2

Preliminaries

The overall objective of this thesis is to derive methods that increase the likelihood of discovery of *innovative solutions*, in using automated search. This chapter provides an overview of established techniques and approaches that will be applied or from which ideas are derived, reported on in later chapters. As underlying search method, an advanced *Evolutionary Algorithm* will be used that adapts its search distribution to the search landscape, the *CMA-ES*. It is introduced in Section 2.1.

The goal of automated discovery of innovative solutions is approached by strenghtening the explorative behavior in the search induced by the CMA-ES. Exploration schemes compatible with this algorithm, but not limited to it, will be examined, including schemes that use a *surrogate model* to determine a path through the search space. In this usage, though, technically speaking, it is not a surrogate model: While the model approximates the quality function, it is not intended to replace the quality function, but to operate complementary to it, for inducing exploration. Nevertheless, the meaning of a model that approximates the quality function stands, and therefore this term will be used throughout the thesis.

Two techniques for approximation modeling, *feed-forward neural networks* and *Kriging*, are described in Section 2.2. Next, to distinguish the explorative usage of surrogate models from their default application in partially replacing the quality function in optimization, we briefly summarize *surrogate-assisted optimization* in Section 2.3. Moreover, related to surrogate-assisted optimization, ideas of how to deal with the trade-off between exploitation and exploration are addressed.

2.1 · Evolutionary Algorithms

Inspired by Darwinian evolution, *Evolutionary Algorithms* (EAs) (see, e.g., [**Bäck, 1996, Eiben and Smith, 2003**]) are *optimization algorithms* that work from a *population* of solutions, using randomness to generate new solutions as slightly-varied combinations of solutions currently in the population. An optimization algorithm aims to find a solution that, given a certain *quality function*, provides the best quality value from the range of that quality function. As such, in EAs, a solution's quality value is used as indicator of *fitness*, scaling its chances in participating in generating new solutions and proceeding into the next *generation*'s population of solutions.

A solution is a possible input to the quality function, where both input and quality function are defined in the broadest sense of the term. It is not uncommon that a different solution representation than the actual input is used *within* the EA, provided that there is an approach to obtain the function input from this *encoded individual*. Typically, an individual in an EA is a vector of values, discrete or continuous, and possibly a combination of these, although other representations such as complete program trees are used as well [**Eiben and Smith, 2003**]. Furthermore, EAs have flexibility in how new solutions are generated, allowing for problem-specific *variation operators*.

An EA performs stochastic sampling, directed by quality via a population of solutions. In its search for the global optimum, being directed by a population of solutions provides moderate robustness to multimodality in the quality landscape [**Loshchilov et al., 2012**] (i.e., the presence of multiple peaks instead of a single increasing or decreasing trend towards the global optimum). Moreover, an EA is able to operate on *black-box* functions [**Kruisselbrink, 2012**], in which the structure of the quality function is not available to the optimization algorithm, for instance when the quality value is determined through simulation. Also, the stochasticity involved makes that not a fixed path is followed through the search space, potentially allowing for repeated execution from the same initial settings to produce alternative solutions.

Evolution Strategies (ESs), constituting a sophisticated class of EAs, feature online adaptations in their probabilistic sampling of new solutions during the search. Typically using a real-valued solution representation, after each iteration, the continuous search distribution from which solutions are sampled is adapted to better meet the search objective. Among ESs, the *Covariance Matrix Adaptation Evolution*

Strategy (CMA-ES) [**Hansen and Ostermeier, 2001**] has proven a successful, widely used variant, in which the dynamic search distribution is an anisotropic, multivariate normal distribution.

2.1.1 · Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

The CMA-ES estimates a covariance matrix to match the local curvature of the quality landscape [**Kruisselbrink, 2012**], but without using or approximating gradient information[1]. For adapting the covariance matrix, the *ranking* of sampled solutions that is induced by the quality function is relied on, instead of the *actual* structure of the quality landscape. As a result, the CMA-ES is applicable to problems on which the usually faster, classical optimization approaches such as *quasi-Newton* and *conjugate gradient* methods fail due to the quality landscape not being convex or being locally highly rugged [**Hansen, 2011**].

