6

Evolutionary Design of Emergent Behavior

Jürgen Branke, Hartmut Schmeck

Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany.
branke@aifb.uni-karlsruhe.de, schmeck@aifb.uni-karlsruhe.de

Summary. Most technical systems envisioned in organic computing are assumed to be complex, consisting of a large number of interacting components, self-organizing and exhibiting *emergent* behavior. As is argued in this chapter, a system's emergent properties surface only after realization or during a simulation of all interacting components. Thus, the usual "top-down" and "bottom-up" design paradigms have severe limitations when it comes to emergence. Instead, the use of evolutionary computation is advocated for the automated, simulation-based design of organic computing systems with emergent behavior.

Key words: Emergence, organic computing, evolutionary computation, design principles

6.1 Introduction

As a result of the continuing technological and scientific progress, the systems created by engineers, computer scientists and others become larger and larger, consisting of more and more highly interconnected and heterogeneous components. In short, they become ever more complex. On the other hand, the demand for reliability, adaptability and cost effectiveness remains or even increases. Being able to design and control such complex systems thus becomes a competitive necessity.

The organic computing initiative envisions computer systems of the future to be more flexible and adaptive, more autonomous, and with a stronger focus on user needs. In short, organic computers are self-organizing systems, able to dynamically adapt to a changing environment. They exhibit the so-called "self-x" properties, including self-organization, self-configuration, self-healing, self-adaptation, and self-protection. To achieve these goals, a life-like structure is envisioned, with many interacting, more or less autonomous subsystems, self-organizing into a coherent global system. This can be viewed to be analogous to complex natural systems, as e.g. societies of ants, termites, or wasps. An observer-controller structure [63, 22] is supposed to supervise and control the overall system.

On the one hand, such a structure relieves the designer from the need to foresee all possible events and rigidly program all system responses in every detail, as for example small faults can be self-repaired, the system can self-adapt to a variety of environments and users, and self-protect even against threats not yet existing at design time. On the other hand, the emergent behavior of such collections of interacting, self-adapting components poses new challenges to the designer.

Usually, emergence is defined as some global behavior of the overall system that can not be observed, and often not even deduced, from looking at the components individually, see, e.g., [46, 28]. Examples include traffic jams, the finding of shortest paths by ants, or the complex functionality of the brain consisting of many rather simple interconnected neurons. Note that the emergent behavior may be desired, as in the example of the brain, or undesired, as in the example of traffic jams.

Since emergent behavior only occurs if components are working together, the system's characteristics can not be derived by analyzing the individual parts, and analytical models of such complex systems usually don't exist[1]. But this also means that the usual "top-down" design principle is no longer appropriate. It is by definition impossible to know how design choices made on the component level affect the overall system behavior. Therefore, designing a set of interacting subsystems such that they generate a desired emergent global behavior , while preventing undesired emergent behavior, is a very challenging task.

This predicament has been discussed also by others, see e.g. [29, 34, 5, 53]. Most researchers agree that the design of complex, self-organizing systems with emergent properties is necessarily an iterative, step-wise refinement process, with extensive use of simulations and experiments. In fact, simulation seems to be the only practicable method of developing an understanding of the properties of organic systems, and we therefore conjecture that any promising design process has to involve simulation to evaluate a system's quality. Some tools have been proposed to support such a process, see, e.g., [9]. Nevertheless, the proper design methodology is still a heavily debated research issue. Eventually, the design process is often driven by trial-and-error and the intuition and ingenuity of an engineer. For inspiration, people often turn to principles from nature when designing self-organizing systems, see [57, 54] for compilations of such principles.

In this paper, we argue that simulation-based design, in its extreme form, basically corresponds to a black box search in the design space, using simulation to evaluate solutions. Although many real-world problems may not be

[1] Exceptions are perhaps physical systems, where the global behavior can be described by appropriately chosen differential equations

a complete black box, there are powerful black box optimization algorithms that could be used to automate simulation-based design of complex systems.

Design and optimization based on simulation have become practical only recently due to the immense increase in computational power. In the realm of simulation, entire complex systems can now be modeled realistically to the point of allowing accurate conclusions about the real world. And in the realm of design and optimization, it allows to develop and run iterative nature-inspired metaheuristics like evolutionary algorithms (EAs), which have proven to be successful in a wide variety of problem domains. Together, this provides, for the first time, a means to automatically optimize and design complex systems despite of the appearance of emergent phenomena.

Simulation-based design, however, poses a number of challenges:

1. **Evaluating a solution is time consuming.** Because simulating a complex organic system usually involves the simulation of a large number of interacting components, such a simulation is time consuming and computationally expensive.
2. **Evaluation is stochastic.** In many cases, a complex system contains some random component, be it only the environment it interacts with. As a consequence, the simulations used for evaluating complex systems involve a pseudo random number generator, and thus the observed performance is stochastic and depends on the random number seed. Such uncertainty in evaluation is a major challenge for design and optimization.
3. **Typical applications involve multiple objectives.** Although this aspect is not restricted to organic systems, many practical complex systems are supposed to satisfy a multitude of usually conflicting criteria. Since no single system is optimal with respect to all criteria, a compromise solution has to be found, which represents a proper trade-off of the different objectives.

