1.9E-1	-	-	**0.02**	0.04	0.05	-	-
10		1.5E+2	**9.3E+1**	1.7E+2	-	-	**4.0E+0**	5.5E+0	1.8E+1	-	-	**0.05**	0.06	0.10	-	-
20		6.0E+2	**4.5E+2**	8.1E+2	-	-	2.5E+2	**1.0E+2**	2.4E+2	-	-	**0.17**	0.18	0.27	-	-
50		**8.0E+3**	9.2E+3	1.4E+4	-	-	**5.2E+3**	5.5E+3	9.2E+3	-	-	**0.20**	0.47	0.55	-	-
100		**3.8E+4**	4.1E+4	5.1E+4	-	-	**3.2E+4**	3.3E+4	4.2E+4	-	-	**0.25**	0.70	0.73	-	-
2	pred3	2.1E+0	**1.1E-1**	2.0E-1	1.7E-1	1.1E-1	**7.9E-5**	1.6E-4	2.2E-4	2.4E-4	1.6E-4	**0.00**	0.02	0.03	0.03	0.03
5		5.0E+0	**3.8E+0**	9.6E+0	6.2E+0	3.8E+0	**3.4E-2**	5.9E-2	2.4E-1	9.7E-2	5.9E-2	**0.02**	0.05	0.07	0.06	0.06
10		1.1E+2	**1.1E+2**	1.9E+2	1.3E+2	1.1E+2	**2.4E+0**	6.6E+0	1.8E+1	7.5E+0	6.7E+0	**0.03**	0.07	0.11	0.08	0.09
20		**4.4E+2**	4.6E+2	8.1E+2	4.9E+2	4.6E+2	1.6E+2	1.1E+2	2.3E+2	**1.1E+2**	1.1E+2	**0.12**	0.19	0.26	0.19	0.20
50		**5.6E+3**	8.2E+3	1.5E+4	9.1E+3	8.2E+3	**2.9E+3**	4.5E+3	9.2E+3	5.2E+3	4.5E+3	**0.15**	0.42	0.56	0.48	0.46
100		**2.8E+4**	3.7E+4	5.2E+4	3.7E+4	3.7E+4	**2.1E+4**	2.8E+4	4.2E+4	2.8E+4	2.8E+4	**0.20**	0.62	0.73	0.68	0.68
2	pred3p	2.1E+0	**1.1E-1**	2.0E-1	1.7E-1	1.2E-1	**8.0E-5**	1.6E-4	2.3E-4	2.5E-4	1.6E-4	**0.00**	0.02	0.03	0.03	0.03
5		5.0E+0	**3.8E+0**	9.5E+0	6.2E+0	3.8E+0	**3.4E-2**	5.9E-2	2.4E-1	9.8E-2	5.9E-2	**0.02**	0.05	0.07	0.06	0.06
10		1.1E+2	1.1E+2	1.9E+2	1.3E+2	**1.1E+2**	**2.4E+0**	6.7E+0	1.8E+1	7.3E+0	6.7E+0	**0.03**	0.07	0.11	0.08	0.08
20		**4.4E+2**	4.6E+2	8.2E+2	5.0E+2	4.6E+2	1.6E+2	1.1E+2	2.3E+2	**1.1E+2**	1.1E+2	**0.12**	0.19	0.26	0.19	0.20
50		**5.6E+3**	8.2E+3	1.5E+4	9.1E+3	8.2E+3	**2.9E+3**	4.5E+3	9.0E+3	5.2E+3	4.5E+3	**0.15**	0.41	0.55	0.48	0.46
100		**2.7E+4**	3.6E+4	5.2E+4	3.6E+4	3.6E+4	**2.0E+4**	2.8E+4	4.2E+4	2.8E+4	2.8E+4	**0.20**	0.62	0.73	0.67	0.68

Table 8.2: MPB-Noisy (noise = 1.0): comparison of parameter settings

Metric		BOG			BEBC			RCS		
Dim		1.05	1.49	2.00	1.05	1.49	2.00	1.05	1.49	2.00
2	dynPSO	-21.59	-104.53	**-171.54**	153.76	71.47	**4.05**	0.91	0.60	**0.14**
20		**-26.31**	-21.90	-11.29	**159.74**	175.45	200.28	**0.86**	0.90	0.95
50		-0.04	**-0.36**	-0.22	194.27	**193.78**	194.01	1.00	**1.00**	1.00
100		0.00	**-0.01**	-0.01	187.29	**187.28**	187.28	1.00	**1.00**	1.00
2	pred2p	-21.76	-79.04	**-167.25**	153.76	97.49	**8.41**	0.91	0.73	**0.19**
20		**-29.27**	-21.66	-11.15	**151.31**	176.18	200.69	**0.85**	0.90	0.95
50		-0.14	**-0.35**	-0.23	194.11	**193.79**	194.00	1.00	**1.00**	1.00
100		**-0.05**	-0.03	-0.01	**187.23**	187.26	187.28	**1.00**	1.00	1.00
2	pred3	-22.42	-111.65	**-166.98**	153.76	64.59	**7.69**	0.91	0.52	**0.18**
20		**-33.21**	-21.90	-11.33	**144.28**	175.34	199.96	**0.83**	0.90	0.95
50		**-0.29**	-0.22	-0.22	**193.80**	193.97	193.99	**1.00**	1.00	1.00
100		**-0.16**	-0.10	-0.02	**187.08**	187.16	187.27	**1.00**	1.00	1.00
2	pred3p	-22.44	-119.38	**-164.79**	153.76	56.87	**9.72**	0.91	0.49	**0.21**
20		**-32.62**	-21.98	-11.35	**145.66**	175.32	199.76	**0.84**	0.90	0.95
50		**-0.30**	-0.23	-0.22	**193.79**	193.96	194.00	**1.00**	1.00	1.00
100		**-0.15**	-0.11	-0.02	**187.10**	187.16	187.27	**1.00**	1.00	1.00

Considering Table 8.3, `pred2p` can be classified as worst of the variants, since it is never the best one. Algorithm `dynPSO` outperforms all other variants regarding $\overline{\text{BOG}}$, but often only when the objective function is low-dimensional. Its results for BEBC are good as well (`pred3p` is equally often best), but it is outperformed by `pred3`. Taking into account the algorithms' convergence properties, `pred3` and `pred3p` clearly surpass `dynPSO` and `pred2p`. Altogether, `pred3` achieves in most cases the best result, followed by `pred3p` and `dynPSO`.

To examine whether the differences between the PSO variants are statistically significant, we perform pairwise Mann-Whitney U tests. Table 8.4 contains the p values, where statistically significant values, i. e., $p < 0.05$, are highlighted bold. The table is constructed as follows. The algorithm whose name is written vertically is tested against the algorithms whose names are in the column headlines, e. g., `dynPSO` is tested pairwise against `pred2p`, `pred3` and `pred3p`. The p values show in nearly all cases statistical significance for tests of `dynPSO` and `pred2p` against the other approaches, whereas the tests `pred3` versus `pred3p` hardly ever yield statistically significant values.

MPB-Noisy

Table 8.5 shows the metric values for the MPB-Noisy experiments. Since here the best achievable $\overline{\text{BOG}}$ depends on the respective fitness function (for SRR it is 0), a column for the expected value is inserted. The benchmark column contains the strength of the noise in parentheses. Interestingly, there is no relationship between the noise strength and which algorithm achieves best results. Like for the SRR benchmarks, `pred2p` least frequently outperforms other algorithms, and `dynPSO` achieves better results than `pred3` and `pred3p` on rather low dimensions. Examining RCS, `pred3p` is most frequently best. Regarding $\overline{\text{BOG}}$ and BEBC, `pred3` and `pred3p` have often very similar metric values whereas `dynPSO` and `pred2p` appear to be different to them and to each other. These findings are confirmed by statistical tests (Table 8.6). Approaches `pred3` and `pred3p` remain too similar, whereas `dynPSO` and `pred2p` appear to be statistically different from the other PSO variants.

Table 8.3: SRR: comparison of PSO variants with best parameter settings

Metric	BOG				BEBC				RCS			
Benchmark	dynPSO	pred2p	pred3	pred3p	dynPSO	pred2p	pred3	pred3p	dynPSO	pred2p	pred3	pred3p
sph. 2 lin.	**6.0E-2**	9.6E-1	5.0E-1	5.0E-1	5.2E-5	3.1E-5	**2.8E-5**	2.9E-5	0.01	0.01	0.00	**0.00**
sph. 5 lin.	**3.1E+0**	8.0E+0	3.1E+0	3.1E+0	2.7E-2	1.9E-2	**1.8E-2**	1.8E-2	0.02	0.02	0.01	**0.01**
sph. 10 lin.	2.2E+1	3.1E+1	**1.5E+1**	1.5E+1	1.4E+0	4.1E-1	**3.2E-1**	3.2E-1	0.07	0.05	**0.02**	0.02
sph. 20 lin.	1.6E+2	1.4E+2	**7.7E+1**	7.7E+1	3.5E+1	1.7E+1	9.6E+0	**9.6E+0**	0.20	0.14	0.07	**0.07**
sph. 50 lin.	3.5E+3	2.7E+3	**1.7E+3**	1.7E+3	2.3E+3	1.8E+3	**1.1E+3**	1.2E+3	0.56	0.15	**0.13**	0.13
sph. 100 lin.	7.0E+4	6.4E+4	**4.0E+4**	4.2E+4	6.3E+4	5.8E+4	**3.6E+4**	3.7E+4	0.83	0.18	**0.16**	0.17
sph. 2 sin.	**1.6E-1**	7.3E+0	2.1E+0	2.1E+0	1.0E-4	9.5E-5	**7.9E-5**	8.0E-5	0.02	0.01	0.00	**0.00**
sph. 5 sin.	**2.7E+0**	1.2E+1	5.0E+0	5.0E+0	3.6E-2	7.1E-2	3.4E-2	**3.4E-2**	0.03	0.02	**0.02**	0.02
sph. 10 sin.	**9.4E+1**	1.5E+2	1.1E+2	1.1E+2	5.7E+0	4.0E+0	**2.4E+0**	2.4E+0	0.06	0.05	**0.03**	0.03
sph. 20 sin.	4.6E+2	6.0E+2	4.4E+2	**4.4E+2**	**1.0E+2**	2.5E+2	1.6E+2	1.6E+2	0.18	0.17	0.12	**0.12**
sph. 50 sin.	9.2E+3	8.0E+3	**5.6E+3**	5.6E+3	5.4E+3	5.2E+3	**2.9E+3**	2.9E+3	0.47	0.20	0.15	**0.15**
sph. 100 sin.	4.1E+4	3.8E+4	2.8E+4	**2.7E+4**	3.3E+4	3.2E+4	2.1E+4	**2.0E+4**	0.71	0.25	0.20	**0.20**
ros. 2 lin.	**3.8E+0**	3.8E+2	1.2E+2	1.2E+2	**1.2E-3**	9.0E-3	5.5E-3	5.5E-3	0.01	0.01	**0.00**	0.00
ros. 5 lin.	**2.5E+3**	1.1E+4	5.4E+3	5.4E+3	**1.8E+1**	2.0E+1	1.9E+1	1.9E+1	0.01	0.01	**0.00**	0.00
ros. 10 lin.	**9.0E+4**	2.2E+5	1.9E+5	1.9E+5	8.4E+2	2.8E+2	1.2E+2	**1.2E+2**	0.02	0.02	**0.01**	0.01
ros. 20 lin.	1.0E+7	2.7E+6	**2.2E+6**	2.2E+6	2.9E+6	1.8E+5	**4.9E+3**	4.9E+3	0.16	0.05	**0.01**	0.01
ros. 50 lin.	2.7E+8	1.3E+8	**6.8E+7**	7.1E+7	1.2E+8	4.8E+7	**1.8E+7**	2.0E+7	0.37	0.20	**0.09**	0.10
ros. 100 lin.	1.4E+10	1.1E+10	7.7E+9	**7.7E+9**	1.1E+10	9.5E+9	**6.8E+9**	6.8E+9	0.72	0.16	**0.14**	0.14
ros. 2 sin.	**3.7E+4**	3.1E+5	5.9E+4	5.9E+4	**1.5E-1**	3.8E+3	3.7E+3	3.7E+3	**0.01**	0.05	0.04	0.04
ros. 5 sin.	**7.6E+3**	4.1E+5	1.2E+5	1.1E+5	8.3E+1	3.7E+1	1.1E+1	**9.2E+0**	0.02	0.00	0.00	**0.00**
ros. 10 sin.	**4.8E+5**	1.6E+6	9.1E+5	9.3E+5	7.3E+3	5.2E+3	2.9E+3	**2.7E+3**	0.02	0.01	**0.01**	0.01
ros. 20 sin.	8.1E+6	5.3E+6	**3.6E+6**	3.6E+6	6.5E+5	1.1E+5	**5.7E+4**	6.0E+4	0.07	0.03	**0.02**	0.02
ros. 50 sin.	8.2E+8	3.2E+8	2.4E+8	**2.3E+8**	3.2E+8	1.2E+8	6.7E+7	**6.6E+7**	0.27	0.14	0.10	**0.10**
ros. 100 sin.	5.6E+9	3.5E+9	2.6E+9	**2.5E+9**	3.9E+9	2.5E+9	1.5E+9	**1.5E+9**	0.52	0.16	0.14	**0.14**
rast. 2 lin.	**1.6E+0**	3.8E+0	2.1E+0	2.1E+0	5.6E-3	1.1E-2	4.7E-3	**4.5E-3**	0.03	0.03	0.01	**0.01**
rast. 5 lin.	2.2E+1	4.9E+1	**2.2E+1**	2.2E+1	7.4E+0	2.0E+1	6.9E+0	**6.9E+0**	0.18	0.08	**0.05**	0.05
rast. 10 lin.	**9.9E+1**	2.2E+2	1.3E+2	1.3E+2	**5.6E+1**	1.3E+2	8.1E+1	8.1E+1	0.33	0.08	**0.08**	0.08
rast. 20 lin.	4.0E+2	6.2E+2	3.9E+2	**3.9E+2**	**2.3E+2**	3.9E+2	2.5E+2	2.5E+2	0.42	0.12	**0.10**	0.10
rast. 50 lin.	**4.1E+3**	6.7E+3	4.9E+3	4.9E+3	**2.8E+3**	5.4E+3	4.1E+3	4.1E+3	0.60	0.17	0.15	**0.15**
rast. 100 lin.	6.2E+4	9.3E+4	5.3E+4	**5.2E+4**	5.5E+4	8.6E+4	4.8E+4	**4.7E+4**	0.81	0.21	0.17	**0.17**
rast. 2 sin.	**5.1E+0**	3.2E+1	1.7E+1	1.7E+1	**1.0E-1**	1.1E+1	9.4E+0	9.4E+0	0.04	0.06	0.04	**0.04**
rast. 5 sin.	**2.6E+1**	1.0E+2	4.6E+1	4.6E+1	**1.0E+1**	6.2E+1	2.5E+1	2.5E+1	0.22	0.07	0.06	**0.06**
rast. 10 sin.	**1.8E+2**	4.1E+2	3.1E+2	3.1E+2	**7.5E+1**	2.2E+2	1.6E+2	1.6E+2	0.17	0.16	**0.12**	0.12
rast. 20 sin.	**6.6E+2**	1.1E+3	8.0E+2	8.0E+2	**3.1E+2**	7.0E+2	4.8E+2	4.8E+2	0.28	0.12	0.10	**0.10**
rast. 50 sin.	1.0E+4	9.8E+3	**6.7E+3**	6.7E+3	6.2E+3	7.0E+3	**4.0E+3**	4.0E+3	0.49	0.18	**0.14**	0.14
rast. 100 sin.	4.3E+4	4.0E+4	**2.9E+4**	2.9E+4	3.5E+4	3.4E+4	**2.2E+4**	2.2E+4	0.71	0.29	**0.23**	0.23
Best frequen.	18	0	11	7	11	0	14	11	1	0	19	16

Table 8.4: SRR: p values for pairwise tests of PSO variants

Algorithm		pred2p			pred3			pred3p				pred3			pred3p				pred3p		
Benchmark		BOG	BEBC	RCS	BOG	BEBC	RCS	BOG	BEBC	RCS		BOG	BEBC	RCS	BOG	BEBC	RCS		BOG	BEBC	RCS
sph. 2 lin.	dynPSO	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	pred2p	**6.8E-8**	**3.3E-5**	**6.8E-8**	**6.8E-8**	**2.3E-3**	**6.8E-8**	pred3	8.2E-1	1.6E-1	3.9E-1
sph. 5 lin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		2.7E-1	4.2E-1	7.8E-1
sph. 10 lin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		2.0E-1	9.9E-1	4.2E-1
sph. 20 lin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		9.0E-1	5.8E-1	6.4E-1
sph. 50 lin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		7.6E-2	**4.1E-2**	**2.9E-2**
sph. 100 lin.		**4.0E-3**	**4.0E-3**	**1.9E-7**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**1.1E-7**		2.3E-1	2.3E-1	3.2E-1
sph. 2 sin.		**8.0E-9**	**4.0E-6**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**1.4E-6**	**6.8E-8**	**6.8E-8**	**4.5E-6**	**6.8E-8**		2.2E-1	8.8E-1	7.1E-1
sph. 5 sin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**4.0E-6**	**6.8E-8**	**8.0E-9**	**2.1E-7**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		7.4E-1	6.2E-1	7.8E-1
sph. 10 sin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		1.3E-1	4.1E-1	3.6E-1
sph. 20 sin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		8.6E-1	5.6E-1	8.4E-1
sph. 50 sin.		**8.0E-9**	**4.0E-3**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		4.9E-1	9.9E-1	8.2E-1
sph. 100 sin.		**4.0E-6**	**2.2E-2**	**1.9E-7**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		**1.5E-2**	**1.9E-2**	3.0E-1
ros. 2 lin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**2.1E-6**	**6.8E-8**	**6.8E-8**	**1.8E-6**	**6.8E-8**		5.8E-1	2.4E-1	1.6E-1
ros. 5 lin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**3.0E-7**	**6.8E-8**		9.7E-1	1.5E-1	9.9E-1
ros. 10 lin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		7.8E-1	6.9E-1	5.2E-1
ros. 20 lin.		**8.0E-9**	**8.0E-9**	**3.0E-8**	**8.0E-9**	**8.0E-9**	**3.0E-8**	**8.0E-9**	**8.0E-9**	**3.0E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		8.8E-1	6.9E-1	4.9E-1
ros. 50 lin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		2.5E-1	3.2E-1	3.4E-1
ros. 100 lin.		**2.1E-7**	**4.0E-6**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		5.4E-1	5.6E-1	6.2E-1
ros. 2 sin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		8.2E-1	8.6E-1	8.6E-1
ros. 5 sin.		**8.0E-9**	**4.0E-6**	**8.6E-6**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**8.6E-6**	**6.8E-8**	**6.8E-8**	**2.2E-7**	**6.8E-8**		2.6E-1	4.9E-1	2.7E-1
ros. 10 sin.		**8.0E-9**	**4.0E-6**	**6.8E-8**	**8.0E-9**	**8.0E-9**	3.9E-1	**8.0E-9**	**8.0E-9**	3.0E-1		**6.8E-8**	**1.8E-6**	**6.8E-8**	**6.8E-8**	**1.4E-6**	**6.8E-8**		7.4E-1	2.0E-1	7.4E-1
ros. 20 sin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**3.9E-7**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		2.5E-1	8.6E-2	1.5E-1
ros. 50 sin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		3.4E-1	3.0E-1	3.6E-1
ros. 100 sin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		**8.4E-3**	**2.2E-2**	4.4E-1
rast. 2 lin.		**8.0E-9**	**4.0E-6**	**6.8E-8**	**8.0E-9**	**4.0E-3**	**7.6E-6**	**8.0E-9**	**2.2E-2**	**1.7E-7**		**6.8E-8**	**5.9E-6**	**6.8E-8**	**6.8E-8**	**4.5E-6**	**6.8E-8**		8.6E-1	4.2E-1	8.6E-1
rast. 5 lin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**4.0E-6**	**2.1E-7**	**7.9E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		5.6E-1	9.0E-1	4.4E-1
rast. 10 lin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		2.2E-1	2.4E-1	3.0E-1
rast. 20 lin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**2.1E-7**	**8.0E-9**	**9.2E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		5.8E-1	9.7E-1	1.2E-1
rast. 50 lin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		6.9E-1	7.8E-1	2.1E-1
rast. 100 lin.		**8.0E-9**	**8.0E-9**	6.0E-1	**2.1E-7**	**4.0E-6**	**6.8E-8**	**2.1E-7**	**2.1E-7**	**1.4E-7**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**1.2E-7**	**1.2E-7**	**6.8E-8**		7.8E-1	7.4E-1	3.2E-1
rast. 2 sin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		3.1E-1	7.8E-1	7.6E-1
rast. 5 sin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		6.6E-1	4.6E-1	4.2E-1
rast. 10 sin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		8.1E-2	9.0E-1	6.8E-1
rast. 20 sin.		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		3.2E-1	3.4E-1	7.6E-1
rast. 50 sin.		**5.5E-5**	**2.1E-7**	**6.2E-4**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		8.2E-1	8.6E-1	7.8E-1
rast. 100 sin.		**4.0E-6**	**5.5E-5**	**1.1E-7**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		3.1E-1	4.7E-1	5.2E-1

Table 8.5: MPB-Noisy: comparison of PSO variants with best parameter settings

Metric		BOG					BEBC				RCS			
Benchmark		dynPSO	pred2p	pred3	pred3p	expected	dynPSO	pred2p	pred3	pred3p	dynPSO	pred2p	pred3	pred3p
noisy (0)	2	**-142.65**	-22.70	-23.11	-23.09	-177.42	**33.75**	153.76	153.76	153.76	**0.35**	0.93	0.91	0.91
noisy (0)	20	-22.31	**-32.82**	-17.07	-16.92	-225.32	176.51	**149.43**	191.60	191.73	**0.90**	0.90	0.91	0.91
noisy (0)	50	-0.40	-0.64	**-1.41**	-1.40	-194.32	193.73	193.36	**191.91**	191.93	1.00	1.00	**0.99**	0.99
noisy (0)	100	-0.02	-0.07	-0.17	**-0.18**	-187.29	187.27	187.21	187.08	**187.06**	1.00	1.00	1.00	**1.00**
noisy (0.1)	2	**-23.53**	-22.67	-23.05	-23.04	-177.42	153.76	153.76	**153.76**	153.76	0.98	**0.87**	0.91	0.91
noisy (0.1)	20	-22.67	**-34.03**	-15.80	-15.93	-225.32	176.79	**146.15**	192.33	192.03	0.90	**0.90**	0.91	0.91
noisy (0.1)	50	-0.17	-0.26	**-0.75**	-0.75	-194.32	194.06	193.90	**193.03**	193.03	1.00	1.00	**1.00**	1.00
noisy (0.1)	100	-0.01	-0.09	-0.12	**-0.15**	-187.29	187.28	187.18	187.13	**187.10**	1.00	1.00	1.00	**1.00**
noisy (1)	2	**-104.53**	-21.76	-22.42	-22.44	-177.42	**71.47**	153.76	153.76	153.76	**0.60**	0.73	0.91	0.91
noisy (1)	20	-21.90	-29.27	**-33.21**	-32.62	-225.32	175.45	151.31	**144.28**	145.66	0.90	0.90	**0.83**	0.84
noisy (1)	50	**-0.36**	-0.14	-0.29	-0.30	-194.32	**193.78**	194.11	193.80	193.79	1.00	1.00	1.00	**1.00**
noisy (1)	100	-0.01	-0.05	**-0.16**	-0.15	-187.29	187.28	187.23	**187.08**	187.10	1.00	1.00	**1.00**	1.00
noisy (10)	2	**-139.73**	-123.95	-126.34	-126.97	-177.42	**28.11**	28.75	31.56	30.68	0.20	0.24	0.20	**0.20**
noisy (10)	20	-0.58	-0.93	-1.11	**-1.13**	-225.32	223.82	222.27	221.58	**221.56**	1.00	1.00	0.99	**0.99**
noisy (10)	50	**-0.02**	-0.01	-0.01	-0.01	-194.32	**194.29**	194.30	194.29	194.29	1.00	1.00	1.00	**1.00**
noisy (10)	100	0.00	0.00	-0.01	**-0.01**	-187.29	187.29	187.29	187.28	**187.28**	1.00	1.00	1.00	**1.00**
Best frequency		6	2	4	4		5	2	5	4	3	2	4	7

Table 8.6: MPB-Noisy: p values for pairwise tests of PSO variants

Algorithm			pred2p			pred3			pred3p				pred3			pred3p				pred3p		
Benchmark			BOG	BEBC	RCS	BOG	BEBC	RCS	BOG	BEBC	RCS		BOG	BEBC	RCS	BOG	BEBC	RCS		BOG	BEBC	RCS
noisy (0)	2	dynPSO	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	pred2p	**6.8E-8**	**1.0E-3**	1.0E-1	**6.8E-8**	**3.1E-3**	1.1E-1	pred3	1.6E-1	9.9E-1	9.7E-1
noisy (0)	20		**5.5E-5**	**5.5E-5**	**1.6E-4**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**1.6E-4**	**1.6E-4**	**1.6E-4**	**1.6E-4**	**1.6E-4**	**1.6E-4**		4.2E-1	5.4E-1	7.6E-1
noisy (0)	50		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		4.4E-1	5.6E-1	5.4E-1
noisy (0)	100		**4.0E-6**	**4.0E-6**	**3.4E-4**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**3.3E-5**	**2.0E-5**	**2.3E-5**	**1.3E-5**	**9.7E-6**	**9.7E-6**		5.8E-1	6.0E-1	6.0E-1
noisy (0.1)	2		**8.0E-9**	**8.0E-9**	**5.6E-3**	**8.0E-9**	**8.0E-9**	1.2E-1	**8.0E-9**	**8.0E-9**	1.0E-1		**6.8E-8**	**2.6E-5**	1.5E-1	**6.8E-8**	**2.3E-3**	1.8E-1		2.4E-1	7.8E-1	9.2E-1
noisy (0.1)	20		**2.1E-7**	**2.1E-7**	**1.2E-6**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**1.2E-6**	**1.2E-6**	**1.2E-6**	**1.2E-6**	**1.2E-6**	**1.2E-6**		3.6E-1	8.6E-2	1.1E-1
noisy (0.1)	50		**2.1E-7**	**2.1E-7**	**1.2E-6**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		6.9E-1	7.6E-1	7.1E-1
noisy (0.1)	100		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		8.4E-1	6.9E-1	6.9E-1	**2.6E-2**	**2.1E-2**	**2.4E-2**		4.6E-1	4.4E-1	4.2E-1
noisy (1)	2		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**3.7E-5**	**9.1E-7**	**6.8E-8**	**2.0E-5**	**1.2E-6**		**1.1E-2**	4.6E-1	7.4E-1
noisy (1)	20		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		6.4E-2	9.2E-1	4.1E-1	**1.5E-2**	9.7E-1	5.1E-1		9.0E-1	9.0E-1	8.2E-1
noisy (1)	50		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**5.5E-4**	**6.8E-8**	**8.0E-9**	9.9E-1	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		2.0E-1	2.6E-1	2.7E-1
noisy (1)	100		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**2.6E-5**	**2.3E-5**	**2.3E-5**		7.4E-1	7.8E-1	8.2E-1
noisy (10)	2		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**7.1E-3**	**2.7E-2**	**7.1E-3**	**2.2E-4**	9.1E-2	**1.2E-3**		5.2E-1	5.8E-1	6.6E-1
noisy (10)	20		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**9.2E-8**	**6.8E-8**		2.6E-1	7.8E-1	3.1E-1
noisy (10)	50		**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		2.6E-1	2.4E-1	2.1E-1
noisy (10)	100		**5.5E-5**	**5.5E-5**	**3.9E-7**	**8.0E-9**	**8.0E-9**	**6.8E-8**	**8.0E-9**	**8.0E-9**	**6.8E-8**		**2.2E-7**	**6.8E-8**	**7.9E-8**	**6.8E-8**	**6.8E-8**	**6.8E-8**		1.6E-1	1.3E-1	1.6E-1

8.5 Summary

In this chapter, we proposed three different extensions of dynamic PSO (pred2p, pred3, pred3p). They integrate a prediction model that forecasts the optimum position of the next change period. Approach pred2p inserts the prediction as particle into the swarm, pred3 adds a third attractor to the velocity function with the predicted optimum serving as attractor, and pred3p combines both approaches.

The experimental study demonstrated that the predicted optimum as third attractor within the velocity function is necessary to successfully integrate prediction into PSO, since pred2p achieved worse results than the basic PSO without prediction. Outstanding results of the prediction variants (pred3, pred3p) could be observed regarding convergence, while the results for tracking accuracy were satisfying as well. Especially, on problems with higher dimensionality positive influence of prediction could be observed.

9 Predictive Uncertainty for Evolution Strategies

In the approaches covered by this thesis, prediction strategies aim at estimating the moving optimum after a change of the fitness function. Considering the predicted optimum for re-initialization of the population, the optimizer is led into the direction of the next optimum. Prediction approaches have in common that they might hamper the optimization in case the predicted optimum differs much from the true one. If all individuals were re-initialized around the falsely predicted optimum after a fitness function change, an ES or PSO possibly needs some extra generations to find a promising region in the solution space. Ideally, the closer the prediction is to the true optimum, the more individuals should be placed close to the prediction. As the true optimum is unknown, it would be useful to have at least an estimate $\hat{\mathbf{u}}_c$ for the predictive uncertainty, that is an estimate of how far the predicted optimum $\hat{\mathbf{o}}_c$ is from the real one (Line 11 in Algorithm 7).

In this chapter, we propose a new re-initialization strategy for ES to control the influence of the prediction depending on its estimated uncertainty. To our knowledge, there exists only the work [RAD08] that takes into account predictive uncertainty. Instead of a Kalman filter, employed by [RAD08], we use a prediction model that might be able to capture more difficult problem dynamics: a temporal convolutional network (TCN) equipped with Monte Carlo dropout. Since in preliminary experiments TCNs seemed to be easier and more stable to train, and, e.g., also [BKK18] reports superiority of TCNs against basic RNN structures, we do not employ recurrent neural networks that have been applied in Chapters 7 and 8. However, the focus of this chapter lies rather on the evaluation of re-initialization strategies than on comparison of prediction methods. This chapter is based on our publication [MK19].

9.1 Related Work

We explain how existing re-initialization strategies try to ensure a diverse population and take into account uncertainty caused by a prediction. After-

Algorithm 7 Dynamic (μ+λ)-ES with predictive uncertainty

```
1:  $\mathbf{P} \leftarrow$ initialize_population()
2:  $s \leftarrow$ initialize_mutation_strength()
3:  $\mathbf{A} \leftarrow [\,]$                                   # found solutions
4:  $c \leftarrow 1$                                      # change period counter
5:  for generations do
6:      $s \leftarrow$ adapt_mutation_strength($s$)        # Rechenberg's 1/5th rule
7:      if change_detected() then
8:          $c \leftarrow c + 1$
9:          $s \leftarrow$ reset_mutation_strength()
10:         $\mathbf{A} \leftarrow \mathbf{A}$.append($\mathbf{x}^*_{c-1}$)          # store best solution
11:         $\hat{\mathbf{o}}_c, \hat{\mathbf{u}}_c \leftarrow$ train_and_predict($\mathbf{A}$)      # predict opt. & unc.
12:         $\mathbf{P} \leftarrow$ reinitialize_population($\mathbf{P}, \hat{\mathbf{o}}_c, \hat{\mathbf{u}}_c$)
13:     $\mathbf{P}' \leftarrow$ create_$\lambda$_offspring_individuals($\mathbf{P}, s$)  # recomb. & mutation
14:     $\mathbf{P} \leftarrow$ select_best_$\mu$_individuals($\mathbf{P}, \mathbf{P}'$)
```

wards, we describe the process of uncertainty estimation for neural networks that we use in our approach.

9.1.1 Re-Initialization Strategies

Population re-initialization after a change is important to support exploration abilities of the ES (Line 12 in Algorithm 7). Different re-initialization strategies exist both for dynamic optimization with [HW06; RAD08; SC14; BLY17] and without prediction [NYB12; CGP11; WY09]. In the following, we introduce re-initialization strategies, that have been proposed for ES and are used in our experimental study. The pattern for the strategies' names is as follows. The first letter signifies whether a prediction model is applied (`p`) or not (`n`). The last letters denote the respective strategy. Vector $\mathbf{x}_c \in \mathbb{R}^d$ is an immigrant of the population in change period c.

