

Chapter 3

Problems and Limitations of Bionic Optimization

Tatiana Popova, Iryna Kmitina, Rolf Steinbuch, and Simon Gekeler

We have seen that Bionic Optimization can be a powerful tool when applied to problems with non-trivial landscapes of goals and restrictions. This, in turn, led us to a discussion of useful methodologies for applying this optimization to real problems. On the other hand, it must be stated that each optimization is a time consuming process as soon as the problem expands beyond a small number of free parameters related to simple parabolic responses. Bionic Optimization is not a quick approach to solving complex questions within short times. In some cases it has the potential to fail entirely, either by sticking to local maxima or by random exploration of the parameter space without finding any promising solutions. The following sections present some remarks on the efficiency and limitations users must be aware of. They aim to increase the knowledge base of using and encountering Bionic Optimization. But they should not discourage potential users from this promising field of powerful strategies to find good or even the best possible designs.

3.1 Efficiency of Bionic Optimization Procedures

Iryna Kmitina and Tatiana Popova

Bionic Optimization strategies have proven to be efficient in many applications, especially where there are many local maxima to be expected in parameter spaces of higher dimensions. In structural mechanics, the central question is whether one particular procedure is to be preferred generally or if there are different problem types where some procedures are more efficient than others. Evolutionary

T. Popova • I. Kmitina • R. Steinbuch • S. Gekeler (✉)
Hochschule Reutlingen, Reutlingen Research Institute, Alteburgstraße 150, 72762 Reutlingen, Germany
e-mail: Tatiana.Popova@Reutlingen-University.DE; Iryna.Kmitina@Reutlingen-University.DE; Rolf.Steinbuch@Reutlingen-University.DE; Simon.Gekeler@Reutlingen-University.DE

Optimization with some sub-strategies, Particle Swarm Optimization, Artificial Neural Nets, along with hybrid approaches that couple the aforementioned methods have been investigated to some extent. These approaches are not uniquely defined, but rather imply many variants in the definition and selection of next-generation members, varying settings of the underlying processes, and criteria for changing strategies. Some simple test problems were used to quantify the performance of these different approaches. The measure of the procedures performance was the number of individuals which needed to be studied in order to come up with a satisfactory solution. As our main concern is problems with many parameters to be optimized, artificial neural nets do not show sufficient convergence velocities in our class of optimization studies to be included. Evolutionary Optimization, including its subclasses of Fern Optimization, and Particle Swarm Optimization prove to be of comparable power when applied to the test problems. It is important to note that, for all these approaches, some experience of the optimization parameters has to be gathered. In consequence, the total number of runs or individuals necessary to do the final optimization is essentially larger than the number of runs during this final optimization. Good initial proposals prove to be the most important factor for all optimization processes.

3.1.1 Comparing Bionic Optimization Strategies

As discussed in many sections of this book, Bionic Optimization may be defined by many different approaches. In this section, we deal with some of the most commonly accepted classifications, without taking into account all the many sub-classifications that might be found in the literature. The central approaches we compare are:

- Evolutionary Strategy (EVO, Sect. 2.1) (Rechenberg 1994; Steinbuch 2010)—where paired or crossed parents have children by the combination and mutation of their properties. These children, or some of them, are parents in the next generation.
- Fern Strategy (FS, Sect. 2.2)—which may be regarded as a simplification of Evolutionary Optimization. Individuals have offspring by mutation only, not by crossing properties with other members of the parent generation.
- Particle Swarm Optimization (PSO, Sect. 2.3) (Coelho and Mariani 2006; Plevris and Papadrakakis 2011)—where a population drifts through the possible solution space. The swarm’s coherence is given by simple rules about the velocity of the individuals.
- Artificial Neural Nets (ANN, Sect. 2.4) (Berke et al. 1993; Lagaros and Papadrakakis 2004; Widmann 2012)—where training of the net yields an understanding of the solution space and allows the prediction of the system’s response to given input. As ANN are not very efficient when applied to problems with many free parameters, we do not discuss them here (Widmann 2012).