Nature-inspired metaheuristics form one group of optimizers able to successfully tackle black box optimization problems. Nature-inspired optimization is a very active field of research, encompassing a number of different optimization approaches inspired by different natural phenomena. Among the most prominent representatives are simulated annealing [1], evolutionary algorithms [32], ant colony optimization [12], tabu search [40], or particle swarm optimization [33]. Their suitability for black box optimization alone would make them promising candidates for the design and optimization of complex systems. But, as will be demonstrated in this chapter, EAs are also able to address all the other challenges involved in the design and optimization of complex organic systems.

As a closely related field, simulation-based optimization has received increasing interest over the past years, good overviews can be found e.g. in [39, 67, 43]. So far, however, the area has been mostly concerned with the uncertainty of evaluations. A typical representative would be stochastic approximation [66], which is a variant of gradient search explicitly taking into account stochastic evaluation functions. Here, we will show how EAs can be adapted to handle such problems, and will additionally look at all the other aspects mentioned above and considered important when designing or optimizing complex organic systems.

This chapter is structured as follows. The next section will provide a brief introduction to EAs. Then we discuss the different challenges with respect to the optimization of complex organic systems, and how EAs can address them. First, in section 6.3, we discuss ways to run the algorithm in reasonable time despite of the usually large time to evaluate a single solution. In particular, we discuss parallelization and the use of approximate models. Then, Section 6.4 looks at ways to allow EAs to cope with stochastic evaluations. The consideration of multiple objectives and the algorithm's ability to focus on the most interesting solutions is treated in Section 6.5. Section 6.6 briefly describes two exemplary applications. The chapter concludes with a summary and some ideas for future work.

6.2 A brief introduction to evolutionary computation

A detailed description of evolutionary algorithms is out of the scope of this chapter, and the interested reader is referred to, e.g., [32, 35]. However, a brief outline of the algorithms' main features shall be provided here.

Evolutionary algorithms are randomized heuristic search methods based on the principles of natural evolution, or more specifically, on Darwin's theory of the survival of the fittest. The two driving forces of EAs are selection and diversification. Starting with a set of candidate solutions (population), in each iteration (generation), new individuals are created based on the current population (diversification). The two primary construction operators are crossover, which combines information from two solutions to form a new solution, and mutation, which modifies an existing solution locally. In the next step, out of this larger set of parents and offspring, the new set of individuals allowed to reproduce is selected. By continually selecting good solutions for reproduction and then creating new solutions based on the knowledge represented in the selected individuals, the solutions "evolve" and become better and better adapted to the problem to be solved, just like in nature, where the individuals become better and better adapted to their environment through the means of evolution.

There are four different main streams of evolutionary computation that have originally been developed independently and focused on different aspects, namely genetic algorithms [42], evolution strategies [58, 64], evolutionary programming [38], and genetic programming [50]. While all can be used for design and optimization, the latter may be particularly interesting for open-ended design, because it allows for variable-length descriptions of solutions, e.g. in the form of LISP-expressions, i.e. it imposes fewer restrictions on the search space.

As all metaheuristics, EAs are more or less black box optimization techniques and thus don't impose any constraints on the optimization problem, e.g., the fitness function need not be differentiable.

6.3 Timely execution despite expensive evaluations

If during the course of optimization, many candidate solutions have to be evaluated, and each evaluation involves a time-consuming simulation, the overall time required for optimization may be excessively long. Therefore, methods are needed to speed up the optimization.

We consider here two fundamental ways to achieve that goal: either the execution itself is accelerated, or the algorithm is modified such that it can work with fewer evaluations (and thus requires fewer simulation runs). The former can be achieved by, e.g., parallelization, the latter by replacing the time-consuming simulation with approximate evaluations. These aspects are discussed in the following subsections.

6.3.1 Parallelization

Parallelization can be implemented on different levels: The lowest level is the level of a single evaluation or simulation. However, parallelization on this level is very problem specific and, in particular, if the interaction between system components is not restricted locally, could turn out to be quite challenging.

The highest level would be to run several instances of the EA in parallel on different processors. Since EAs are randomized search algorithms, they would generate different solutions when started with different random seeds on the different processors. The final solution would then be the best solution found by either of the parallel runs. Although that sort of high level parallelization would be very easy to realize, intuitively, it is not very efficient, as the different runs don't exchange any information.

Parallelizing a single run on the algorithm level seems most promising. Luckily, EAs are relatively easy to parallelize, since the time consuming evaluation of a solution can be done in parallel and independently for different solutions, for surveys see, e.g., [3, 26, 61].

Clearly, all individuals created in one generation can be evaluated independently on different processors. Also, mutation and crossover could be done in parallel. Only for selection, a solution's quality has to be judged relative to to the quality of all other solutions in the population, and thus global knowledge is required. This can be achieved in a master-slave fashion, where the master process maintains the population and selects parents, but sends out the parents for crossover, mutation and evaluation to the slave processes. This very straightforward parallelization scheme incurs significant communication overhead, as individuals have to be sent out and recollected in every iteration. Therefore, the EA community has developed algorithmic variants with a more local selection, alleviating the need for global control. Most prominent among those are the island model and the diffusion model. In the island model, the population is divided into a number of subpopulations, which can run independent EAs on different processors. Only at regular intervals the subpopulations exchange some (usually the best) individuals with their neighbors in a so-called migration step. Communication is thus reduced to occasional migration. In the diffusion model, individuals are distributed spatially. In every generation, each individual selects a mating partner from its local neighborhood. The model is called diffusion model because the neighborhoods are overlapping, and a very good individual can spread only slowly (diffuse) from one local neighborhood to the next. The island model is ideally suited for workstation clusters with powerful processors and slow communication. The diffusion model with a local interaction structure is particularly well suited for massively parallel computers with a very fast local interconnection network.