We apply the (μ, λ)-CMA-ES, as implemented in the Shark Machine Learning Library v2.3.43[2], thereby following the procedure and parameter settings described in [**Hansen and Kern, 2004**]. The prefix (μ, λ) indicates a parent population size of μ individuals and an offspring population size of λ individuals, and the use of *comma-selection*: At the end of each generation, all parents are discarded and a new set of parents is selected from the lastly-generated offspring. An initial comparison of the setup of [**Hansen and Kern, 2004**] to the updated procedure and parameter settings described in [**Hansen, 2011**] was performed on the real-world airfoil test case described in Chapter 6. It showed that by inducing less locally-oriented search, the original settings delivered better results for this application.

An overview is given of the inner workings of the (μ, λ)-CMA-ES, see Technical Note 2.1, together with the parameter settings adopted for it, see Technical Note 2.2, as used in the experiments reported on in this thesis. Utilizing the (μ, λ)-CMA-ES requires a problem-specific setting for the initial solution from which to start the optimization, to provide an initial global stepsize for the search distribution, and to define termination criteria, for instance, the number of generations to use for the optimization run. For assisting in choosing the parent and offspring population sizes μ and λ, a minimal setting for λ and default setting for μ are provided for the (μ, λ)-CMA-ES, see Equation 2.10.

[1] According to Nikolaus Hansen, on `https://www.lri.fr/~hansen/cmaesintro.html`.
[2] `http://shark-project.sourceforge.net`

Technical Note 2.1 The (μ, λ) CMA Evolution Strategy (CMA-ES)

Initialization:

- Initialize evolution path, $\mathbf{p}_\sigma = \mathbf{0}$ and $\mathbf{p}_\mathrm{c} = \mathbf{0}$, and covariance matrix, $\boldsymbol{C} = \mathbf{I}_n$;
- Problem-specifically, initialize the distribution mean $\mathbf{m} \in \mathbb{R}^n$ (i.e., set to the initial solution) and stepsize $\sigma \in \mathbb{R}_+$, where n is the dimension of the solution vectors.

Until termination:

- Sample λ new individuals $\mathbf{x}_k$:

$$\mathbf{z}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_n), \tag{2.1}$$

$$\mathbf{y}_k = \boldsymbol{C}^{\frac{1}{2}}\mathbf{z}_k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{C}), \tag{2.2}$$

$$\mathbf{x}_k = \mathbf{m} + \sigma\mathbf{y}_k \sim \mathcal{N}(\mathbf{m}, \sigma^2\boldsymbol{C}); \tag{2.3}$$

- Select the μ best-ranked individuals with index $i{:}\lambda$, $i = 1, \ldots, \mu$ and recombine them using weights w_i to form the new distribution mean $\mathbf{m}$:

$$\mathbf{m} \leftarrow \langle \mathbf{x} \rangle_\mathrm{w} = \sum_{i=1}^{\mu} w_i \mathbf{x}_{i:\lambda}, \tag{2.4}$$

$$\langle \mathbf{y} \rangle_\mathrm{w} = \sum_{i=1}^{\mu} w_i \mathbf{y}_{i:\lambda}; \tag{2.5}$$

- Update the evolution path, a memory of the steps taken by the population, using exponential smoothing with different update rates for the stepsize and the covariance matrix:

$$\mathbf{p}_\sigma = (1 - c_\sigma) \cdot \mathbf{p}_\sigma + \sqrt{c_\sigma(2 - c_\sigma) \cdot \mu_\mathrm{eff}} \cdot \langle \mathbf{y} \rangle_\mathrm{w}, \tag{2.6}$$

$$\mathbf{p}_\mathrm{c} = (1 - c_\mathrm{c}) \cdot \mathbf{p}_\mathrm{c} + \sqrt{c_\mathrm{c}(2 - c_\mathrm{c}) \cdot \mu_\mathrm{eff}} \cdot \boldsymbol{C}^{-\frac{1}{2}} \cdot \langle \mathbf{y} \rangle_\mathrm{w}; \tag{2.7}$$

- Update the stepsize, comparing the length $\|\mathbf{p}_\sigma\|$ of the evolution path to the expected length $\mathbf{E}[\|\mathcal{N}(\mathbf{0}, \mathbf{I}_n)\|]$ under random selection:

$$\sigma \leftarrow \sigma \cdot \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\mathbf{p}_\sigma\|}{\mathbf{E}[\|\mathcal{N}(\mathbf{0}, \mathbf{I}_n)\|]} - 1\right)\right); \tag{2.8}$$