`nRND`

The new individuals $\mathbf{x}_c$ are randomly sampled within the lower bound b^l and upper bound b^u of the solution space: $\mathbf{x}_c \sim \mathcal{U}\left(b^l, b^u\right)$ [Zho+07].

nVAR

The new individuals are the old ones with additional noise: $\mathbf{x}_c = \mathbf{x}_{c-1} + \varepsilon$. The noise is sampled with $\varepsilon \sim \mathcal{N}\left(0, \frac{1}{4d}\|\mathbf{x}_{c-1} - \mathbf{x}_{c-2}\|_2^2\right)$ and depends on the difference between the current position $\mathbf{x}_{c-1}$ and the position of the nearest individual $\mathbf{x}_{c-2}$ in the previous population [Zho+07].

nPRE

This strategy does not require a separate prediction model but serves itself as a simple prediction approach [Zho+07]. For each individual $\mathbf{x}_{c-1}$ its next position $\hat{\mathbf{x}}_c$ is predicted with $\hat{\mathbf{x}}_c = \mathbf{x}_{c-1} + (\mathbf{x}_{c-1} - \mathbf{x}_{c-2})$ where $\mathbf{x}_{c-2}$ is defined as in nVAR. The individuals are re-initialized at their predicted positions that are perturbed with noise ε as in nVAR: $\mathbf{x}_c = \hat{\mathbf{x}}_c + \varepsilon$.

pKAL

The only approach that considers uncertainty $\hat{\mathbf{u}}_c$ of prediction $\hat{\mathbf{o}}_c$, is the work of [RAD08] with a Kalman filter prediction model. Based on the estimated prediction uncertainty they adapt the number of individuals that are placed around the predicted optimum. The new individuals $\mathbf{x}_c$ are sampled from $\mathbf{x}_c \sim \mathcal{N}(\hat{\mathbf{o}}_c, \hat{\mathbf{u}}_c)$ leading to a larger spread in dimensions with high uncertainty. Here, the a priori state estimation $\hat{\boldsymbol{a}}_c^-$ is the predicted optimum $\hat{\mathbf{o}}_c$, and the a priori error variance, i.e., the diagonal of the a priori error covariance $\hat{\boldsymbol{E}}_c^-$, serves as estimate for the predictive uncertainty $\hat{\mathbf{u}}_c$ (see Paragraph 5.3).

Rossi *et al.* [RAD08] propose to re-initialize only $\lfloor \delta \cdot \mu \rfloor$ individuals around the predicted optimum, the remaining ones are re-initialized with a standard method, e.g., nRND. Factor δ decreases with increasing uncertainty, where $\delta = \frac{\chi}{1+\hat{u}_{\max}}$ and $0 < \chi < 1 + \hat{u}_{\max}$. Here, $\hat{u}_{\max}$ denotes the maximum entry of $\hat{\mathbf{u}}_c$, μ the population size, and χ a selectable constant.

9.1.2 Uncertainty Estimation with Neural Networks

By design, artificial neural networks (NNs) output a point prediction $\hat{\mathbf{y}}$ for a given input $\mathbf{x} \in \mathbb{R}^d$. In order to get an estimate for the uncertainty of the output, NNs of any type can be combined with Monte Carlo (MC) dropout without changing the network architecture [Gal16]. With MC dropout, neurons are dropped not only during training but also for prediction. After training of the NN, for a given input the output and its uncertainty are predicted by conducting m so-called Monte Carlo runs. The NN output is

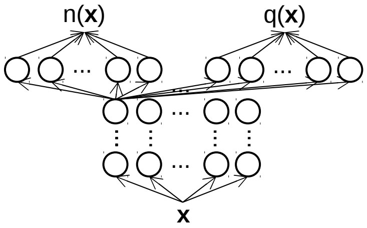


Figure 9.1: Feed-forward neural network with additional output layer for aleatoric uncertainty

computed m times for the same input with other neurons dropped in each run. This leads to m different network outputs for the given input. The average output and its variance, i.e., predictive mean and predictive variance, respectively, are computed with

$$\mathrm{E}\left[\hat{\mathbf{y}}\right] = \frac{1}{m}\sum_{i=1}^{m} n_i(\mathbf{x}) \tag{9.1}$$

$$\mathrm{Var}\left[\hat{\mathbf{y}}\right] = \frac{1}{m}\sum_{i=1}^{m} q_i(\mathbf{x}) + n_i(\mathbf{x})^2 - \mathrm{E}\left[\hat{\mathbf{y}}\right]^2 , \tag{9.2}$$

where $n_i(\mathbf{x})$ denotes the network output of the ith MC run for input $\mathbf{x}$. The predictive variance represents the uncertainty of the prediction. It consists of the sample variance plus noise $q_i(\mathbf{x})$ that is inherently present in the data, i.e., aleatoric uncertainty [KG17]. Aleatoric uncertainty is data-dependent and can automatically be learned by the NN without further information. Only an additional output layer for $q(\mathbf{x})$ and a corresponding loss function are required, see Figure 9.1. For more information on uncertainty estimation for NNs, see [OZK18; KG17; Gal16].

In the experimental study, we use a TCN combined with Monte Carlo dropout. However, also recurrent neural networks can be extended by the uncertainty estimation method. The ES employs the predictive mean $\mathrm{E}\left[\hat{\mathbf{y}}\right]$ as predicted optimum $\hat{\mathbf{o}}_c$ and the predictive variance $\mathrm{Var}\left[\hat{\mathbf{y}}\right]$ as predictive uncertainty $\hat{\mathbf{u}}_c$.

9.2 Uncertainty-Aware Re-Initialization

We propose new population re-initialization strategies for ES with prediction (`pUNC`, `pDEV`, `pRND`): one with and two without predictive uncertainty estimation. They are described in the following.

pUNC

This is our new re-initialization strategy with predictive uncertainty. In contrast to pKAL, we propose to sample not only some as in [RAD08] but all new individuals from confidence intervals around the predicted optimum with $\mathbf{x}_c \sim z \cdot \mathcal{N}\left(\hat{\mathbf{o}}_c, \hat{\mathbf{u}}_c\right)$. In pKAL, the problem that a poor prediction might lead to a population concentrated in a region far from the real optimum is circumvented by also sampling immigrants uniformly in the whole solution space. Since this strategy often might place many individuals in regions not containing promising solutions, we propose to sample the individuals from various intervals around the prediction, e.g., with $z \in \{0.1, 0.5, 1.0, 2.0\}$ that represent the probability intervals 0.09, 0.38, 0.68, and 0.95 of the normal distribution. For each interval, an equal proportion of the overall number of immigrants is generated. By this means, the population covers a large area of the solution space but still is centered around the prediction. This might be advantageous as prediction often provides some information about the direction of the optimum movement even if the predicted position does not exactly represent the real one. Nevertheless, the population is spread widely so that it is able to explore other regions.

In our publication [MK19], we accidentally implemented pUNC in a different way: $\mathbf{x}_c \sim \mathcal{N}\left(\hat{\mathbf{o}}_c, z \cdot \sqrt{\hat{\mathbf{u}}_c}\right)$. Therefore, the concrete results of the experimental study in this thesis differ slightly from the published ones, nevertheless the conclusion is same.

pDEV

In order to examine whether the uncertainty estimation of the Kalman filter and the TCN, respectively, are useful we also propose a simpler kind of uncertainty estimation. Here, the deviation of the predicted and found optimum is interpreted as uncertainty $\sigma = \sqrt{\frac{1}{d}\|\mathbf{x}^*_{c-1} - \hat{\mathbf{o}}_{c-1}\|^2_2}$. We sample with different z values like in pUNC: $\mathbf{x}_c \sim z \cdot \mathcal{N}\left(\hat{\mathbf{o}}_c, \sigma^2\right)$. In contrast to pKAL and pUNC, here only one uncertainty estimate for all d dimensions is available.

pRND

Our last strategy does not consider predictive uncertainty and is only for the sake of comparison. The predicted optimum is randomly perturbed with different scales z: $\mathbf{x}_c \sim z \cdot \mathcal{N}\left(\hat{\mathbf{o}}_c, \mathbf{I}\right)$, where $\mathbf{I}$ is the identity. We use this strategy in Chapter 7 as well.

9.3 Experimental Setup

We equip the same base ES (Algorithm 7) with five different prediction methods: no prediction model (`npm`), a linear autoregressive model (`ar`) [HW06], TCN without (`tcn`) and with uncertainty estimation (`unc`), and a Kalman filter (`kal`). We combine the prediction methods with following re-initialization strategies:

- `npm` with `nRND`, `nVAR`, and `nPRE`
- `ar` and `tcn` with `pRND`, and `pDEV`
- `kal` and `unc` with `pRND`, `pDEV`, `pKAL`, and `pUNC`

For `pKAL`, we set $\chi = 0.1$ since this setting turned out to be good in the original work. All re-initialization strategies with probability intervals have $z \in \{0.1, 0.5, 1.0, 2.0\}$.

We employ a (50+100)-ES with mutation strength set to 1.0 initially. After a change, 50 immigrants are generated according to the respective re-initialization strategy and are inserted into the population; one third of them is randomly placed in the solution space to support exploration. The fitness function changes every 30th generation and the ES is conducted for 554 change periods. We chose this number in order to get a number of training data that is divisible by the batch size of the TCNs such that all batches have the same size. All experiments are repeated 20 times.

We predict the next optimum after each change but re-train the prediction models only every 75 changes to circumvent excessive runtimes. Overall, five training phases are conducted. In each of them we use the most recent 128 training patterns. This is different to the experiments in Chapters 7 and 8 where training took place after each change but only with the last new training pattern. The training data are scaled into interval $[-1, 1]$. Different from the experiments in the previous chapters, now the data are not the absolute positions but differences between each two succeeding solutions. This is a common strategy to ease prediction [Zha12]. We train the TCNs for 100 epochs and conduct 50 and 10 Monte Carlo runs during training and prediction, respectively. Since the training patterns have a window size of 50 time steps, the TCN requires 3 blocks according to Equation 5.7. Nevertheless, we use four blocks since slightly overparameterized networks seem to achieve better results. In preliminary experiments, we tuned the hyperparameters of the TCNs, see Appendix D.3.1. The best setting we found is: 27 filters, filter size 6, learning rate 0.001, batch size 32, and dropout probability 0.1.

We compare the algorithms on the dynamic sine benchmark (DSB) and the moving peaks benchmark (MPB). We initialize the benchmarks for dimensions $d \in \{2, 5, 10, 20\}$, the solution space is within $[0, 100]^d$. We combine DSB with the well-known fitness functions Sphere, Rastrigin, and Rosenbrock. The parameterization of DSB is: $C = 10$, $V = 0.5$, $\rho_{\max} = 4$. We instantiate MPB with ten peaks and noise $\nu \in \{0.0, 0.01, 0.05\}$, $\eta = 1 - \nu$.

Unlike in Chapters 7 and 8, here we measure not only the performance of the ES but also the error of the prediction models. We introduce prediction error (PE) as root mean squared error of the predicted and true optimum positions over all change periods. The best PE value is 0. The metrics are computed with respect to the generations in which a prediction is conducted.

9.4 Experimental Results

Both for DSB and MPB we first identify for each prediction model the best re-initialization strategy. Afterwards, we compare the different prediction approaches combined with the identified settings. We conduct pairwise Mann-Whitney U tests with significance level $\alpha = 0.05$ to examine statistical significance. The structure of the tables containing the results is the same as in Paragraph 7.4.6, only the employed metrics differ. Here, the order of metrics listed for the algorithms in the columns from left to right is: $\overline{\text{BOG}}$, BEBC, RCS, PE. With '·' we indicate that the PE measure is not computed for `npm`, since `npm` has no prediction model. The algorithms' names consist of the prediction model followed by the re-initialization strategy. For all metrics low values are desirable. Hence, an algorithm whose name is in the row significantly outperforms another algorithm if the respective metric contains '▼'. The columns 'Alg.', 'Bmk.', and "Dim." contain the algorithm's name, the benchmark function, and the problem dimensionality, respectively.

9.4.1 Dynamic Sine Benchmark

Comparison of Re-Initialization Strategies

ES without prediction (`npm`) performs best with re-initialization strategy `nVAR` because it almost never is worse than `nPRE` and `nRND` (Table 9.1). For prediction model `ar`, the choice of re-initialization strategy seems to be less important (Table 9.2). Here, strategy `pRND` leads to better results than `pDEV` only for $\overline{\text{BOG}}$. In contrast, `pRND` clearly outperforms `pDEV` if the ES is equipped with `tcn` (Table 9.3). Interestingly, the ES equipped with `kal` performs worse with `pKAL` than with those re-initialization strategies

Table 9.1: DSB: comparison of re-initialization strategies for `npm`. Order of metrics from left to right: $\overline{\text{BOG}}$, BEBC, RCS, PE

Alg.	Bmk.	Dim.	npm-nVAR	npm-nPRE
npm-nRND	Sphere	2	– – △ ·	△ – △ ·
		5	– △ △ ·	– △ △ ·
		10	△ △ △ ·	△ – – ·
		20	△ △ △ ·	△ – – ·
	Rosenbrock	2	– △ △ ·	– – △ ·
		5	– △ △ ·	– △ △ ·
		10	▼ – △ ·	– △ △ ·
		20	– – – ·	– – – ·
	Rastrigin	2	△ △ △ ·	△ △ △ ·
		5	△ △ △ ·	△ – – ·
		10	△ △ △ ·	▼ – – ·
		20	△ – △ ·	– – – ·
npm-nVAR	Sphere	2		– – – ·
		5		– – ▼ ·
		10		– ▼ ▼ ·
		20		– ▼ ▼ ·
	Rosenbrock	2		– ▼ – ·
		5		– – ▼ ·
		10		– – – ·
		20		– – – ·
	Rastrigin	2		▼ – – ·
		5		▼ ▼ ▼ ·
		10		▼ ▼ ▼ ·
		20		▼ – ▼ ·

Table 9.2: DSB: comparison of re-initialization strategies for `ar`

Alg.	Bmk.	Dim.	ar-pDEV
ar-pRND	Sphere	2	▼ – – –
		5	▼ – – –
		10	– – – –
		20	▼ – – –
	Rosenbrock	2	▼ – – –
		5	▼ – – –
		10	▼ – – –
		20	▼ ▼ – –
	Rastrigin	2	▼ – – –
		5	– – – –
		10	– – – –
		20	– – – –

Table 9.3: DSB: comparison of re-initialization strategies for `tcn`

Alg.	Bmk.	Dim.	tcn-pDEV
tcn-pRND	Sphere	2	▼ ▼ ▼ –
		5	▼ – – –
		10	▼ ▼ ▼ –
		20	▼ ▼ ▼ –
	Rosenbrock	2	▼ ▼ ▼ –
		5	▼ ▼ – –
		10	▼ ▼ ▼ –
		20	▼ – – –
	Rastrigin	2	▼ ▼ ▼ –
		5	▼ – – –
		10	▼ – – –
		20	▼ – – –

(`pRND`, `pDEV`) that do not utilize the uncertainty estimation of the Kalman filter (Table 9.4). In contrast, `pUNC`, also considering predictive uncertainty, frequently outperforms the other strategies. This shows the advantage of predictive uncertainty for the ES. Whether the predictive uncertainty can be utilized by the ES obviously depends on the choice of re-initialization strategy. Even with `unc`, `pKAL` is worse than all other strategies (Table 9.5). For `unc`, `pRND` and `pUNC` seem to have nearly equal results while `pDEV` is slightly worse than these two. Since the intention of `unc` was to examine the influence of predictive uncertainty, we choose `pUNC` for `unc` in the following analysis although the results so far would also justify the choice of `pRND`.

Comparison of Prediction Methods

After having identified the best re-initialization strategy for each prediction method, we compare these ES variants in the following, see Table 9.6. In Appendix D.3, tables with all pairwise comparisons can be found (Tables D.22, D.23, D.24).

First, we consider the prediction-based algorithms. Algorithm `ar-pRND`

Table 9.4: DSB: comparison of reinitialization strategies for `kal`

Alg.	Bmk.	Dim.	kal-pDEV	kal-pUNC	kal-pKAL
kal-pRND	Sphere	2	▼ – – –	△ – – ▼	▼ – ▼ ▼
		5	▼ ▼ ▼ –	△ △ ▼ –	▼ ▼ ▼ –
		10	▼ – ▼ △	△ △ ▼ △	▼ ▼ ▼ ▼
		20	▼ – ▼ –	△ △ ▼ –	▼ ▼ ▼ –
	Rosenbrock	2	▼ – – –	△ △ – –	▼ ▼ ▼ ▼
		5	▼ – – –	△ △ – –	▼ ▼ ▼ ▼
		10	▼ ▼ ▼ –	△ △ ▼ –	▼ ▼ ▼ –
		20	▼ ▼ ▼ –	△ – ▼ –	▼ ▼ ▼ △
	Rastrigin	2	▼ – – –	△ △ – –	▼ ▼ ▼ –
		5	▼ – – –	– – – –	▼ – – –
		10	– – – –	– – – –	▼ – – –
		20	△ △ △ △	– – △ –	– – – △
kal-pDEV	Sphere	2		△ – – –	▼ ▼ ▼ –
		5		△ △ – –	▼ ▼ ▼ –
		10		△ △ – –	▼ ▼ ▼ ▼
		20		△ △ – –	▼ ▼ ▼ –
	Rosenbrock	2		△ △ – –	▼ ▼ ▼ ▼
		5		△ △ – –	▼ ▼ ▼ –
		10		△ △ – –	▼ ▼ ▼ –
		20		△ △ – –	▼ – ▼ –
	Rastrigin	2		△ △ – –	▼ ▼ ▼ –
		5		△ – – –	▼ – – –
		10		– – – –	▼ – – △
		20		▼ ▼ – ▼	▼ ▼ ▼ –
kal-pUNC	Sphere	2			▼ ▼ ▼ –
		5			▼ ▼ ▼ –
		10			▼ ▼ ▼ ▼
		20			▼ ▼ ▼ –
	Rosenbrock	2			▼ ▼ ▼ ▼
		5			▼ ▼ ▼ –
		10			▼ ▼ ▼ –
		20			▼ ▼ ▼ △
	Rastrigin	2			▼ ▼ ▼ –
		5			▼ – – –
		10			▼ – – –
		20			– – ▼ △

Table 9.5: DSB: comparison of reinitialization strategies for `unc`

Alg.	Bmk.	Dim.	unc-pDEV	unc-pUNC	unc-pKAL
unc-pRND	Sphere	2	△ △ △ △	△ △ △ △	– – – △
		5	▼ ▼ ▼ –	△ – △ –	▼ ▼ ▼ ▼
		10	▼ ▼ ▼ ▼	▼ – – –	▼ ▼ ▼ ▼
		20	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼
	Rosenbrock	2	▼ ▼ ▼ –	△ △ △ –	▼ ▼ ▼ –
		5	▼ ▼ ▼ –	– – – –	▼ ▼ ▼ –
		10	▼ ▼ ▼ –	▼ ▼ ▼ –	▼ ▼ ▼ –
		20	▼ – ▼ –	▼ ▼ ▼ –	▼ ▼ ▼ –
	Rastrigin	2	– – – –	△ △ △ –	▼ ▼ ▼ –
		5	▼ – – –	▼ – – –	▼ ▼ ▼ –
		10	– – – △	▼ – ▼ –	▼ ▼ ▼ –
		20	– – – △	▼ – – △	▼ – ▼ △
unc-pDEV	Sphere	2		△ △ △ △	▼ ▼ ▼ –
		5		△ △ △ –	▼ ▼ ▼ ▼
		10		– – – –	▼ ▼ ▼ ▼
		20		▼ – ▼ ▼	▼ ▼ ▼ ▼
	Rosenbrock	2		△ △ △ –	▼ ▼ ▼ –
		5		△ △ △ –	▼ ▼ ▼ –
		10		– – – –	▼ ▼ ▼ –
		20		– ▼ ▼ –	▼ ▼ ▼ –
	Rastrigin	2		△ △ △ –	▼ ▼ ▼ –
		5		△ – – –	▼ ▼ ▼ –
		10		▼ ▼ ▼ –	▼ ▼ ▼ –
		20		– – – –	▼ – – –
unc-pUNC	Sphere	2			▼ ▼ ▼ ▼
		5			▼ ▼ ▼ ▼
		10			▼ ▼ ▼ ▼
		20			▼ ▼ ▼ –
	Rosenbrock	2			▼ ▼ ▼ ▼
		5			▼ ▼ ▼ –
		10			▼ ▼ ▼ –
		20			▼ ▼ ▼ –
	Rastrigin	2			▼ ▼ ▼ –
		5			▼ ▼ ▼ –
		10			▼ – ▼ –
		20			▼ – – –

is almost always outperformed by the other approaches, and `tcn-pRND` is superior to `kal-pUNC` although it does not take into account predictive uncertainty. The reason for this is that TCNs are better suited to these kinds of dynamics than linear models like Kalman filters and autoregressive models. This is confirmed by the results on the MPB benchmark, see Paragraph 9.4.2. The comparison of `tcn-pRND` and `unc-pUNC` shows that for TCN-based ES predictive uncertainty has a positive effect especially on lower-dimensional functions with few local optima.

No prediction, i.e., `npm-nRND`, often is the worst algorithm except for good RCS values compared to `ar` and `kal`. The good RCS values can be explained as follows. Prediction-based re-initialization leads to a population that is concentrated in a region probably close to the true optimum. Therefore, these algorithms find solutions with better fitness after a fitness function change. But it might be the case that this concentration hampers finding the true optimum, since the population has to be spread again. In contrast, `npm-nRND` distributes the population over the whole solution space so that it might need more generations after a change to find a promising region which leads to worse average fitness ($\overline{\text{BOG}}$). Since it approaches this region probably from different directions, it might be easier at the end to exploit the region causing a good RCS. It is possible that `npm-nRND` has a better RCS but worse $\overline{\text{BOG}}$ because $\overline{\text{BOG}}$ is based on average (good for `ar` and `kal`), while RCS punishes poor fitness values especially in later generations of a change period due to multiplication with the generation number, see Paragraph 6.2.4. Generally, `npm-nRND` does not outperform the TCN-based predictors regarding RCS because, as already described, TCNs are better predictors for this benchmark and probably yield a prediction that is nearer to the true optimum.

Overall, the order of performance reaches from best to worst `tcn-pRND`, `unc-pUNC`, `kal-pUNC`, `ar-pRND`, `npm-nVAR`. The fact that `tcn` outperforms `kal` emphasizes that, in case a prediction approach is not suited to the problem dynamics, even uncertainty estimation cannot compensate the weaknesses of the prediction approach. In addition, a smaller advantage of uncertainty could be observed for multimodal and high dimensional problems.

Influence of Dimensionality

In the previous paragraph it turned out that for TCN-based ES predictive uncertainty might be helpful only in lower-dimensional problems. Therefore, we examine on the Sphere function, up to which dimensionality predictive uncertainty provides useful information both for Kalman filter and TCN.

Table 9.6: DSB: comparison of prediction methods

Alg.	Bmk.	Dim.	ar-pRND	tcn-pRND	kal-pUNC	unc-pUNC
npm-nVAR	Sphere	2	△ – ▼ ·	△ △ △ ·	△ △ ▼ ·	△ △ △ ·
		5	△ – ▼ ·	△ △ △ ·	△ △ ▼ ·	△ △ △ ·
		10	△ ▼ ▼ ·	△ △ △ ·	△ △ – ·	△ △ △ ·
		20	△ △ – ·	△ △ △ ·	△ △ – ·	△ △ △ ·
	Rosenbrock	2	△ – ▼ ·	△ △ △ ·	△ △ – ·	△ △ △ ·
		5	△ – ▼ ·	△ △ △ ·	△ △ ▼ ·	△ △ △ ·
		10	△ △ ▼ ·	△ △ △ ·	△ △ – ·	△ △ △ ·
		20	△ △ ▼ ·	△ △ △ ·	△ △ ▼ ·	△ △ – ·
	Rastrigin	2	– – ▼ ·	△ △ △ ·	△ △ – ·	△ △ △ ·
		5	– – ▼ ·	△ – – ·	– – ▼ ·	△ – – ·
		10	△ – ▼ ·	△ – – ·	△ – ▼ ·	△ – ▼ ·
		20	△ – – ·	△ △ △ ·	△ ▼ – ·	△ △ – ·
ar-pRND	Sphere	2		△ △ △ △	△ – – △	△ △ △ △
		5		△ △ △ △	△ △ △ △	△ △ △ △
		10		△ △ △ △	△ △ △ △	△ △ △ △
		20		△ △ △ △	△ △ – ▼	△ △ △ △
	Rosenbrock	2		△ △ △ △	△ △ – △	△ △ △ △
		5		△ △ △ △	△ △ △ △	△ △ △ △
		10		△ △ △ △	△ △ △ △	△ △ △ △
		20		△ △ △ △	△ – – ▼	△ △ △ △
	Rastrigin	2		△ △ △ △	△ △ – △	△ △ △ △
		5		△ △ △ △	– – ▼ ▼	△ – △ △
		10		△ – – △	▼ – – ▼	– – – △
		20		△ – △ –	▼ ▼ – ▼	– – – △
tcn-pRND	Sphere	2			▼ ▼ ▼ ▼	△ △ △ △
		5			▼ ▼ ▼ ▼	△ △ △ –
		10			▼ ▼ ▼ ▼	▼ ▼ ▼ –
		20			▼ ▼ ▼ ▼	▼ ▼ ▼ –
	Rosenbrock	2			▼ ▼ ▼ ▼	△ △ △ –
		5			▼ ▼ ▼ ▼	– – △ –
		10			▼ ▼ ▼ ▼	▼ ▼ ▼ –
		20			▼ ▼ ▼ ▼	▼ ▼ ▼ –
	Rastrigin	2			▼ ▼ ▼ ▼	△ △ △ –
		5			▼ ▼ ▼ ▼	▼ – – –
		10			▼ – ▼ ▼	▼ – – –
		20			▼ ▼ ▼ ▼	▼ – – –
kal-pUNC	Sphere	2				△ △ △ △
		5				△ △ △ △
		10				△ △ △ △
		20				△ △ △ △
	Rosenbrock	2				△ △ △ △
		5				△ △ △ △
		10				△ △ △ △
		20				△ △ △ △
	Rastrigin	2				△ △ △ △
		5				△ – △ △
		10				△ – – △
		20				△ △ – △

Table 9.7: DSB (Sphere, various dimensionalities): `kal-pRND` and `kal-pUNC`

Alg.	Bmk.	Dim.	kal-pUNC
kal-pRND	Sphere	2	△ – – ▼
		3	△ – – △
		4	△ △ – △
		5	△ △ △ –
		6	△ △ – –
		7	△ △ – –
		8	△ △ – –
		9	△ △ – –
		10	△ △ △ △
		11	△ △ – –
		12	△ △ – –
		13	△ △ – –
		14	△ △ – –
		15	△ △ – –
		16	△ – – –
		17	△ △ – –
		18	△ △ – –
		19	△ △ – –
		20	△ △ △ –

Table 9.8: DSB (Sphere, various dimensionalities): `unc-pRND` and `unc-pUNC`

Alg.	Bmk.	Dim.	unc-pUNC
unc-pRND	Sphere	2	△ △ △ △
		3	△ △ △ –
		4	△ – △ –
		5	△ – △ –
		6	△ △ △ –
		7	△ △ △ –
		8	– – – –
		9	▼ ▼ ▼ –
		10	▼ – ▼ –
		11	– – ▼ –
		12	▼ ▼ ▼ –
		13	▼ ▼ ▼ –
		14	▼ ▼ ▼ –
		15	▼ ▼ ▼ –
		16	▼ ▼ ▼ –
		17	▼ ▼ ▼ –
		18	▼ ▼ ▼ –
		19	▼ ▼ ▼ –
		20	▼ ▼ ▼ ▼

For the Kalman-based algorithms no dependency between dimensionality and effect of predictive uncertainty can be observed (Table 9.7). Furthermore, RCS seems not to be influenced by the uncertainty estimate. From Table 9.8 it is obvious that `unc` outperforms `tcn` for lower dimensions, and the use of predictive uncertainty decreases with increasing dimensions. From $d = 8$, `unc` has no advantage over `tcn`. The reason for this is that the uncertainty estimate of the TCN becomes very large in high dimensions, while the uncertainty of the Kalman filter remains in the same magnitude. Therefore, in TCN-based ES the population is spread so widely that the optimization is hampered.

In order to scrutinize the effect of uncertainty estimation further, in Figure 9.2 we plot the best fitness achieved in the respective generation averaged over the runs. The algorithms are ES with `kal` and `unc` as prediction model combined with re-initialization strategies `pRND` and `pUNC`. Due to restricted space not all generations are shown. It can be observed that `unc-pUNC` and `kal-pUNC` often start after a change with a lower fitness value than their counterpart without predictive uncertainty. Thus, considering predictive uncertainty for re-initialization often prevents high fitness peaks during the first generations of change periods. After some generations, the algorithms without uncertainty estimation often achieve quite good fitness values as well and sometimes even converge to better solutions.

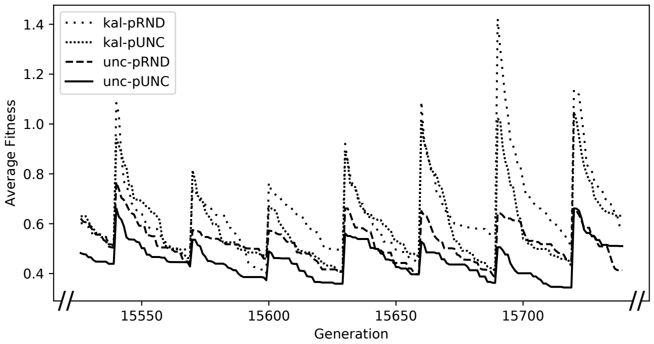


Figure 9.2: Best fitness for some generations on Sphere function ($d = 7$)

9.4.2 Moving Peaks Benchmark

Comparison of Re-Initialization Strategies

Also for MPB we first identify the best re-initialization strategy for each prediction approach. The same combinations as on DSB are best except for `npm`. Overall, `npm` has better results with `nRND` though `nVAR` would be a reasonable choice on high-dimensional problems with low or medium-sized noise (Table 9.9). Prediction method `tcn` again performs better with `pRND` but on problems with high noise the re-initialization strategy has not much influence (Table 9.12). For `ar` and `unc`, the re-initialization strategy seems to be less important than on DSB (Tables 9.10 and 9.13). While `unc` performs with `pRND` and `pUNC` nearly equally, `kal` with `pUNC` clearly outperforms the other re-initialization strategies (Table 9.11). As already motivated in the paragraph on DSB, the reason for this is that Kalman filters are better suited to noisy linear dynamics than TCNs, and proper re-initialization cannot mitigate poor prediction to such a large extent. Interestingly, for both `unc` and `kal` re-initialization strategy `pKAL` performs not as poor as on DSB. This is indicated by the fact that the respective columns for `pKAL` contain more often ‘–’ or ‘△’.