Either model localizes selection, thereby creating temporary niches, in which also inferior individuals have a chance to survive for some time. As experience has shown, this effect is so beneficial that many people today actually implement either the island or the diffusion model even on a single processor.

All the approaches above more or less assume a dedicated parallel computer with equally powerful processors. However, in recent years, computer grids, i.e., the combination of available computers connected via Internet to form a virtual supercomputer, became a much cheaper and more accessible alternative. The power of computer grids has been demonstrated, e.g., by the project [65], which connected thousands of computers to search for extraterrestrial life, and companies like [56] or [74] commercialize the idea of networked computing. Clearly, computer grids have the potential to resolve the problem of high computer resources required by EAs, and will help pave the way to their still more widespread use.

However, the above parallelization schemes have to be adapted to a heterogeneous computer architecture with processors of vastly different power (with processor power actually varying over time, as only the computers' idle cycles are utilized), and slow and unreliable communication, see [20] for first steps in this direction.

6.3.2 Use of approximate models

Another way to reduce computational complexity and to speed up the procedure is to replace the usual evaluation by an approximate one. Such an approximate evaluation can be obtained through, e.g., response surface modeling. In the simplest case, methods from experimental design are used to determine a suitable set of potential solutions, called *design points*, that are evaluated and then used to construct a *metamodel*, a simple approximate model of the true evaluation function. Typical types of approximation models include regression, kriging, or artificial neural networks.

Given such a metamodel, the application of EAs is straightforward. However, such a two-step process of first constructing a metamodel and then using it for optimization assumes that a good solution with respect to the metamodel also represents a good solution with respect to the true evaluation. The validity of this assumption clearly depends on the quality of the metamodel. The dilemma is that constructing an accurate metamodel for the whole search space may require even more evaluations than running the optimizer directly on the original evaluation function.

A promising alternative to that two-step process is to interweave model construction and optimization: in the beginning of the optimization process, a rather crude model may be sufficient, and later on, information from the optimization run can be used to identify the most promising regions of the search space, where the model can be repeatedly refined. In most cases solutions are generated by the optimizer and evaluated using the metamodel. Then, it is decided which of the solutions should be evaluated accurately, and finally, the information gained by evaluating some solutions accurately is used to update the metamodel. EAs are particularly suitable for combination with approximation models because they are black box optimizers (and thus the accurate evaluation can easily be exchanged with an approximate model on a solution-by-solution basis), and because they repeatedly resample promising regions of the search space, thereby gathering information over time in the most interesting regions, which can be used to refine the model exactly there. Comprehensive overviews of this rather large research area can be found e.g. in [48, 47].

The use of a metamodel is particularly helpful in the context of applications with noisy evaluation functions, or when searching for robust solutions. In those cases, in order to obtain sufficiently accurate estimates of a solution's quality, repeated evaluation of each solution is required, which makes optimization particularly costly. Here, metamodels can help reduce the number of evaluations per solution, see, e.g., [25, 55, 59]. These issues are discussed in more detail in the following section.

6.4 Stochastic fitness

EAs rely on an appropriate balance between exploration and exploitation, i.e., between testing new regions of the search space and concentrating the search on the most promising regions. The primary operator for exploitation is selection, which is a prerequisite for the advancement of search. In EAs, there are two potential selection steps: Good individuals have a higher probability to be selected as parents (parent selection), and bad individuals are removed from the population to make way for new individuals (usually termed environmental selection).

Selection implies the ability to discriminate alternative solutions accurately by their quality, in order to separate the good from the bad. However, as has been stated above, when designing complex organic systems, evaluation is done by randomized simulation, and thus selection is subject to uncertainty. This obviously impacts the algorithm's ability to select. Many authors have addressed this issue, and, in this section, we discuss ways that allow nature-inspired metaheuristics to work despite the noise.

There are a number of reasonable optimization goals in the presence of uncertainty, ranging from worst case performance over expected performance to the probability of being above a specified level. The by far most thoroughly studied and arguably most important criterion is expected performance, which will be assumed for the remainder of this section.

In most of the literature, the issue of uncertain evaluation is divided into two categories:

1. **Noisy evaluation** generally assumes an underlying objective function $f(x)$, which is unknown and disturbed by additive noise, i.e., the observed fitness function can be described as $F(x) = f(x) + \delta$, with δ a random variable (usually normally distributed with zero mean). In this case the expected fitness is the underlying (unknown) function: $E(F(x)) = f(x)$.
2. **Search for robust solutions** usually assumes that the underlying objective function $f(x)$ can be accurately evaluated during optimization, but the final solution is subject to noise when it is implemented, and the fitness obtained is thus $F(x) = f(x + \delta)$. Such a setting is typical in the case of, e.g., manufacturing tolerances. However, even if the probability distribution of δ is assumed to be known, it is not possible to calculate the expected fitness $E(F(x)) = \displaystyle\int_{-\infty}^{+\infty} f(x+\delta)p(\delta)d\delta$, because an analytically closed form of the underlying fitness function is not available.