- Update the covariance matrix, by exponential smoothing, in the direction of evolution path $\mathbf{p}_\mathrm{c}$, with the covariances used for obtaining the best μ individuals:

$$\boldsymbol{C} \leftarrow (1 - c_\mathrm{cov}) \cdot \boldsymbol{C} + c_1 \cdot \mathbf{p}_\mathrm{c}\mathbf{p}_\mathrm{c}^\mathrm{T} + c_\mu \cdot \sum_{i=1}^{\mu} w_i \mathbf{y}_{i:\lambda}\mathbf{y}_{i:\lambda}^\mathrm{T}. \tag{2.9}$$

Technical Note 2.2 Parameter Settings of the CMA-ES

Minimal settings for population sizes μ and λ [**Hansen and Kern, 2004**], where n is the dimension of the solution vectors:

$$\lambda = 4 + \lfloor 3 \ln n \rfloor, \quad \mu = \left\lfloor \frac{\lambda}{2} \right\rfloor; \tag{2.10}$$

Recombination weights (deviating slightly from [**Hansen and Kern, 2004**], cf. $\ln(i)$):

$$w_i = \frac{w_i'}{\sum_{j=1}^{\mu} w_j'}, \quad w_i' = \ln(\mu + 1) - \ln(1 + i), \quad i = 1, \ldots, \mu; \tag{2.11}$$

The *effective mass* μ_{eff} after selection and recombination using weights w_i:

$$\mu_{\text{eff}} = \left(\sum_{i=1}^{\mu} w_i^2 \right)^{-1}, \text{ since } \sum_{i=1}^{\mu} w_i = 1, \text{ it holds that } 1 \leq \mu_{\text{eff}} \leq \mu; \tag{2.12}$$

The evolution-path update rate c_σ for the stepsize and c_{c} for the covariance matrix:

$$c_\sigma = \frac{\mu_{\text{eff}} + 2}{n + \mu_{\text{eff}} + 3}, \quad c_{\text{c}} = \frac{4}{n + 4}; \tag{2.13}$$

The stepsize-update dampening parameter d_σ, decreasing the effect of update rate c_σ:

$$d_\sigma = 1 + 2 \cdot \max \left(0, \sqrt{\frac{\mu_{\text{eff}} - 1}{n + 1}} - 1 \right) + c_\sigma; \tag{2.14}$$

The covariance-matrix update rate c_{cov}, and the influence of the evolution path and of the covariances in each update:

$$c_{\text{cov}} = \frac{1}{\mu_{\text{eff}}} \frac{2}{\left(n + \sqrt{2}\right)^2} + \left(1 - \frac{1}{\mu_{\text{eff}}}\right) \min \left(1, \frac{2\mu_{\text{eff}} - 1}{(n + 2)^2 + \mu_{\text{eff}}} \right), \tag{2.15}$$

$$c_1 = c_{\text{cov}} \frac{1}{\mu_{\text{eff}}}, \quad c_\mu = c_{\text{cov}} \left(1 - \frac{1}{\mu_{\text{eff}}}\right). \tag{2.16}$$

After setting the problem-specific parameters and initializing the covariance matrix to the n-by-n identity matrix, optimization is started, divided into iterations in which five recurring steps are performed:

1. λ offspring individuals are generated based on the current distribution mean $\mathbf{m}$,

covariance matrix $\boldsymbol{C}$, and global stepsize σ;

2. From these individuals, the μ best-ranked individuals are selected and recombined into the new distribution mean, with more influence for the better ranked solutions, see Equation 2.11. Notation $i:\lambda$ stands for the i-th best-ranked individual from the λ total;

3. The evolution-path variables are updated, separately for use later in the stepsize and covariance matrix adaptation. Through exponential smoothing, these variables represent a weighted average of the last search steps performed by the optimization. Here μ_{eff} indicates the resulting *effective mass* of applying the weighted recombination, in terms of the amount of individuals that the recombined mean represents, see Equation 2.12. In the covariance-matrix evolution-path update rule, $\boldsymbol{C}^{-\frac{1}{2}}$ rescales the last step taken based on the width of the search distribution per dimension, to make the length of the step independent of its direction (the H_σ parameter, enforcing a maximum for the last step taken in [**Hansen and Kern, 2004**], is not used);