Table 9.9: MPB: comparison of re-initialization strategies for `npm`

Alg.	Noise	Dim.	npm-nVAR	npm-nPRE
npm-nRND	0.00	2	▼ ▼ ▼ ·	▼ ▼ ▼ ·
		5	– – ▼ ·	– – ▼ ·
		10	△ △ △ ·	– – – ·
		20	△ △ △ ·	– – △ ·
	0.01	2	▼ ▼ ▼ ·	▼ ▼ ▼ ·
		5	– – – ·	– – – ·
		10	▼ ▼ ▼ ·	▼ ▼ ▼ ·
		20	△ △ △ ·	– – △ ·
	0.05	2	▼ ▼ ▼ ·	▼ ▼ ▼ ·
		5	▼ ▼ ▼ ·	▼ ▼ ▼ ·
		10	▼ ▼ – ·	▼ ▼ ▼ ·
		20	– – △ ·	– – – ·
npm-nVAR	0.00	2		– – – ·
		5		– – – ·
		10		▼ ▼ ▼ ·
		20		▼ ▼ – ·
	0.01	2		– – – ·
		5		– – – ·
		10		▼ ▼ ▼ ·
		20		▼ ▼ – ·
	0.05	2		– – – ·
		5		– – – ·
		10		– – ▼ ·
		20		– – ▼ ·

Table 9.10: MPB: comparison of re-initialization strategies for `ar`

Alg.	Noise	Dim.	ar-pDEV
ar-pRND	0.00	2	▼ – – –
		5	– – – –
		10	– – – –
		20	▼ ▼ – –
	0.01	2	▼ ▼ – –
		5	– – – –
		10	– – – –
		20	▼ ▼ – –
	0.05	2	– – – –
		5	– – – –
		10	– – – –
		20	– – – –

Table 9.11: MPB: comparison of re-initialization strategies for `kal`

Alg.	Noise	Dim.	kal-pDEV	kal-pUNC	kal-pKAL
kal-pRND	0.00	2	– – – –	– – – –	– – – –
		5	– – – –	– – – –	– – – –
		10	▼ ▼ – –	△ △ △ –	▼ ▼ △ –
		20	▼ ▼ – –	△ △ △ –	▼ ▼ – –
	0.01	2	▼ ▼ – ▼	△ △ △ △	▼ ▼ △ –
		5	– – – –	– – – –	– – – –
		10	– – – –	– – – ▼	– – – –
		20	▼ ▼ – –	△ △ △ –	– – – –
	0.05	2	– – – –	– – – –	△ △ – △
		5	▼ ▼ – –	△ △ – –	▼ ▼ – –
		10	– – – –	– – – –	– – – –
		20	– – – –	– – △ –	– – – –
kal-pDEV	0.00	2		– – – –	– – – –
		5		– – – –	– – – –
		10		△ △ △ –	△ – △ –
		20		△ △ △ –	– – – –
	0.01	2		△ △ △ △	△ △ △ –
		5		– – – –	– – – –
		10		– – – ▼	– – – –
		20		△ △ △ –	– – – –
	0.05	2		– – – –	△ △ – △
		5		△ △ △ –	△ △ △ –
		10		△ △ – –	– – – –
		20		△ △ △ –	– – – –
kal-pUNC	0.00	2			– – – –
		5			– – – –
		10			▼ ▼ ▼ –
		20			▼ ▼ ▼ –
	0.01	2			▼ ▼ – ▼
		5			– – – –
		10			– – – △
		20			▼ ▼ ▼ –
	0.05	2			△ △ – △
		5			▼ ▼ – –
		10			– – – –
		20			▼ ▼ ▼ –

Table 9.12: MPB: comparison of re-initialization strategies for `tcn`

Alg.	Noise	Dim.	tcn-pDEV
tcn-pRND	0.00	2	– – – –
		5	– – – –
		10	▼ ▼ ▼ –
		20	▼ ▼ ▼ –
	0.01	2	▼ ▼ ▼ –
		5	▼ ▼ – –
		10	– – – –
		20	▼ ▼ ▼ –
	0.05	2	– – – –
		5	▼ ▼ – –
		10	– – – –
		20	– – ▼ –

Table 9.13: MPB: comparison of re-initialization strategies for `unc`

Alg.	Noise	Dim.	unc-pDEV	unc-pUNC	unc-pKAL
unc-pRND	0.00	2	– – – –	– – – –	– – – –
		5	– – – –	– – – –	– – – –
		10	▼ ▼ ▼ –	– – – –	▼ ▼ – –
		20	▼ ▼ ▼ –	– – – –	▼ ▼ – –
	0.01	2	▼ ▼ ▼ –	– △ – –	▼ ▼ – –
		5	– – – –	– – – –	– – – –
		10	– – – –	– – – –	– – – –
		20	– – ▼ –	– – – –	– – – –
	0.05	2	– – – –	– – – –	△ △ – –
		5	▼ ▼ – –	– – – –	▼ ▼ – –
		10	– – – –	– – – –	– – – –
		20	– – ▼ –	– – – –	– – – –
unc-pDEV	0.00	2		– – – –	– – – –
		5		– – – –	– – – –
		10		△ △ △ –	– – △ –
		20		△ △ △ –	– – △ –
	0.01	2		△ △ △ –	– – △ –
		5		– – – –	– – – –
		10		– – – –	– – – –
		20		– – △ –	– – △ –
	0.05	2		– – – –	△ △ – △
		5		△ △ – –	– – – –
		10		– – – –	– – – –
		20		– – △ –	– – △ –
unc-pUNC	0.00	2			– – – –
		5			– – – –
		10			▼ ▼ – –
		20			▼ ▼ – –
	0.01	2			▼ ▼ – –
		5			– – – –
		10			– – – –
		20			– – – –
	0.05	2			– △ – △
		5			▼ ▼ – –
		10			– – – –
		20			– – – –

Comparison of Prediction Methods

No prediction (`npm`) outperforms the other approaches frequently regarding BEBC (Table 9.14). The reason for this might be that the prediction-based approaches follow a local optimum but not the global one. Therefore, they have lower fitness values at the beginning of the change period resulting in lower $\overline{\text{BOG}}$ and RCS. In contrast to that, `npm` shows more diversity in the population. Hence, it is more likely to explore the global optimum leading to a better BEBC. On the multimodal Rastrigin function these effects do not appear, see Table 9.6. Possibly, this can be explained by the fitness landscape. MPB is rather flat with some small basins of attraction whereas Rastrigin exhibits strong slopes everywhere enabling the ES to find promising directions in the solution space. Therefore, `npm` might find good solutions faster, and the prediction-based approaches could easier leave their tracked local optimum. This might lead to a better RCS for `npm`, and to a better BEBC for the prediction-based approaches on Rastrigin compared to MPB.

Prediction methods `ar`, `tcn`, and `unc` exhibit only marginal differences on MPB and often perform worse than `kal` showing Kalman filters' superiority for problems with linear dynamics. The fact that no differences between `tcn` and `unc` exist confirms the previous finding that uncertainty-based re-initialization does not support the optimizer if the predictor is not good enough (Paragraph 9.4.1). The overall order of performance on MPB is from best to worst `kal`, `unc` & `tcn`, `ar`, `npm`.

9.5 Summary

In this chapter, we proposed a new re-initialization strategy (`pUNC`) to consider predictive uncertainty for population re-initialization. We applied a temporal convolutional network (TCN) with Monte Carlo dropout as new prediction model with uncertainty estimation for dynamic optimization.

The results demonstrate the advantage of TCNs with uncertainty estimation on rather complex problems whereas Kalman filters are superior on noisy linear problem dynamics. Our new re-initialization strategy turned out to outperform the existing one that considers predictive uncertainty (`pKAL`). We could show that predictive uncertainty only has a positive effect on the ES if the prediction approach is combined with the proper re-initialization strategy, and, vice versa, that uncertainty-based re-initialization can not mitigate poor prediction. In general, the positive effect of predictive uncertainty vanishes with increasing problem dimensionality.

Table 9.14: MPB: comparison of prediction methods

Alg.	Noise	Dim.	ar-pRND	tcn-pRND	kal-pUNC	unc-pUNC
npm-nRND	0.00	2	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
		5	– – – ·	– – – ·	– – △ ·	– – △ ·
		10	– ▼ △ ·	– – △ ·	△ – △ ·	– – △ ·
		20	– ▼ – ·	– ▼ △ ·	△ ▼ △ ·	– ▼ △ ·
	0.01	2	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
		5	△ – – ·	△ – – ·	– – – ·	– – – ·
		10	– ▼ – ·	– ▼ – ·	– ▼ – ·	– ▼ – ·
		20	△ ▼ △ ·	△ ▼ △ ·	△ ▼ △ ·	△ ▼ △ ·
	0.05	2	△ ▼ – ·	△ ▼ – ·	△ ▼ – ·	△ ▼ – ·
		5	△ – – ·	△ – – ·	△ – △ ·	△ – – ·
		10	– ▼ – ·	– ▼ – ·	– ▼ – ·	– ▼ – ·
		20	– ▼ – ·	– ▼ – ·	– ▼ – ·	– ▼ – ·
ar-pRND	0.00	2		– – – –	– – – △	– – – –
		5		– – – –	– – △ –	– – △ –
		10		– – – –	△ △ △ –	– – – –
		20		– – – –	△ △ △ –	– – – –
	0.01	2		△ △ △ –	△ △ △ △	△ △ △ –
		5		△ – – –	△ – – –	– – – –
		10		– – – –	– – – –	– – – –
		20		– – – –	– – △ –	– – – –
	0.05	2		– – – –	– – – –	– – – –
		5		△ – – –	△ △ △ △	△ △ – –
		10		– – – –	– – – –	– – – –
		20		– – – –	– – △ –	– – – –
tcn-pRND	0.00	2			△ – – △	– – – –
		5			– – △ –	– – – –
		10			△ △ △ –	– – – –
		20			△ △ △ –	– – – –
	0.01	2			△ △ △ △	– – – –
		5			– – – –	– – – –
		10			– – – –	– – – –
		20			– – △ –	– – – –
	0.05	2			– – – –	– – – –
		5			△ △ △ △	– – – –
		10			– – – –	– – – –
		20			– – △ –	– – – –
kal-pUNC	0.00	2				▼ – – –
		5				– – – –
		10				▼ ▼ ▼ –
		20				▼ ▼ ▼ –
	0.01	2				▼ ▼ ▼ ▼
		5				– – – –
		10				– – – –
		20				– – ▼ –
	0.05	2				– – – –
		5				▼ ▼ ▼ ▼
		10				▼ ▼ – –
		20				– – – –

Part III

Summary

10 Conclusion

Dynamic optimization problems typically appear in real-world systems underlying environmental influence. Solving this kind of problems requires algorithms considering relationships between problem instances at different points in time. Nature-inspired optimization is frequently used to solve dynamic optimization problems as they store information about past environments in the population by a natural means. In order to circumvent the premature convergence of nature-inspired meta-heuristics in dynamic problems, prediction is one among other approaches.

This thesis investigated prediction-based nature-inspired optimization and neural networks as prediction methods. The main contributions are summarized in the following. Furthermore, we depict their transferability to other kinds of optimization algorithms and dynamic optimization problems.

10.1 Contributions

Prediction for ES We equipped an ES with a recurrent neural network and incorporated the predicted optimum as individual into the population. In our experimental study, prediction-based ES outperform the simple ES without prediction. Depending on the problem dynamics, either RNN or autoregressive prediction is better. The results show that neural networks are applicable as predictors in dynamic optimization despite their specific requirements regarding amount of training data and training time. Neural networks extend the pool of applicable prediction techniques for dynamic optimization covering types of dynamics that are difficult to model by other techniques. Hybridizing RNN and autoregressive prediction combines advantages of both methods leading to overall best performance.

Prediction for PSO While work on prediction for dynamic ES is rather mature, for PSO in dynamic optimization only two approaches exist that adapt the swarm according to the predicted optimum. In contrast, we focused on extending the PSO function by the predicted optimum as third attractor. Our results show that this approach leads to better performance than inserting the prediction as individual into the swarm.

Predictive Uncertainty for ES If the predicted optimum is located far from the true one, prediction would mislead the optimizer and might drastically deteriorate optimization performance. To cope with that, we constructed a new re-initialization strategy taking into account predictive uncertainty. After a change, it samples the new population from intervals of a normal distribution that is centered to the predicted optimum, while the intervals' spread is based on predictive uncertainty. The higher the uncertainty, the wider the population is distributed in the solution space increasing the optimizer's exploration abilities. Our re-initialization strategy lowers fitness peaks after fitness function changes and outperforms the already available uncertainty-based re-initialization strategy. Its benefit depends on the quality of the predictor's uncertainty estimate.

Dynamic Sine Benchmark (DSB) We proposed DSB as new benchmark generator equipping static fitness functions with randomized trigonometric movement. It provides the configurable parameters velocity and curviness that determine problem difficulty for the optimizer and the predictor, respectively. In contrast to other benchmark generators for dynamic optimization, DSB constructs predictable dynamics with difficulty that is appropriately balanced between too easy (linear) and unpredictable (random) dynamics. Therefore, DSB is well suited to evaluate prediction-based optimization algorithms.

Relative Convergence Speed (RCS) RCS is our new measure to evaluate the convergence properties of optimization algorithms. It takes into account how fast an algorithm approaches the global optimum fitness. By this means, the behavior of prediction-based approaches that typically begin a change period on a lower fitness level than other algorithms is better represented than with existing measures.

10.2 Transferability

We discuss whether prediction and prediction with uncertainty estimation can be used in other types of optimization algorithms than those described in Chapter 3. Then, we show which modifications are necessary to apply our approaches to kinds of dynamic optimization problems that are not covered by this thesis. Finally, applicability to use cases with available domain knowledge is depicted.

10.2.1 Prediction for Other Optimization Algorithms

In optimization algorithms solving the problem analytically, prediction is not necessary, since they already are able to find the global optimum. Moreover, it would be hard or even impossible to integrate an approximated optimum in an exact solution method. Neither exploration-based algorithms are able to benefit from prediction. Even if they started evaluating candidate solutions close to the prediction, they would have to enumerate the remaining solution space in order to possibly find better solutions.

Local optimizers could be supported by prediction to choose an appropriate starting point. If the optimizer starts near the global optimum without local optima in between, they are likely to converge to the global one. Global methods, i.e., blind and guided random search, could benefit from prediction as well. Blind random search could start at the predicted position. If the algorithm is stopped after some time, it is likely to have found a better solution than if the search would have started somewhere else. Guided random search, comprising meta-heuristics, are suited for incorporating prediction because the prediction could easily be introduced as candidate solution, like done in this thesis. Prediction in meta-heuristics can also be combined with other specialized techniques. For example, in multi-population approaches different local optima are tracked at the same time leading to a separate time series for each one. Here, it has to be considered whether the same prediction model is trained on the time series of all peaks, or whether a separate model is instantiated for each peak.

Local and global optimizers could employ predictive uncertainty as well. It could be used to adapt parameters of the optimizer that influence where new solutions in the next iteration should be generated. Such a parameter could be the step size in Newton's method or in nature-inspired techniques.

10.2.2 Prediction for Other Optimization Problems

In Paragraph 2.3, we restricted this thesis to certain kinds of optimization problems. Here, we consider whether prediction can be used in nature-inspired optimizers to solve problems other than those.

In dynamic constrained or noisy problems, the prediction mechanism does not need modifications as the fitness and constrained functions are considered as black-box. Even if specialized search mechanisms circumventing the individual challenges in constrained and noisy optimization are applied, the optimizer still produces a time-series of optimum positions that the prediction could rely on.

In multi-objective optimization, various conflicting objectives are optimized, e.g., with sophisticated algorithms like NSGA-II [Kra17], leading to a Pareto set of non-comparable solutions instead of one solution in single-objective optimization. Thus, not a single optimum has to be predicted but the Pareto set. Therefore, similar considerations as for multi-population optimizers have to be done, for example, whether for each solution in the Pareto set a separate predictor is employed.

Another kind of problems are changing number of dimensions which would affect the individuals' representation. Since prediction methods usually expect input in a defined format, preprocessing the time series for the predictor would be necessary. Missing dimensions could be padded with zero values if the minimum and maximum number of dimensions is known.

In time-linkage problems and problems with robust optimization as goal, the optimum prediction could be done as usual. Nevertheless, better solutions might be found by the optimizer if the fitness value of solutions in the next time step(s) is predicted as described in Paragraph 4.2. By this means, the optimizer could also take into account the future performance of a solution and thus optimize the average fitness of solutions. This would involve a completely different prediction mechanism because the data would form a mapping from a position to a fitness for different points in time.

10.2.3 Prediction for Application Problems

As common practice in the optimization domain, we evaluated our methods on generic artificial benchmark sets. Since nature-inspired optimization techniques handle the problem at hand as black-box, they are applicable to a wide range of problems. As motivated in Chapter 3, the operators and parameters of meta-heuristic optimization algorithms have to be chosen problem-dependent to achieve best performance on a certain application. Even with our extensions, the optimizers are adjustable to application problems. Utilizing domain knowledge does not only support refining the optimizer's operators and parameters but also choosing an appropriate predictor and employing the prediction in the best way. Since neither of our extensions depends on a specific prediction method, the predictor is exchangeable. If the type of dynamics, e.g., linear or recurrent, is known for a certain application problem, a matching prediction method could be chosen.

11 Outlook

This thesis presented the application of neural network prediction to ES, extensions of PSO with prediction and improvement of optimization performance by predictive uncertainty. It leaves space for further investigation; two possible research directions are described in the following.

PSO with Predictive Uncertainty We considered predictive uncertainty in ES by adapting the re-initialization operator, see Chapter 9. In PSO, this approach could be applied as well if the predicted optimum would be incorporated as particle into the swarm. However, employing the prediction as third attractor turned out to be more successful, see Chapter 8. An approach to take into account predictive uncertainty in this setting would be to adapt parameters ω, θ_1, θ_2 and θ_3 of the velocity function (Equation (8.1)) according to the uncertainty, since these parameters influence the intensity of exploitation and exploration as described in Paragraph 3.4.

In case of a large predictive uncertainty, the influence of the prediction should be reduced by decreasing θ_3, while inertia weight ω and θ_1 could be increased to support exploration. Because the parameter adaptation is prone to lead to divergent behavior of the PSO, it has to be done carefully considering existing proofs for convergent parameter values as described in Paragraph 8.2.5. An automatic adaptation depending on the uncertainty and subject to the convergence bounds would be desired.

CMA-ES with Prediction Covariance matrix adaptation evolution strategy (CMA-ES) is the most famous variant of ES with a specialized mutation operator. Different to ES that employ a univariate normal distribution for generating offspring individuals, see Equation (3.3), CMA-ES samples a multivariate normal distribution to create an offspring individual $\mathbf{x}'$ in generation t:

$$\mathbf{x}' = \mathbf{m}_t + s_t \cdot \mathcal{N}\left(\mathbf{0}, \mathbf{C}_t\right) \tag{11.1}$$

The weighted mean of the population is denoted by $\mathbf{m}_t$, while s_t is the step size and $\mathbf{C}_t$ the covariance matrix. CMA-ES consists of sophisticated mechanisms to adapt step size and covariance in order to compute successful offspring individuals. So far, two attempts have been made to apply CMA-ES to dynamic optimization problems. On the one hand, the static CMA-ES is employed without adaptations [Bou05; AL12; NCP16]. This approach is called `statCMA` in the following. On the other hand, after a fitness function change the step size is set according to the estimated shift severity, while internal variables belonging to the adaptation mechanism of covariance matrix and step size are reset to their initial values [Yaz+19]. We refer to this strategy with `yazCMA`. As far as we know, no prediction approaches have been incorporated into CMA-ES so far.

The most obvious approach to equip CMA-ES with prediction, here called `predCMA`, is to reset all variables of CMA-ES to their initial values and to modify the sampling after a fitness function change to:

$$\mathbf{x}' = \hat{\mathbf{o}}_c + \sqrt{\hat{\mathbf{u}}_c} \cdot \mathcal{N}\left(\mathbf{0}, \mathbf{I}\right) \tag{11.2}$$

By this means, the predicted optimum $\hat{\mathbf{o}}_c$ and the predictive uncertainty $\hat{\mathbf{u}}_c$ are used similar to our re-initialization strategy `pUNC`, see Paragraph 9.2.

Preliminary experiments show superiority of our approach to both existing ones, see Table 11.1. The structure of the tables is the same as in Paragraph 9.4. We use three DSB instances with $C = 10$ and $V \in \{0.5, 2, 5\}$. The `statCMA` approach shows strength in high-dimensional problems, while `predCMA` outperforms the other approaches on problems with lower dimensions. With increasing velocity, `predCMA` becomes better in relation to `yazCMA`.

There is still potential to improve our approach. Open questions are, for example, how to set the covariance matrix and other parameters of the CMA-ES after a change, or whether a better choice for the step size could be done. Another idea is to pull the population in direction of the prediction in each generation. This could be realized by a mechanism similar to the one we proposed with `pred3` for PSO (Chapter 8).

Table 11.1: CMA-ES variants on DSB. From left to right $V \in \{0.5, 2, 5\}$

Alg.	Bmk.	Dim.	`yazCMA`	`predCMA`
`statCMA`	Sphere	2	△ △ △ ·	△ △ △ ·
		5	△ △ △ ·	△ △ △ ·
		10	▼ ▼ ▼ ·	▼ ▼ ▼ ·
		20	▼ ▼ ▼ ·	▼ ▼ ▼ ·
	Rosenbrock	2	△ △ △ ·	△ △ △ ·
		5	△ △ – ·	△ △ – ·
		10	▼ ▼ – ·	▼ ▼ ▼ ·
		20	▼ ▼ ▼ ·	▼ ▼ ▼ ·
	Rastrigin	2	△ △ △ ·	△ △ △ ·
		5	△ △ △ ·	△ △ △ ·
		10	△ △ △ ·	△ △ △ ·
		20	△ △ △ ·	▼ △ △ ·
`yazCMA`	Sphere	2		△ △ △ ·
		5		△ △ △ ·
		10		▼ ▼ ▼ ·
		20		▼ ▼ ▼ ·
	Rosenbrock	2		– △ – ·
		5		▼ ▼ – ·
		10		▼ ▼ ▼ ·
		20		– ▼ – ·
	Rastrigin	2		△ △ △ ·
		5		– – – ·
		10		▼ ▼ – ·
		20		▼ ▼ ▼ ·

Alg.	Bmk.	Dim.	`yazCMA`	`predCMA`
`statCMA`	Sphere	2	△ △ △ ·	△ △ △ ·
		5	△ △ △ ·	△ △ △ ·
		10	▼ ▼ ▼ ·	▼ ▼ ▼ ·
		20	▼ ▼ ▼ ·	▼ ▼ ▼ ·
	Rosenbrock	2	△ △ △ ·	△ △ △ ·
		5	▼ – ▼ ·	▼ △ – ·
		10	▼ ▼ ▼ ·	▼ ▼ ▼ ·
		20	▼ ▼ ▼ ·	▼ ▼ ▼ ·
	Rastrigin	2	△ △ △ ·	△ △ △ ·
		5	△ △ △ ·	△ △ △ ·
		10	△ △ △ ·	△ △ △ ·
		20	▼ ▼ ▼ ·	▼ ▼ ▼ ·
`yazCMA`	Sphere	2		△ △ △ ·
		5		△ – ▼ ·
		10		△ – △ ·
		20		△ △ ▼ ·
	Rosenbrock	2		– △ – ·
		5		– △ △ ·
		10		△ △ △ ·
		20		△ △ – ·
	Rastrigin	2		△ ▼ △ ·
		5		– – ▼ ·
		10		▼ ▼ – ·
		20		△ △ ▼ ·

Alg.	Bmk.	Dim.	`yazCMA`	`predCMA`
`statCMA`	Sphere	2	△ △ – ·	△ △ △ ·
		5	△ △ △ ·	△ △ △ ·
		10	▼ ▼ – ·	– – △ ·
		20	▼ ▼ ▼ ·	▼ ▼ ▼ ·
	Rosenbrock	2	△ △ – ·	△ △ △ ·
		5	– – ▼ ·	– △ – ·
		10	▼ ▼ – ·	▼ ▼ – ·
		20	▼ ▼ ▼ ·	▼ ▼ ▼ ·
	Rastrigin	2	△ △ – ·	△ △ △ ·
		5	△ △ △ ·	△ △ △ ·
		10	▼ ▼ – ·	▼ ▼ ▼ ·
		20	▼ ▼ ▼ ·	▼ ▼ ▼ ·
`yazCMA`	Sphere	2		△ – △ ·
		5		△ – △ ·
		10		△ △ △ ·
		20		△ △ △ ·
	Rosenbrock	2		△ – △ ·
		5		△ △ – ·
		10		△ △ – ·
		20		△ △ △ ·
	Rastrigin	2		△ – △ ·
		5		△ △ △ ·
		10		△ △ ▼ ·
		20		△ △ △ ·

Part IV

Appendix

A Fitness Functions

For static nature-inspired optimization, a set of fitness functions exists that frequently are applied to benchmark optimization algorithms, see [Kra08; LQS13] for a comprehensive list of functions. Since the functions Sphere, Rosenbrock and Rastrigin are used in this thesis, we depict their mathematical description and visualize their fitness landscape for a two-dimensional solution space. For all functions holds $\mathbf{x} \in \mathbb{R}^d$.

Sphere

$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i^2 \tag{A.1}$$

Rosenbrock

$$f(\mathbf{x}) = \sum_{i=1}^{d-1} \left((100 \cdot (x_i^2 - x_{i+1})^2 + (x_i - 1)^2\right) \tag{A.2}$$

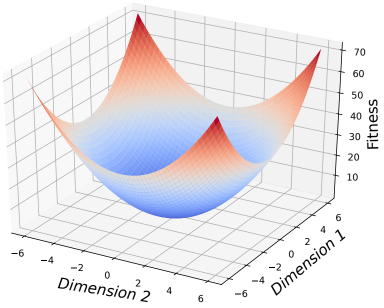


Figure A.1: Sphere function

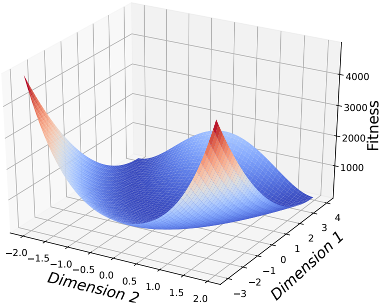


Figure A.2: Rosenbrock function

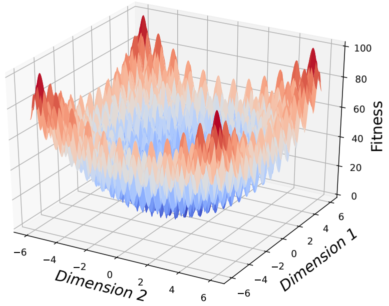


Figure A.3: Rastrigin function

Rastrigin

$$f(\mathbf{x}) = \sum_{i=1}^{d} \left(x_i^2 - 10 \cdot \cos(2\pi x_i) + 10\right) \tag{A.3}$$

B Examination of RCS and ARR

We examine the differences between ARR and RCS by means of a simplified optimization scenario where three discrete fitness values are possible (0,1, and 2) with zero as best fitness, and the optimization is conducted for one change period with two generations. Therefore, 3^2 possible optimization runs exist for one algorithm. For two algorithms, called A and B, there are $\binom{9}{2} = 36$ possible combinations disregarding the cases where both algorithms have the same sequence of fitness values. For each of these combinations, Figure B.1 shows the ARR, while Figure B.2 visualizes the RCS. The plots depict for each generation the best fitness value the respective algorithm has. The legend contains the ARR or RCS value, respectively, while the color of the lines for the fitness progress is chosen according to the metric value. The darker the color, the better the metric value. Thus, a dark color represents a high ARR and a low RCS, respectively. Due to the exponentially increasing number of plots, we do not visualize scenarios with more fitness levels or generations. To signify a certain plot, for example the third from left in the second row, we write (2,3).

The most obvious disadvantage of ARR is that negative ARR values are possible though the authors proposing this measure claim that it ranges between zero and one [NY12]. In case the fitness level increases during a change period, the ARR becomes negative, e.g., in plot (5,2). That plot also shows the second limitation of ARR. If an algorithm starts a change period with the best possible fitness, division by zero occurs so that the ARR cannot be computed. In our implementation, we handle that case by setting ARR to one. However, this does not reflect the algorithm's behavior because the algorithm has ARR $= 1$ independent of whether its fitness stays low or increases, see e.g. plot (1,2). The third drawback are unintuitive ARR values in case algorithms start at different fitness levels, and their fitness decreases or stays the same. If the algorithm that started on a lower fitness always has equal or better fitness than the other algorithm, its ARR unexpectedly is not better. This happens in plots (8,1), (8,2), and (6,4). This behavior especially disadvantages prediction-based optimizers, since they might often obtain a lower fitness level at the beginning of change periods than algorithms without prediction because the prediction is supposed to lead them to promising regions in the solution space.

Comparing the results for ARR and RCS, it can be obtained that RCS solves these disadvantages. RCS cannot become negative, since in minimization problems both $f(\mathbf{x}^*_{ct}, c)$ and $f(\mathbf{x}^*_{c_{\text{worst}}}, c)$ always are greater or equal to $f(\mathbf{o}_c, c)$ so that $f(\mathbf{x}^*_{ct}, c) - f(\mathbf{o}_c, c)$ and $f(\mathbf{x}^*_{c_{\text{worst}}}, c) - f(\mathbf{o}_c, c)$ are positive. For maximization, these differences have a negative result, hence RCS is positive as well. Thus $| \cdot |$, used in the original publication [MK18a], is not required. Division by zero takes place in the computation of RCS if all algorithms achieve the best possible fitness in all generations of the respective change period. Then, all algorithms get RCS $= 0$ for that change period. In contrast to ARR's behavior when division by zero occurs, this represents the situation that all algorithms perfectly converge to the global optimum. Furthermore, RCS does not disadvantage prediction-based optimizers as can be observed in plots (8,1), (8,2), and (6,4).

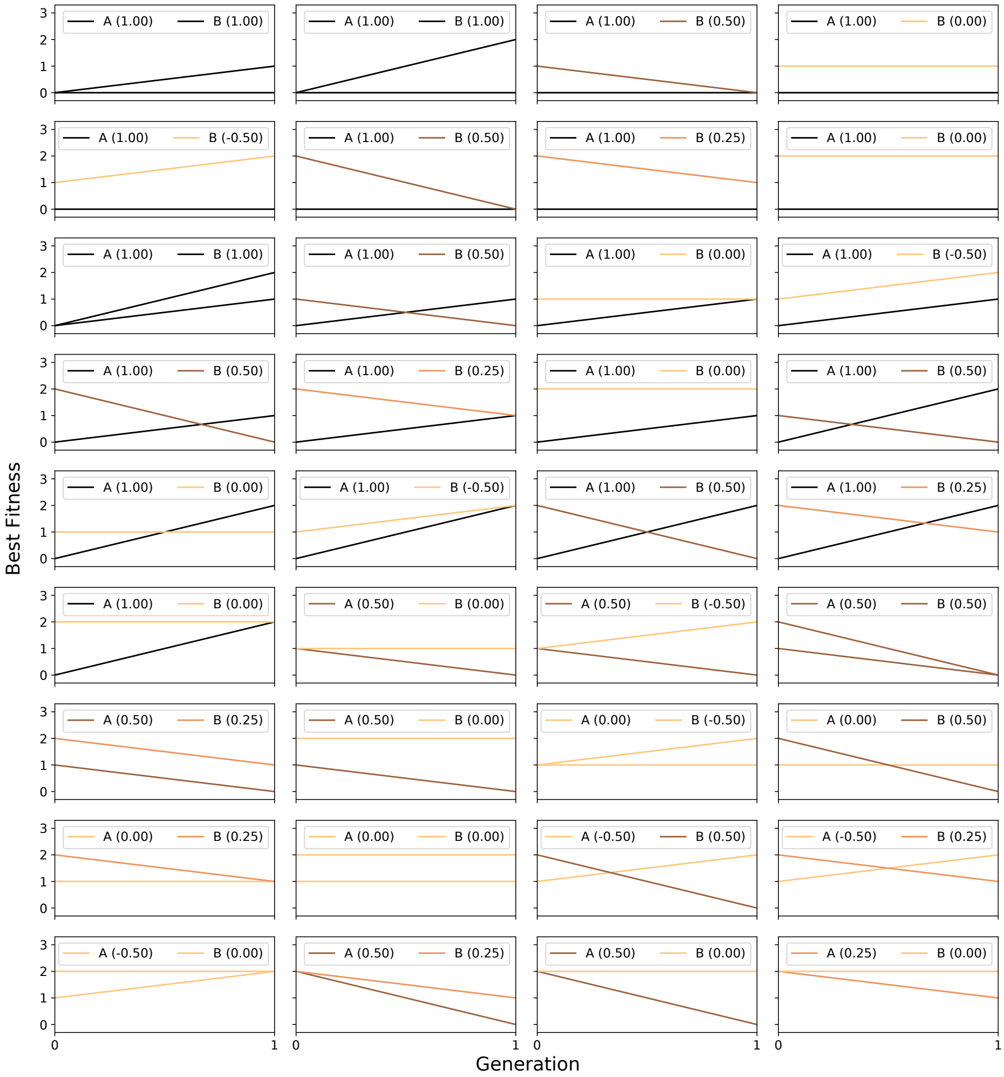


Figure B.1: ARR (in brackets) for all possible combinations of two algorithms (A and B) with three fitness levels $(0, 1, 2)$ and two generations. Fitness plots with better ARR values, i.e., larger ones, have darker colors

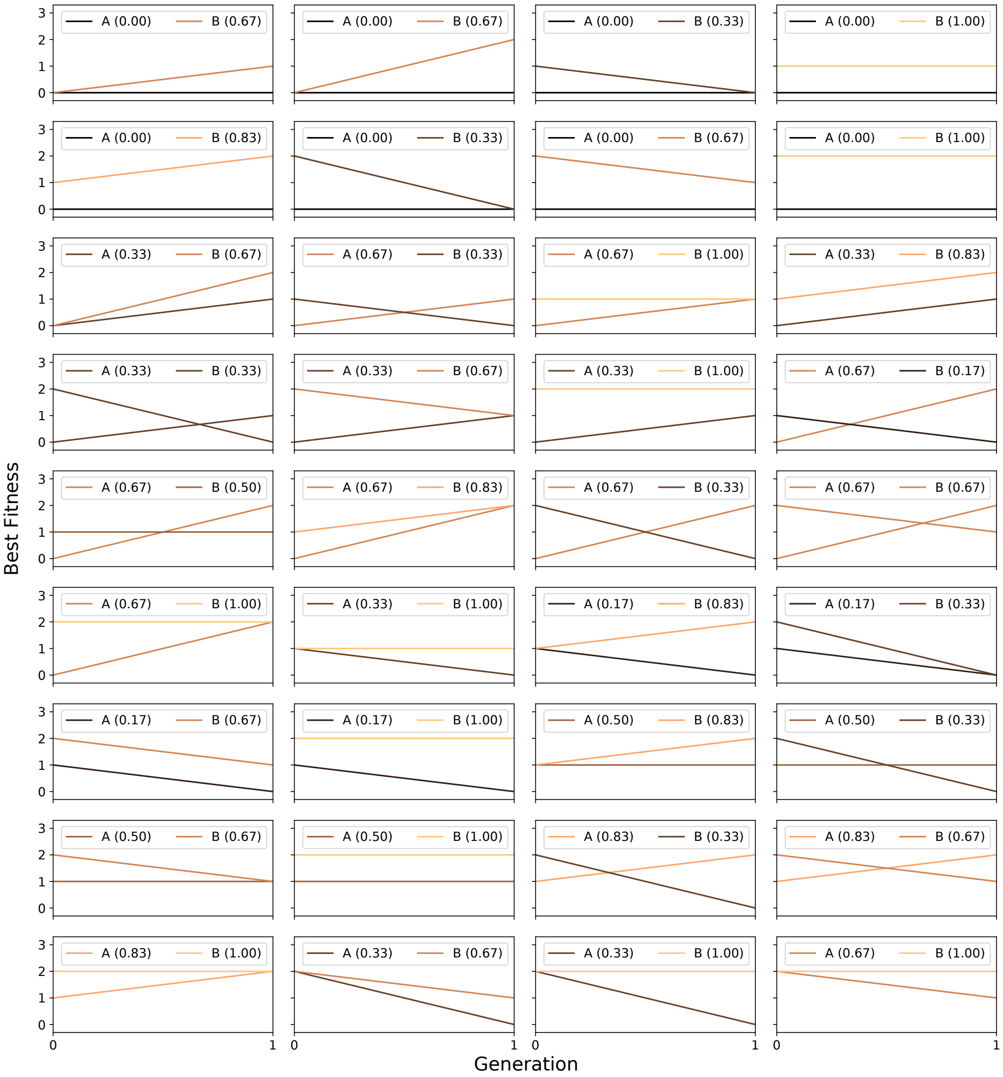


Figure B.2: RCS (in brackets) for all possible combinations of two algorithms (A and B) with three fitness levels $(0, 1, 2)$ and two generations. Fitness plots with better RCS values, i.e., lower ones, have darker colors

C Empirical Comparison of Prediction Methods

As described in Paragraph 7.1.2, the work of [Wol19] compared different time series prediction methods (persistence model, Kalman-Filter, TCN, and LSTM) on simple dynamics. They are called 2DE and Trigo, and are described below. Here, we show results of applying the same methods to the dynamic sine benchmark (DSB). We also re-compute the results on the benchmarks employed in [Wol19] leading to small deviations but the same conclusions.