The above categorization makes sense, as it addresses different application areas, but the border between the categories is blurred, and very similar techniques have been successfully applied for both scenarios. The main differences are that, in the case of noisy evaluation, the noise is assumed to be uncontrollable, and it is impossible to evaluate without noise, while in the case of searching for robust solutions, the disturbances applied *during optimization* can be chosen deliberately, and only the final solution is subject to uncontrollable noise. Controllability allows for the use of statistical variance reduction techniques. Furthermore, the distribution of *fitness* values for a particular solution is often assumed normally distributed in noisy optimization, while it is usually quite irregular (skewed and non-normal) when searching for robust solutions (because the noise enters the fitness function). In the following, we will discuss the issue of uncertain evaluation in general, and specify the assumptions underlying all approaches. For a survey see, e.g., [48].

6.4.1 Multiple samples

The simplest way to reduce uncertainty is by evaluating a solution repeatedly and using the average as an estimate for the true mean fitness. Sampling n times reduces a random variable's standard deviation by a factor of $\sqrt{n}$, but on the other hand increases the computation time by a factor of n. This is a generally perceived trade-off: either one can use relatively exact estimates but evaluate only a small number of individuals (because a single estimate requires many evaluations), or one can let the algorithm work with relatively crude fitness estimates and allow for more evaluations (as each estimate requires less effort). For examples of papers using this simple approach see, e.g., [44, 75, 71]. Depending on the application, variance reduction techniques like common random numbers or Latin hypercube sampling can help improving the estimates [15].

6.4.2 Implicit averaging

Instead of removing the noise by averaging over multiple samples, one might just let the algorithm cope with the uncertainty. Already many years ago, researchers have argued that EAs should be relatively robust against noise, see e.g., [37], and recently a number of publications have appeared which support that claim at least partially [6, 7, 8]. In [52] it is shown that for infinite population size proportional[2] selection is not affected by noise. Similarly, in [73] it was shown that a genetic algorithm with random perturbations applied to the design variables behaves identically to a genetic algorithm working on the expected fitness values if the population size is infinite.

The reason is that promising areas of the search space are sampled repeatedly by the EA, and the population usually contains many similar solutions. When the population is large, the noise in evaluating an individual is very likely compensated by that of a similar individual. This effect has been termed "implicit averaging" [48]. A natural question is whether explicit averaging in the form of re-sampling or implicit averaging in the form of a larger population size would be more efficient, given a fixed total number of fitness evaluations per generation. Conflicting conclusions have been drawn in different investigations [37, 10, 45]. In [51] and [52], some simplified theoretical models are developed, which allow to simultaneously optimize both population and sample size.

6.4.3 Response surface modeling

Instead of implicitly averaging over neighboring solutions, one can explicitly exploit information about previously evaluated similar solutions. And since

[2] With proportional selection, an individual's probability to be selected is proportional to its quality relative to the sum of qualities of all other individuals in the population.

nature-inspired optimization heuristics repeatedly sample promising regions of the search space, such data is usually available. In the simplest case of noise applied to the decision variables, neighboring solutions can be regarded as samples, and a weighted average over neighboring individuals can approximate the integral over possible disturbances [13]. Assuming a locally smooth fitness landscape, this idea has recently been extended in [55], where local metamodels are constructed based on previous evaluations in the neighborhood. Then, numerical integration over the metamodel can be used to approximate the expected fitness. In the case of noise applied to the fitness values, smooth metamodels, which average out the noise, can also be applied. In [25], we have successfully used local regression for that purpose, similar ideas can also be found in [59, 60]. Such techniques improve the fitness estimates without requiring additional samples.

6.4.4 Statistical ranking and selection techniques

The probability of erroneous selection depends not only on the uncertainty, but more on the signal-to-noise ratio, i.e., fitness difference relative to fitness variance. If the signal-to-noise ratio is large, selection is unlikely to make any errors. If it is rather small, selection is very uncertain. Thus, it seems promising to adapt the effort spent reducing the noise by repeated sampling to the uncertainty in a particular selection decision, rather than using a fixed number of samples. Consequently, it has been suggested to use a higher sample size for individuals with higher estimated variance [2]. Similarly, [68] bases the sample size on an individual's probability to be among the best (and thus to survive to the next generation).

While these attempts certainly represent improvements over the simple strategy of sampling each solution a constant number of times, they ignore the huge and well-developed field of statistically sound ranking and selection techniques. The primary difference between ranking and selection procedures and optimization procedures is that the former assume a given, usually small set of systems that are exhaustively examined, while the latter attempt to search efficiently through a search space too large for exhaustive search. But selection among a given small set of alternatives is done in every iteration of nature-inspired optimization, namely when the memory is updated. A survey of the most important selection techniques, together with an extensive comparison and demonstration of the respective strengths and weaknesses is provided in [16]. They are all based on the idea of sequential sampling, i.e., they take some samples, and then iteratively decide which systems should be sampled next until a termination criterion is met. One first effort to integrate methods from ranking and selection into EAs can be found in [11], where sequential sampling techniques are used to divide the individuals in the population into groups of similar quality, which then receive the same probability to be selected as parents.