4. The stepsize is updated, by comparing the length of the evolution path, i.e., the exponentially-weighted average of the last steps taken, to the expected length of a random step. When the length of the evolution path is larger than the expected length under random selection, the stepsize is increased, and the stepsize is decreased when the observed length is smaller than the expected length (for an illustrated motivation, see [**Hansen, 2011**]); $\mathbf{E}[\|\mathcal{N}(\mathbf{0},\mathbf{I}_n)\|]$ is approximated as $\sqrt{n}\left(1-\frac{1}{4n}-\frac{1}{21n^2}\right)$ [**Hansen and Kern, 2004**];

5. The covariance matrix is updated, also using exponential smoothing, based on the exponentially-weighted average of the last steps taken, and the covariances that were used in obtaining the new distribution mean.

Optimization continues until a termination criterion is met. For a discussion of the parameters listed in Technical Note 2.2 and the derivation of possible settings, see [**Hansen and Ostermeier, 2001**].

2.2 · Approximation Models

An *approximation model* predicts a *mapping* from inputs to outputs by *generalizing* from a *training set* of inputs with corresponding outputs, to approximated outputs for inputs not encountered in training [**Haykin, 1998**]. As such, it represents an approximation of the actual function from input to output. The inputs are vectors that are associated with singular or higher-dimensional output vectors. Both type of vectors can consist of discrete or continuous variables, or a combination of both. For employing approximation modeling, a data set of inputs mapped to outputs is required. To then predict outputs for unseen inputs, a *modeling technique* is to be chosen, as well as an *iterative training approach*, which fits the model to the available data guided by an *error measure* that indicates the attained generalization performance.

Two modeling techniques are presented, *feed-forward neural networks* and *Kriging*. Whereas Kriging is a good *interpolator*, i.e., in general delivers good predictions for points located between training points, and provides an estimate of the associated prediction error, neural networks require less computational effort for training and do usually not converge to a single output vector in *extrapolating*, i.e., predicting outputs for points located outside of the space covered by the training data.

Feed-forward Neural Networks

Artificial neural networks (ANNs) (e.g., see [**Haykin, 1998**]) are inspired from neural networks in animals and consist of interconnected nodes or *neurons*. A neuron is a signal-propagating unit that sends a signal over its outgoing connections if its input signal, i.e., the total strength of the signals received on its incoming connections, exceeds a certain *threshold*. In using ANNs for approximation modeling, the strengths or *weights* of the connections between nodes are adapted in fitting the model to training data. The threshold is implemented as an *activation function* that transforms the sum of the weighted incoming values to an output value, either via a discrete step function or as a transformed continuous output value.

In an ANN, nodes can be grouped in *layers*, with connections only existing between nodes in adjacent layers. *Feed-forward neural networks* (FFNNs) are defined to have signals propagate forward between subsequent layers. They consist of a layer of input nodes and a layer of output nodes, and typically have one or more *hidden layers* in between, in which case they are termed *multilayer perceptrons* [**Haykin,**

1998]. The input nodes have linear activation, i.e., they just propagate the actual values of the input vectors from the training set to the nodes in the first hidden layer.

Multilayer perceptrons are trained using *error back-propagation* [Rumelhart et al., 1986, Haykin, 1998]: The prediction error between the FFNN-approximated output and the actual output of a training example is propagated back through the network, slightly adjusting the weights along the way to make the approximated output closer to the desired output. Training is continued in steps that feed the whole training set to the network, i.e., one *epoch*, while back-propagating the prediction error for each training example, until a required precision that is expressed by an error measure, or an available number of epochs is reached.

In the experiments described in this thesis, we apply FFNNs as implemented in the Shark Machine Learning Library v2.3.43[3]. Nodes in the hidden layer are sigmoidally-activated, i.e., they have a continuous *logistic* activation function [Haykin, 1998], while the output nodes have linear activation. The network is fully connected, meaning that there are connections going from each node in a certain layer to all nodes in all following layers. A *bias node*, i.e., always producing a signal with value 1.0, is connected to all nodes, except those in the input layer. For training, the iRprop$^+$ algorithm by [Igel and Hüsken, 2003] is applied, an improved version of the *resilient back-propagation* algorithm that uses an adaptive update scheme for the connection weights, based on the sequence of recent changes for a certain weight [Riedmiller and Braun, 1993].