All time series $\mathbf{S} \in \mathbb{R}^{T \times d}$ are d-variate with independent dimensions, each following a separate univariate time series $\mathbf{s} \in \mathbb{R}^{T}$ with $T = 1000$ time steps. In the noisy setting, the time series are disturbed by multiplicative Gaussian noise with zero mean and standard deviation as listed in Table C.1. The evaluated time series (2DE, Trigo, DSB) are described in the following, visualizations for five-dimensional instances of the time series without noise are given in Figures C.1,C.2, and C.3, where each line represents one dimension.

2DE Second-order differential equation [Wol19]:

$$\begin{aligned} s_t &= s_{t-1} + \alpha_1 \cdot s'_t \\ s'_t &= s'_{t-1} + \alpha_2 \end{aligned}$$

with $s_0 = s'_0 = 0$. Parameters $\alpha_i \in [-1, 1]$ are randomly chosen every 100 steps. For each of the d dimensions of series $\mathbf{S}$ an instance of this series is created.

Trigo Trigonometric function with randomly parameterized additive sine functions [Wol19]:

$$s_t = \iota_1 \sin\left(\frac{t}{T}\beta_1\right) + \iota_2 \sin\left(\frac{t}{T}\beta_2\right)$$

with amplitudes $\iota_1 \in [30, 100]$ and $\iota_2 \in [3, 10]$, and frequencies $\beta_1 \in [5, 25]$ and $\beta_2 \in [50, 250]$. This procedure is repeated for each dimension of series $\mathbf{S}$.

DSB Same parameterization as in Paragraph 9.3, i.e., $C = 10$, $V = 0.5$, $\rho = 4$. Compared to the benchmarks 2DE and Trigo, these series exhibit higher frequencies.

The performance of the prediction models is evaluated by the root-mean-square-error (RMSE), a frequently employed measure in regression problems [Bis07]:

$$\text{RMSE} = \sqrt{\frac{1}{d \cdot T} \sum_{i=1}^{d} \sum_{t=1}^{T} (s_{ti} - \hat{s}_{ti})^2},$$

where s_{ti} denotes the true value in dimension i at time step t, i.e., the ith entry in the tth row of $\mathbf{S}$, and $\hat{s}_{ti}$ is the predicted one.

The prediction models are initially trained with the first 250 time steps. Afterwards, in an iterative manner the next step is predicted, and the model is re-trained on the true value. The window size is 15, and the neural network models are trained for 15 epochs.

The results are listed in Table C.1. The sign "$-$" signifies that due to an exception during runtime no results could be computed for the respective prediction model. The results reproduce the conclusion of [Wol19] that Kalman filters and the persistence model are the best choice for 2DE and Trigo series, whereat Kalman filters outperform the persistence model on noisy series. Comparing Kalman filter and persistence model on DSB shows similar behavior. Interestingly, both are outperformed by LSTM and also TCN outperforms Kalman filter and persistence model on series without noise. This result suggests, that neural network-based prediction might be a good choice for high frequency series.

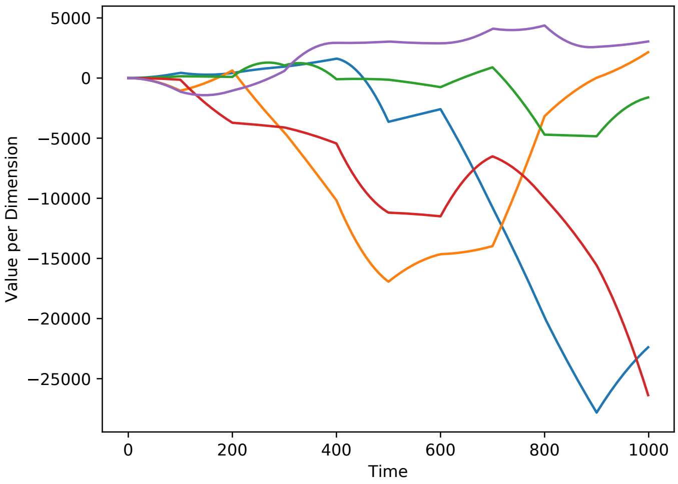


Figure C.1: 2DE time series

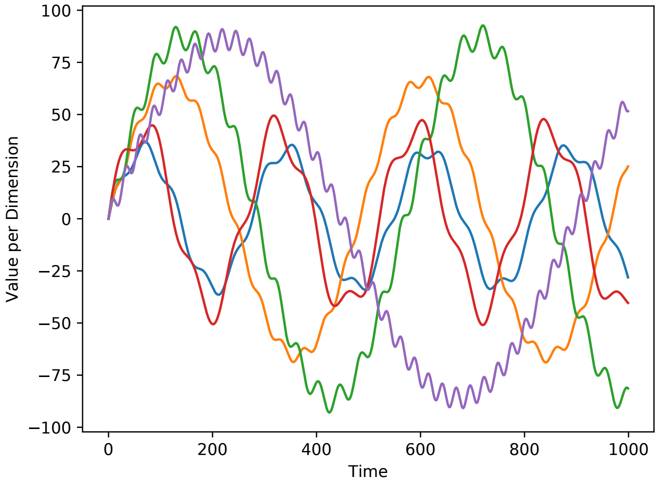


Figure C.2: Trigo time series

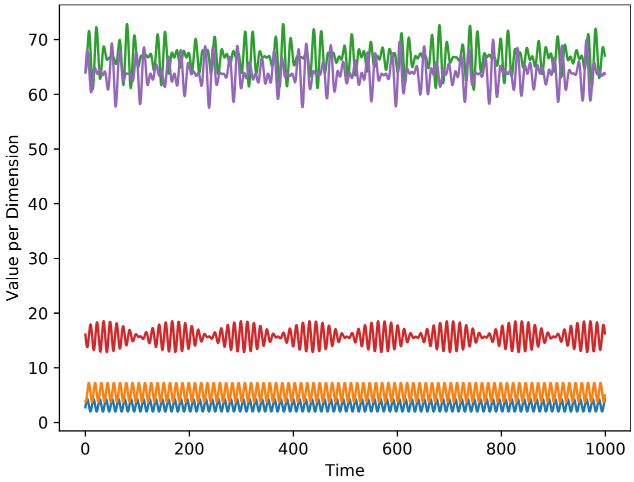


Figure C.3: DSB time series

Table C.1: Comparison of time series prediction methods

Series Type	Noise	Dim.	Persistence	Kalman	LSTM	TCN
2DE	0	2	**0.0030**	0.0053	0.0534	0.1331
		5	**0.0026**	0.0097	0.1187	0.1979
		10	**0.0032**	0.0575	0.1195	0.2080
		20	**0.0031**	–	0.1393	0.2553
	0.1	2	0.0471	**0.0378**	0.0799	0.1472
		5	0.0501	**0.0410**	0.1185	0.1843
		10	0.0583	**0.0477**	0.1354	0.1974
		20	0.0580	**0.0486**	0.1468	0.2292
Trigo	0	2	**0.0083**	0.0115	0.0754	0.1302
		5	**0.0065**	0.0084	0.1661	0.2303
		10	**0.0082**	0.0110	0.1775	0.2757
		20	**0.0072**	0.0529	0.2039	0.3427
	5	2	0.0516	**0.0440**	0.0845	0.1277
		5	0.0507	**0.0419**	0.1557	0.2179
		10	0.0478	**0.0412**	0.1620	0.2377
		20	0.0499	**0.0421**	0.1801	0.2881
DSB	0	2	0.1867	0.1995	**0.0028**	0.0881
		5	0.1387	0.1463	**0.0407**	0.1210
		10	0.1466	0.1531	**0.0491**	0.1403
		20	0.1358	–	**0.0986**	0.1634
	5	2	0.1916	0.1454	**0.1382**	0.1562
		5	0.1958	0.1535	**0.1489**	0.1640
		10	0.2034	0.1567	**0.1524**	0.1655
		20	0.2104	0.1635	**0.1538**	0.1681

D Further Results

This appendix contains the complete results for the experiments in Chapters 7, 8, and 9.

D.1 Prediction for Evolution Strategies

This paragraph lists for Chapter 7 the results regarding the convergence measures ARR and RCS averaged over all runs. The results for group SRR are contained in Tables D.1–D.4. Tables D.5 and D.6, and Tables D.7 and D.8 belong to group MPB-Noisy and Ros-Length, respectively. Table D.9 comprises the ARR values for the group SRR-Neurons. For SRR-Neurons, the RCS measure cannot be computed, since it requires at least two algorithms, but group SRR-Neurons is conducted only for `rnnPred`.

Table D.1: SRR (Rosenbr.): ARR

Mov.	Dim.	`npm`	`rnnPred`	`autoPred`
linear	2	**8.96E-1**	8.26E-1	2.22E-1
	5	**9.10E-1**	8.58E-1	6.31E-1
	10	**8.36E-1**	8.22E-1	4.14E-1
	20	7.23E-1	**7.27E-1**	2.29E-1
	50	**5.29E-1**	5.24E-1	1.57E-1
	100	**3.61E-1**	2.84E-1	1.37E-1
sine	2	**8.96E-1**	8.55E-1	8.66E-1
	5	**9.16E-1**	8.70E-1	8.95E-1
	10	**8.74E-1**	8.27E-1	8.59E-1
	20	**8.14E-1**	7.71E-1	7.94E-1
	50	**7.29E-1**	7.01E-1	7.13E-1
	100	**6.42E-1**	6.22E-1	6.30E-1

Table D.2: SRR (Rosenbr.): RCS

Mov.	Dim.	`npm`	`rnnPred`	`autoPred`
linear	2	2.33E-2	1.08E-2	**4.77E-3**
	5	1.80E-2	6.08E-3	**5.01E-3**
	10	4.68E-2	**1.45E-2**	7.16E-2
	20	1.25E-1	3.28E-2	**2.83E-2**
	50	3.29E-1	7.63E-2	**5.06E-2**
	100	5.31E-1	2.60E-1	**8.02E-2**
sine	2	1.78E-2	**9.92E-3**	1.22E-2
	5	1.37E-2	**8.45E-3**	9.55E-3
	10	3.41E-2	**2.64E-2**	2.90E-2
	20	6.82E-2	**5.44E-2**	5.90E-2
	50	1.32E-1	**1.01E-1**	1.08E-1
	100	2.18E-1	1.71E-1	**1.66E-1**

Table D.3: SRR (Rastrigin): ARR

Mov.	Dim.	`npm`	`rnnPred`	`autoPred`
linear	2	4.90E-1	**5.92E-1**	2.21E-1
	5	3.27E-1	**3.75E-1**	3.24E-1
	10	2.69E-1	**2.96E-1**	2.70E-1
	20	2.29E-1	2.31E-1	**2.31E-1**
	50	1.81E-1	1.60E-1	**1.81E-1**
	100	1.52E-1	1.36E-1	**1.66E-1**
sine	2	**8.27E-1**	8.11E-1	8.12E-1
	5	5.72E-1	**6.00E-1**	5.87E-1
	10	3.26E-1	**3.75E-1**	3.32E-1
	20	1.19E-1	1.53E-1	**1.70E-1**
	50	**4.41E-2**	4.22E-2	3.97E-2
	100	2.37E-2	**2.49E-2**	1.94E-2

Table D.4: SRR (Rastrigin): RCS

Mov.	Dim.	`npm`	`rnnPred`	`autoPred`
linear	2	4.12E-1	1.97E-1	**9.08E-2**
	5	4.64E-1	**3.36E-1**	4.10E-1
	10	5.06E-1	**4.08E-1**	4.54E-1
	20	5.16E-1	**4.65E-1**	4.71E-1
	50	5.39E-1	5.26E-1	**5.14E-1**
	100	6.28E-1	5.51E-1	**3.89E-1**
sine	2	7.36E-2	**4.19E-2**	5.77E-2
	5	2.95E-1	**1.85E-1**	2.48E-1
	10	4.79E-1	**3.45E-1**	4.49E-1
	20	7.65E-1	**4.99E-1**	5.28E-1
	50	9.29E-1	5.39E-1	**5.36E-1**
	100	9.51E-1	7.58E-1	**5.34E-1**

Table D.5: MPB-Noisy: ARR

Dim.	Noise	`npm`	`rnnPred`	`autoPred`
2	0.0	**6.25E-1**	4.86E-1	1.27E-1
	0.1	**6.64E-1**	4.77E-1	4.10E-1
	1.0	**7.82E-1**	6.91E-1	6.96E-1
	10.0	**6.79E-1**	6.61E-1	6.64E-1
20	0.0	6.34E-2	**1.31E-1**	3.85E-2
	0.1	7.36E-2	1.50E-1	**2.08E-1**
	1.0	4.06E-2	4.85E-2	**5.37E-2**
	10.0	2.26E-3	2.43E-3	**2.46E-3**

Table D.6: MPB-Noisy: RCS

Dim.	Noise	`npm`	`rnnPred`	`autoPred`
2	0.0	**3.10E-1**	3.38E-1	3.21E-1
	0.1	**2.71E-1**	3.23E-1	3.13E-1
	1.0	**1.40E-1**	1.67E-1	1.70E-1
	10.0	**1.90E-1**	1.95E-1	1.93E-1
20	0.0	9.06E-1	7.82E-1	**6.20E-1**
	0.1	8.91E-1	7.52E-1	**5.87E-1**
	1.0	9.39E-1	9.26E-1	**9.18E-1**
	10.0	9.96E-1	9.96E-1	**9.96E-1**

Table D.7: Ros-Length: ARR

Dim.	v	`npm`	`rnnPred`	`autoPred`
2	5	**4.63E-1**	4.59E-1	4.59E-1
	10	**7.03E-1**	6.99E-1	6.91E-1
	20	**8.99E-1**	8.59E-1	8.67E-1
	40	**9.52E-1**	9.28E-1	9.36E-1
	60	**9.68E-1**	9.51E-1	9.57E-1
20	5	3.43E-1	3.60E-1	**3.62E-1**
	10	**6.15E-1**	5.91E-1	6.11E-1
	20	**8.15E-1**	7.73E-1	7.95E-1
	40	**9.17E-1**	8.89E-1	9.00E-1
	60	**9.48E-1**	9.29E-1	9.37E-1
50	5	2.52E-1	2.61E-1	**2.67E-1**
	10	4.93E-1	4.88E-1	**4.99E-1**
	20	**7.32E-1**	7.02E-1	7.15E-1
	40	**8.66E-1**	8.42E-1	8.48E-1
	60	**9.12E-1**	8.95E-1	8.95E-1

Table D.8: Ros-Length: RCS

Dim.	v	`npm`	`rnnPred`	`autoPred`
2	5	1.23E-1	**6.38E-2**	7.59E-2
	10	1.64E-2	**9.57E-3**	1.12E-2
	20	4.12E-3	**2.19E-3**	2.75E-3
	40	2.80E-1	**1.86E-1**	1.98E-1
	60	1.99E-3	**1.09E-3**	1.44E-3
20	5	2.09E-1	**1.29E-1**	1.59E-1
	10	6.76E-2	**5.35E-2**	5.90E-2
	20	1.90E-2	**1.63E-2**	1.75E-2
	40	4.96E-1	**2.74E-1**	3.22E-1
	60	8.98E-3	**7.74E-3**	8.18E-3
50	5	3.47E-1	**2.45E-1**	2.49E-1
	10	1.30E-1	**9.66E-2**	1.06E-1
	20	4.50E-2	**4.12E-2**	4.21E-2
	40	6.28E-1	4.49E-1	**4.16E-1**
	60	2.45E-2	**2.20E-2**	2.34E-2

Table D.9: SRR-Neurons: ARR

Dim.	Sphere	Rosenbrock	Rastrigin
2	8.93E-1	**8.67E-1**	8.10E-1
5	8.50E-1	**8.76E-1**	**6.02E-1**
10	7.85E-1	**8.29E-1**	3.73E-1
20	**6.85E-1**	**7.75E-1**	**1.65E-1**
50	**4.83E-1**	**7.02E-1**	**4.42E-2**
100	**3.30E-1**	6.17E-1	**2.56E-2**

D.2 Prediction for Particle Swarm Optimization

The following pages contain the experimental results obtained by executing all PSO variants with different parameter settings in order to find the most suitable setting as described in Chapter 8. Tables D.10–D.13 are for the SRR benchmark, while Tables D.14–D.17 are for MPB-Noisy.

Table D.10: SRR: comparison of parameter settings for `dynPSO`

Metric	BOG			BEBC			RCS		
Benchmark	1.05	1.49	2.00	1.05	1.49	2.00	1.05	1.49	2.00
sph. 2 lin.	9.2E-1	6.0E-2	**5.4E-2**	**3.5E-5**	5.2E-5	5.8E-5	**0.01**	0.01	0.03
sph. 5 lin.	8.2E+0	**3.1E+0**	4.3E+0	**1.9E-2**	2.7E-2	7.2E-2	**0.02**	0.02	0.04
sph. 10 lin.	3.2E+1	**2.2E+1**	3.9E+1	**4.1E-1**	1.4E+0	4.1E+0	**0.05**	0.07	0.12
sph. 20 lin.	**1.5E+2**	1.6E+2	3.0E+2	**2.3E+1**	3.5E+1	9.3E+1	**0.15**	0.20	0.32
sph. 50 lin.	6.7E+4	**3.5E+3**	6.4E+3	6.5E+4	**2.3E+3**	4.5E+3	0.89	**0.56**	0.64
sph. 100 lin.	3.2E+6	7.0E+4	**6.3E+4**	3.2E+6	6.3E+4	**5.4E+4**	0.97	0.83	**0.81**
sph. 2 sin.	8.1E+0	1.6E-1	**1.3E-1**	1.3E-4	**1.0E-4**	1.4E-4	**0.01**	0.02	0.02
sph. 5 sin.	1.2E+1	**2.7E+0**	7.2E+0	1.9E-1	**3.6E-2**	1.8E-1	**0.03**	0.03	0.05
sph. 10 sin.	4.5E+2	**9.4E+1**	1.7E+2	2.9E+2	**5.7E+0**	1.8E+1	0.15	**0.06**	0.10
sph. 20 sin.	1.3E+3	**4.6E+2**	8.3E+2	9.3E+2	**1.0E+2**	2.5E+2	0.31	**0.18**	0.27
sph. 50 sin.	6.6E+4	**9.2E+3**	1.4E+4	6.3E+4	**5.4E+3**	9.1E+3	0.82	**0.47**	0.55
sph. 100 sin.	4.9E+5	**4.1E+4**	5.2E+4	4.8E+5	**3.3E+4**	4.2E+4	0.94	**0.71**	0.73
ros. 2 lin.	3.8E+2	3.8E+0	**1.7E+0**	6.4E-3	**1.2E-3**	1.5E-2	**0.01**	0.01	0.03
ros. 5 lin.	1.1E+4	2.5E+3	**2.4E+3**	2.1E+1	**1.8E+1**	3.9E+1	**0.01**	0.01	0.02
ros. 10 lin.	2.2E+5	**9.0E+4**	1.8E+5	**5.2E+2**	8.4E+2	2.5E+3	**0.02**	0.02	0.04
ros. 20 lin.	**4.2E+6**	1.0E+7	1.6E+7	**8.2E+5**	2.9E+6	2.9E+6	0.18	0.16	**0.14**
ros. 50 lin.	7.2E+9	**2.7E+8**	5.2E+8	6.7E+9	**1.2E+8**	3.1E+8	0.57	**0.37**	0.50
ros. 100 lin.	4.6E+11	**1.4E+10**	1.7E+10	4.5E+11	**1.1E+10**	1.3E+10	0.90	0.72	**0.71**
ros. 2 sin.	3.1E+6	3.7E+4	**4.4E+1**	2.1E+6	**1.5E-1**	6.3E+0	0.43	**0.01**	0.05
ros. 5 sin.	1.3E+9	**7.6E+3**	1.7E+4	1.2E+9	**8.3E+1**	4.1E+2	0.35	**0.02**	0.04
ros. 10 sin.	1.6E+6	**4.8E+5**	1.4E+6	**4.6E+3**	7.3E+3	4.1E+4	**0.01**	0.02	0.04
ros. 20 sin.	**6.7E+6**	8.1E+6	2.0E+7	**3.8E+5**	6.5E+5	1.9E+6	**0.04**	0.07	0.11
ros. 50 sin.	1.6E+9	**8.2E+8**	2.0E+9	1.4E+9	**3.2E+8**	9.1E+8	0.55	**0.27**	0.37
ros. 100 sin.	3.9E+11	**5.6E+9**	1.2E+10	3.9E+11	**3.9E+9**	8.1E+9	0.91	**0.52**	0.61
rast. 2 lin.	3.8E+0	1.6E+0	**1.4E+0**	1.2E-2	**5.6E-3**	2.2E-2	0.03	**0.03**	0.07
rast. 5 lin.	2.4E+5	**2.2E+1**	2.7E+1	2.4E+5	**7.4E+0**	1.1E+1	0.64	**0.18**	0.26
rast. 10 lin.	5.9E+5	**9.9E+1**	1.3E+2	5.9E+5	**5.6E+1**	6.8E+1	0.90	**0.33**	0.36
rast. 20 lin.	3.0E+5	**4.0E+2**	5.6E+2	3.0E+5	**2.3E+2**	3.1E+2	0.87	**0.42**	0.46
rast. 50 lin.	2.0E+5	**4.1E+3**	7.2E+3	2.0E+5	**2.8E+3**	5.2E+3	0.93	**0.60**	0.67
rast. 100 lin.	3.2E+6	**6.2E+4**	6.2E+4	3.2E+6	5.5E+4	**5.3E+4**	0.98	0.81	**0.80**
rast. 2 sin.	9.9E+1	5.1E+0	**2.3E+0**	6.4E+1	**1.0E-1**	1.3E-1	0.18	**0.04**	0.09
rast. 5 sin.	4.7E+4	**2.6E+1**	3.6E+1	4.7E+4	**1.0E+1**	1.5E+1	0.83	**0.22**	0.25
rast. 10 sin.	1.3E+3	**1.8E+2**	2.6E+2	1.1E+3	**7.5E+1**	1.0E+2	0.41	**0.17**	0.19
rast. 20 sin.	6.9E+4	**6.6E+2**	1.0E+3	6.8E+4	**3.1E+2**	4.5E+2	0.88	**0.28**	0.33
rast. 50 sin.	1.8E+5	**1.0E+4**	1.5E+4	1.7E+5	**6.2E+3**	9.6E+3	0.89	**0.49**	0.56
rast. 100 sin.	2.1E+5	**4.3E+4**	5.1E+4	2.1E+5	**3.5E+4**	4.2E+4	0.93	**0.71**	0.73
Best frequen.	3	25	8	8	26	2	11	21	4

Table D.11: SRR: comparison of parameter settings for `pred2p`

Metric	BOG			BEBC			RCS		
Benchmark	1.05	1.49	2.00	1.05	1.49	2.00	1.05	1.49	2.00
sph. 2 lin.	9.6E-1	6.4E-2	**5.5E-2**	**3.1E-5**	4.9E-5	6.3E-5	**0.01**	0.01	0.03
sph. 5 lin.	8.0E+0	**3.0E+0**	4.2E+0	**1.9E-2**	2.8E-2	7.1E-2	**0.02**	0.02	0.04
sph. 10 lin.	3.1E+1	**2.2E+1**	3.9E+1	**4.1E-1**	1.3E+0	4.1E+0	**0.05**	0.07	0.12
sph. 20 lin.	**1.4E+2**	1.6E+2	3.0E+2	**1.7E+1**	3.6E+1	9.2E+1	**0.14**	0.20	0.31
sph. 50 lin.	**2.7E+3**	3.4E+3	6.2E+3	**1.8E+3**	2.2E+3	4.3E+3	**0.15**	0.56	0.62
sph. 100 lin.	6.4E+4	**5.5E+4**	6.2E+4	5.8E+4	**4.8E+4**	5.3E+4	**0.18**	0.69	0.80
sph. 2 sin.	7.3E+0	1.5E-1	**1.4E-1**	**9.5E-5**	1.1E-4	1.5E-4	**0.01**	0.02	0.02
sph. 5 sin.	1.2E+1	**2.8E+0**	7.0E+0	7.1E-2	**3.8E-2**	1.9E-1	**0.02**	0.04	0.05
sph. 10 sin.	1.5E+2	**9.3E+1**	1.7E+2	**4.0E+0**	5.5E+0	1.8E+1	**0.05**	0.06	0.10
sph. 20 sin.	6.0E+2	**4.5E+2**	8.1E+2	2.5E+2	**1.0E+2**	2.4E+2	**0.17**	0.18	0.27
sph. 50 sin.	**8.0E+3**	9.2E+3	1.4E+4	**5.2E+3**	5.5E+3	9.2E+3	**0.20**	0.47	0.55
sph. 100 sin.	**3.8E+4**	4.1E+4	5.1E+4	**3.2E+4**	3.3E+4	4.2E+4	**0.25**	0.70	0.73
ros. 2 lin.	3.8E+2	4.0E+0	**1.6E+0**	9.0E-3	**4.2E-3**	1.1E-2	**0.01**	0.01	0.03
ros. 5 lin.	1.1E+4	2.6E+3	**2.4E+3**	2.0E+1	**2.0E+1**	3.7E+1	**0.01**	0.01	0.02
ros. 10 lin.	2.2E+5	**9.0E+4**	3.4E+5	**2.8E+2**	9.5E+2	8.7E+4	**0.02**	0.02	0.06
ros. 20 lin.	**2.7E+6**	1.1E+7	1.9E+7	**1.8E+5**	3.2E+6	3.6E+6	**0.05**	0.17	0.17
ros. 50 lin.	**1.3E+8**	2.6E+8	5.5E+8	**4.8E+7**	1.1E+8	3.2E+8	**0.20**	0.36	0.52
ros. 100 lin.	1.1E+10	**9.1E+9**	1.8E+10	9.5E+9	**7.1E+9**	1.4E+10	**0.16**	0.55	0.74
ros. 2 sin.	3.1E+5	3.0E+4	**2.9E+1**	3.8E+3	**1.5E+0**	3.4E+0	0.05	**0.02**	0.04
ros. 5 sin.	4.1E+5	**5.1E+3**	1.6E+4	**3.7E+1**	7.8E+1	6.7E+2	**0.00**	0.02	0.04
ros. 10 sin.	1.6E+6	**5.6E+5**	1.4E+6	**5.2E+3**	1.2E+4	3.6E+4	**0.01**	0.02	0.04
ros. 20 sin.	**5.3E+6**	7.8E+6	2.3E+7	**1.1E+5**	6.2E+5	2.3E+6	**0.03**	0.07	0.12
ros. 50 sin.	**3.2E+8**	7.8E+8	2.1E+9	**1.2E+8**	3.0E+8	9.4E+8	**0.14**	0.26	0.38
ros. 100 sin.	**3.5E+9**	5.5E+9	1.1E+10	**2.5E+9**	3.7E+9	8.0E+9	**0.16**	0.50	0.60
rast. 2 lin.	3.8E+0	1.7E+0	**1.4E+0**	1.1E-2	**6.6E-3**	2.6E-2	0.03	**0.03**	0.07
rast. 5 lin.	4.9E+1	**2.2E+1**	2.7E+1	2.0E+1	**7.5E+0**	1.1E+1	**0.08**	0.18	0.26
rast. 10 lin.	2.2E+2	**1.0E+2**	1.3E+2	1.3E+2	**5.6E+1**	6.8E+1	**0.08**	0.33	0.36
rast. 20 lin.	6.2E+2	**4.0E+2**	5.6E+2	3.9E+2	**2.3E+2**	3.1E+2	**0.12**	0.42	0.46
rast. 50 lin.	6.7E+3	**4.0E+3**	7.0E+3	5.4E+3	**2.8E+3**	5.0E+3	**0.17**	0.60	0.65
rast. 100 lin.	9.3E+4	**5.3E+4**	6.3E+4	8.6E+4	**4.6E+4**	5.4E+4	**0.21**	0.71	0.81
rast. 2 sin.	3.2E+1	5.2E+0	**2.3E+0**	1.1E+1	**1.2E-1**	1.4E-1	0.06	**0.04**	0.09
rast. 5 sin.	1.0E+2	**2.6E+1**	3.5E+1	6.2E+1	**1.0E+1**	1.5E+1	**0.07**	0.22	0.25
rast. 10 sin.	4.1E+2	**1.8E+2**	2.6E+2	2.2E+2	**7.5E+1**	9.9E+1	**0.16**	0.17	0.19
rast. 20 sin.	1.1E+3	**6.7E+2**	1.0E+3	7.0E+2	**3.1E+2**	4.5E+2	**0.12**	0.28	0.33
rast. 50 sin.	**9.8E+3**	1.0E+4	1.5E+4	7.0E+3	**6.1E+3**	9.6E+3	**0.18**	0.49	0.56
rast. 100 sin.	**4.0E+4**	4.3E+4	5.2E+4	**3.4E+4**	3.5E+4	4.2E+4	**0.29**	0.71	0.73
Best frequen.	11	18	7	18	18	0	33	3	0