In [24], we have integrated a selection technique, KN++ [49], into an EA's tournament selection. Also, we have shown how to numerically derive an even better selection procedure tailored to a specific noise level. In our test environment, both approaches, KN++ as well as our new procedure, showed approximately the same performance as the standard procedure, but required only half the number of samples.

6.4.5 Noise-adapted selection

In [23], we followed a completely different approach based on the observation that many standard EA variants include a form of randomized selection. For example, in rank-based selection, the probability to select an individual as parent is proportional to its rank in the population. In stochastic tournament selection, two individuals are compared and the better one is selected with a probability $p > 50\%$. Randomized selection is usually motivated by the wish to maintain diversity and to escape local optima. Noise has a similar effect as stochastic selection, namely that the inferior solution is selected with some probability. Thus, it should be possible to accept the noise inherent in the optimization problem and to use it to (at least partially) replace the randomness in the optimization algorithm. This has been achieved with our noise-adjusted tournament selection (NATS) presented in [23]. In NATS, the probability to accept a solution depends on the observed fitness difference between the two solutions. We used bootstrapping to generate suitable acceptance probabilities such that the expected acceptance rate is as close as possible to the desired acceptance rate.

6.4.6 Further issues

In [2] it was probably first suggested that the sample size, and thus the accuracy of evaluation, should be increased over the run. [14] looks at this problem more closely, and, in an extensive computational study, observes that it is best to have high accuracy in the beginning of a run (presumably because in that phase, the algorithm selects a subregion of the search space to work on), and towards the end of the run (when local fine-tuning and selection of the final solution require more precision). [4] look at a slightly different problem, but also conclude that the sample size should increase over the run.

For the case of multiple uncertain objectives, [70] modifies the usual Pareto dominance criterion to take uncertainty into account.

6.5 Multiple objectives

Design and optimization of complex organic systems often involves the consideration of multiple, usually conflicting objectives. There is usually not one solution which is optimal with respect to all objectives, but a set of alternative solutions with different trade-offs. These solutions are generally called *Pareto optimal* or *efficient* whenever it is impossible to improve on such a solution in any criterion without suffering in at least one other criterion. Which of these solutions is the desired one depends on the preferences of the *decision maker* (DM). If these were known beforehand, e.g. in the form of a weighted combination of the objectives, the problem could be transformed into a single objective problem and solved in a standard way. However, very often the DM is unable to specify his or her preferences before the alternatives are known. It is therefore very convenient to have an optimization method capable of generating the whole set of Pareto-optimal solutions and to allow the DM to select among those afterwards.

EAs are population-based methods and thus capable of searching for all or a representative subset of the Pareto-optimal solutions in one run. In recent years, the field of evolutionary multi-objective optimization (EMO) has seen a dramatic rise in interest with thousands of papers published, a dedicated conference, and several books. For a comprehensive survey on the field, the reader is referred to [31]. An extensive and frequently updated repository of references can be found online [36].

The application of EAs to multi-objective problems is more or less straightforward. The main challenges are

1. to ensure convergence towards better solutions, and
2. to maintain a representative set of good alternatives.

To ensure convergence, one has to be able to determine when one solution should be preferred over another. Here, most approaches rely on the concept of non-dominance. A solution A is said to *dominate* a solution B, if A is at least as good as B in all objectives, and better in at least one objective. To ensure diversity along the front, among the non-dominated solutions, those in sparse areas are favored over those in crowded areas.

While generating the whole front of Pareto-optimal solution ensures that a DM's preferred solution is part of the solution set, such a solution set can contain a large number of alternative solutions, and selecting one may be tedious. Generating so many solutions is usually also time-consuming and would require rather large populations.

If we assume that for practical reasons the number of generated solutions should be small, one immediate question is how they should be selected from the large set of Pareto-optimal solutions, and whether the search can be focused on the most interesting solutions from the beginning. Most multi-objective approaches assume that the best representative set is equally distributed along the Pareto-optimal front. But although usually the DM can not a priori specify his or her preferences completely, some vague information is often available. Integrating this information into the optimization

and thereby biasing search towards the most interesting solutions holds the promise of reducing computation time and generating more relevant solutions.

There have been several attempts in this direction, see, e.g., [30, 21, 17]. Furthermore, it is possible to interactively learn user preferences during the optimization run, see, e.g., [72].

But even if no information about the DM's preferences is available, some solutions are more likely to be useful to the DM than others. In [18], we have proposed to evaluate solutions according to their expected marginal utility to a virtual DM. That is, we calculate how much additional value a solution provides according to a DM's utility function, averaged over an assumed distribution of possible utility functions. As a result, the algorithm using this diversity preservation mechanism exhibits a clear bias towards "knees", i.e., regions of the Pareto-optimal front with strong curvature. It has been argued before that these solution have particularly high practical relevance [27] because even a small improvement in either objective requires a significant worsening of the other objectives.

6.6 Exemplary applications

In this section, we will briefly discuss two exemplary applications where EAs have been successfully used to design complex systems.

