SPOT—A Toolbox for Visionary Ideas

Thomas Bartz-Beielstein

Abstract

We present SPOT, an open-source toolbox for the experimental analysis of optimization algorithms. Evolution strategies (ES) have been severely criticized before they started their triumphal procession as optimization algorithms. Schwefel discussed ideas which appeared to be provocative at the first sight. SPOT provides a modern framework to test these ideas. The extreme programming methodology was used to implement the toolbox. Schwefel's idea that forgetting is as important as learning, which laid the cornerstone for comma-strategies in ES, is analyzed.

1. Introduction

Schwefel proposed innovative ideas that have "been laughed at" first. For example, Beyer and Schwefel (2002) wrote: "Had the (μ+1)-ES already been laughed at because it makes use not only of the so far best individual to produce an offspring, but also of the second best, even the worst out of μ parents, ..."

However, many of these ideas belong to the standard repertoire of modern optimization practitioners. We present a framework to test new ideas and to derive interesting conclusions: SPOT (sequential parameter optimization toolbox), a toolbox for testing new ideas in simulation and optimization.

Extreme programming is presented as a valuable tool for testing stochastic algorithms. As an example, Schwefel's idea that survival of the best ancestor is not always a good advice is analyzed. SPOT was able to reproduce results from Schwefel's original experiments.

2. The Sequential Parameter Optimization Toolbox

SPOT has been developed as a toolbox for the experimental analysis of algorithms. It consists of three modules:

1. optimization
2. prediction
3. report and analysis

Each module is extensible and can be used separately. Interfaces are kept as simple as possible.

The optimization module contains an implementation of an evolution strategy as presented in Beyer and Schwefel (2002). Optimization algorithms require the specification of exogenous strategy parameters, e.g. population size in ES, before they can be started. One parameter setting is considered as a design point. The prediction module generates design points that might improve the algorithm's performance. It uses the sequential parameter optimization (SPO) framework (Bartz-Beielstein 2006), which combines classical and modern statistical techniques such as DOE (design of experiments) and DACE (design and analysis of computer experiments). The third module is a collection of report generators and visualization routines for the analysis of the results.

Before we discuss the applicability of SPOT to analyze new ideas in simulation and optimization, the implementation of the ES in the optimization module is presented.

3. Extreme Programming

Because ES are stochastic search algorithms, it is difficult to test their correctness. Extreme programming (XP) tools have been proven to be valuable for our implementation. In contrast to waterfall-development approaches, XP demands tests should be automated and performed continuously. Code does not consist of solid blocks, it is in a liquid state: it can be refactored or even completely thrown away and rewritten. “Everything flows, nothing stands still” (Fig.2).

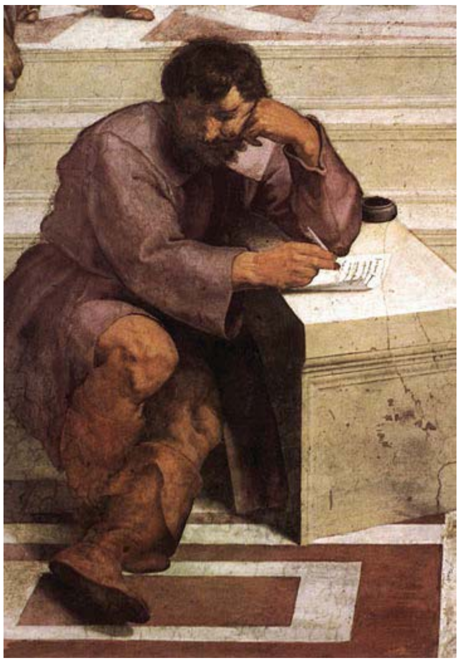
Figure 2: Heraclitus, Detail of Rafaello Santi's "The School of Athens" (1510), Vatican collection, Rome. For Heraclitus everything is "in flux", as exemplified in his famous aphorism “Everything flows, nothing stands still". Fb78, Wikimedia Commons, lizenziert unter CreativeCommons-Lizenz by-sa-2.0-de,URL: http://creativecommons.org/licenses/by-sa/2.0/de/legalcode

XP comprehends four basic techniques: Coding, testing, listening, and designing (Beck 1999). Features of the software do not exist until they are verified and validated by tests. We take a glance at testing, because it is a key practice in XP.