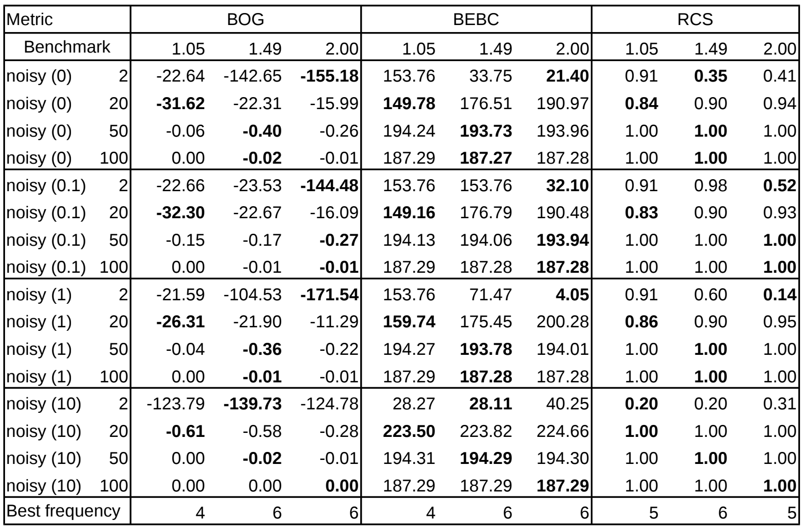
Table D.12: SRR: comparison of parameter settings for `pred3`

Metric	BOG					BEBC					RCS				
Benchmark	1.05	1.49	2.00	additive	adaptive	1.05	1.49	2.00	additive	adaptive	1.05	1.49	2.00	additive	adaptive
sph. 2 lin.	5.0E-1	4.8E-2	5.7E-2	**4.1E-2**	4.8E-2	**2.8E-5**	5.2E-5	6.3E-5	5.4E-5	5.1E-5	**0.00**	0.01	0.03	0.04	0.02
sph. 5 lin.	3.1E+0	2.1E+0	3.4E+0	2.4E+0	**2.1E+0**	**1.8E-2**	2.6E-2	6.3E-2	3.2E-2	2.7E-2	**0.01**	0.02	0.04	0.06	0.04
sph. 10 lin.	**1.5E+1**	1.7E+1	3.1E+1	2.1E+1	1.7E+1	**3.2E-1**	1.3E+0	3.3E+0	1.8E+0	1.3E+0	**0.02**	0.06	0.09	0.11	0.10
sph. 20 lin.	**7.7E+1**	1.4E+2	2.4E+2	1.4E+2	1.4E+2	**9.6E+0**	3.5E+1	6.9E+1	3.1E+1	3.5E+1	**0.07**	0.18	0.24	0.22	0.26
sph. 50 lin.	**1.7E+3**	2.2E+3	5.7E+3	2.5E+3	2.2E+3	**1.1E+3**	1.4E+3	3.9E+3	1.5E+3	1.4E+3	**0.13**	0.34	0.57	0.49	0.51
sph. 100 lin.	4.0E+4	2.8E+4	5.1E+4	2.8E+4	**2.7E+4**	3.6E+4	2.3E+4	4.3E+4	2.3E+4	**2.2E+4**	**0.16**	0.37	0.65	0.72	0.70
sph. 2 sin.	2.1E+0	**1.1E-1**	2.0E-1	1.7E-1	1.1E-1	**7.9E-5**	1.6E-4	2.2E-4	2.4E-4	1.6E-4	**0.00**	0.02	0.03	0.03	0.03
sph. 5 sin.	5.0E+0	**3.8E+0**	9.6E+0	6.2E+0	3.8E+0	**3.4E-2**	5.9E-2	2.4E-1	9.7E-2	5.9E-2	**0.02**	0.05	0.07	0.06	0.06
sph. 10 sin.	1.1E+2	**1.1E+2**	1.9E+2	1.3E+2	1.1E+2	**2.4E+0**	6.6E+0	1.8E+1	7.5E+0	6.7E+0	**0.03**	0.07	0.11	0.08	0.09
sph. 20 sin.	**4.4E+2**	4.6E+2	8.1E+2	4.9E+2	4.6E+2	1.6E+2	1.1E+2	2.3E+2	**1.1E+2**	1.1E+2	**0.12**	0.19	0.26	0.19	0.20
sph. 50 sin.	**5.6E+3**	8.2E+3	1.5E+4	9.1E+3	8.2E+3	**2.9E+3**	4.5E+3	9.2E+3	5.2E+3	4.5E+3	**0.15**	0.42	0.56	0.48	0.46
sph. 100 sin.	**2.8E+4**	3.7E+4	5.2E+4	3.7E+4	3.7E+4	**2.1E+4**	2.8E+4	4.2E+4	2.8E+4	2.8E+4	**0.20**	0.62	0.73	0.68	0.68
ros. 2 lin.	1.2E+2	7.8E-1	1.6E+0	**6.0E-1**	7.9E-1	5.5E-3	**1.0E-3**	2.2E-2	1.2E-3	1.2E-3	**0.00**	0.01	0.03	0.03	0.02
ros. 5 lin.	5.4E+3	**2.1E+3**	2.4E+3	2.2E+3	2.1E+3	**1.9E+1**	1.9E+1	3.5E+1	2.1E+1	2.0E+1	**0.00**	0.01	0.02	0.03	0.03
ros. 10 lin.	1.9E+5	8.4E+4	3.8E+5	9.0E+4	**8.3E+4**	**1.2E+2**	6.3E+2	8.1E+4	6.7E+2	5.8E+2	**0.01**	0.02	0.05	0.04	0.04
ros. 20 lin.	**2.2E+6**	1.1E+7	1.6E+7	9.5E+6	1.1E+7	**4.9E+3**	3.1E+6	2.8E+6	2.5E+6	3.0E+6	**0.01**	0.17	0.14	0.18	0.19
ros. 50 lin.	**6.8E+7**	1.4E+8	4.5E+8	1.7E+8	1.4E+8	**1.8E+7**	5.2E+7	2.6E+8	6.3E+7	5.1E+7	**0.09**	0.19	0.39	0.33	0.32
ros. 100 lin.	7.7E+9	3.8E+9	1.1E+10	**3.5E+9**	3.6E+9	6.8E+9	2.7E+9	8.5E+9	**2.5E+9**	2.5E+9	**0.14**	0.26	0.50	0.58	0.59
ros. 2 sin.	5.9E+4	6.2E+3	**2.9E+1**	6.2E+3	6.3E+3	3.7E+3	**1.3E-1**	1.5E+0	1.5E-1	1.8E-1	0.04	**0.01**	0.04	0.04	0.04
ros. 5 sin.	1.2E+5	**5.4E+3**	2.8E+4	1.3E+4	5.4E+3	**1.1E+1**	9.8E+1	7.7E+2	1.9E+2	9.7E+1	**0.00**	0.03	0.05	0.04	0.04
ros. 10 sin.	9.1E+5	6.4E+5	1.9E+6	8.9E+5	**6.3E+5**	**2.9E+3**	1.3E+4	4.2E+4	1.6E+4	1.3E+4	**0.01**	0.03	0.04	0.03	0.03
ros. 20 sin.	**3.6E+6**	8.4E+6	2.4E+7	1.0E+7	8.4E+6	**5.7E+4**	6.7E+5	2.2E+6	6.7E+5	6.5E+5	**0.02**	0.08	0.12	0.09	0.09
ros. 50 sin.	**2.4E+8**	6.4E+8	2.0E+9	7.9E+8	6.5E+8	**6.7E+7**	2.2E+8	8.6E+8	2.8E+8	2.2E+8	**0.10**	0.21	0.36	0.27	0.26
ros. 100 sin.	**2.6E+9**	5.1E+9	1.1E+10	5.2E+9	5.1E+9	**1.5E+9**	3.2E+9	7.3E+9	3.3E+9	3.2E+9	**0.14**	0.46	0.56	0.51	0.52
rast. 2 lin.	2.1E+0	1.2E+0	1.4E+0	**1.1E+0**	1.2E+0	**4.7E-3**	6.1E-3	2.9E-2	5.9E-3	5.7E-3	**0.01**	0.02	0.07	0.07	0.04
rast. 5 lin.	2.2E+1	**2.0E+1**	2.6E+1	2.2E+1	2.0E+1	**6.9E+0**	7.2E+0	1.1E+1	8.8E+0	7.3E+0	**0.05**	0.17	0.25	0.29	0.26
rast. 10 lin.	1.3E+2	**9.5E+1**	1.2E+2	9.9E+1	9.5E+1	8.1E+1	**5.6E+1**	6.4E+1	5.7E+1	5.6E+1	**0.08**	0.33	0.33	0.39	0.41
rast. 20 lin.	3.9E+2	3.6E+2	5.0E+2	3.7E+2	**3.6E+2**	2.5E+2	2.2E+2	2.9E+2	**2.2E+2**	2.2E+2	**0.10**	0.39	0.41	0.44	0.48
rast. 50 lin.	4.9E+3	**2.6E+3**	6.4E+3	3.0E+3	2.6E+3	4.1E+3	**1.8E+3**	4.6E+3	2.0E+3	1.8E+3	**0.15**	0.39	0.59	0.55	0.58
rast. 100 lin.	5.3E+4	2.9E+4	5.1E+4	3.0E+4	**2.9E+4**	4.8E+4	2.4E+4	4.3E+4	2.4E+4	**2.4E+4**	**0.17**	0.39	0.66	0.72	0.71
rast. 2 sin.	1.7E+1	**2.4E+0**	2.7E+0	2.7E+0	2.4E+0	9.4E+0	8.5E-2	2.0E-1	1.3E-1	**8.5E-2**	**0.04**	0.04	0.10	0.09	0.09
rast. 5 sin.	4.6E+1	**2.8E+1**	4.0E+1	3.3E+1	2.8E+1	2.5E+1	**1.1E+1**	1.6E+1	1.3E+1	1.1E+1	**0.06**	0.25	0.27	0.27	0.29
rast. 10 sin.	3.1E+2	**2.0E+2**	2.8E+2	2.2E+2	2.0E+2	1.6E+2	**7.9E+1**	1.0E+2	8.0E+1	7.9E+1	**0.12**	0.18	0.20	0.18	0.20
rast. 20 sin.	8.0E+2	**6.8E+2**	1.0E+3	7.2E+2	6.8E+2	4.8E+2	**3.1E+2**	4.4E+2	3.1E+2	3.1E+2	**0.10**	0.28	0.33	0.28	0.30
rast. 50 sin.	**6.7E+3**	8.8E+3	1.5E+4	9.8E+3	8.9E+3	**4.0E+3**	5.1E+3	9.4E+3	5.8E+3	5.1E+3	**0.14**	0.43	0.56	0.49	0.48
rast. 100 sin.	**2.9E+4**	3.9E+4	5.2E+4	3.9E+4	3.9E+4	**2.2E+4**	3.0E+4	4.2E+4	3.0E+4	3.0E+4	**0.23**	0.63	0.74	0.69	0.69
Best frequen.	13	12	1	4	6	23	7	0	3	3	35	1	0	0	0

Table D.13: SRR: comparison of parameter settings for `pred3p`

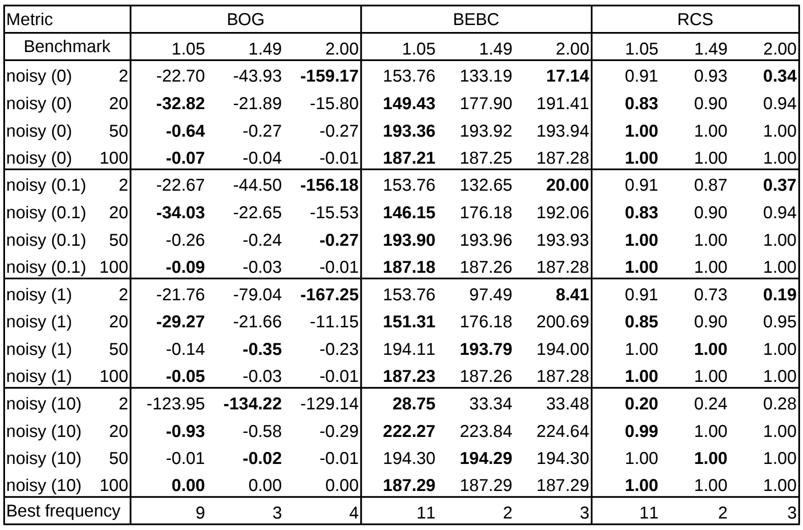
Metric	BOG					BEBC					RCS				
Benchmark	1.05	1.49	2.00	additive	adaptive	1.05	1.49	2.00	additive	adaptive	1.05	1.49	2.00	additive	adaptive
sph. 2 lin.	5.0E-1	4.8E-2	5.7E-2	**4.2E-2**	4.9E-2	**2.9E-5**	5.1E-5	6.7E-5	5.3E-5	5.3E-5	**0.00**	0.01	0.03	0.04	0.03
sph. 5 lin.	3.1E+0	**2.1E+0**	3.4E+0	2.4E+0	2.1E+0	**1.8E-2**	2.7E-2	6.5E-2	3.2E-2	2.7E-2	**0.01**	0.02	0.04	0.06	0.04
sph. 10 lin.	**1.5E+1**	1.7E+1	3.1E+1	2.1E+1	1.7E+1	**3.2E-1**	1.3E+0	3.3E+0	1.7E+0	1.3E+0	**0.02**	0.06	0.09	0.11	0.10
sph. 20 lin.	**7.7E+1**	1.4E+2	2.4E+2	1.4E+2	1.4E+2	**9.6E+0**	3.5E+1	6.9E+1	3.0E+1	3.4E+1	**0.07**	0.18	0.24	0.21	0.26
sph. 50 lin.	**1.7E+3**	2.2E+3	5.7E+3	2.5E+3	2.2E+3	**1.2E+3**	1.4E+3	3.9E+3	1.5E+3	1.3E+3	**0.13**	0.34	0.57	0.50	0.50
sph. 100 lin.	4.2E+4	2.9E+4	5.2E+4	**2.8E+4**	2.8E+4	3.7E+4	2.4E+4	4.4E+4	**2.3E+4**	2.3E+4	**0.17**	0.38	0.67	0.71	0.73
sph. 2 sin.	2.1E+0	**1.1E-1**	2.0E-1	1.7E-1	1.2E-1	**8.0E-5**	1.6E-4	2.3E-4	2.5E-4	1.6E-4	**0.00**	0.02	0.03	0.03	0.03
sph. 5 sin.	5.0E+0	**3.8E+0**	9.5E+0	6.2E+0	3.8E+0	**3.4E-2**	5.9E-2	2.4E-1	9.8E-2	5.9E-2	**0.02**	0.05	0.07	0.06	0.06
sph. 10 sin.	1.1E+2	1.1E+2	1.9E+2	1.3E+2	**1.1E+2**	**2.4E+0**	6.7E+0	1.8E+1	7.3E+0	6.7E+0	**0.03**	0.07	0.11	0.08	0.08
sph. 20 sin.	**4.4E+2**	4.6E+2	8.2E+2	5.0E+2	4.6E+2	1.6E+2	1.1E+2	2.3E+2	**1.1E+2**	1.1E+2	**0.12**	0.19	0.26	0.19	0.20
sph. 50 sin.	**5.6E+3**	8.2E+3	1.5E+4	9.1E+3	8.2E+3	**2.9E+3**	4.5E+3	9.0E+3	5.2E+3	4.5E+3	**0.15**	0.41	0.55	0.48	0.46
sph. 100 sin.	**2.7E+4**	3.6E+4	5.2E+4	3.6E+4	3.6E+4	**2.0E+4**	2.8E+4	4.2E+4	2.8E+4	2.8E+4	**0.20**	0.62	0.73	0.67	0.68
ros. 2 lin.	1.2E+2	7.7E-1	1.6E+0	**6.1E-1**	7.8E-1	5.5E-3	9.9E-4	1.7E-2	1.2E-3	**9.6E-4**	**0.00**	0.01	0.03	0.03	0.02
ros. 5 lin.	5.4E+3	**2.1E+3**	2.3E+3	2.2E+3	2.1E+3	1.9E+1	1.9E+1	3.6E+1	2.3E+1	**1.9E+1**	**0.00**	0.01	0.02	0.03	0.02
ros. 10 lin.	1.9E+5	8.4E+4	3.0E+5	8.9E+4	**8.3E+4**	**1.2E+2**	7.3E+2	4.5E+4	6.1E+2	5.6E+2	**0.01**	0.02	0.05	0.04	0.04
ros. 20 lin.	**2.2E+6**	1.0E+7	1.5E+7	9.2E+6	1.1E+7	**4.9E+3**	2.8E+6	2.5E+6	2.4E+6	3.1E+6	**0.01**	0.15	0.13	0.17	0.20
ros. 50 lin.	**7.1E+7**	1.4E+8	4.6E+8	1.7E+8	1.5E+8	**2.0E+7**	5.2E+7	2.7E+8	6.3E+7	5.3E+7	**0.10**	0.19	0.40	0.33	0.33
ros. 100 lin.	7.7E+9	3.7E+9	1.1E+10	**3.4E+9**	3.7E+9	6.8E+9	2.7E+9	8.5E+9	**2.4E+9**	2.6E+9	**0.14**	0.26	0.50	0.57	0.60
ros. 2 sin.	5.9E+4	6.2E+3	**3.6E+1**	6.2E+3	6.2E+3	3.7E+3	**1.3E-1**	2.6E+0	1.6E-1	1.4E-1	0.04	**0.01**	0.05	0.04	0.04
ros. 5 sin.	1.1E+5	5.5E+3	2.9E+4	1.4E+4	**5.3E+3**	**9.2E+0**	1.0E+2	7.3E+2	2.1E+2	9.9E+1	**0.00**	0.03	0.05	0.04	0.04
ros. 10 sin.	9.3E+5	6.6E+5	1.8E+6	8.7E+5	**6.3E+5**	**2.7E+3**	1.6E+4	3.9E+4	1.6E+4	1.3E+4	**0.01**	0.03	0.04	0.03	0.03
ros. 20 sin.	**3.6E+6**	1.0E+7	2.5E+7	9.8E+6	9.2E+6	**6.0E+4**	9.4E+5	2.3E+6	6.4E+5	7.6E+5	**0.02**	0.08	0.12	0.09	0.09
ros. 50 sin.	**2.3E+8**	6.4E+8	2.0E+9	8.0E+8	6.5E+8	**6.6E+7**	2.2E+8	8.7E+8	2.8E+8	2.2E+8	**0.10**	0.21	0.36	0.28	0.26
ros. 100 sin.	**2.5E+9**	5.2E+9	1.1E+10	5.2E+9	5.0E+9	**1.5E+9**	3.2E+9	7.2E+9	3.2E+9	3.2E+9	**0.14**	0.47	0.56	0.51	0.51
rast. 2 lin.	2.1E+0	1.2E+0	1.4E+0	**1.1E+0**	1.2E+0	**4.5E-3**	5.8E-3	3.0E-2	5.8E-3	6.3E-3	**0.01**	0.02	0.07	0.07	0.04
rast. 5 lin.	2.2E+1	**2.0E+1**	2.6E+1	2.2E+1	2.0E+1	**6.9E+0**	7.3E+0	1.1E+1	8.8E+0	7.2E+0	**0.05**	0.17	0.25	0.30	0.26
rast. 10 lin.	1.3E+2	9.5E+1	1.2E+2	9.9E+1	**9.5E+1**	8.1E+1	5.7E+1	6.4E+1	5.7E+1	**5.6E+1**	**0.08**	0.33	0.33	0.39	0.41
rast. 20 lin.	3.9E+2	3.6E+2	5.0E+2	3.7E+2	**3.6E+2**	2.5E+2	2.2E+2	2.9E+2	**2.2E+2**	2.2E+2	**0.10**	0.39	0.41	0.43	0.48
rast. 50 lin.	4.9E+3	2.7E+3	6.4E+3	3.0E+3	**2.6E+3**	4.1E+3	1.8E+3	4.6E+3	2.0E+3	**1.8E+3**	**0.15**	0.39	0.59	0.55	0.57
rast. 100 lin.	5.2E+4	3.0E+4	5.1E+4	**3.0E+4**	3.0E+4	4.7E+4	2.5E+4	4.3E+4	**2.4E+4**	2.5E+4	**0.17**	0.40	0.66	0.72	0.74
rast. 2 sin.	1.7E+1	**2.4E+0**	2.7E+0	2.7E+0	2.4E+0	9.4E+0	9.0E-2	2.0E-1	1.2E-1	**8.9E-2**	**0.04**	0.04	0.11	0.10	0.09
rast. 5 sin.	4.6E+1	2.8E+1	4.0E+1	3.3E+1	**2.8E+1**	2.5E+1	1.1E+1	1.6E+1	1.3E+1	**1.1E+1**	**0.06**	0.25	0.27	0.27	0.29
rast. 10 sin.	3.1E+2	**2.0E+2**	2.8E+2	2.2E+2	2.0E+2	1.6E+2	**7.9E+1**	1.0E+2	8.0E+1	7.9E+1	**0.12**	0.18	0.20	0.18	0.20
rast. 20 sin.	8.0E+2	6.8E+2	1.0E+3	7.2E+2	**6.8E+2**	4.8E+2	3.1E+2	4.4E+2	**3.1E+2**	3.1E+2	**0.10**	0.28	0.33	0.29	0.30
rast. 50 sin.	**6.7E+3**	8.8E+3	1.5E+4	9.9E+3	8.8E+3	**4.0E+3**	5.1E+3	9.4E+3	5.8E+3	5.1E+3	**0.14**	0.43	0.56	0.50	0.48
rast. 100 sin.	**2.9E+4**	3.9E+4	5.2E+4	3.9E+4	3.9E+4	**2.2E+4**	3.0E+4	4.2E+4	3.0E+4	3.0E+4	**0.23**	0.63	0.74	0.69	0.70
Best frequen.	13	7	1	6	9	22	2	0	6	6	35	1	0	0	0

Table D.14: MPB-Noisy: comparison of parameter settings for `dynPSO`



Metric		BOG			BEBC			RCS		
Benchmark		1.05	1.49	2.00	1.05	1.49	2.00	1.05	1.49	2.00
noisy (0)	2	-22.64	-142.65	**-155.18**	153.76	33.75	**21.40**	0.91	**0.35**	0.41
noisy (0)	20	**-31.62**	-22.31	-15.99	**149.78**	176.51	190.97	**0.84**	0.90	0.94
noisy (0)	50	-0.06	**-0.40**	-0.26	194.24	**193.73**	193.96	1.00	**1.00**	1.00
noisy (0)	100	0.00	**-0.02**	-0.01	187.29	**187.27**	187.28	1.00	**1.00**	1.00
noisy (0.1)	2	-22.66	-23.53	**-144.48**	153.76	153.76	**32.10**	0.91	0.98	**0.52**
noisy (0.1)	20	**-32.30**	-22.67	-16.09	**149.16**	176.79	190.48	**0.83**	0.90	0.93
noisy (0.1)	50	-0.15	-0.17	**-0.27**	194.13	194.06	**193.94**	1.00	1.00	**1.00**
noisy (0.1)	100	0.00	-0.01	**-0.01**	187.29	187.28	**187.28**	1.00	1.00	**1.00**
noisy (1)	2	-21.59	-104.53	**-171.54**	153.76	71.47	**4.05**	0.91	0.60	**0.14**
noisy (1)	20	**-26.31**	-21.90	-11.29	**159.74**	175.45	200.28	**0.86**	0.90	0.95
noisy (1)	50	-0.04	**-0.36**	-0.22	194.27	**193.78**	194.01	1.00	**1.00**	1.00
noisy (1)	100	0.00	**-0.01**	-0.01	187.29	**187.28**	187.28	1.00	**1.00**	1.00
noisy (10)	2	-123.79	**-139.73**	-124.78	28.27	**28.11**	40.25	**0.20**	0.20	0.31
noisy (10)	20	**-0.61**	-0.58	-0.28	**223.50**	223.82	224.66	**1.00**	1.00	1.00
noisy (10)	50	0.00	**-0.02**	-0.01	194.31	**194.29**	194.30	1.00	**1.00**	1.00
noisy (10)	100	0.00	0.00	**0.00**	187.29	187.29	**187.29**	1.00	1.00	**1.00**
Best frequency		4	6	6	4	6	6	5	6	5

Table D.15: MPB-Noisy: comparison of parameter settings for `pred2p`



Metric		BOG			BEBC			RCS		
Benchmark		1.05	1.49	2.00	1.05	1.49	2.00	1.05	1.49	2.00
noisy (0)	2	-22.70	-43.93	**-159.17**	153.76	133.19	**17.14**	0.91	0.93	**0.34**
noisy (0)	20	**-32.82**	-21.89	-15.80	**149.43**	177.90	191.41	**0.83**	0.90	0.94
noisy (0)	50	**-0.64**	-0.27	-0.27	**193.36**	193.92	193.94	**1.00**	1.00	1.00
noisy (0)	100	**-0.07**	-0.04	-0.01	**187.21**	187.25	187.28	**1.00**	1.00	1.00
noisy (0.1)	2	-22.67	-44.50	**-156.18**	153.76	132.65	**20.00**	0.91	0.87	**0.37**
noisy (0.1)	20	**-34.03**	-22.65	-15.53	**146.15**	176.18	192.06	**0.83**	0.90	0.94
noisy (0.1)	50	-0.26	-0.24	**-0.27**	**193.90**	193.96	193.93	**1.00**	1.00	1.00
noisy (0.1)	100	**-0.09**	-0.03	-0.01	**187.18**	187.26	187.28	**1.00**	1.00	1.00
noisy (1)	2	-21.76	-79.04	**-167.25**	153.76	97.49	**8.41**	0.91	0.73	**0.19**
noisy (1)	20	**-29.27**	-21.66	-11.15	**151.31**	176.18	200.69	**0.85**	0.90	0.95
noisy (1)	50	-0.14	**-0.35**	-0.23	194.11	**193.79**	194.00	1.00	**1.00**	1.00
noisy (1)	100	**-0.05**	-0.03	-0.01	**187.23**	187.26	187.28	**1.00**	1.00	1.00
noisy (10)	2	-123.95	**-134.22**	-129.14	**28.75**	33.34	33.48	**0.20**	0.24	0.28
noisy (10)	20	**-0.93**	-0.58	-0.29	**222.27**	223.84	224.64	**0.99**	1.00	1.00
noisy (10)	50	-0.01	**-0.02**	-0.01	194.30	**194.29**	194.30	1.00	**1.00**	1.00
noisy (10)	100	**0.00**	0.00	0.00	**187.29**	187.29	187.29	**1.00**	1.00	1.00
Best frequency		9	3	4	11	2	3	11	2	3

Table D.16: MPB-Noisy: comparison of parameter settings for `pred3`

Metric		BOG			BEBC			RCS		
Benchmark		1.05	1.49	2.00	1.05	1.49	2.00	1.05	1.49	2.00
noisy (0)	2	-23.11	-104.97	**-160.75**	153.76	71.79	**15.32**	0.91	0.58	**0.31**
noisy (0)	20	-17.07	**-24.18**	-16.15	191.60	**175.46**	191.06	0.91	**0.90**	0.93
noisy (0)	50	**-1.41**	-0.32	-0.28	**191.91**	193.82	193.92	**0.99**	1.00	1.00
noisy (0)	100	**-0.17**	-0.11	-0.03	**187.08**	187.15	187.26	**1.00**	1.00	1.00
noisy (0.1)	2	-23.05	-99.15	**-153.68**	153.76	77.57	**22.29**	0.91	0.55	**0.41**
noisy (0.1)	20	-15.80	**-24.67**	-16.30	192.33	**174.82**	190.40	0.91	**0.89**	0.93
noisy (0.1)	50	**-0.75**	-0.31	-0.28	**193.03**	193.83	193.92	**1.00**	1.00	1.00
noisy (0.1)	100	**-0.12**	-0.12	-0.02	**187.13**	187.14	187.26	**1.00**	1.00	1.00
noisy (1)	2	-22.42	-111.65	**-166.98**	153.76	64.59	**7.69**	0.91	0.52	**0.18**
noisy (1)	20	**-33.21**	-21.90	-11.33	**144.28**	175.34	199.96	**0.83**	0.90	0.95
noisy (1)	50	**-0.29**	-0.22	-0.22	**193.80**	193.97	193.99	**1.00**	1.00	1.00
noisy (1)	100	**-0.16**	-0.10	-0.02	**187.08**	187.16	187.27	**1.00**	1.00	1.00
noisy (10)	2	**-126.34**	-124.87	-118.78	**31.56**	41.15	39.48	**0.20**	0.29	0.33
noisy (10)	20	**-1.11**	-0.53	-0.28	**221.58**	223.90	224.64	**0.99**	1.00	1.00
noisy (10)	50	-0.01	**-0.02**	-0.01	194.29	**194.29**	194.30	**1.00**	1.00	1.00
noisy (10)	100	**-0.01**	0.00	0.00	**187.28**	187.29	187.29	**1.00**	1.00	1.00
Best frequency		10	3	3	10	3	3	11	2	3

Table D.17: MPB-Noisy: comparison of parameter settings for `pred3p`

Metric		BOG			BEBC			RCS		
Benchmark		1.05	1.49	2.00	1.05	1.49	2.00	1.05	1.49	2.00
noisy (0)	2	-23.09	-100.37	**-155.65**	153.76	76.36	**20.44**	0.91	0.62	**0.38**
noisy (0)	20	-16.92	**-24.59**	-16.92	191.73	**174.56**	189.46	0.91	**0.89**	0.93
noisy (0)	50	**-1.40**	-0.31	-0.28	**191.93**	193.83	193.93	**0.99**	1.00	1.00
noisy (0)	100	**-0.18**	-0.11	-0.02	**187.06**	187.15	187.26	**1.00**	1.00	1.00
noisy (0.1)	2	-23.04	-109.83	**-154.42**	153.76	66.87	**21.46**	0.91	0.49	**0.39**
noisy (0.1)	20	-15.93	**-25.96**	-15.16	192.03	**171.89**	192.80	0.91	**0.89**	0.94
noisy (0.1)	50	**-0.75**	-0.32	-0.27	**193.03**	193.82	193.93	**1.00**	1.00	1.00
noisy (0.1)	100	**-0.15**	-0.11	-0.02	**187.10**	187.16	187.27	**1.00**	1.00	1.00
noisy (1)	2	-22.44	-119.38	**-164.79**	153.76	56.87	**9.72**	0.91	0.49	**0.21**
noisy (1)	20	**-32.62**	-21.98	-11.35	**145.66**	175.32	199.76	**0.84**	0.90	0.95
noisy (1)	50	**-0.30**	-0.23	-0.22	**193.79**	193.96	194.00	**1.00**	1.00	1.00
noisy (1)	100	**-0.15**	-0.11	-0.02	**187.10**	187.16	187.27	**1.00**	1.00	1.00
noisy (10)	2	**-126.97**	-120.45	-121.70	**30.68**	46.24	36.86	**0.20**	0.32	0.31
noisy (10)	20	**-1.13**	-0.53	-0.27	**221.56**	223.90	224.66	**0.99**	1.00	1.00
noisy (10)	50	-0.01	**-0.02**	-0.01	194.29	**194.29**	194.30	**1.00**	1.00	1.00
noisy (10)	100	**-0.01**	0.00	0.00	**187.28**	187.29	187.29	**1.00**	1.00	1.00
Best frequency		10	3	3	10	3	3	11	2	3

Table D.18: Parameter settings for experiments

	Input Range	Hyperparameter Tuning		Final Comparison
		Architecture	Training	
Range	$[0,2], [0,1], [-1,1]$	$[-1,1]$	$[-1,1]$	$[-1,1]$
Filters	16	$[8,16,27]$	27	27
Filter Size k	3	$[2,3,4,5,6,7]$	6	6
Learning Rate	0.002	0.002	$[0.001, 0.002, 0.004]$	0.001
Epochs	5	80	80	100
Batch Size	32	32	$[8,16,32]$	32
Dropout	0.1	0.05	$[0.01, 0.05, 0.1]$	0.1

D.3 Predictive Uncertainty for Evolution Strategies

As mentioned in Chapter 9, we tune the network hyper-parameters before conducting the final experiments. Procedure and results of hyperparameter tuning are presented in Paragraph D.3.1, while Paragraph D.3.2 contains the results of the final comparison.

D.3.1 Tuning of Hyperparameters

We tune the hyperparameters of the TCNs only on `tcn`. Since the training process for `tcn` and `unc` is the same except for the output layer of the aleatoric uncertainty, the identified settings on `tcn` should also be suited for `unc`. Re-initialization strategy is `pRND`. We employ a ten-dimensional DSB instance with the Sphere function as benchmark which differs from the one used in the final comparison. All experiments are repeated five times.