The first example is the design of en-route caching strategies. In the Internet, document requests are routed from the requesting node point-to-point through the network to the node storing the document, then the document is sent all the way back to the requesting node. When hubs in the network become over-utilized, slowdowns and timeouts can disrupt the process. It is thus worthwhile to think about ways to minimize these effects. Caching, i.e., storing replicas of previously-seen objects for later reuse, has the potential to generate large bandwidth savings and in turn a significant decrease in response time. With en-route caching, each router in the network is equipped with a cache and may opt to store copies of some documents for future reuse [69]. The rules used for such decisions are called "caching strategies". Designing such strategies is a challenging task, because the different nodes interact, resulting in a complex dynamic system. The quality of a caching strategy can only be determined by simulation. [19] have demonstrated that genetic programming can be used successfully to design new effective caching strategies. The newly discovered caching strategy significantly outperformed all other state-of-the-art caching strategies tested.

Another example is the design of traffic light controllers. For simple controllers, where different phases are given fixed time intervals, theoretical analysis may still be possible. However, for adaptive controllers that adjust the signals based on traffic sensor data, a rigorous analysis does not seem to be possible and traffic planners usually rely on simulation to evaluate the quality of a traffic light controller. Goldate [41] has used EAs and traffic simulation to design a traffic light controller in a multi-objective setting, attempting to minimize travel time as well as the number of stops. Although his diploma thesis can only be seen as a feasibility study, the resulting adaptive controller outperformed a controller designed by a human expert. The nature-inspired design of traffic light controllers is now explored with more rigor in the DFG project on "Organic Traffic Control"[3], including aspects like dynamic adaptation to changing (macro-)traffic patterns and interplay between the traffic light controllers of neighboring crossroads.

6.7 Conclusion

Organic computer systems with life-like structure are advocated as a way to handle the increasingly complex adaptive systems created by engineers and computer scientists. Consisting of a large number of dynamically interacting components, these systems exhibit emergent global behavior, which is not deducible from the local actions of a single component. Organic Computing thus calls for a methodology to determine appropriate local actions leading to the desired global behavior. Another major challenge for the design of organic systems is to control this emergent global behavior such that undesired effects of emergence can be prevented. In this chapter, we have argued that due to the emergent behavior, simulation seems to be required to evaluate a system's quality, which makes simulation a central component of design.

As has been demonstrated, EAs are particularly suitable for simulation-based design and optimization for the following reasons:

- The quality of a complex adaptive system can usually only be evaluated by simulation, as no closed analytical description is available. Since EAs are black box optimization heuristics, they do not impose any restrictions on the fitness function and naturally satisfy this requirement.
- EAs can be run efficiently in parallel, even on heterogeneous hardware platforms like computer grids. Furthermore, it is possible to partly substitute costly simulations by approximate models that are easier to compute. These two aspects allow to significantly reduce the running time if necessary.
- Even standard EAs can cope well with uncertainty in evaluation (e.g. due to stochastic simulations). Furthermore, they can be enhanced with advanced statistical methods from, e.g., ranking and selection to perform even better.
- EAs maintain a population of solutions throughout the run. As such, they are particularly suitable to handle multiple objectives and uncertainty about user preferences by searching for different trade-offs in parallel.

However, even though the suitability of this methodology has been demonstrated in several exemplary applications, more work is needed to further refine the methods and to make them ready for widespread easy use. Also,

[3] The project is part of the DFG priority program SPP 1183

while most of the above aspects have been examined independently so far, integrating them into one method seems a natural next step.

Finally, it seems promising to integrate the engineer's knowledge more closely into the process, e.g., by using interactive EAs [5].

Acknowledgments

The authors gratefully acknowledge the funding of their research on organic computing by the German Science Foundation (DFG) under SCHM752/12-1 and SCHM752/14-1.

References

1. E. Aarts and J. Korst. *Simulated annealing and Boltzmann machines.* Wiley, 1989.
2. A. N. Aizawa and B. W. Wah. Scheduling of genetic algorithms in a noisy environment. *Evolutionary Computation*, pages 97–122, 1994.
3. E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–461, 2002.
4. L. A. Albert and D. E. Goldberg. Efficient evaluation genetic algorithms under integrated fitness functions. Technical Report 2001024, Illinois Genetic Algorithms Laboratory, Urbana-Champaign, USA, 2001.
5. C. Anderson. Creation of desirable complexity: strategies for designing self-organized systems. In D. Braha et al., editors, *Complex Engineered Systems*, pages 101–121. Springer, 2006.
6. D. V. Arnold and H.-G. Beyer. Efficiency and mutation strength adaptation of the $(\mu/\mu_i, \lambda)$-ES in a noisy environment. In Schoenauer et al. [62], pages 39–48.
7. D. V. Arnold and H.-G. Beyer. Local performance of the $(\mu/\mu_i, \lambda)$-ES in a noisy environment. In W. Martin and W. Spears, editors, *Foundations of Genetic Algorithms*, pages 127–142. Morgan Kaufmann, 2000.
8. D. V. Arnold and H.-G. Beyer. A comparison of evolution strategies with other direct search methods in the presence of noise. *Computational Optimization and Applications*, 24:135–159, 2003.
9. C. Bernon, V. Camps, M.-P. Gleizes, and G. Picard. Tools for self-organizing applications engineering. In G. D. Serugendo et al., editors, *Engineering Self-Organizing Systems*, volume 2977 of *LNAI*, page 283.298. Springer, 2004.
10. H.-G. Beyer. Toward a theory of evolution strategies: Some asymptotical results from the $(1 \overset{+}{,} \lambda)$-theory. *Evolutionary Computation*, 1(2):165–188, 1993.
11. J. Boesel. *Search and Selection for Large-Scale Stochastic Optimization.* PhD thesis, Northwestern University, Evanston, Illinois, USA, 1999.
12. E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: From natural to artificial systems.* Oxford University Press, 1999.
13. J. Branke. Creating robust solutions by means of an evolutionary algorithm. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature*, volume 1498 of *LNCS*, pages 119–128. Springer, 1998.