Kriging

In *Kriging*, a technique originally intended for predicting ore concentrations in mining, approximation modeling is performed based on a *Gaussian random field* [Emmerich, 2005]. A Gaussian random field defines a *separate* Gaussian distribution at each possible input point, i.e., *index* of the random field, in a given domain. A to-be-modeled function with one-dimensional output is assumed to be a *realization* of a Gaussian random field. This enables using the Gaussian distribution at each index for modeling the associated output value.

The procedure is illustrated in Figure 2.1. The training points, for which the actual output value is available, are represented by the white dots. Assuming spatial correlation between these output values [Jones et al., 1998, Emmerich, 2005], an

[3] http://shark-project.sourceforge.net

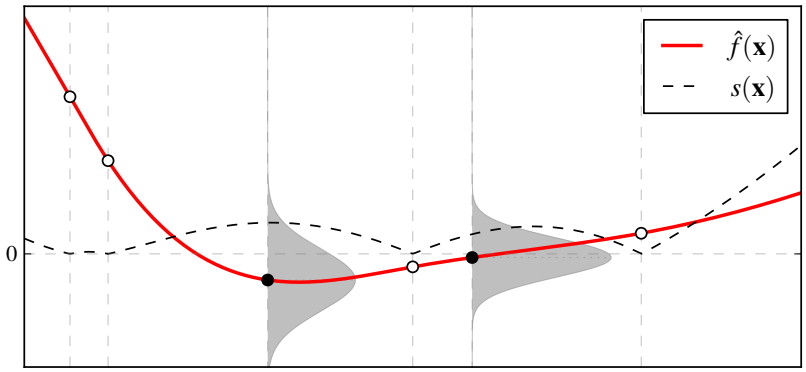

Figure 2.1 Kriging. The underlying output function of training points (white dots) is taken as a realization of a Gaussian random field (i.e., a random field defining Gaussian distributions in each input $\mathbf{x}$). By assuming spatial correlation between the training points, an approximation $\hat{f}(\mathbf{x})$ (red line) of the underlying function is calculated. Together with the prediction error $s(\mathbf{x})$ (black dashed line) that is estimated for each input as well, this provides the likeliness (via the probability density function of $\mathcal{N}\left(\hat{f}(\mathbf{x}), s^2(\mathbf{x})\right)$, in gray) of all possible output values of an unseen input $\mathbf{x}$ (black dots), where $\hat{f}(\mathbf{x})$ is the most-likely realization of the output value.

approximation $\hat{f}(\mathbf{x})$ is calculated of the to-be-modeled function, plotted as a red line. This red line indicates the mean of the Gaussian distribution at each index (i.e., input point) in the random field. The standard deviation $s(\mathbf{x})$ per Gaussian distribution, termed the *conditional* or *local* standard deviation, is also determined from the assumed correlation between the training points and their spatial distribution, and is plotted as a black dashed line. Altogether, for an unseen input point $\mathbf{x}$, this provides the likelihood for a certain value to be the actual output value, as it is considered to be a realization of the output distribution $\mathcal{N}\left(\hat{f}(\mathbf{x}), s^2(\mathbf{x})\right)$. Two unseen input points are indicated by the black dots, with the plotted probability density functions showing the likelihood of all their possible output values.

Kriging is not a branch of *Gaussian Process modeling* (GP) [**Rasmussen and Williams, 2006**] as it is often presented: Kriging aims to predict the output of an unseen input point as a linearly-weighted sum of the outputs in the training data, based on the spatial relationship between the training data and the unseen input point (i.e., the *best linear unbiased predictor* [**Stein, 1999**]). GP, on the other hand, aims to provide the mean and conditional standard deviation of the Gaussian distribution of the output of an unseen input point. Nevertheless, we restrict Kriging to using Gaussian-type random fields and therefore both interpretations can be used interchangeably, as

is done in the previous paragraph. In case of multiple output variables, i.e., the to-be-modeled function has multiple outputs, multiple separate instances of univariate Kriging, described above, can be used. This, however, ignores the correlations between the output variables, which can be taken into account by using multivariate Kriging (see, e.g., [**Kleijnen and Mehdad, 2012**]).