Each part of the program is isolated in XP. The goal of the so-called unit testing is to show that these individual parts are correct. Unit tests are pieces of source code written by a programmer to test a small and clearly specified part of software (Hunt and Thomas 2003). Assertions indicate that the source code behaves as expected. The following assertion `assertTrue` tests whether the condition is true. Figure 3 presents a simple example in JAVA. If the condition is not fulfilled (Fig. 4), a program terminates.

```
public void assertTrue (boolean cond){
if (!cond){
         abort();
         }
}
```

Figure 3: A simple assertion. The program is aborted if the test fails

The concepts discussed so far consider deterministic computer programs—evolution strategies are by definition stochastic search algorithms. The deterministic behaviour of the test in Fig. 4 can only be guaranteed if the value 2 is assigned deterministically to the variable a. The test would produce nonsense if `a` is a random variable.

```
int a=2;
xxxx xxxx
xxx xxxx
assertTrue(a == 2);
xxx xxxxxx xx
xxxxxxxxxx x
```

Figure 4: This program terminates if the test fails

Randomness is a fundamental concept for evolution strategies. Randomness complicates testing, because the arguments for the assertions are not deterministic, their results are not predictable. However, results are certain if the random stream seeds are set deterministically. One can go even one step further: replace ES-functions (methods) with deterministic functions. This concept is referred to as “de-randomization” in the following. De-randomization here is loosely related to techniques in complexity theory, but not to the de-randomized step-size adaptation in ES. De-randomization generates mock objects from random objects. Mock objects have dummy behaviour, they replace domain objects. In JAVA, Mock objects implement an interface (Fig. 5).

```
public interface IMutation {
  public Individual mutate(Individual
x);
}
```

Figure 5: Mock objects implement an interface. They are used in our example to imitate the behaviour of mutation operators

To give an example, we consider the mutation operator in ES. The real mutation operator is implemented as shown in Fig. 6, whereas the mock implementation is shown in Fig. 7. The code can be tested if the ES-mutation operator is initialized with a `Mock-Mutation` object. The real code uses the `Mutation` operator.

```
public class Mutation implements IMuta-
tion{
 public Individual mutate(Individual x){
 //the real mutation which uses
 // randomness
 xxxxxxxxxxx…
 xxxx
 xxxxxxxx
 }
}
```

Figure 6: The real mutation operator can be implemented without any modification. It does not “see” the mock routines

Further ES operators, e.g., recombination or selection, can be implemented and tested in a similar manner. The reader is referred to the SPOT documentation for details. To show that SPOT can be applied to analyze new ideas, we now investigate one idea from "Natural Evolution and Collective Optimum-Seeking".

```
public class MockMutation implements
IMutation{

 private Individual b;

 public void setMutationRsult(Individual
x) {

  // method to control the deterministic

  // behavior during the test

  b=a;

 }

 public Individual mutate(Individual x){
 // the mock mutation which produces
 // deterministic results
  return b;
  }
}
```

Figure 7: The mock mutation operator returns deterministic results

4. Forgetting and Learning: Schwefel’s Results

The idea "forgetting is as important as learning" (Schwefel 1992) sounds provocative at the first sight. Why should we skip available information in decision making? Is there any context where forgetting is beneficial?

Schwefel wrote: " 'Survival of the fittest', often taken as Darwin's view, turns out to be a bad advice. Forgetting, i.e. individual death, and even regression show up to be necessary ingredients of the life game."

He described the (1+1)-ES as a representation of Darwin's 'survival of the fittest' selection principle (Schwefel 1992):

"According to a given selection criterion, a descendant is rejected if its vitality is less than that of its ancestor, the ancestor otherwise. This scheme may be called a (1+1) or a two-membered evolution strategy [...], resembling the "struggle for life" between one ancestor *and* one descendent."

The (1+1)-scheme can be extended to (1+λ), (μ+1), or even (μ+λ) selection schemes—all these plus schemes use the survival of the best. Schwefel introduced the so-called comma-strategies, the (μ,λ) versions: parents are no longer included in the selection. This ES version requires a birth-surplus. Note, that information concerning the best found position so far (object parameters) is forgotten. However, information concerning "good" internal models (strategy parameters) are inherited. Formulated as a hypothesis, Schwefel's idea reads:

(H) *Survival of the best ancestor is not always a good advice*.

Schwefel used the following experimental setup (experimental design in the jargon of statistics) to test (H): An (1,10)-ES is compared to an (1+10)-ES on the 30-dimensional sphere function $f_1 = \sum x_i^2$. The progress rate was measured as $\log\sqrt{(F_0/F_g)}$, where F_0 denotes the start value, and F_g the current value at generation g. Schwefel demonstrated that the (1,10)-ES performs better than the (1+10)-ES (Fig.8).

He explained this behaviour as follows (Schwefel 1987):

"If an ancestor happens to arrive at a superior position, this might be - by chance - in spite of a non-optimum step size, or a step size which is not suitable for further generations. The (1+λ) scheme preserves the unsuitable step size as long as with it a further success is placed. This leads to periods of stagnation. Within a (1,λ) ES the good position, occasionally won with an unsuitable step size, is lost, together with the latter, during the next generation. This short term regression, however, enhances the long term velocity of the whole process by a stronger selection with respect to the suitable step size (strategy parameter). In other words: Forgetting is as important as learning, the former must be seen as a necessary integral part of the latter. One might interpret the fact of an inherent finite life time of living beings (preprogrammed maximum number of cell divisions) as an appropriate measure of nature to overcome the difficulties of undeserved success - or, in a changing environment, of forgetting outdated `knowledge'."