Before we tune the hyperparameters, we identify the best range for scaling the input data. The hyperparameter tuning is divided into two parts: tuning of architecture and training parameters. The best found setting is employed for the final comparison. See Table D.18 for an overview of the employed parameter settings.

Input Range

For NNs it is important to scale the inputs into a reasonable range. Therefore, we experimentally identify the best among three common ranges before we tune the hyperparameters. The interval $[-1, 1]$ turns out to be more suitable than $[0, 2]$ and $[0, 1]$ (Table D.19).

Table D.19: Comparison of different input ranges

Range	$\overline{\text{BOG}}$	BEBC	RCS	PE
$[0, 2]$	2.01 ± 0.01	1.55 ± 0.02	0.53 ± 0.01	0.97 ± 0.17
$[0, 1]$	2.00 ± 0.02	1.54 ± 0.02	0.53 ± 0.01	0.85 ± 0.02
$[-1, 1]$	**1.78 ± 0.02**	**1.43 ± 0.01**	**0.48 ± 0.01**	**0.62 ± 0.01**

Table D.20: Results for tuning of architecture parameters

Filters	k	$\overline{\text{BOG}}$	BEBC	RCS	PE
8	2	**1.75 ±0.024**	1.42 ±0.022	0.71 ±0.010	0.62 ±0.006
	3	1.77 ±0.030	1.43 ±0.017	0.71 ±0.013	0.62 ±0.009
	4	1.76 ±0.020	**1.41 ±0.012**	0.71 ±0.005	**0.62 ±0.013**
	5	1.76 ±0.030	1.43 ±0.036	**0.70 ±0.010**	0.62 ±0.009
	6	1.78 ±0.015	1.44 ±0.010	0.71 ±0.007	0.63 ±0.006
	7	1.78 ±0.023	1.44 ±0.015	0.70 ±0.007	0.63 ±0.011
16	2	1.79 ±0.021	1.44 ±0.011	**0.70 ±0.014**	0.64 ±0.010
	3	1.79 ±0.011	1.45 ±0.013	0.71 ±0.012	0.63 ±0.010
	4	1.73 ±0.037	1.42 ±0.023	0.72 ±0.013	0.61 ±0.014
	5	**1.72 ±0.018**	**1.40 ±0.014**	0.72 ±0.005	**0.60 ±0.013**
	6	1.73 ±0.034	1.41 ±0.024	0.72 ±0.014	0.60 ±0.011
	7	1.77 ±0.036	1.45 ±0.031	0.72 ±0.011	0.62 ±0.018
27	2	1.75 ±0.039	1.43 ±0.027	**0.72 ±0.021**	0.61 ±0.014
	3	1.73 ±0.038	1.40 ±0.037	0.72 ±0.003	0.60 ±0.007
	4	1.72 ±0.010	1.41 ±0.007	0.73 ±0.003	0.59 ±0.005
	5	**1.67 ±0.024**	**1.38 ±0.015**	0.73 ±0.010	0.58 ±0.014
	6	1.70 ±0.027	1.39 ±0.018	0.74 ±0.012	**0.58 ±0.017**
	7	1.73 ±0.027	1.41 ±0.020	0.73 ±0.011	0.59 ±0.014

Tuning of Architecture Parameters

We tune the architecture parameters number and size k of filters by grid searching over the values listed in Table D.18. Table D.20 contains the average results and their standard deviation. A bold value denotes that this setting is the best among all filter sizes k for a certain number of filters regarding the respective metric. The best value of the whole column is signified by an underlined bold value.

The results show that 27 filters are the best choice for all metrics except for RCS. Regarding PE, $k = 6$ is best whereas $k = 5$ is best for the other metrics. Nevertheless, in the following experiments we employ $k = 6$, since we aim at optimizing the prediction ability of the TCN which mainly is measured by PE.

Tuning of Training Parameters

We conduct a grid search over different values for learning rate, batch size and dropout rate (Table D.18). The results are highlighted as follows (Table D.21). A bold value denotes the best dropout rate with fixed learning rate and batch size whereas an underlined bold value represents the best batch size for a certain learning rate. A boxed value represents the best setting of the whole column, i.e., the best learning rate.

Regarding $\overline{\text{BOG}}$, BEBC and PE, 0.001 is the best learning rate. It can be observed that batch size 8 is almost never the best, whereas for learning rates 0.001 and 0.002 batch size 32 is superior. For learning rate 0.004, batch size 16 is reasonable as well. Since learning rate 0.001 and batch size 32 seem to be promising, we only consider these settings to identify the best dropout rate. Regarding RCS, dropout rate 0.01 is best but for all other metrics rate 0.1 has better values.

Overall the tuning of architecture and training hyperparameters yields 27 filters, filter size 6, learning rate 0.001, batch size 32 and dropout rate 0.1 as best parameters. This setting is employed in the final comparison for both `tcn` and `unc`.

D.3.2 Final Comparisons on DSB and MPB

Tables D.22, D.23 and D.24 comprise pairwise statistical tests between all combinations of prediction method and re-initialization strategy on DSB. Results for corresponding experiments on MPB are listed in Tables D.25, D.26 and D.27.

Table D.21: Results for tuning of training parameters

Learnig R.	Batch S.	Dropout	$\overline{\text{BOG}}$	BEBC	RCS	PE
0.001	8	0.01	**1.68 ±0.009**	**1.38 ±0.015**	0.74 ±0.010	**0.58 ±0.011**
		0.05	1.70 ±0.016	1.40 ±0.011	0.74 ±0.008	0.58 ±0.004
		0.1	1.69 ±0.014	1.40 ±0.017	**0.73 ±0.005**	0.58 ±0.009
	16	0.01	1.71 ±0.030	1.40 ±0.017	**0.73 ±0.016**	0.59 ±0.021
		0.05	**1.66 ±0.042**	**1.37 ±0.030**	0.75 ±0.017	0.56 ±0.015
		0.1	1.68 ±0.017	1.39 ±0.022	0.75 ±0.009	**0.56 ±0.009**
	32	0.01	1.69 ±0.023	1.39 ±0.023	**0.73 ±0.010**	0.58 ±0.003
		0.05	1.68 ±0.018	1.38 ±0.024	0.74 ±0.007	0.56 ±0.009
		0.1	**1.65 ±0.033**	**1.37 ±0.031**	0.75 ±0.012	**0.56 ±0.012**
0.002	8	0.01	1.76 ±0.012	1.43 ±0.013	**0.71 ±0.007**	0.62 ±0.012
		0.05	**1.72 ±0.026**	**1.41 ±0.018**	0.73 ±0.009	**0.60 ±0.014**
		0.1	1.72 ±0.030	1.41 ±0.018	0.72 ±0.007	0.60 ±0.013
	16	0.01	1.71 ±0.034	1.40 ±0.030	**0.72 ±0.015**	0.60 ±0.018
		0.05	**1.70 ±0.020**	1.40 ±0.009	0.73 ±0.014	0.59 ±0.008
		0.1	1.70 ±0.036	**1.39 ±0.032**	0.73 ±0.008	**0.58 ±0.008**
	32	0.01	1.71 ±0.028	1.40 ±0.024	**0.73 ±0.013**	0.59 ±0.019
		0.05	1.69 ±0.032	1.39 ±0.014	0.74 ±0.011	0.58 ±0.020
		0.1	**1.67 ±0.026**	**1.38 ±0.013**	0.73 ±0.020	**0.58 ±0.007**
0.004	8	0.01	1.84 ±0.017	1.47 ±0.015	0.70 ±0.011	0.70 ±0.017
		0.05	1.85 ±0.018	1.49 ±0.015	0.70 ±0.012	0.68 ±0.023
		0.1	**1.79 ±0.001**	**1.45 ±0.015**	**0.69 ±0.006**	**0.65 ±0.016**
	16	0.01	1.84 ±0.042	1.46 ±0.035	**0.69 ±0.018**	0.70 ±0.029
		0.05	1.76 ±0.023	1.43 ±0.011	0.71 ±0.009	0.63 ±0.012
		0.1	**1.75 ±0.021**	**1.42 ±0.024**	0.71 ±0.011	**0.63 ±0.010**
	32	0.01	1.81 ±0.030	1.47 ±0.023	**0.70 ±0.005**	0.68 ±0.019
		0.05	1.76 ±0.034	**1.43 ±0.022**	0.70 ±0.013	0.63 ±0.022
		0.1	**1.75 ±0.022**	1.43 ±0.011	0.71 ±0.007	**0.62 ±0.012**

Table D.22: DSB: pairwise test results for all combinations of re-initialization strategy and prediction method; continued in Table D.23

Alg.	Bmk.	Dim.	npm-nVAR	npm-nPRE	ar-pRND	ar-pDEV	tcn-pRND	tcn-pDEV	kal-pRND	kal-pDEV	kal-pUNC	kal-pKAL	unc-pRND	unc-pDEV	unc-pUNC	unc-pKAL
npm-nRND	Sphere	2	– – △ ·	△ – △ ·	△ – – ·	△ – – ·	△ △ △ ·	△ △ △ ·	△ – △ ·	△ – △ ·	△ △ △ ·	△ – ▼ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
		5	– △ △ ·	– △ △ ·	△ △ – ·	△ △ – ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ – ▼ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
		10	△ △ △ ·	△ – – ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
		20	△ △ △ ·	△ – – ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
	Rosenbrock	2	– △ △ ·	– – △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ – ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
		5	– △ △ ·	– △ △ ·	△ △ – ·	△ △ – ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ ▼ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ – ·
		10	▼ – △ ·	– △ △ ·	△ △ – ·	△ △ – ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ – ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
		20	– – – ·	– – – ·	△ △ – ·	△ △ – ·	△ △ △ ·	△ △ △ ·	△ △ – ·	△ △ – ·	△ △ – ·	△ △ – ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ – ·
	Rastrigin	2	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ – ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
		5	△ △ △ ·	△ – – ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ – – ·	△ △ – ·	△ △ – ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ – – ·
		10	△ △ △ ·	▼ – – ·	△ – △ ·	△ – △ ·	△ △ △ ·	△ △ △ ·	△ – – ·	△ – △ ·	△ – △ ·	△ – – ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ – – ·
		20	△ – △ ·	– – – ·	△ – △ ·	△ – △ ·	△ △ △ ·	△ △ △ ·	△ – △ ·	△ △ △ ·	△ – △ ·	△ – △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
npm-nVAR	Sphere	2		– – – ·	△ – ▼ ·	△ – ▼ ·	△ △ △ ·	△ △ △ ·	△ – ▼ ·	△ – ▼ ·	△ △ ▼ ·	△ – ▼ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
		5		– – ▼ ·	△ – ▼ ·	△ – ▼ ·	△ △ △ ·	△ △ △ ·	△ △ ▼ ·	△ △ ▼ ·	△ △ ▼ ·	△ – ▼ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ – ·
		10		– ▼ ▼ ·	△ ▼ ▼ ·	△ ▼ ▼ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ – ·	△ △ – ·	△ – ▼ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
		20		– ▼ ▼ ·	△ △ – ·	△ △ – ·	△ △ △ ·	△ △ △ ·	△ △ – ·	△ △ – ·	△ △ – ·	△ – ▼ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ ▼ ·
	Rosenbrock	2		– ▼ – ·	△ – ▼ ·	△ – ▼ ·	△ △ △ ·	△ △ △ ·	△ △ – ·	△ – – ·	△ △ – ·	△ – ▼ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ – ·
		5		– – ▼ ·	△ – ▼ ·	△ – ▼ ·	△ △ △ ·	△ △ △ ·	△ △ ▼ ·	△ △ ▼ ·	△ △ ▼ ·	△ – ▼ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ – ▼ ·
		10		– – – ·	△ △ ▼ ·	△ △ ▼ ·	△ △ △ ·	△ △ △ ·	△ △ – ·	△ △ – ·	△ △ – ·	△ △ ▼ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ ▼ ·
		20		– – – ·	△ △ ▼ ·	△ △ ▼ ·	△ △ △ ·	△ △ △ ·	△ △ – ·	△ △ ▼ ·	△ △ ▼ ·	△ △ ▼ ·	△ △ △ ·	△ △ – ·	△ △ – ·	△ △ ▼ ·
	Rastrigin	2		▼ – – ·	– – ▼ ·	▼ – ▼ ·	△ △ △ ·	△ △ △ ·	△ △ – ·	– – – ·	△ △ – ·	▼ ▼ ▼ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
		5		▼ ▼ ▼ ·	– – ▼ ·	▼ – ▼ ·	△ – – ·	– – – ·	– – ▼ ·	▼ – ▼ ·	– – ▼ ·	▼ – ▼ ·	△ – – ·	– – ▼ ·	△ – – ·	▼ ▼ ▼ ·
		10		▼ ▼ ▼ ·	△ – ▼ ·	△ – ▼ ·	△ – – ·	△ – – ·	△ ▼ ▼ ·	△ – ▼ ·	△ – ▼ ·	△ – ▼ ·	△ △ – ·	△ △ – ·	△ – ▼ ·	△ – ▼ ·
		20		▼ – ▼ ·	△ – – ·	△ – – ·	△ △ △ ·	△ △ △ ·	△ – ▼ ·	△ △ – ·	△ ▼ – ·	△ – ▼ ·	△ △ △ ·	△ – – ·	△ △ – ·	△ △ – ·
npm-nPRE	Sphere	2			△ – ▼ ·	△ – ▼ ·	△ △ △ ·	△ △ △ ·	△ – ▼ ·	△ – ▼ ·	△ – ▼ ·	△ – ▼ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
		5			△ △ ▼ ·	△ – ▼ ·	△ △ △ ·	△ △ △ ·	△ △ – ·	△ △ ▼ ·	△ △ ▼ ·	△ – ▼ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
		10			△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
		20			△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
	Rosenbrock	2			△ △ – ·	△ △ – ·	△ △ △ ·	△ △ △ ·	△ △ – ·	△ △ – ·	△ △ – ·	△ – ▼ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ – ·
		5			△ – ▼ ·	△ – ▼ ·	△ △ △ ·	△ △ △ ·	△ △ ▼ ·	△ △ ▼ ·	△ △ ▼ ·	△ △ ▼ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ ▼ ·
		10			△ △ – ·	△ △ – ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ – ·	△ △ – ·	△ △ ▼ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ – ·
		20			△ △ – ·	△ △ – ·	△ △ △ ·	△ △ △ ·	△ △ – ·	△ △ – ·	△ △ – ·	△ △ ▼ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ – ·
	Rastrigin	2			△ – – ·	△ – – ·	△ △ △ ·	△ △ △ ·	△ △ – ·	△ – – ·	△ △ – ·	▼ ▼ ▼ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
		5			△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ – ·	△ – – ·	△ △ – ·	△ △ – ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ – ▼ ·
		10			△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ – △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
		20			△ – △ ·	△ – △ ·	△ △ △ ·	△ △ △ ·	△ – – ·	△ △ △ ·	△ – △ ·	△ – – ·	△ △ △ ·	△ △ △ ·	△ △ △ ·	△ △ △ ·
ar-pRND	Sphere	2				▼ – – –	△ △ △ △	△ △ △ △	△ – – △	△ – – △	△ – – △	▼ – ▼ △	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		5				▼ – – –	△ △ △ △	△ △ △ △	△ △ △ △	△ – △ △	△ △ △ △	△ ▼ ▼ △	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		10				– – – –	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △	△ – – △	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		20				▼ – – –	△ △ △ △	△ △ △ △	△ – – ▼	– – – ▼	△ △ – ▼	▼ ▼ ▼ ▼	△ △ △ △	△ △ △ △	△ △ △ △	△ ▼ – △
	Rosenbrock	2				▼ – – –	△ △ △ △	△ △ △ △	△ △ △ △	△ △ – △	△ △ – △	▼ – ▼ △	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		5				▼ – – –	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △	– – ▼ △	△ △ △ △	△ △ △ △	△ △ △ △	△ – – △
		10				▼ – – –	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △	– – ▼ △	△ △ △ △	△ △ △ △	△ △ △ △	△ – △ △
		20				▼ ▼ – –	△ △ △ △	△ △ △ △	△ – – ▼	– ▼ – ▼	△ – – ▼	▼ ▼ ▼ ▼	△ △ △ △	△ △ △ △	△ △ △ △	– ▼ – △
	Rastrigin	2				▼ – – –	△ △ △ △	△ △ △ △	△ – △ △	– – – △	△ △ – △	▼ ▼ ▼ △	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		5				– – – –	△ △ △ △	△ △ △ △	– – – ▼	▼ – ▼ ▼	– – ▼ ▼	▼ – ▼ ▼	△ – △ △	– – △ △	△ – △ △	▼ ▼ ▼ △
		10				– – – –	△ – – △	– – – △	▼ – – ▼	▼ – – ▼	▼ – – ▼	▼ – ▼ ▼	△ – △ △	△ △ △ △	– – – △	▼ – – △
		20				– – – –	△ – △ –	– – – –	▼ – ▼ ▼	▼ – – ▼	▼ ▼ – ▼	▼ – ▼ ▼	△ △ △ –	– – – –	– – – △	▼ – – △
ar-pDEV	Sphere	2					△ △ △ △	△ △ △ △	△ – – △	△ – – △	△ – – △	▼ – ▼ △	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		5					△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △	△ – ▼ △	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		10					△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △	△ – – △	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		20					△ △ △ △	△ △ △ △	△ △ – ▼	△ – – ▼	△ △ – ▼	▼ ▼ ▼ ▼	△ △ △ △	△ △ △ △	△ △ △ △	△ – – △
	Rosenbrock	2					△ △ △ △	△ △ △ △	△ △ △ △	△ – – △	△ △ – △	▼ ▼ ▼ △	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		5					△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △	△ – ▼ △	△ △ △ △	△ △ △ △	△ △ △ △	△ – – △
		10					△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △	△ – ▼ △	△ △ △ △	△ △ △ △	△ △ △ △	△ – △ △
		20					△ △ △ △	△ △ △ △	△ △ – ▼	△ – – ▼	△ △ – ▼	– – ▼ ▼	△ △ △ △	△ △ △ △	△ △ △ △	△ – – △
	Rastrigin	2					△ △ △ △	△ △ △ △	△ – △ △	△ – – △	△ △ – △	▼ – ▼ △	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		5					△ – △ △	△ – △ △	– – – ▼	– – ▼ ▼	– – ▼ ▼	▼ – ▼ ▼	△ – △ △	△ – △ △	△ – △ △	▼ ▼ ▼ △
		10					△ – – △	△ △ – △	▼ – – ▼	– – – ▼	▼ – – ▼	▼ – ▼ ▼	△ △ △ △	△ △ △ △	– – – △	▼ – – △
		20					△ △ △ –	△ △ – –	▼ – ▼ ▼	– △ – ▼	▼ ▼ – ▼	▼ – ▼ ▼	△ △ △ ▼	△ – – –	△ – – △	– – – –

Table D.23: Continues Table D.22; continued in Table D.24

Alg.	Bmk.	Dim.	npm-nVAR	npm-nPRE	ar-pRND	ar-pDEV	tcn-pRND	tcn-pDEV	kal-pRND	kal-pDEV	kal-pUNC	kal-pKAL	unc-pRND	unc-pDEV	unc-pUNC	unc-pKAL
tcn-pRND	Sphere	2						▼ ▼ ▼ –	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	– – – –	△ △ △ △	△ △ △ △	– – – △
		5						▼ – – –	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	– – – –	▼ – ▼ –	△ △ △ –	▼ ▼ ▼ ▼
		10						▼ ▼ ▼ –	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	– – – –	▼ ▼ ▼ ▼	▼ ▼ ▼ –	▼ ▼ ▼ ▼
		20						▼ ▼ ▼ –	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	– – – –	▼ ▼ ▼ –	▼ ▼ ▼ –	▼ ▼ ▼ ▼
	Rosenbrock	2						▼ ▼ ▼ –	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	– – – –	▼ ▼ ▼ –	△ △ △ –	▼ ▼ ▼ –
		5						▼ ▼ – –	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	– – △ –	▼ – – –	– – △ –	▼ ▼ ▼ –
		10						▼ ▼ ▼ –	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	– – – –	▼ ▼ ▼ –	▼ ▼ ▼ –	▼ ▼ ▼ –
		20						▼ – – –	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	– – – –	▼ – ▼ –	▼ ▼ ▼ –	▼ ▼ ▼ –
	Rastrigin	2						▼ ▼ ▼ –	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	– ▼ – –	– – – –	△ △ △ –	▼ ▼ ▼ –
		5						▼ – – –	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	– – – –	▼ – – –	▼ – – –	▼ ▼ ▼ –
		10						▼ – – –	▼ ▼ ▼ ▼	▼ – ▼ ▼	▼ – ▼ ▼	▼ – ▼ ▼	– – – –	– △ – –	▼ – – –	▼ ▼ ▼ –
		20						▼ – – –	▼ ▼ ▼ ▼	▼ – ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	– – – ▼	– – – –	▼ – – –	▼ – ▼ –
tcn-pDEV	Sphere	2							▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	△ △ △ –	△ △ △ △	△ △ △ △	△ △ △ △
		5							▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	△ △ – –	– – ▼ –	△ △ △ –	▼ ▼ ▼ ▼
		10							▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	△ – △ –	– ▼ ▼ –	– – – –	▼ ▼ ▼ ▼
		20							▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	△ △ △ △	– – △ –	– – – ▼	▼ ▼ ▼ ▼
	Rosenbrock	2							▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	△ △ △ –	– – – –	△ △ △ –	▼ ▼ ▼ –
		5							▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ – ▼ ▼	▼ ▼ ▼ ▼	△ △ △ –	– – – –	△ △ △ –	▼ ▼ ▼ –
		10							▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	△ △ △ –	– – – –	– – – –	▼ ▼ ▼ –
		20							▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	– – △ ▼	– – – –	▼ ▼ ▼ –	▼ ▼ ▼ –
	Rastrigin	2							▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	△ △ △ –	△ △ △ –	△ △ △ –	▼ ▼ ▼ –
		5							▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	△ – – –	– – – –	– – – –	▼ ▼ ▼ △
		10							▼ ▼ ▼ ▼	▼ – ▼ ▼	▼ ▼ ▼ ▼	▼ – ▼ ▼	– – – ▼	– △ – –	– – – –	▼ ▼ ▼ –
		20							▼ ▼ ▼ ▼	▼ – – ▼	▼ ▼ – ▼	▼ ▼ ▼ ▼	△ – – –	– – – –	– – – △	▼ – – –
kal-pRND	Sphere	2								▼ – – –	△ – – ▼	▼ – ▼ ▼	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		5								▼ ▼ ▼ –	△ △ ▼ –	▼ ▼ ▼ –	△ △ △ △	△ △ △ △	△ △ △ △	△ – △ △
		10								▼ – ▼ △	△ △ ▼ △	▼ ▼ ▼ ▼	△ △ △ △	△ △ △ △	△ △ △ △	△ – – △
		20								▼ – ▼ –	△ △ ▼ –	▼ ▼ ▼ –	△ △ △ △	△ △ △ △	△ △ △ △	– ▼ ▼ △
	Rosenbrock	2								▼ – – –	△ △ – –	▼ ▼ ▼ ▼	△ △ △ △	△ △ △ △	△ △ △ △	▼ – – △
		5								▼ – – –	△ △ – –	▼ ▼ ▼ ▼	△ △ △ △	△ △ △ △	△ △ △ △	▼ – ▼ △
		10								▼ ▼ ▼ –	△ △ ▼ –	▼ ▼ ▼ –	△ △ △ △	△ △ △ △	△ △ △ △	▼ ▼ ▼ △
		20								▼ ▼ ▼ –	△ – ▼ –	▼ ▼ ▼ △	△ △ △ △	△ △ △ △	△ △ △ △	▼ ▼ ▼ △
	Rastrigin	2								▼ – – –	△ △ – –	▼ ▼ ▼ –	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		5								▼ – – –	– – – –	▼ – – –	△ – △ △	△ – △ △	△ – △ △	▼ ▼ ▼ △
		10								– – – –	– – – –	▼ – – –	△ △ △ △	△ △ △ △	△ – – △	▼ – – △
		20								△ △ △ △	– – △ –	– – – △	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
kal-pDEV	Sphere	2									△ – – –	▼ ▼ ▼ –	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		5									△ △ – –	▼ ▼ ▼ –	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		10									△ △ – –	▼ ▼ ▼ ▼	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		20									△ △ – –	▼ ▼ ▼ –	△ △ △ △	△ △ △ △	△ △ △ △	– ▼ – △
	Rosenbrock	2									△ △ – –	▼ ▼ ▼ ▼	△ △ △ △	△ △ △ △	△ △ △ △	▼ – – △
		5									△ △ – –	▼ ▼ ▼ –	△ △ △ △	△ △ △ △	△ △ △ △	▼ ▼ – △
		10									△ △ – –	▼ ▼ ▼ –	△ △ △ △	△ △ △ △	△ △ △ △	▼ ▼ ▼ △
		20									△ △ – –	▼ – ▼ –	△ △ △ △	△ △ △ △	△ △ △ △	– – – △
	Rastrigin	2									△ △ – –	▼ ▼ ▼ –	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		5									△ – – –	▼ – – –	△ △ △ △	△ △ △ △	△ △ △ △	▼ – – △
		10									– – – –	▼ – – △	△ △ △ △	△ △ △ △	△ – – △	▼ – – △
		20									▼ ▼ – ▼	▼ ▼ ▼ –	△ – △ △	△ – – △	△ – – △	– – – △
kal-pUNC	Sphere	2										▼ ▼ ▼ –	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		5										▼ ▼ ▼ –	△ △ △ △	△ △ △ △	△ △ △ △	▼ ▼ △ △
		10										▼ ▼ ▼ ▼	△ △ △ △	△ △ △ △	△ △ △ △	▼ ▼ △ △
		20										▼ ▼ ▼ –	△ △ △ △	△ △ △ △	△ △ △ △	▼ ▼ – △
	Rosenbrock	2										▼ ▼ ▼ ▼	△ △ △ △	△ △ △ △	△ △ △ △	▼ – – △
		5										▼ ▼ ▼ –	△ △ △ △	△ △ △ △	△ △ △ △	▼ ▼ – △
		10										▼ ▼ ▼ –	△ △ △ △	△ △ △ △	△ △ △ △	▼ ▼ ▼ △
		20										▼ ▼ ▼ △	△ △ △ △	△ △ △ △	△ △ △ △	▼ ▼ – △
	Rastrigin	2										▼ ▼ ▼ –	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		5										▼ – – –	△ – △ △	– – △ △	△ – △ △	▼ ▼ – △
		10										▼ – – –	△ △ △ △	△ △ △ △	△ – – △	▼ – – △
		20										– – ▼ △	△ △ △ △	△ △ – △	△ △ – △	△ △ – △

Table D.24: Continues Table D.23

Alg.	Bmk.	Dim.	npm-nVAR	npm-nPRE	ar-pRND	ar-pDEV	tcn-pRND	tcn-pDEV	kal-pRND	kal-pDEV	kal-pUNC	kal-pKAL	unc-pRND	unc-pDEV	unc-pUNC	unc-pKAL
kal-pKAL	Sphere	2											△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		5											△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		10											△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		20											△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
	Rosenbrock	2											△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		5											△ △ △ △	△ △ △ △	△ △ △ △	△ – △ △
		10											△ △ △ △	△ △ △ △	△ △ △ △	△ – △ △
		20											△ △ △ △	△ △ △ △	△ △ △ △	△ – △ △
	Rastrigin	2											△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
		5											△ △ △ △	△ – △ △	△ – △ △	– ▼ – △
		10											△ △ △ △	△ △ △ △	△ – △ △	– – – △
		20											△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △
unc-pRND	Sphere	2												△ △ △ △	△ △ △ △	– – – △
		5												▼ ▼ ▼ –	△ – △ –	▼ ▼ ▼ ▼
		10												▼ ▼ ▼ ▼	▼ – – –	▼ ▼ ▼ ▼
		20												▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼
	Rosenbrock	2												▼ ▼ ▼ –	△ △ △ –	▼ ▼ ▼ –
		5												▼ ▼ ▼ –	– – – –	▼ ▼ ▼ –
		10												▼ ▼ ▼ –	▼ ▼ ▼ –	▼ ▼ ▼ –
		20												▼ – ▼ –	▼ ▼ ▼ –	▼ ▼ ▼ –
	Rastrigin	2												– – – –	△ △ △ –	▼ ▼ ▼ –
		5												▼ – – –	▼ – – –	▼ ▼ ▼ –
		10												– – – △	▼ – ▼ –	▼ ▼ ▼ –
		20												– – – △	▼ – – △	▼ – ▼ △
unc-pDEV	Sphere	2													△ △ △ △	▼ ▼ ▼ –
		5													△ △ △ –	▼ ▼ ▼ ▼
		10													– – – –	▼ ▼ ▼ ▼
		20													▼ – ▼ ▼	▼ ▼ ▼ ▼
	Rosenbrock	2													△ △ △ –	▼ ▼ ▼ –
		5													△ △ △ –	▼ ▼ ▼ –
		10													– – – –	▼ ▼ ▼ –
		20													– ▼ ▼ –	▼ ▼ ▼ –
	Rastrigin	2													△ △ △ –	▼ ▼ ▼ –
		5													△ – – –	▼ ▼ ▼ –
		10													▼ ▼ ▼ –	▼ ▼ ▼ –
		20													– – – –	▼ – – –
unc-pUNC	Sphere	2														▼ ▼ ▼ ▼
		5														▼ ▼ ▼ ▼
		10														▼ ▼ ▼ ▼
		20														▼ ▼ ▼ –
	Rosenbrock	2														▼ ▼ ▼ ▼
		5														▼ ▼ ▼ –
		10														▼ ▼ ▼ –
		20														▼ ▼ ▼ –
	Rastrigin	2														▼ ▼ ▼ –
		5														▼ ▼ ▼ –
		10														▼ – ▼ –
		20														▼ – – –

Table D.25: MPB: pairwise test results for all combinations of re-initialization strategy and prediction method; continued in Table D.26