In this work *Ordinary Kriging* is employed, which assumes a constant trend ("offset") for the Gaussian random field. The spatial correlation is represented by a power-exponential multiparameter kernel [**Forrester et al., 2008**],

$$\mathrm{corr}(\mathbf{x},\mathbf{x}') = \exp\left(-\sum_{i=1}^{\dim(\mathbf{x})} \theta_i |x_i - x_i'|^{p_i}\right), \tag{2.17}$$

$$\theta_i \in [10^{-10}, 10^2],\ p_i \in [1,2], \tag{2.18}$$

where $\dim(\mathbf{x})$ is the number of elements in vector $\mathbf{x}$ and where the θ_i and p_i are allowed to vary per dimension. The Gaussian random field is trained by tuning the correlation function: The parameters θ_i and p_i of the correlation function are chosen such that the likelihood that the available outputs were sampled from the Gaussian random field is maximized. Finding settings for the θ_i and p_i is done using *Differential Evolution* [**Storn and Price, 1997**] with population size 50, for 50 generations, followed by a hill-climbing phase. The used Kriging implementation was provided by Giles Endicott, implemented according to [**Forrester et al., 2008**].

2.3 · Surrogate-assisted Optimization

In *surrogate-assisted optimization*, approximation models are employed for improving efficiency in terms of the number of consumed quality evaluations by using model approximations as a *surrogate* indication of quality. In this usage, approximation models are hence addressed as *surrogate models*, or as *metamodels*, the latter referring to the fact that the quality function that is approximated can be seen as a model in its own right, namely, of actual real-world usefulness or applicability of a solution. In an attempt to more extensively utilize the information implicitly available from the sampled data, a surrogate model is trained on the solutions that were evaluated on quality during optimization, and its quality predictions are used for ranking solutions. Potentially, this way, less evaluations of the quality function are required for the same performance as in an optimization scheme that is not model-assisted, either enabling

to decrease the number of evaluations necessary for obtaining comparable results, or to find better solutions under the same evaluation budget.

In replacing quality function evaluations by model approximations, one should bear in mind that models are susceptible to introducing false and obscuring actual high-quality regions, depending on the interplay between the regularity of the quality landscape, the number and distribution of the training points, and the used (settings for the) modeling technique. Therefore, it is important to keep evaluating new solutions on actual quality during the optimization process for verification of the followed path, either via presenting the new information to the optimizer directly [**Jin et al., 2002, Emmerich et al., 2006**], followed by a model update (more "local" use of the model), or through restarting the search on an updated model [**Jones et al., 1998, Sóbester et al., 2004**] (more "global" use of the model). Selecting the most-promising solutions for on-line evaluation should be guided by the model predictions of the quality *and* by information on the distribution of the training points. This way, areas with relatively-low predicted quality, while nevertheless likely to harbor high-quality solutions, are considered as well, decreasing the tendency to skip over potential high-quality regions due to lack of information.

Next to the predicted quality of a solution, certain modeling techniques therefore provide the *uncertainty* associated with the prediction based on the distance to the available training points (e.g., Kriging, see Section 2.2, and *radial basis function networks* (RBFNs) [**Sóbester et al., 2004**]). The uncertainty of a predicted value $\hat{f}(\mathbf{x})$ for a solution $\mathbf{x}$ is provided as the standard deviation $s(\mathbf{x})$ of a normal distribution $\mathcal{N}\left(\hat{f}(\mathbf{x}), s^2(\mathbf{x})\right)$ around the predicted value $\hat{f}(\mathbf{x})$. The actual quality value $f(\mathbf{x})$ will be a realization of a random variable that is distributed according to $\mathcal{N}\left(\hat{f}(\mathbf{x}), s^2(\mathbf{x})\right)$ [**Jones et al., 1998**].