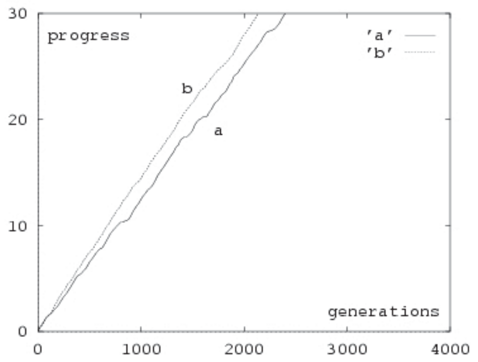

Figure 8: Self learning of one common mutation step size for the sphere function f_1. a) (1+10) evolution strategy, b) (1,10) evolution strategy. Higher progress [rate] values are better (Schwefel 1987).

5. Forgetting and Learning: SPOT Results

We repeated Schwefel's experiments with SPOT. The results from the previous section could be reproduced (Fig. 9).

Although Schwefel did not specify the exact experimental design, i.e., starting points, initial step sizes, and other exogenous strategy parameters, his results are reproducible—at least qualitatively. The curves depicted in Figs. 8 and 9 look similar even though the exogenous parameters were varied.

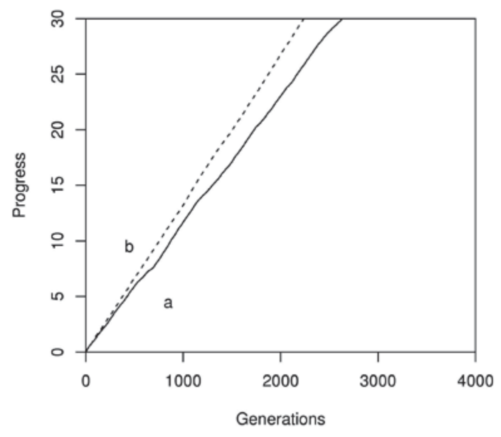

Figure 9: Self learning of one common mutation step size for the sphere function f_1. a) (1+10) evolution strategy, b) (1,10) evolution strategy. Higher progress rate values are better. This figure was generated with SPOT, whereas Fig. 8 is the original figure presented in Schwefel (1987)

6. Summary

We have introduced SPOT, a toolbox for the experimental analysis of algorithms, to discuss briefly one visionary idea, Schwefel's "forgetting is as important as learning" principle. SPOT provides a set of modern software tools to test randomized search algorithms and to confront visionary ideas with reality.

SPOT can be used to test further visionary ideas, e.g. the idea that "many wrongs do make a right" which is related to recombination. Schwefel (1992) comments: "If more than one, i.e. not only the best of the descendants, become parents of the next generation and recombination by sexual propagation takes place, i.e. mixing of the information gathered by different individuals during the course of evolution, then over-adaptation and consecutive stagnation can be overcome. Now the convergence rate steeply goes up with the population size." SPOT enables interested readers to perform related experiments.

Schwefel propagated many other ideas, some of them have been "re-invented" several decades later, e.g. random search, asynchronous parallelism, self-adaptation, varying population sizes, meta-strategies, the importance of robustness (effectivity), ageing concepts, neighbourhood models, prey-predator models, The reader is referred to Schwefel's articles, many of them are freely available on the internet.

The SPOT software and the results discussed in this article are available for other researchers to test the huge potential of evolution strategies.

We close with an anecdote: One of Schwefel's friends, a researcher who enjoys an excellent reputation, stayed as a guest in Dortmund. Inspired by studies in Schwefel's famous library he improved his own optimization algorithm. After the first day had passed, he proudly presented his improved algorithm: it used a modified selection operator that worked in a similar manner as selection in ES. On the next day, his algorithm even worked better, because he modified a second operator in an ES-like manner. On the third day, he reported that taking over the real-valued representation from ES resulted in a significant improvement.

This went on until the end of the week. Finally, he confessed: “Hans-Paul, my algorithm works much better now—but it looks like your ES.”

Literature

Beck, Kent (1999) Extreme Programming Explained: Embrace Change. Addison-Wesley, New York.
Hunt, Andrew and Thomas, David (2003) Pragmatic Unit Testing - in Java with JUnit, The Pragmatic Bookshelf, Lewisville.
Schwefel, Hans-Paul (1981) Numerical Optimization of Computer Models. Wiley, Chichester.
Schwefel, Hans-Paul (1987) Collective phenomena in evolutionary systems. In P. Checkland and I. Kiss, editors, Problems of Constancy and Change - The Complementarity of Systems Approaches to Complexity, Proc. 31st Annual Meeting, volume 2, pages 1025-1033, Int'l Soc. for General System Research, Budapest.
Schwefel, Hans-Paul (1992) Natural evolution and collective optimum seeking. In A. Sydow, editor, Computational Systems Analysis - Topics and Trends, pages 5-14. Elsevier, Amsterdam.
Bartz-Beielstein, Thomas (2006) Experimental Research in Evolutionary Computation, Springer, Heidelberg.
Beyer, Hans-Georg and Schwefel, Hans-Paul (2002) Evolution strategies – A comprehensive introduction, Natural computing 1:3-52.