Alg.	Noise	Dim.	npm-nVAR	npm-nPRE	ar-pRND	ar-pDEV	tcn-pRND	tcn-pDEV	kal-pRND	kal-pDEV	kal-pUNC	kal-pKAL	unc-pRND	unc-pDEV	unc-pUNC	unc-pKAL
npm-nRND	0.00	2	▼▼▼·	▼▼▼·	△△△·	△–△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·
		5	––▼·	––▼·	–––·	△▼–·	–––·	△–△·	––△·	△–△·	––△·	△–△·	–––·	△–△·	––△·	△–△·
		10	△△△·	–––·	–▼△·	–▼△·	––△·	–▼–·	△––·	–▼–·	△–△·	–▼–·	––△·	–▼△·	––△·	–▼△·
		20	△△△·	––△·	–▼–·	–▼–·	–▼△·	–▼–·	–▼–·	–▼–·	△▼△·	–▼–·	–▼△·	–▼–·	–▼△·	–▼△·
	0.01	2	▼▼▼·	▼▼▼·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·
		5	–––·	–––·	△––·	△––·	△––·	–––·	△––·	△––·	–––·	△––·	–––·	–––·	–––·	△––·
		10	▼▼▼·	▼▼▼·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·
		20	△△△·	––△·	△▼△·	△▼△·	△▼△·	△▼△·	△▼–·	△▼–·	△▼△·	△▼△·	△▼△·	△▼△·	△▼△·	△▼△·
	0.05	2	▼▼▼·	▼▼▼·	△▼–·	△▼–·	△▼–·	△▼–·	△▼–·	△▼–·	△▼–·	△––·	△▼–·	△▼–·	△▼–·	△––·
		5	▼▼▼·	▼▼▼·	△––·	△––·	△––·	△––·	△–△·	△––·	△–△·	△–△·	△––·	△––·	△––·	△––·
		10	▼▼–·	▼▼▼·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·
		20	––△·	–––·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·
npm-nVAR	0.00	2		–––·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·
		5		–––·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·
		10		▼▼▼·	–▼–·	–▼–·	–▼△·	–▼–·	△––·	–▼–·	△–△·	–▼–·	–▼△·	–▼–·	–▼△·	–▼△·
		20		▼▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼▼·	–▼▼·	△▼△·	–▼▼·	–▼–·	–▼–·	–▼–·	–▼–·
	0.01	2		–––·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·
		5		–––·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·
		10		▼▼▼·	△▼–·	△▼–·	△▼△·	△▼△·	△▼–·	△▼–·	△▼△·	△▼–·	△▼△·	△▼–·	△▼△·	△▼△·
		20		▼▼–·	△▼△·	–▼△·	△▼△·	–▼–·	–▼▼·	–▼▼·	△▼△·	–▼▼·	△▼△·	△▼–·	△▼△·	–▼△·
	0.05	2		–––·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·
		5		–––·	△–△·	△–△·	△–△·	△–△·	△△△·	△–△·	△△△·	△–△·	△–△·	△–△·	△–△·	△–△·
		10		––▼·	△▼–·	△▼–·	△▼–·	△▼–·	△▼–·	△▼–·	△▼△·	△▼–·	△▼–·	△▼–·	△▼–·	△▼–·
		20		––▼·	–▼–·	–▼–·	–▼–·	–▼–·	–▼▼·	–▼▼·	–▼–·	–▼▼·	–▼–·	–▼▼·	–▼–·	–▼–·
npm-nPRE	0.00	2			△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·
		5			△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·
		10			–▼△·	–▼△·	––△·	–▼△·	△––·	–▼–·	△–△·	–▼–·	––△·	–▼△·	––△·	–▼△·
		20			–▼–·	–▼–·	–▼–·	–▼–·	–▼▼·	–▼▼·	△▼△·	–▼▼·	–▼–·	–▼–·	–▼–·	–▼–·
	0.01	2			△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·
		5			△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·	△–△·
		10			△▼△·	△▼△·	△▼△·	△▼△·	△▼△·	△▼△·	△▼△·	△▼△·	△▼△·	△▼△·	△▼△·	△▼△·
		20			△▼△·	△▼△·	△▼△·	△▼–·	△▼▼·	△▼▼·	△▼△·	△▼▼·	△▼△·	△▼–·	△▼△·	△▼△·
	0.05	2			△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·	△△△·
		5			△–△·	△–△·	△–△·	△–△·	△△△·	△–△·	△△△·	△△△·	△–△·	△–△·	△–△·	△–△·
		10			△▼△·	△▼△·	△▼△·	△▼–·	△▼–·	△▼–·	△▼△·	△▼–·	△▼△·	△▼–·	△▼△·	△▼△·
		20			–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·	–▼–·
ar-pRND	0.00	2				▼–––	––––	▼–––	△––△	–––△	–––△	△△–△	––––	▼–––	––––	––––
		5				––––	––––	––––	––△–	––△–	––△–	△△△–	––––	––––	––△–	––△–
		10				––––	––––	––––	△△▼–	▼–▼–	△△△–	–––△	––––	––––	––––	––––
		20				▼▼––	––––	▼▼▼–	––▼△	▼▼▼△	△△△–	▼▼▼△	––––	▼▼▼–	––––	▼▼––
	0.01	2				▼▼––	△△△–	▼▼▼–	△△△△	△△△△	△△△△	△△△△	△△△–	▼▼▼–	△△△–	▼–△–
		5				––––	△–––	––––	△△––	△△––	△–––	△△––	––––	––––	––––	––––
		10				––––	––––	––––	△△––	––––	––––	––––	△–△–	––––	––––	––––
		20				▼▼––	––––	––▼–	––▼–	▼▼▼–	––△–	––▼–	––––	––▼–	––––	––––
	0.05	2				––––	––––	––––	––––	––––	––––	△△–△	––––	––––	––––	△△––
		5				––––	△–––	––––	△△△△	△△–△	△△△△	△△△△	△–––	––––	△△––	––––
		10				––––	––––	––––	––––	––––	––––	––––	––––	––––	––––	––––
		20				––––	––––	––▼–	––▼–	––▼–	––△–	––▼–	––––	––▼–	––––	––––
ar-pDEV	0.00	2					△△––	––––	△△–△	△△–△	△△–△	△△–△	△–––	△–––	△–––	––––
		5					––––	––––	––△–	––△–	––△–	––△–	––––	––––	––△–	––△–
		10					△△––	––––	△△▼–	––▼–	△△△–	––––	△△––	––––	△△––	––––
		20					△△––	––▼–	△△▼△	▼▼▼–	△△△–	▼▼▼△	△△––	––▼–	△△––	––––
	0.01	2					△△△–	△–▼–	△△△△	△△△△	△△△△	△△△△	△△△–	△△▼–	△△△–	△△△–
		5					––––	––––	––––	△–––	––––	△△––	––––	––––	––––	––––
		10					––––	––––	––––	––––	––––	––––	––△–	––––	––––	––––
		20					△△––	––▼–	––▼–	––▼–	△△△–	––▼–	△△––	––▼–	△△––	––––
	0.05	2					––––	––––	––––	––––	––––	△△–△	––––	––––	––––	–△–△
		5					△△––	––––	△△△△	△△–△	△△△△	△△△△	△△––	––––	△△––	––––
		10					––––	––––	––––	––––	△△––	––––	––––	––––	––––	––––
		20					––––	––▼–	––▼–	––▼–	△△△–	––▼–	––––	––▼–	––––	––––

Table D.26: Continues Table D.25; continued in Table D.27

Alg.	Noise	Dim.	npm-nVAR	npm-nPRE	ar-pRND	ar-pDEV	tcn-pRND	tcn-pDEV	kal-pRND	kal-pDEV	kal-pUNC	kal-pKAL	unc-pRND	unc-pDEV	unc-pUNC	unc-pKAL
tcn-pRND	0.00	2						– – – –	△ – – △	– – – △	△ – – △	△ △ – △	– – – –	– – – –	– – – –	– – – –
		5						– – – –	– – – –	– – – –	– – △ –	– – △ –	– – – –	– – – –	– – – –	– – – –
		10						▼ ▼ ▼ –	△ △ ▼ –	▼ ▼ ▼ –	△ △ △ –	– – ▼ –	– – – –	▼ ▼ ▼ –	– – – –	▼ ▼ – –
		20						▼ ▼ ▼ –	– – ▼ –	▼ ▼ ▼ –	△ △ △ –	▼ ▼ ▼ –	– – – –	▼ ▼ ▼ –	– – – –	▼ ▼ – –
	0.01	2						▼ ▼ ▼ –	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △	– ▼ – –	▼ ▼ ▼ –	– – – –	▼ ▼ – –
		5						▼ ▼ – –	△ – – –	– – – –	– – – –	– – – △	– – – –	– – – –	– – – –	– – – –
		10						– – – –	△ △ – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –
		20						▼ ▼ ▼ –	– – ▼ –	▼ ▼ ▼ –	– – △ –	– – ▼ –	– – – –	– – ▼ –	– – – –	– – – –
	0.05	2						– – – –	– – – –	– – – –	– – – –	△ △ – △	– – – –	– – – –	– – – –	△ △ – –
		5						▼ ▼ – –	△ △ △ △	△ △ – △	△ △ △ △	△ △ △ △	– – – –	▼ ▼ – –	– – – –	▼ ▼ – –
		10						– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –
		20						– – ▼ –	– – ▼ –	– – ▼ –	– – △ –	– – ▼ –	– – – –	– – ▼ –	– – – –	– – – –
tcn-pDEV	0.00	2							△ – – △	△ – – △	△ – – –	△ △ – △	– – – –	– – – –	– – – –	– – – –
		5							– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –
		10							△ △ ▼ –	– – ▼ –	△ △ △ –	– – – –	△ △ △ –	– – – –	△ △ △ –	– – △ –
		20							△ △ ▼ –	▼ ▼ ▼ –	△ △ △ –	▼ ▼ ▼ –	△ △ △ –	– – – –	△ △ △ –	– – △ –
	0.01	2							△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ △	△ △ △ –	– – – –	△ △ △ –	– – △ –
		5							△ △ △ –	△ △ – –	△ – – –	△ △ – –	– – – –	– – – –	– – – –	– – – –
		10							△ △ – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –
		20							– – ▼ –	▼ ▼ ▼ –	△ △ △ –	– – ▼ –	△ △ △ –	– – – –	△ △ △ –	– – △ –
	0.05	2							– – – –	– – – –	– – – –	△ △ – △	– – – –	– – – –	– – – –	– – – –
		5							△ △ △ △	△ △ – △	△ △ △ △	△ △ △ △	△ △ – –	– – – –	△ △ – –	– – – –
		10							△ – – –	– – – –	△ △ △ –	– – – –	– – – –	– – – –	– – – –	– – – –
		20							– – – –	– – – –	△ △ △ –	– – ▼ –	– – △ –	– – – –	– – △ –	– – △ –
kal-pRND	0.00	2								– – – –	– – – –	– – – –	▼ – – ▼	▼ – – ▼	▼ – – ▼	▼ – – ▼
		5								– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –
		10								▼ ▼ – –	△ △ △ –	▼ ▼ △ –	▼ ▼ △ –	▼ ▼ △ –	▼ ▼ △ –	▼ ▼ △ –
		20								▼ ▼ – –	△ △ △ –	▼ ▼ – –	– – △ –	– – △ –	– – △ –	▼ ▼ △ –
	0.01	2								▼ ▼ – ▼	△ △ △ △	▼ ▼ △ –	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼
		5								– – – –	– – – –	– – – –	▼ ▼ – –	▼ ▼ – –	▼ ▼ – –	– – – –
		10								– – – –	– – – ▼	– – – –	▼ ▼ – –	▼ ▼ – –	▼ ▼ – –	– – – –
		20								▼ ▼ – –	△ △ △ –	– – – –	△ △ △ –	– – △ –	△ △ △ –	– – △ –
	0.05	2								– – – –	– – – –	△ △ – △	– – – –	– – – –	– – – –	△ △ – –
		5								▼ ▼ – –	△ △ – –	▼ ▼ – –	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼
		10								– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –
		20								– – – –	– – △ –	– – – –	– – △ –	– – – –	– – △ –	– – △ –
kal-pDEV	0.00	2									– – – –	– – – –	▼ – – ▼	▼ – – ▼	– – – ▼	▼ – – ▼
		5									– – – –	– – – –	– – – –	– – – –	– – – –	– – – –
		10									△ △ △ –	△ – △ –	△ △ △ –	△ – △ –	△ △ △ –	– – △ –
		20									△ △ △ –	– – – –	△ △ △ –	△ △ △ –	△ △ △ –	△ △ △ –
	0.01	2									△ △ △ △	△ △ △ –	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼
		5									– – – –	– – – –	– – – –	▼ – – –	– – – –	– – – –
		10									– – – ▼	– – – –	– – – –	– – – –	– – – –	– – – –
		20									△ △ △ –	– – – –	△ △ △ –	△ △ △ –	△ △ △ –	△ △ △ –
	0.05	2									– – – –	△ △ – △	– – – –	– – – –	– – – –	△ △ – △
		5									△ △ △ –	△ △ △ –	▼ – – ▼	▼ ▼ – ▼	▼ ▼ – ▼	▼ ▼ – ▼
		10									△ △ – –	– – – –	– – – –	– – – –	– – – –	– – – –
		20									△ △ △ –	– – – –	– – △ –	– – – –	– – △ –	– – △ –
kal-pUNC	0.00	2										– – – –	▼ – – ▼	▼ – – –	▼ – – –	▼ – – ▼
		5										– – – –	▼ – ▼ –	– – – –	– – – –	– – – –
		10										▼ ▼ ▼ –	▼ ▼ ▼ ▼	▼ ▼ ▼ –	▼ ▼ ▼ –	▼ ▼ ▼ –
		20										▼ ▼ ▼ –	▼ ▼ ▼ –	▼ ▼ ▼ –	▼ ▼ ▼ –	▼ ▼ ▼ –
	0.01	2										▼ ▼ – ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼
		5										– – – –	▼ – – –	– – – –	– – – –	– – – –
		10										– – – △	– – – –	– – – –	– – – –	– – – –
		20										▼ ▼ ▼ –	– – ▼ –	▼ ▼ ▼ –	– – ▼ –	▼ ▼ ▼ –
	0.05	2										△ △ – △	– – – –	– – – –	– – – –	△ △ – –
		5										▼ ▼ – –	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼
		10										– – – –	▼ ▼ – –	– – – –	▼ ▼ – –	– – – –
		20										▼ ▼ ▼ –	– – ▼ –	▼ ▼ ▼ –	– – – –	▼ ▼ – –

Table D.27: Continues Table D.26

Alg.	Noise	Dim.	npm-nVAR	npm-nPRE	ar-pRND	ar-pDEV	tcn-pRND	tcn-pDEV	kal-pRND	kal-pDEV	kal-pUNC	kal-pKAL	unc-pRND	unc-pDEV	unc-pUNC	unc-pKAL
kal-pKAL	0.00	2											▼ ▼ – ▼	▼ ▼ – ▼	▼ ▼ – ▼	▼ ▼ – ▼
		5											– – ▼ –	– – – –	– – – –	– – – –
		10											– – △ ▼	– – – –	– – △ –	– – △ –
		20											△ △ △ –	△ △ △ –	△ △ △ –	△ △ △ –
	0.01	2											▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼
		5											– – – –	▼ ▼ – –	– – – ▼	▼ ▼ – –
		10											– – – –	– – – –	– – – –	– – – –
		20											– – △ –	– – △ –	– – △ –	– – △ –
	0.05	2											▼ ▼ – ▼	▼ ▼ – ▼	▼ ▼ – ▼	– – – ▼
		5											▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼	▼ ▼ ▼ ▼
		10											– – – –	– – – –	– – – –	– – – –
		20											– – △ –	– – – –	– – △ –	– – △ –
unc-pRND	0.00	2												– – – –	– – – –	– – – –
		5												– – – –	– – – –	– – – –
		10												▼ ▼ ▼ –	– – – –	▼ ▼ – –
		20												▼ ▼ ▼ –	– – – –	▼ ▼ – –
	0.01	2												▼ ▼ ▼ –	– △ – –	▼ ▼ – –
		5												– – – –	– – – –	– – – –
		10												– – – –	– – – –	– – – –
		20												– – ▼ –	– – – –	– – – –
	0.05	2												– – – –	– – – –	△ △ – –
		5												▼ ▼ – –	– – – –	▼ ▼ – –
		10												– – – –	– – – –	– – – –
		20												– – ▼ –	– – – –	– – – –
unc-pDEV	0.00	2													– – – –	– – – –
		5													– – – –	– – – –
		10													△ △ △ –	– – △ –
		20													△ △ △ –	– – △ –
	0.01	2													△ △ △ –	– – △ –
		5													– – – –	– – – –
		10													– – – –	– – – –
		20													– – △ –	– – △ –
	0.05	2													– – – –	△ △ – △
		5													△ △ – –	– – – –
		10													– – – –	– – – –
		20													– – △ –	– – △ –
unc-pUNC	0.00	2														– – – –
		5														– – – –
		10														▼ ▼ – –
		20														▼ ▼ – –
	0.01	2														▼ ▼ – –
		5														– – – –
		10														– – – –
		20														– – – –
	0.05	2														– △ – △
		5														▼ ▼ – –
		10														– – – –
		20														– – – –

E Implementation

The algorithms, the benchmark generator, and the convergence measure proposed in this thesis are implemented in PYTHON; the code is available on GitHub.[1] The PYTHON packages `numpy`, `scipy` and `scikit-learn` are mainly used. For the prediction methods, we employ the following packages and libraries:

- Autoregressive model: `statsmodels`[2]
- Kalman filter: `pykalman`[3]
- RNN: Keras[4] library for neural networks
- TCN: Our implementation relies on the code written by [Hsi18] for Tensorflow;[5] it corresponds to the architecture proposed by [BKK18]. In order to estimate the predictive uncertainty we modified the code: we added an output layer for aleatoric uncertainty, see Paragraph 9.1.2, and trained it with the loss function suggested by [OZK18].

[1] `https://github.com/almuthmeier/DynOpt`, last access 2019/12/05

[2] `http://www.statsmodels.org/0.6.1/vector_ar.html`, last access 2019/12/05

[3] `https://pykalman.github.io/`, last access 2019/12/05

[4] `https://keras.io/`, last access 2019/12/05

[5] `https://www.tensorflow.org/`, last access 2019/12/05

Bibliography

[Ahm+10] Nesreen K. Ahmed et al. “An empirical comparison of machine learning models for time series forecasting.” In: *Econometric Reviews* 29.5-6 (2010), pp. 594–621 (cit. on pp. 32, 55, 56).

[Ahr+19] Ali Ahrari et al. “A new prediction approach for dynamic multiobjective optimization.” In: *Congress on Evolutionary Computation (CEC)*. 2019, pp. 2268–2275 (cit. on pp. 9, 10, 25, 26, 54).

[AL12] Chun-Kit Au and Ho-Fung Leung. “An empirical comparison of CMA-ES in dynamic environments.” In: *Parallel Problem Solving from Nature (PPSN)*. 2012, pp. 529–538 (cit. on p. 108).

[And+18] Simon Anderer et al. “Meta heuristics for dynamic machine scheduling: A review of research efforts and Industrial requirements.” In: *International Joint Conference on Computational Intelligence (IJCCI)*. 2018, pp. 192–203 (cit. on p. 1).

[AP98] Uri M. Ascher and Linda R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, 1998 (cit. on p. 2).

[Ase+13] Daniel Asenjo et al. “Visualizing basins of attraction for different minimization algorithms.” In: *The Journal of Physical Chemistry B* 117.42 (2013), pp. 12717–12723 (cit. on p. 25).

[ATE17] Lokman Altin, Haluk Rahmi Topcuoglu, and Murat Ermis. “Hybridizing change detection schemes for dynamic optimization problems.” In: *Congress on Evolutionary Computation (CEC)*. 2017, pp. 2086–2093 (cit. on pp. 26, 57).

[BB04] Tim Blackwell and Jürgen Branke. “Multi-swarm optimization in dynamic environments.” In: *Applications of Evolutionary Computing, EvoWorkshops*. 2004, pp. 489–500 (cit. on p. 69).

[BBL08] Tim Blackwell, Jürgen Branke, and Xiaodong Li. “Particle swarms for dynamic optimization problems.” In: *Swarm Intelligence: Introduction and Applications*. Springer, 2008, pp. 193 –217 (cit. on pp. 22, 25).

[BE06] Frans van den Bergh and Andries P. Engelbrecht. “A study of particle swarm optimization particle trajectories.” In: *Information Sciences* 176.8 (2006), pp. 937–971 (cit. on p. 74).

[Bel15] Richard Bellman. *Adaptive Control Processes - A Guided Tour (Reprint from 1961)*. Vol. 2045. Princeton Legacy Library. Princeton University Press, 2015 (cit. on p. 11).

[BH12] Robert G. Brown and Patrick Y. C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering*. 4th Edition. John Wiley & Sons, Inc., 2012 (cit. on p. 31).

[Bis07] Christopher M. Bishop. *Pattern recognition and machine learning*. Information science and statistics. Springer, 2007 (cit. on pp. 29, 56, 120).

[BKK18] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling.” In: *CoRR* abs/1803.01271 (2018) (cit. on pp. 33, 34, 55, 85, 143).

[BL08] Christian Blum and Xiaodong Li. “Swarm intelligence in optimization.” In: *Swarm Intelligence: Introduction and Applications*. Springer, 2008, pp. 43–85 (cit. on p. 21).

[Bla07] Tim Blackwell. “Particle swarm optimization in dynamic environments.” In: *Evolutionary Computation in Dynamic and Uncertain Environments*. Springer, 2007, pp. 29–49 (cit. on p. 69).

[BLA16] Yesnier Bravo, Gabriel Luque, and Enrique Alba. “Global memory schemes for dynamic optimization.” In: *Natural Computing* 15.2 (2016), pp. 319–333 (cit. on pp. 19, 26).

[BLP07] Peter A. N. Bosman and Han La Poutré. “Learning and anticipation in online dynamic optimization with evolutionary algorithms: The stochastic case.” In: *Conference on Genetic and Evolutionary Computation (GECCO)*. 2007, pp. 1165–1172 (cit. on p. 28).

[BLY17] Chenyang Bu, Wenjian Luo, and Lihua Yue. “Continuous dynamic constrained optimization with ensemble of locating and tracking feasible regions strategies.” In: *Transactions on Evolutionary Computation* 21.1 (2017), pp. 14–33 (cit. on pp. 70, 71, 75, 86).

[Bos+19] Jakob Bossek et al. “Bi-objective orienteering: Towards a dynamic multi-objective evolutionary algorithm.” In: *Evolutionary Multi-Criterion Optimization (EMO)*. 2019, pp. 516–528 (cit. on p. 1).

[Bos05] Peter A. N. Bosman. “Learning, anticipation and time-deception in evolutionary online dynamic optimization.” In: *Genetic and Evolutionary Computation Conference (GECCO) Workshops*. 2005, pp. 39–47 (cit. on pp. 28, 54).

[Bos07] Peter A. N. Bosman. “Learning and anticipation in online dynamic optimization.” In: *Evolutionary Computation in Dynamic and Uncertain Environments*. Springer, 2007, pp. 129–152 (cit. on p. 13).

[Bou05] Amine Boumaza. “Learning environment dynamics from self-adaptation: A preliminary investigation.” In: *Genetic and Evolutionary Computation Conference (GECCO) Workshops*. 2005, pp. 48–54 (cit. on p. 108).

[Bra+00] Jürgen Branke et al. “A multi-population approach to dynamic optimization problems.” In: *Evolutionary Design and Manufacture*. Ed. by I. C. Parmee. Springer, 2000, pp. 299–307 (cit. on p. 26).

[Bra02] Jürgen Branke. *Evolutionary Optimization in Dynamic Environments*. Springer, 2002 (cit. on p. 10).

[Bra99] Jürgen Branke. “Memory enhanced evolutionary algorithms for changing optimization problems.” In: *Congress on Evolutionary Computation (CEC)*. 1999, pp. 1875–1882 (cit. on pp. 35, 36, 60).

[BRAK13] Hajer Ben-Romdhane, Enrique Alba, and Saoussen Krichen. “Best practices in measuring algorithm performance for dynamic optimization problems.” In: *Soft Computing* 17.6 (2013), pp. 1005–1017 (cit. on pp. 25, 47).

[BS02] Hans-Georg Beyer and Hans-Paul Schwefel. “Evolution strategies–A comprehensive introduction.” In: *Natural Computing* 1.1 (2002), pp. 3–52 (cit. on pp. 19, 58).

[BS07] Hans-Georg Beyer and Bernhard Sendhoff. “Robust optimization–A comprehensive survey.” In: *Computer Methods in Applied Mechanics and Engineering* 196.33 (2007), pp. 3190–3218 (cit. on p. 13).

[BSU05] Jürgen Branke, Erdem Salihoglu, and Sima Uyar. “Towards an analysis of dynamic environments.” In: *Genetic and Evolutionary Computation Conference (GECCO)*. 2005, pp. 1433–1440 (cit. on p. 11).

[BTB12] Gianluca Bontempi, Souhaib Ben Taieb, and Yann-Aël Le Borgne. “Machine learning strategies for time series forecasting.” In: *Business Intelligence–Second European Summer School (eBISS)*. 2012, pp. 62–77 (cit. on pp. 29, 55).

[Bu+17] Chenyang Bu et al. “Solving online dynamic time-linkage problems under unreliable prediction.” In: *Applied Soft Computing* 56 (2017), pp. 702–716 (cit. on pp. 27, 28).

[BV11] Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. 9th edition. Cambridge University Press, 2011 (cit. on pp. 9, 11, 17).

[Cao+17] Leilei Cao et al. “A first-order difference model-based evolutionary dynamic multiobjective optimization.” In: *Simulated Evolution and Learning (SEAL)*. 2017, pp. 644–655 (cit. on p. 54).

[Cao+18] Leilei Cao et al. “A differential prediction model for evolutionary dynamic multiobjective optimization.” In: *Genetic and Evolutionary Computation Conference (GECCO)*. 2018, pp. 601–608 (cit. on p. 54).

[CB07] Eduardo F. Camacho and Carlos Bordons. *Model Predictive Control*. Springer, 2007 (cit. on p. 2).

[CE14a] Christopher W. Cleghorn and Andries P. Engelbrecht. “Particle swarm convergence: An empirical investigation.” In: *Congress on Evolutionary Computation (CEC)*. 2014, pp. 2524–2530 (cit. on p. 74).

[CE14b] Christopher Wesley Cleghorn and Andries P. Engelbrecht. “A generalized theoretical deterministic particle swarm model.” In: *Swarm Intelligence* 8.1 (2014), pp. 35–59 (cit. on p. 74).

[CE15] Christopher W. Cleghorn and Andries P. Engelbrecht. “Particle swarm variants: Standardized convergence analysis.” In: *Swarm Intelligence* 9.2 (2015), pp. 177–203 (cit. on pp. 22, 74).

[CE17] Christopher W. Cleghorn and Andries P. Engelbrecht. “Fitness-distance-ratio particle swarm optimization: Stability analysis.” In: *Genetic and Evolutionary Computation Conference (GECCO)*. 2017, pp. 12–18 (cit. on p. 74).

[Cel91] François E. Cellier. *Continuous system modeling*. Springer, 1991 (cit. on p. 31).

[CGP11] Carlos Cruz, Juan R. González, and David A. Pelta. “Optimization in dynamic environments: A survey on problems, methods and measures.” In: *Soft Computing* 15.7 (2011), pp. 1427–1448 (cit. on pp. 25, 26, 47, 86).

[CHN11] Sven F. Crone, Michèle Hibon, and Konstantinos Nikolopoulos. “Advances in forecasting with neural networks? Empirical evidence from the NN3 competition on time series prediction.” In: *International Journal of Forecasting* 27.3 (2011), pp. 635 –660 (cit. on p. 55).

[Cho+18] Li-Der Chou et al. “DPRA: Dynamic power-saving resource allocation for cloud data center using particle swarm optimization.” In: *IEEE Systems Journal* 12.2 (2018), pp. 1554–1565 (cit. on pp. 1, 22).

[CLY18] Renzhi Chen, Ke Li, and Xin Yao. “Dynamic multiobjectives optimization with a changing number of objectives.” In: *Transactions on Evolutionary Computation* 22.1 (2018), pp. 157–171 (cit. on p. 12).

[CMP16] Jenny F. Calderín, Antonio D. Masegosa, and David A. Pelta. “Dynamic optimization with restricted and unrestricted moves between changes: A study on the dynamic maximal covering location problem.” In: *Congress on Evolutionary Computation (CEC)*. 2016, pp. 570–577 (cit. on p. 13).

[Cob90] Helen G. Cobb. *An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments.* Tech. rep. Naval Research Lab Washington DC, 1990 (cit. on p. 26).

[CY13] Hui Cheng and Shengxiang Yang. "Genetic algorithms for dynamic routing problems in mobile ad hoc networks." In: *Evolutionary Computation for Dynamic Optimization Problems.* Ed. by Shengxiang Yang and Xin Yao. Springer, 2013, pp. 343–375 (cit. on p. 26).

[Dha+19] Krishna G. Dhal et al. "A survey on nature-inspired optimization algorithms and their application in image enhancement domain." In: *Archives of Computational Methods in Engineering* 26.5 (2019), pp. 1607–1638 (cit. on p. 18).

[DS04] Marco Dorigo and Thomas Stützle. *Ant colony optimization.* MIT Press, 2004 (cit. on p. 21).

[Eng10] Andries P. Engelbrecht. "Heterogeneous particle swarm optimization." In: *Swarm Intelligence (ANTS).* 2010, pp. 191–202 (cit. on p. 69).

[Eng13] Andries P. Engelbrecht. "Particle swarm optimization: Global best or local best?" In: *BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence.* 2013, pp. 124–135 (cit. on p. 22).

[ES15] A. E. Eiben and James E. Smith. *Introduction to Evolutionary Computing.* 2nd edition. Natural Computing Series. Springer, 2015 (cit. on pp. 11, 17, 19, 21).

[EYG17] Jayne Eaton, Shengxiang Yang, and Mario Gongora. "Ant colony optimization for simulated dynamic multi-objective railway junction rescheduling." In: *Transactions on Intelligent Transportation Systems* 18.11 (2017), pp. 2980–2992 (cit. on p. 1).

[FKG18] Bjarne A. Foss, Brage R. Knudsen, and Bjarne Grimstad. "Petroleum production optimization–A static or dynamic problem?" In: *Computers & Chemical Engineering* 114 (2018), pp. 245–253 (cit. on p. 1).

[FO07] Robert Fildes and Keith Ord. “Forecasting competitions: Their role in improving forecasting practice and research.” In: *A Companion to Economic Forecasting.* John Wiley & Sons, Ltd, 2007. Chap. 15, pp. 322–353 (cit. on p. 55).

[FS17] Xiaogang Fu and Jianyong Sun. “A new learning based dynamic multi-objective optimisation evolutionary algorithm.” In: *Congress on Evolutionary Computation (CEC).* 2017, pp. 341–348 (cit. on pp. 27, 54).

[Fu+12] Haobo Fu et al. “Characterizing environmental changes in robust optimization over time.” In: *Congress on Evolutionary Computation (CEC).* 2012, pp. 1–8 (cit. on pp. 11, 13).

[Fu+13] Haobo Fu et al. “Finding robust solutions to dynamic optimization problems.” In: *Applications of Evolutionary Computation.* 2013, pp. 616–625 (cit. on p. 13).

[Fu+14] Haobo Fu et al. “What are dynamic optimization problems?” In: *Congress on Evolutionary Computation (CEC).* 2014, pp. 1550–1557 (cit. on pp. 10, 13).

[Gal16] Yarin Gal. “Uncertainty in Deep Learning.” PhD thesis. University of Cambridge, 2016 (cit. on pp. 87, 88).

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016 (cit. on pp. 31, 32).

[Geo+17] Koshy George et al. “Comparison of neural-network learning algorithms for time-series prediction.” In: *International Conference on Advances in Computing, Communications and Informatics (ICACCI).* 2017, pp. 7–13 (cit. on p. 55).

[GL97] Fred W. Glover and Manuel Laguna. *Tabu Search.* Kluwer, 1997 (cit. on p. 18).

[Gre92] John J. Grefenstette. “Genetic algorithms for changing environments.” In: *Parallel Problem Solving from Nature (PPSN).* 1992, pp. 139–146 (cit. on p. 26).

[Gre99] John J. Grefenstette. “Evolvability in dynamic fitness landscapes: A genetic algorithm approach.” In: *Congress on Evolutionary Computation (CEC)* (1999) (cit. on p. 27).

[GT09] Chi Keong Goh and Kay Chen Tan. “A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization.” In: *Transactions on Evolutionary Computation* 13.1 (2009), pp. 103–127 (cit. on p. 25).

[GTA17] Sen B. Gee, Kay C. Tan, and Hussein A. Abbass. “A benchmark test suite for dynamic evolutionary multiobjective optimization.” In: *Transaction on Cybernetics* 47.2 (2017), pp. 461–472 (cit. on p. 35).

[HA13] Rob J. Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts: Melbourne, Australia. Last access 2019/12/05. 2013. URL: http://otexts.org/fpp/ (cit. on pp. 30, 31).

[HAA15] Nikolaus Hansen, Dirk V. Arnold, and Anne Auger. “Evolution strategies.” In: *Springer Handbook of Computational Intelligence*. Springer, 2015, pp. 871–898 (cit. on pp. 19, 21).

[Has+19] Maryam Hasani-Shoreh et al. “On the behaviour of differential evolution for problems with dynamic linear constraints.” In: *Congress on Evolutionary Computation (CEC)*. 2019, pp. 3045–3052 (cit. on p. 25).

[HDM13] Udit Halder, Swagatam Das, and Dipankar Maity. “A cluster-based differential evolution algorithm with external archive for optimization in dynamic environments.” In: *Transactions on Cybernetics* 43.3 (2013), pp. 881–897 (cit. on p. 25).

[HE02] Xiaohui Hu and Russel C. Eberhart. “Adaptive particle swarm optimization: Detection and response to dynamic systems.” In: *Congress on Evolutionary Computation (CEC)*. 2002, pp. 1666–1670 (cit. on pp. 70, 72).

[Hem+01] Jano van Hemert et al. *A “Futurist” approach to dynamic environments*. 2001 (cit. on p. 27).

[HEOB16a] Kyle R. Harrison, Andries P. Engelbrecht, and Beatrice M. Ombuki-Berman. “Inertia weight control strategies for particle swarm optimization.” In: *Swarm Intelligence* 10.4 (2016), pp. 267–305 (cit. on pp. 23, 70).

[HEOB16b] Kyle R. Harrison, Andries P. Engelbrecht, and Beatrice M. Ombuki-Berman. “The sad state of self-adaptive particle swarm optimizers.” In: *Congress on Evolutionary Computation (CEC)*. 2016, pp. 431–439 (cit. on pp. 23, 74).

[HMR09] Tim Hendtlass, Irene Moser, and Marcus Randall. “Dynamic problems and nature-inspired meta-heuristics.” In: *Biologically-Inspired Optimisation Methods*. Vol. 210. Springer, 2009 (cit. on pp. 1, 2).