2.3.1 · Surrogate Utility Functions

The most-straightforward utility function is applying the approximated quality function $\hat{f}$ for ranking solutions directly, thus relying on the predicted quality values without considering uncertainty, i.e., for a quality function f that is to be minimized,

$$\operatorname*{argmin}_{\mathbf{x}} \hat{f}(\mathbf{x}). \tag{2.19}$$

According to [**Kushner, 1962**], the purpose of a utility function is to introduce a trade-off between sampling in expected high-quality regions versus sampling in

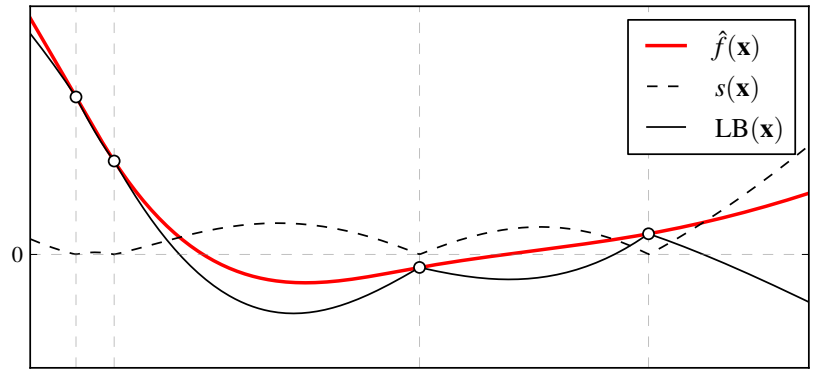

Figure 2.2 Lower Confidence Bound. A surrogate model is used that, fitted to training data of a to-be-minimized quality function, represented by white dots, provides both a quality prediction $\hat{f}(\mathbf{x})$ and an estimated associated prediction error $s(\mathbf{x})$ for unseen solutions $\mathbf{x}$. In minimizing LB$(\mathbf{x})$, here weighting both prediction and error equally, i.e., $w = 1$, solutions in the far-right corner get a good ranking, even though their expected quality values $\hat{f}(\mathbf{x})$ are suboptimal.

under-explored regions [**Emmerich, 2005**]. In Equation 2.19 the balance is tilted completely towards the first, selecting directly on the expected quality values. An intuitive ranking criterion taking both into account, as the predicted quality value $\hat{f}(\mathbf{x})$ and the associated uncertainty $s(\mathbf{x})$ are of the same scale, is combining them using a linear weighting. This results, for a to-be-minimized quality function, in ranking on the *lower confidence bound* (LB) [**Cox and John, 1997**, **Emmerich et al., 2006**, **Emmerich, 2005**],

$$\underset{\mathbf{x}}{\operatorname{argmin}}\, \mathrm{LB}(\mathbf{x}), \quad \mathrm{LB}(\mathbf{x}) = \hat{f}(\mathbf{x}) - w \cdot s(\mathbf{x}), \quad w > 0. \tag{2.20}$$

A lower w-value makes that there is more emphasis on exploitation of the expected high-quality regions, while a higher w-value drives the search towards areas with higher uncertainty, those areas that have been sampled less densely. Taking $w = 1$, the LB criterion is illustrated in Figure 2.2.

The most-frequently encountered utility function in literature that takes the prediction $\hat{f}(\mathbf{x})$ and associated error $s(\mathbf{x})$ into account, is the *expected improvement* (ExI) criterion. ExI quantifies how much an unseen solution's quality value is expected to improve upon the currently best-found quality value. It does so based on probability distributions that are parameterized according to these predictions and errors, e.g., Gaussian distributions $\mathcal{N}\left(\hat{f}(\mathbf{x}), s^2(\mathbf{x})\right)$, depending on what probability distributions are assumed in the used modeling technique.

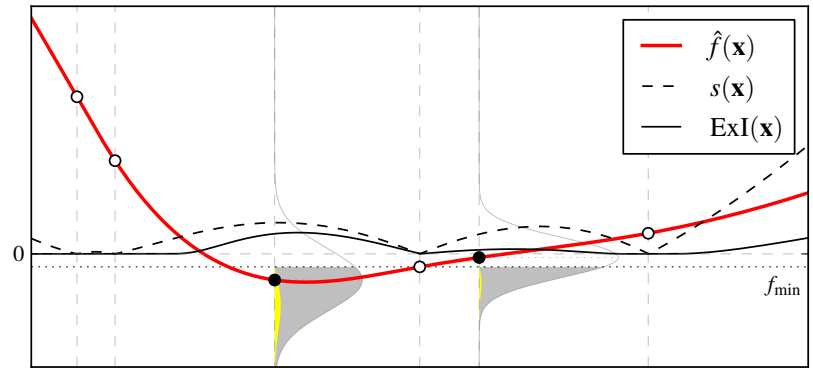

Figure 2.3 Expected Improvement. Based on prediction $\hat{f}(\mathbf{x})$ and estimated prediction error $s(\mathbf{x})$, the improvement that is expected to be realized in solution $\mathbf{x}$, with respect to $f_{\min}$, is calculated by summing over all possible improvements in $\mathbf{x}$ times their probability. Training points are indicated by white dots, $f_{\min}$ is the quality value of the best-quality training point. For two unseen solutions, indicated by black dots, the expected improvement ExI($\mathbf{x}$) is depicted by the size of the yellow surface.