14. J. Branke. *Evolutionary Optimization in Dynamic Environments.* Kluwer, 2001.
15. J. Branke. Reducing the sampling variance when searching for robust solutions. In L. S. et al., editor, *Genetic and Evolutionary Computation Conference (GECCO'01)*, pages 235–242. Morgan Kaufmann, 2001.
16. J. Branke, S. Chick, and C. Schmidt. Selecting a selection procedure. Technical report, Fontainebleau, 2005.
17. J. Branke and K. Deb. Integrating user preferences into evolutionary multi-objective optimization. In Y. Jin, editor, *Knowledge Incorporation into Evolutionary Algorithms*, pages 461–478. Springer, 2004.
18. J. Branke, K. Deb, H. Dierolf, and M. Osswald. Finding knees in multi-objective optimization. In *Parallel Problem Solving from Nature*, number 3242 in LNCS, pages 722–731. Springer, 2004.
19. J. Branke, P. Funes, and F. Thiele. Evolving en-route caching strategies for the internet. In *Genetic and Evolutionary Computation Conference*, volume 3103 of *LNCS*, pages 434–446. Springer, 2004.
20. J. Branke, A. Kamper, and H. Schmeck. Distribution of evolutionary algorithms in heterogeneous networks. In *Genetic and Evolutionary Computation Conference*, volume 3102 of *LNCS*, pages 923–934, 2004.
21. J. Branke, T. Kaußler, and H. Schmeck. Guidance in evolutionary multi-objective optimization. *Advances in Engineering Software*, 32(6):499–508, 2001.
22. J. Branke, M. Mnif, C. Müller-Schloer, H. Prothmann, U. Richter, F. Rochner, and H. Schmeck. Organic computing - addressing complexity by controlled self-organization. In *International Symposium on Leveraging Applications of Formal Methods, Verification and Validation.* ACM, 2007.
23. J. Branke and C. Schmidt. Selection in the presence of noise. In E. Cantu-Paz, editor, *Genetic and Evolutionary Computation Conference*, volume 2723 of *LNCS*, pages 766–777. Springer, 2003.
24. J. Branke and C. Schmidt. Sequential sampling in noisy environments. In X. Yao et al., editor, *Parallel Problem Solving from Nature*, volume 3242 of *LNCS*, pages 202–211. Springer, 2004.
25. J. Branke, C. Schmidt, and H. Schmeck. Efficient fitness estimation in noisy environments. In L. Spector et al., editors, *Genetic and Evolutionary Computation Conference*, pages 243–250. Morgan Kaufmann, 2001.
26. E. Cantu-Paz. *Efficient and Accurate Parallel Genetic Algorithms.* Kluwer, 2000.
27. I. Das. On characterizing the 'knee' of the pareto curve based on normal-boundary intersection. *Structural Optimization*, 18(2/3):107–115, 1999.
28. T. De Wolf and T. Holvoet. Emergence versus self-organization: Different concepts but promising when combined. In S. A. Brueckner et al., editors, *Engineering Self-Organising Systems: Methodologies and Applications*, number 3464 in LNCS, pages 1–15. Springer, 2005.
29. T. De Wolf and T. Holvoet. Towards a methodology for engineering self-organising emergent systems. In H. Czap et al., editors, *Self-Organization and Autonomic Informatics*, pages 18–34. Springer, 2005.
30. K. Deb. Solving goal programming problems using multi-objective genetic algorithms. In *Congress on Evolutionary Computation*, volume 1, pages 77–84. IEEE, 1999.
31. K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms.* Wiley, 2001.