[HO01] Nikolaus Hansen and Andreas Ostermeier. “Completely derandomized self-aptation in evolution strategies.” In: *Evolutionary Computation* 9.2 (2001), pp. 159–195 (cit. on p. 20).

[Hro04] Juraj Hromkovič. *Algorithmics for Hard Problems.* 2nd edition. Texts in Theoretical Computer Science. Springer, 2004 (cit. on p. 17).

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory.” In: *Neural Computation* 9.8 (1997), pp. 1735–1780 (cit. on p. 32).

[Hsi18] Yuan-Ting Hsieh. *Tensorflow Temporal Convolutional Network*. Version 2018/09/04. 2018. URL: `https://github.com/YuanTingHsieh/TF_TCN` (cit. on p. 143).

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Series in Statistics. Springer, 2009 (cit. on pp. 29, 56).

[Hu+19] Weizhen Hu et al. “Solving dynamic multi-objective optimization problems using incremental support vector machine.” In: *Congress on Evolutionary Computation (CEC).* 2019, pp. 2794–2799 (cit. on p. 54).

[HW06] Iason Hatzakis and David Wallace. “Dynamic multi-objective optimization with evolutionary algorithms: A forward-looking approach.” In: *Genetic and Evolutionary Computation Conference (GECCO).* 2006, pp. 1201–1208 (cit. on pp. 27, 30, 53, 86, 90).

[Jam+13] Gareth James et al. *An Introduction to Statistical Learning with Applications in R.* Springer, 2013 (cit. on p. 30).

[JB05] Yaochu Jin and Jürgen Branke. “Evolutionary optimization in uncertain environments–A survey.” In: *Transactions on Evolutionary Computation* 9.3 (2005), pp. 303–317 (cit. on pp. 10, 12, 25, 26).

[JH75] Arthur E. Bryson Jr. and Yu-Chi Ho. *Applied Optimal Control: Optimization, Estimation and Control.* Halsted Press, 1975 (cit. on p. 2).

[Jia+18a] Min Jiang et al. “Solving dynamic multi-objective optimization problems via support vector machine.” In: *International Conference on Advanced Computational Intelligence (ICACI)*. 2018, pp. 819–824 (cit. on p. 54).

[Jia+18b] Min Jiang et al. “Transfer learning-based dynamic multiobjective optimization algorithms.” In: *Transactions on Evolutionary Computation* 22.4 (2018), pp. 501–514 (cit. on p. 54).

[Jin+13] Yaochu Jin et al. “A framework for finding robust optimal solutions over time.” In: *Memetic Computing* 5.1 (2013), pp. 3–18 (cit. on p. 13).

[Jin+16] Yaochu Jin et al. “Reference point based prediction for evolutionary dynamic multiobjective optimization.” In: *Congress on Evolutionary Computation (CEC)*. 2016, pp. 3769–3776 (cit. on pp. 27, 54).

[Jon06] Kenneth A. De Jong. *Evolutionary Computation–A Unified Approach*. MIT Press, 2006 (cit. on p. 12).

[Jor14] Ahmad R. Jordehi. “Particle swarm optimisation for dynamic optimisation problems: A review.” In: *Neural Computing and Applications* 25.7-8 (2014), pp. 1507–1516 (cit. on pp. 25, 69).

[JY17] Shouyong Jiang and Shengxiang Yang. “Evolutionary dynamic multiobjective optimization: Benchmarks and algorithm comparisons.” In: *Transactions on Cybernetics* 47.1 (2017), pp. 198–211 (cit. on p. 35).

[Kag17] Kaggle. *Web Traffic Time Series Forecasting*. 2017. URL: `https://www.kaggle.com/c/web-traffic-time-series-forecasting/leaderboard` (cit. on p. 56).

[Kal60] Rudolph E. Kalman. “A new approach to linear filtering and prediction problems.” In: *Journal of Basic Engineering* 82.1 (1960), pp. 35–45 (cit. on p. 31).

[KB11] Mehdi Khashei and Mehdi Bijari. “A novel hybridization of artificial neural networks and ARIMA models for time series forecasting.” In: *Applied Soft Computing* 11.2 (2011), pp. 2664–2675 (cit. on p. 56).

[KB12] Danil Koryakin and Martin V. Butz. “Reservoir sizes and feedback weights interact non-linearly in echo state networks.” In: *International Conference on Artificial Neural Networks (ICANN)*. 2012, pp. 499–506 (cit. on p. 39).

[KBC14] Nikolaos Kourentzes, Devon K. Barrow, and Sven F. Crone. "Neural network ensemble operators for time series forecasting." In: *Expert Systems with Applications* 41.9 (2014), pp. 4235–4244 (cit. on p. 56).

[KE95] James Kennedy and Russell Eberhart. "Particle swarm optimization." In: *International Conference on Neural Networks (ICNN)*. Vol. 4. 1995, pp. 1942–1948 (cit. on p. 21).

[KG17] Alex Kendall and Yarin Gal. "What uncertainties do we need in Bayesian deep learning for computer vision?" In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, pp. 5580–5590 (cit. on p. 88).

[KGT10] Wee T. Koo, Chi K. Goh, and Kay C. Tan. "A predictive gradient strategy for multiobjective evolutionary algorithms in a fast changing environment." In: *Memetic Computing* 2.2 (2010), pp. 87–110 (cit. on p. 54).

[KLT13] Przemyslaw Kazienko, Edwin Lughofer, and Bogdan Trawinski. "Hybrid and ensemble methods in machine learning." In: *Universal Computer Science (UCS)* 19.4 (2013), pp. 457–461 (cit. on p. 56).

[Kra08] Oliver Kramer. *Self-Adaptive Heuristics for Evolutionary Computation*. Vol. 147. Studies in Computational Intelligence. Springer, 2008 (cit. on pp. 12, 19–21, 113).

[Kra16] Oliver Kramer. *Machine Learning for Evolution Strategies*. Springer, 2016 (cit. on p. 59).

[Kra17] Oliver Kramer. *Genetic Algorithm Essentials*. Vol. 679. Studies in Computational Intelligence. Springer, 2017 (cit. on pp. 19, 106).

[Kru+16] Rudolf Kruse et al. *Computational Intelligence–A Methodological Introduction*. 2nd edition. Texts in Computer Science. Springer, 2016 (cit. on pp. 17, 18, 20).

[LD19] Qiang Liu and Jinliang Ding. "Reference vector based multidirectional prediction for evolutionary dynamic multiobjective optimization." In: *Congress on Evolutionary Computation (CEC)*. 2019, pp. 1081–1087 (cit. on p. 54).

[LE13] Barend J. Leonard and Andries P. Engelbrecht. "On the optimality of particle swarm parameters in dynamic environments." In: *Congress on Evolutionary Computation (CEC)*. 2013, pp. 1564–1569 (cit. on p. 23).

[Lev16] Steven Levy. *The iBrain is here–and it's already inside your phone*. Last access 2019/10/14. 2016. URL: `https://www.wired.com/2016/08/an-exclusive-look-at-how-ai-and-machine-learning-work-at-apple/` (cit. on pp. 32, 55).

[Li+08] Changhe Li et al. *Benchmark Generator for CEC 2009 Competition on Dynamic Optimization*. Tech. rep. University of Leicester, U.K., 2008 (cit. on p. 37).

[Li+13] Changhe Li et al. *Benchmark Generator for the IEEE WCCI-2014 Competition on Evolutionary Computation for Dynamic Optimization Problems*. Tech. rep. De Montfort University, UK, 2013 (cit. on p. 37).

[Li+14] Qiuying Li et al. "Global prediction-based adaptive mutation particle swarm optimization." In: *International Conference on Natural Computation (ICNC)*. 2014, pp. 268–273 (cit. on p. 71).

[Li+15] Changhe Li et al. "Multi-population methods in unconstrained continuous dynamic environments: The challenges." In: *Information Sciences* 296 (2015), pp. 95–118 (cit. on pp. 26, 36).

[Li+16] Changhe Li et al. "An adaptive multipopulation framework for locating and tracking multiple optima." In: *Transactions on Evolutionary Computation* 20.4 (2016), pp. 590–605 (cit. on p. 26).

[Li+18] Changhe Li et al. "An open framework for constructing continuous optimization problems." In: *Transactions on Cybernetics* (2018), pp. 1–15 (cit. on p. 38).

[Li+19a] Jianxia Li et al. "A special points-based hybrid prediction strategy for dynamic multi-objective optimization." In: *IEEE Access* 7 (2019), pp. 62496–62510 (cit. on p. 54).

[Li+19b] Qingya Li et al. "A predictive strategy based on special points for evolutionary dynamic multi-objective optimization." In: *Soft Computing* 23.11 (2019), pp. 3723–3739 (cit. on pp. 53, 54).

[Liu+14] Ruochen Liu et al. “A novel cooperative coevolutionary dynamic multi-objective optimization algorithm using a new predictive model.” In: *Soft Computing* 18.10 (2014), pp. 1913–1929 (cit. on pp. 25, 28, 54).

[Liu+15] Ruochen Liu et al. “An orthogonal predictive model-based dynamic multi-objective optimization algorithm.” In: *Soft Computing* 19.11 (2015), pp. 3083–3107 (cit. on p. 54).

[Liu+18] Ruochen Liu et al. “A dynamic multiple populations particle swarm optimization algorithm based on decomposition and prediction.” In: *Applied Soft Computing* 73 (2018), pp. 434–459 (cit. on p. 71).

[LMR09] Andrew Lewis, Sanaz Mostaghim, and Marcus Randall, eds. *Biologically-Inspired Optimisation Methods: Parallel Algorithms, Systems and Applications.* Vol. 210. Studies in Computational Intelligence. Springer, 2009 (cit. on p. 17).

[LQS13] Jing J. Liang, Bo Y. Qu, and Ponnuthurai N. Suganthan. *Problem Definitions and Evaluation Criteria for the CEC 2014 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization.* Tech. rep. Zhengzhou University (China) and Nanyang Technological University (Singapore), 2013 (cit. on p. 113).

[Luo+19] Wenjian Luo et al. “Surrogate-assisted evolutionary framework for data-driven dynamic optimization.” In: *Transactions on Emerging Topics in Computational Intelligence* 3.2 (2019), pp. 137–150 (cit. on p. 14).

[LY08] Changhe Li and Shengxiang Yang. “A generalized approach to construct benchmark problems for dynamic optimization.” In: *Simulated Evolution and Learning (SEAL).* Springer, 2008, pp. 391–400 (cit. on p. 35).

[LYP11] Changhe Li, Shengxiang Yang, and David A Pelta. *Benchmark generator for the IEEE WCCI-2012 competition on evolutionary computation for dynamic optimization problems.* Tech. rep. Brunel University, UK, 2011 (cit. on p. 37).

[MB13] Sergio Morales-Enciso and Jürgen Branke. “Response surfaces with discounted information for global optima tracking in dynamic environments.” In: *Nature Inspired Cooperative*

Strategies for Optimization (NICSO). 2013, pp. 57–69 (cit. on p. 14).

[MB15] Sergio Morales-Enciso and Jürgen Branke. "Tracking global optima in dynamic environments with efficient global optimization." In: *European Journal of Operational Research* 242.3 (2015), pp. 744–755 (cit. on p. 14).

[MC13] Irene Moser and Raymond Chiong. "Dynamic function optimization: The moving peaks benchmark." In: *Metaheuristics for Dynamic Optimization*. Springer, 2013, pp. 35–59 (cit. on pp. 35, 36).

[MD99] Ronald W. Morrison and Kenneth A. De Jong. "A test problem generator for non-stationary environments." In: *Congress on Evolutionary Computation (CEC)*. 1999, pp. 2047–2053 (cit. on p. 35).

[Mei+15] Stephan Meisel et al. "Evaluation of a multi-objective EA on benchmark instances for dynamic routing of a vehicle." In: *Genetic and Evolutionary Computation Conference (GECCO)*. 2015, pp. 425–432 (cit. on p. 1).

[MH00] Spyros Makridakis and Michèle Hibon. "The M3-Competition: results, conclusions and implications." In: *International Journal of Forecasting* 16.4 (2000), pp. 451–476 (cit. on p. 55).

[MJK15] Douglas C. Montgomery, Cheryl L. Jennings, and Murat Kulahci. *Introduction to Time Series Analysis and Forecasting*. Ed. by David J. Balding. Wiley, 2015 (cit. on pp. 29, 30).

[MK18a] Almuth Meier and Oliver Kramer. "Prediction with recurrent neural networks in evolutionary dynamic optimization." In: *Applications of Evolutionary Computation (EvoApplications)*. 2018, pp. 848–863 (cit. on pp. 5, 49, 53, 116).

[MK18b] Almuth Meier and Oliver Kramer. "Recurrent neural network-predictions for PSO in dynamic optimization." In: *Genetic and Evolutionary Computation Conference (GECCO)*. 2018, pp. 29–36 (cit. on pp. 5, 27, 30, 69–71).

[MK19] Almuth Meier and Oliver Kramer. "Predictive uncertainty estimation with temporal convolutional networks for dynamic evolutionary optimization." In: *International Conference on Artificial Neural Networks (ICANN)*. 2019, pp. 409–421 (cit. on pp. 5, 38, 39, 85, 89).

[MK20] Almuth Meier and Oliver Kramer. “Prediction in nature-inspired dynamic optimization.” In: *Frontier Applications of Nature Inspired Computation*. Ed. by Mahdi Khosravy et al. Springer, 2020, pp. 34–52 (cit. on pp. 4, 27).

[MLY17] Michalis Mavrovouniotis, Changhe Li, and Shengxiang Yang. “A survey of swarm intelligence for dynamic optimization: Algorithms and applications.” In: *Swarm and Evolutionary Computation* 33 (2017), pp. 1–17 (cit. on pp. 1, 2, 9, 12, 13, 21, 22, 25, 69).

[MM05] Rui Mendes and Arvind S. Mohais. “DynDE: A differential evolution for dynamic optimization problems.” In: *Congress on Evolutionary Computation (CEC)*. 2005, pp. 2808–2815 (cit. on p. 25).

[Mor04] Ronald W. Morrison. *Designing Evolutionary Algorithms for Dynamic Environments*. Natural Computing Series. Springer, 2004 (cit. on p. 2).

[MSA18a] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. “Statistical and machine learning forecasting methods: Concerns and ways forward.” In: *PLOS ONE* 13.3 (Mar. 2018), pp. 1–26 (cit. on pp. 30, 55).

[MSA18b] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. “The M4 competition: Results, findings, conclusion and way forward.” In: *International Journal of Forecasting* 34.4 (2018), pp. 802–808 (cit. on p. 55).

[MTV16] Arrchana Muruganantham, Kay C. Tan, and P. Vadakkepat. “Evolutionary dynamic multiobjective optimization via Kalman filter prediction.” In: *Transactions on Cybernetics* 46.12 (2016), pp. 2862–2873 (cit. on pp. 27, 54).

[Mül+17] David Müller et al. “Dynamic real-time optimization under uncertainty of a hydroformylation mini-plant.” In: *Computers & Chemical Engineering* 106 (2017), pp. 836–848 (cit. on p. 1).

[NCP16] Pavel Novoa-Hernández, Carlos Cruz Corona, and David A. Pelta. “Self-adaptation in dynamic environments–A survey and open issues.” In: *International Journal of Bio-Inspired Computation (IJBIC)* 8.1 (2016), pp. 1–13 (cit. on p. 108).

[Ngu11] Trung Thanh Nguyen. “Continuous dynamic optimization using evolutionary algorithms.” PhD thesis. University of Birmingham, 2011 (cit. on p. 48).

[NS01] Arnold Neumaier and Tapio Schneider. “Estimation of parameters and eigenmodes of multivariate autoregressive models.” In: *Transactions on Mathematical Software (TOMS)* 27.1 (2001), pp. 27–57 (cit. on p. 31).

[NW99] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization.* Springer, 1999 (cit. on p. 17).

[NY09a] Trung Thanh Nguyen and Xin Yao. “Benchmarking and solving dynamic constrained problems.” In: *Congress on Evolutionary Computation (CEC).* 2009, pp. 690–697 (cit. on p. 35).

[NY09b] Trung Thanh Nguyen and Xin Yao. “Dynamic time-linkage problems revisited.” In: *Applications of Evolutionary Computing.* Ed. by Mario Giacobini et al. Springer, 2009, pp. 735–744 (cit. on p. 28).

[NY12] Trung Thanh Nguyen and Xin Yao. “Continuous dynamic constrained optimization–The challenges.” In: *Transactions on Evolutionary Computation* 16.6 (2012), pp. 769–786 (cit. on pp. 48, 115).

[NY13] Trung Thanh Nguyen and Xin Yao. “Dynamic time-linkage evolutionary optimization: Definitions and potential solutions.” In: *Metaheuristics for Dynamic Optimization.* Springer, 2013, pp. 371–395 (cit. on p. 13).

[NYB12] Trung Thanh Nguyen, Shengxiang Yang, and Jürgen Branke. “Evolutionary dynamic optimization: A survey of the state of the art.” In: *Swarm and Evolutionary Computation* 6 (2012), pp. 1–24 (cit. on pp. 2, 12, 13, 25–27, 35, 47, 71, 86).

[OJOMN19] Domingos S. de O. Junior, João F. L. de Oliveira, and Paulo S. G. de Mattos Neto. “An intelligent hybridization of ARIMA with machine learning models for time series forecasting.” In: *Knowledge-Based Systems* 175 (2019), pp. 72–86 (cit. on p. 56).

[OP09] Djamila Ouelhadj and Sanja Petrovic. “A survey of dynamic scheduling in manufacturing systems.” In: *Journal of Scheduling* 12.4 (2009), pp. 417–431 (cit. on p. 1).

[ORB19] Sebastian Otte, Patricia Rubisch, and Martin V. Butz. “Gradient-based learning of compositional dynamics with modular RNNs.” In: *International Conference on Artificial Neural Networks (ICANN)*. 2019, pp. 484–496 (cit. on p. 39).

[OZK18] Stefan Oehmcke, Oliver Zielinski, and Oliver Kramer. “Direct training of dynamic observation noise with UMarineNet.” In: *International Conference on Artificial Neural Networks (ICANN)*. Ed. by Věra Kůrková et al. Springer, 2018, pp. 123–133 (cit. on pp. 88, 143).

[Pas02] Kevin M. Passino. “Biomimicry of bacterial foraging for distributed optimization and control.” In: *IEEE Control Systems Magazine* 22.3 (2002), pp. 52–67 (cit. on p. 21).

[PE19] Gary Pamparà and Andries P. Engelbrecht. “A generator for dynamically constrained optimization problems.” In: *Genetic and Evolutionary Computation Conference (GECCO) Companion*. 2019, pp. 1441–1448 (cit. on p. 35).

[Pen+15] Zhou Peng et al. “Novel prediction and memory strategies for dynamic multiobjective optimization.” In: *Soft Computing* 19.9 (2015), pp. 2633–2653 (cit. on p. 54).

[PH99] Hartmut Pohlheim and Adolf Heiéner. “Optimal control of greenhouse climate using real-world weather data and evolutionary algorithms.” In: *Genetic and Evolutionary Computation Conference (GECCO)*. Orlando, Florida, 1999, pp. 1672–1677 (cit. on p. 1).

[PKB07] Riccardo Poli, James Kennedy, and Tim Blackwell. “Particle swarm optimization.” In: *Swarm Intelligence* 1.1 (2007), pp. 33–57 (cit. on p. 22).

[Pol09] Riccardo Poli. “Mean and variance of the sampling distribution of particle swarm optimizers during stagnation.” In: *IEEE Transactions on Evolutionary Computation* 13.4 (2009), pp. 712–721 (cit. on p. 74).

[PSB19] Antonio R. S. Parmezan, Vinícius M. A. de Souza, and Gustavo E. A. P. A. Batista. “Evaluation of statistical and machine learning models for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model.” In: *Information Sciences* 484 (2019), pp. 302–337 (cit. on p. 55).

[PV10] Konstantinos E. Parsopoulos and Michael N. Vrahatis. *Particle Swarm Optimization and Intelligence: Advances and Applications.* Information Science Reference - Imprint of IGI Publishing, 2010 (cit. on p. 22).

[RAD08] Claudio Rossi, Mohamed Abderrahim, and Julio César Díaz. "Tracking moving optima using Kalman-based predictions." In: *Evolutionary Computation* 16.1 (2008), pp. 1–30 (cit. on pp. 27, 30, 54, 85–87, 89).

[RBK12] Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok, eds. *Handbook of Natural Computing.* Springer, 2012 (cit. on p. 17).

[Rec73] Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzbog, Stuttgart. 1973 (cit. on pp. 19, 21, 58).

[RGZ16] Miao Rong, Dun-Wei Gong, and Yong Zhang. "A multi-direction prediction approach for dynamic multi-objective optimization." In: *International Conference on Intelligent Computing (ICIC).* 2016, pp. 629–636 (cit. on pp. 27, 54).

[RI10] Benjamin Roeschies and Christian Igel. "Structure optimization of reservoir networks." In: *Logic Journal of the IGPL* 18.5 (2010), pp. 635–669 (cit. on p. 39).

[Ric09] Hendrik Richter. "Detecting change in dynamic fitness landscapes." In: *Congress on Evolutionary Computation (CEC).* 2009, pp. 1613–1620 (cit. on pp. 26, 57).

[Ric10] Hendrik Richter. "Evolutionary optimization and dynamic fitness landscapes." In: *Evolutionary Algorithms and Chaotic Systems.* Springer, 2010, pp. 409–446 (cit. on pp. 11, 14).

[Roj96] Raul Rojas. *Neural Networks: A Systematic Introduction.* Springer, 1996 (cit. on p. 32).

[Ron+19] Miao Rong et al. "Multidirectional prediction approach for dynamic multiobjective optimization problems." In: *Transactions on Cybernetics* 49.9 (2019), pp. 3362–3374 (cit. on pp. 27, 28, 54).

[Rot11] Franz Rothlauf. *Design of Modern Heuristics.* Natural Computing Series. Springer, 2011 (cit. on pp. 11, 12, 17).

[RS19] Karthik Ramasubramanian and Abhishek Singh. *Machine Learning Using R. With Time Series and Industry-Based Use Cases in R.* 2nd edition. Apress, 2019 (cit. on p. 29).

[Rua+17] Gan Ruan et al. “The effect of diversity maintenance on prediction in dynamic multi-objective optimization.” In: *Applied Soft Computing* 58 (2017), pp. 631–647 (cit. on p. 54).

[Rud12] Günter Rudolph. “Evolutionary strategies.” In: *Handbook of Natural Computing.* Springer, 2012, pp. 673–698 (cit. on p. 20).

[RY13] Hendrik Richter and Shengxiang Yang. “Dynamic optimization using analytic and evolutionary approaches: A comparative review.” In: *Handbook of Optimization–From Classical to Modern Approach.* Springer, 2013, pp. 1–28 (cit. on p. 2).

[SC08a] Anabela Simões and Ernesto Costa. “Evolutionary algorithms for dynamic environments: Prediction using linear regression and Markov chains.” In: *Parallel Problem Solving from Nature (PPSN).* 2008, pp. 306–315 (cit. on p. 28).

[SC08b] Ingo Steinwart and Andreas Christmann. *Support Vector Machines.* Information Science and Statistics. Springer, 2008 (cit. on p. 54).

[SC09] Anabela Simões and Ernesto Costa. “Improving prediction in evolutionary algorithms for dynamic environments.” In: *Genetic and Evolutionary Computation Conference (GECCO).* 2009, pp. 875–882 (cit. on p. 28).

[SC14] Anabela Simões and Ernesto Costa. “Prediction in evolutionary algorithms for dynamic environments.” In: *Soft Computing* 18.8 (2014), pp. 1471–1497 (cit. on pp. 12, 13, 28, 86).

[Sha49] Claude E. Shannon. “Communication in the presence of noise.” In: *Proceedings of the IRE* 37.1 (1949), pp. 10–21 (cit. on p. 44).

[Shi12] Ofer M. Shir. “Niching in evolutionary algorithms.” In: *Handbook of Natural Computing.* Springer, 2012, pp. 1035–1069 (cit. on p. 25).

[Smy19] Slawek Smyl. “A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting.” In: *International Journal of Forecasting* (2019) (cit. on pp. 32, 56).

[Sor70] Harold W. Sorenson. “Least-squares estimation: From Gauss to Kalman.” In: *IEEE Spectrum* 7.7 (1970), pp. 63–68 (cit. on p. 31).

[SS17] Robert H. Shumway and David S. Stoffer. *Time Series Analysis and Its Applications. With R examples.* 4th edition. Springer Texts in Statistics. Springer, 2017 (cit. on p. 29).

[Ste07] Jochen J. Steil. “Several ways to solve the MSO problem.” In: *European Symposium on Artificial Neural Networks (ESANN).* 2007, pp. 489–494 (cit. on p. 39).

[Teo09] Dusan Teodorovic. “Bee colony optimization (BCO).” In: *Innovations in Swarm Intelligence.* Springer, 2009, pp. 39–60 (cit. on p. 21).

[TM99] Krzysztof Trojanowski and Zbigniew Michalewicz. “Searching for optima in non-stationary environments.” In: *Congress on Evolutionary Computation (CEC).* Vol. 3. 1999, pp. 1843–1850 (cit. on p. 48).

[TY07] Renato Tinós and Shengxiang Yang. “A self-organizing random immigrants genetic algorithm for dynamic optimization problems.” In: *Genetic Programming and Evolvable Machines* 8.3 (2007), pp. 255–286 (cit. on p. 26).

[UFK02] Rasmus K. Ursem, Bogdan Filipič, and Thiemo Krink. “Exploring the performance of an evolutionary algorithm for greenhouse control.” In: *Journal of Computing and Information Technology* 10.3 (2002), pp. 195–201 (cit. on p. 2).

[Urs00] Rasmus K. Ursem. “Multinational GAs: Multimodal optimization techniques in dynamic environments.” In: *Genetic and Evolutionary Computation Conference (GECCO).* 2000, pp. 19–26 (cit. on p. 27).

[Wal02] David M. Walker. “Kalman filtering of time series data.” In: *Modelling and Forecasting Financial Data: Techniques of Nonlinear Dynamics.* Ed. by Abdol S. Soofi and Liangyue Cao. Springer, 2002, pp. 137–157 (cit. on p. 31).

[Wan+16] Zi-Jia Wang et al. “Orthogonal learning particle swarm optimization with variable relocation for dynamic optimization.” In: *Congress on Evolutionary Computation (CEC).* 2016, pp. 594–600 (cit. on p. 54).

[Wei03] Karsten Weicker. *Evolutionary Algorithms and Dynamic Optimization Problems.* Der Andere Verlag Berlin, 2003 (cit. on p. 12).

[WGS05] Daan Wierstra, Faustino J. Gomez, and Jürgen Schmidhuber. "Modeling systems with internal state using evolino." In: *Genetic and Evolutionary Computation Conference (GECCO).* 2005, pp. 1795–1802 (cit. on p. 39).

[WJL15] Yan Wu, Yaochu Jin, and Xiaoxiong Liu. "A directed search strategy for evolutionary dynamic multiobjective optimization." In: *Soft Computing* 19.11 (2015), pp. 3221–3235 (cit. on pp. 27, 54).

[WM97] David H. Wolpert and William G. Macready. "No free lunch theorems for optimization." In: *Transactions on Evolutionary Computation* 1.1 (1997), pp. 67–82 (cit. on p. 53).

[Wol19] Mattis Wolf. *Experimenteller Vergleich von Vorhersagemethoden von multidimensionalen Zeitreihen.* Student Thesis. 2019 (cit. on pp. 56, 119, 120).

[Wu+16] Yonghui Wu et al. "Google's neural machine translation system: Bridging the gap between human and machine translation." In: *CoRR* abs/1609.08144 (2016) (cit. on pp. 32, 55).

[WY09] Yonas G. Woldesenbet and Gary G. Yen. "Dynamic evolutionary algorithm with variable relocation." In: *IEEE Transactions on Evolutionary Computation* 13.3 (2009), pp. 500–513 (cit. on p. 86).

[Yan11] Xin-She Yang. "Optimization algorithms." In: *Computational Optimization, Methods and Algorithms.* Springer, 2011, pp. 13–31 (cit. on p. 18).

[Yan14] Xin-She Yang. *Nature-Inspired Optimization Algorithms.* Elsevier, 2014 (cit. on pp. 18, 19).

[Yaz+18] Danial Yazdani et al. "A multi-objective time-linkage approach for dynamic optimization problems with previous-solution displacement restriction." In: *Applications of Evolutionary Computation (EvoApplications).* 2018, pp. 864–878 (cit. on p. 14).

[YK11] Xin-She Yang and Slawomir Koziel. “Computational Optimization: An Overview.” In: *Computational Optimization, Methods and Algorithms*. Springer, 2011, pp. 1–11 (cit. on pp. 12, 18).

[YM00] Robert A. Yaffee and Monnie McGee. *Introduction to Time Series Analysis and Forecasting with Applications of SAS and SPSS*. Academic Press, Inc., 2000 (cit. on pp. 29, 31, 55).

[YNB19] Danial Yazdani, Trung Thanh Nguyen, and Jürgen Branke. “Robust optimization over time by learning problem space characteristics.” In: *Transactions on Evolutionary Computation* 23.1 (2019), pp. 143–155 (cit. on p. 13).

[Yu+10] Xin Yu et al. “Robust optimization over time-A new perspective on dynamic optimization problems.” In: *Congress on Evolutionary Computation (CEC)*. 2010, pp. 1–6 (cit. on pp. 13, 14, 36).

[YY03] Shengxiang Yang and Xin Yao. “Dual population-based incremental learning for problem optimization in dynamic environments.” In: *Asia Pacific Symposium on Intelligent and Evolutionary Systems*. 2003, pp. 49–56 (cit. on p. 35).

[ZE14] E. T. van Zyl and Andries P. Engelbrecht. “Comparison of self-adaptive particle swarm optimizers.” In: *Swarm Intelligence Symposioum (SIS)*. 2014, pp. 48–56 (cit. on p. 23).

[Zen+06] Sangyou Zeng et al. “A dynamic multi-objective evolutionary algorithm based on an orthogonal design.” In: *Congress on Evolutionary Computation (CEC)*. 2006, pp. 573–580 (cit. on p. 54).

[Zha01] Guoqiang P. Zhang. “An investigation of neural networks for linear time-series forecasting.” In: *Computers & OR* 28.12 (2001), pp. 1183–1202 (cit. on p. 55).

[Zha12] Guoqiang P. Zhang. “Neural networks for time-series forecasting.” In: *Handbook of Natural Computing*. Springer, 2012, pp. 461–477 (cit. on pp. 29, 56, 90).

[Zho+07] Aimin Zhou et al. “Prediction-based population re-initialization for evolutionary dynamic multi-objective optimization.” In: *Evolutionary Multi-Criterion Optimization (EMO)*. Ed. by Shigeru Obayashi et al. 2007, pp. 832–846 (cit. on pp. 28, 54, 86, 87).

[Zho+18] Jianwei Zhou et al. "An evolutionary dynamic multi-objective optimization algorithm based on center-point prediction and sub-population autonomous guidance." In: *Symposium Series on Computational Intelligence (SSCI).* 2018, pp. 2148–2154 (cit. on pp. 27, 54).

[ZJZ14] Amin Zhou, Yaochu Jin, and Qingfu Zhang. "A population prediction strategy for evolutionary dynamic multiobjective optimization." In: *Transactions on Cybernetics* 44.1 (2014), pp. 40–53 (cit. on pp. 27, 53, 71).

[ZLB04] Eckart Zitzler, Marco Laumanns, and Stefan Bleuler. "A tutorial on evolutionary multiobjective optimization." In: *Metaheuristics for multiobjective optimisation.* Springer, 2004, pp. 3–37 (cit. on p. 11).

[Zou+17] Juan Zou et al. "A prediction strategy based on center points and knee points for evolutionary dynamic multi-objective optimization." In: *Applied Soft Computing* 61 (2017), pp. 806 –818 (cit. on pp. 53, 54).

[ZWJ15] Yudong Zhang, Shuihua Wang, and Genlin Ji. "A comprehensive survey on particle swarm optimization algorithm and its applications." In: *Mathematical Problems in Engineering* 2015 (2015) (cit. on p. 22).

[Yaz+19] D. Yazdani et al. "Scaling up dynamic optimization problems: A divide-and-conquer approach." In: *IEEE Transactions on Evolutionary Computation* (2019) (cit. on pp. 35, 108).