Ranking solutions on expected improvement can be done using the ExI($\mathbf{x}$) utility function [**Mockus et al., 1978, Emmerich, 2005**],

$$\underset{\mathbf{x}}{\operatorname{argmax}}\, \mathrm{ExI}(\mathbf{x}), \quad \mathrm{ExI}(\mathbf{x}) = \int_{-\infty}^{f_{\min}} (f_{\min} - y) \cdot \mathrm{pdf}_{\mathcal{N}\left(\hat{f}(\mathbf{x}), s^2(\mathbf{x})\right)}(y)\, \mathrm{d}y, \tag{2.21}$$

$$= \int_{-\infty}^{f_{\min}} (f_{\min} - y) \cdot \mathrm{pdf}_{\mathcal{N}(0,1)}\left(\frac{y - \hat{f}(\mathbf{x})}{s(\mathbf{x})}\right) \mathrm{d}y. \tag{2.22}$$

For a candidate solution $\mathbf{x}$, all possible improvements with respect to the best-found quality value $f_{\min}$ are weighted by their probability, and summed-up. The possible improvements can be represented by the part of a vertical line through $\mathbf{x}$ that extends under the line $f_{\min}$, see Figure 2.3. The probabilities of these improvements are then indicated by the height of the probability density function at the same vertical position. Note that it is the standard deviation $s(\mathbf{x})$ that appears in the Equation 2.22, not variance $s^2(\mathbf{x})$. The expression can be simplified [**Jones et al., 1998, Emmerich, 2005**] to

$$\mathrm{ExI}(\mathbf{x}) = \left(f_{\min} - \hat{f}(\mathbf{x})\right) \cdot \mathrm{CDF}_{\mathcal{N}(0,1)}\left(\frac{f_{\min} - \hat{f}(\mathbf{x})}{s(\mathbf{x})}\right) + s(\mathbf{x}) \cdot \mathrm{pdf}_{\mathcal{N}(0,1)}\left(\frac{f_{\min} - \hat{f}(\mathbf{x})}{s(\mathbf{x})}\right). \tag{2.23}$$

The implicit trade-off that is present in ranking on ExI, i.e., between sampling expected high-quality regions and under-explored regions, depends on the ruggedness that is estimated from the training points. If this ruggedness is estimated greater than

the actual irregularity in the quality landscape, time might be wasted on solutions with suboptimal quality, as the errors $s(\mathbf{x})$ are too large, and thereby sampling in low-quality regions is encouraged. On the other hand, if the ruggedness is estimated smaller than the actual irregularity, this can cause the global optimum to be ignored.

2.4 · Summary

This chapter introduced techniques that will be used in the remainder of the thesis, either applying them directly or using them as inspiration, with the goal of assisting automated search in targeting *innovative solutions*. In applying the *CMA-ES*, an iterated adaptive search method used as underlying optimizer in all experiments, a problem-specific initial mean (i.e., initial solution) and stepsize are to be set for its adaptive search distribution.

Feed-forward neural networks (FFNNs) and *Kriging*, two approximation-modeling techniques, differ from each other in the sense that FFNNs are computationally less expensive to apply, and hence better applicable in scenarios with high-dimensional inputs or a large amount of training data. While Kriging is better at interpolating points located in-between training data, it converges to the average output value of the training points in extrapolating.

Surrogate-assisted optimization replaces the quality function by a surrogate model of it in ranking candidate solutions. During surrogate-assisted optimization, it is important to keep evaluating solutions on actual quality to verify the search path, because of possible modeling errors. Certain surrogate utility functions, such as the *expected improvement* (ExI), account for estimated modeling errors by including them in calculating the expected quality value of solutions.