32. K. A. DeJong. *Evolutionary Computation*. MIT Press, 2002.
33. R. C. Eberhart and Y. Shi. *Swarm Intelligence*. Morgan Kaufmann, 2001.
34. B. Edmonds. Using the experimental method to produce reliable self-organised systems. In S. Brueckner et al., editors, *Engineering Self-Organising Systems*, volume 3464 of *LNAI*, pages 84–99. Springer, 2005.
35. A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.
36. Repository on multi-objective evolutionary algorithms. online, `http://www.lania.mx/$\sim$ccoello/EMOO/`.
37. J. M. Fitzpatrick and J. J. Grefenstette. Genetic algorithms in noisy environments. *Machine Learning*, 3:101–120, 1988.
38. L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley, 1966.
39. M. C. Fu. Optimizationn for simulation: Theory vs. practice. *INFORMS Journal of Computing*, 14(3):192–215, 2002.
40. F. Glover. Tabu search - part I. *ORSA Journal of Computing*, 1(3):190–206, 1989.
41. P. Goldate. Optimierung einer Ampelsteuerung mit Hilfe von evolutionären Algorithmen. Master's thesis, Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany, August 2003.
42. D. E. Goldberg. *Genetic Algorithms*. Addison-Wesley, 1989.
43. A. Gosavi. *Simulation-based optimization*. Kluwer Academic, 2003.
44. H. Greiner. Robust optical coating design with evolutionary strategies. *Applied Optics*, 35(28):5477–5483, 1996.
45. U. Hammel and T. Bäck. Evolution strategies on noisy functions, how to improve convergence properties. In Y. Davidor, H. P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature*, volume 866 of *LNCS*. Springer, 1994.
46. J. Holland. *Emergence - From chaos to order*. Addison-Wesley, 1998.
47. Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, 9:3–12, 2005.
48. Y. Jin and J. Branke. Evolutionary optimization in uncertain environments – a survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.
49. S.-H. Kim and B. Nelson. A fully sequential procedure for indifference-zone selection in simulation. *ACM Transactions on Modelin and Computer Simulation*, 11(3):251–273, 2001.
50. J. R. Koza. *Genetic Programming*. MIT Press, 1991.
51. B. L. Miller. *Noise, Sampling, and Efficient Genetic Algorithms*. PhD thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, 1997. available as TR 97001.
52. B. L. Miller and D. E. Goldberg. Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, 4(2):113–131, 1996.
53. A. A. Minai, D. Braha, and Y. Bar-Yam. Complex engineered systems: A new paradigm. In D. Braha et al., editors, *Complex Engineered Systems*, pages 1–21. Springer, 2006.
54. R. Nagpal. A catalog of biologically-inspired primitives for engineering self-organization. In G. D. Serugendo et al., editors, *Engineering Self-Organizing Systems*, volume 2977 of *LNAI*, pages 53–62. Springer, 2004.
55. I. Paenke, J. Branke, and Y. Jin. Efficient search for robust solutions by means of evolutionary algorithms and fitness approximation. *IEEE Transactions on Evolutionary Computation*, 10(4):405–420, 2006.

56. Parabon Inc. Company homepage. Online. http://www.parabon.com.
57. H. V. D. Parunak. "go to the ant": Engineering principles from natural multi-agent systems. *Annals of Operations Research*, 75:69–101, 1997.
58. I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipen der biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973.
59. Y. Sano and H. Kita. Optimization of noisy fitness functions by means of genetic algorithms using history of search. In Schoenauer et al. [62], pages 571–580.
60. Y. Sano and H. Kita. Optimization of noisy fitness functions by means of genetic algorithms using history of search with test of estimation. In *Congress on Evolutionary Computation*, pages 360–365. IEEE Press, 2002.
61. H. Schmeck, U. Kohlmorgen, and J. Branke. Parallel implementations of evolutionary algorithms. In A. Zomaya, F. Ercal, and S. Olariu, editors, *Solutions to Parallel and Distributed Computing Problems*, pages 47–66. Wiley, 2000.
62. M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors. *Parallel Problem Solving from Nature*, volume 1917 of *LNCS*. Springer, 2000.
63. T. Schöler and C. Müller-Schloer. An observer/controller architecture for adaptive reconfigurable stacks. In M. Beigl and P. Lukowicz, editors, *International Conference on Architecture Of Computing Systems*, volume 3432 of *LNCS*, pages 139–153. Springer, 2005.
64. H.-P. Schwefel. *Evolutionsstrategie und numerische Optimierung*. PhD thesis, Technische Universität Berlin, Germany, 1975.
65. Seti@home. Project homepage. Online. http://setiathome.ssl.berkeley.edu/.
66. J. C. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 31:332–341, 1992.
67. J. C. Spall. *Introduction to stochastic search and optimization*. John Wiley and Sons, 2003.
68. P. Stagge. Averaging efficiently in the presence of noise. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature V*, volume 1498 of *LNCS*, pages 188–197. Springer, 1998.
69. X. Tang and S. T. Chanson. Coordinated en-route web caching. *IEEE Transactions on Computers*, 51(6):595–607, 2002.
70. J. Teich. Pareto-front exploration with uncertain objectives. In E. Zitzler, K. Deb, L. Thiele, C. A. C. Coello, and D. Corne, editors, *Evolutionary Multi-Criterion Optimization*, volume 1993 of *LNCS*, pages 314–328. Springer, 2001.
71. A. Thompson. On the automatic design of robust elektronics through artificial evolution. In M. Sipper, D. Mange, and A. Peres-Urike, editors, *International Conference on Evolvable Systems*, pages 13 – 24. Springer, 1998.
72. D. S. Todd and P. Sen. Directed multiple objective search of design spaces using genetic algorithms and neural networks. In W. B. et al., editor, *Genetic and Evolutionary Computation Conference*, pages 1738–1743. Morgan Kaufmann, San Francisco, California, 1999.
73. S. Tsutsui and A. Ghosh. Genetic algorithms with a robust solution searching scheme. *IEEE Transactions on Evolutionary Computation*, 1(3):201–208, 1997.
74. United Devices. Company homepage. Online. http://www.ud.com.
75. D. Wiesmann, U. Hammel, and T. Bäck. Robust design of multilayer optical coatings by means of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 2(4):162–167, 1998.