3.1.2 Measuring the Efficiency of Procedures

To quantify the efficiency of the different optimization strategies, we must introduce a measure that allows us to uniquely define the amount of work required to achieve a predefined quality. From previous experience, we propose using the number of individuals to be analyzed before coming close to an accepted good value. This requires knowledge of what a good solution would be, which is generally not known when we start studying new problems.

To consider the violation of boundary conditions (cf. Sect. 2.9) we restrict our present study to the use of penalty functions. The geometric input is set to the minimum or maximum value, if the randomly produced data exceed the respective limits. For PSO, we invert the particle's previous velocity, if it violates given limits in addition to the penalty value. This combined approach has the advantage of simple applicability.

3.1.3 Comparing the Efficiency of Bionic Optimization Strategies

Optimization is an expensive and time consuming process. We need to understand which procedure and which combinations of parameters may lead to a good and acceptable result within a reasonable amount of time.

Test Examples

Figure 3.1 depicts the five test examples used while Table 3.1 summarizes their data. We want to minimize the mass of the frames by varying the rods' cross sections without exceeding their maximum stresses and displacements. The grid size of the examples is 1000 mm, except for example F2 where the grid size is 360 in. Example F2 used imperial units (in., kip) the other frames use mm and Newton.

To come up with comparable results, we performed a series of 20 loops for each problem and each strategy to avoid having only one or few very good or very bad results. Also, the optimization settings we used were based on previous experience with the underlying problems, so the number of runs presented does not come from naïvely starting a procedure, but includes some preliminary work which is impossible to quantify.

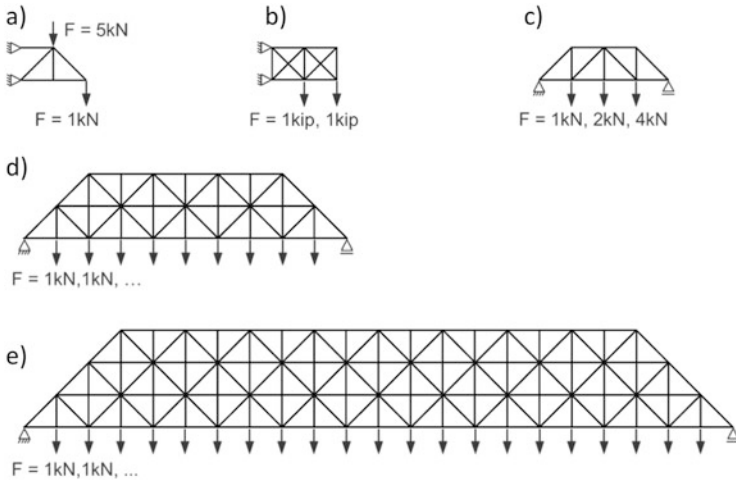


Fig. 3.1 Test frames with loads and supports. (a) F1: 6 rods frame. (b) F2: 10 rods frame. (c) F3: 13 rods frame. (d) F4: 58 rods frame. (e) F5: 193 rods frame

Table 3.1 Data of test problems

Frame	Free param.	Grid size	Amax/Amin	E-Mod	σ_{max}	d_{max}
F1	6	1000 mm	600/20 mm ²	200 GPa	120 MPa	0.5 mm
F2	10	360 in.	35/0.1 in. ²	10 Msi	25 ksi	2.0 in.
F3	13	1000 mm	400/20 mm ²	200 GPa	50 MPa	0.5 mm
F4	58	1000 mm	400/20 mm ²	200 GPa	100 MPa	2.0 mm
F5	193	1000 mm	600/20 mm ²	200 GPa	450 MPa	20 mm

Parameters: # of rods in frame; grid size: horizontal or vertical distance between the nodes; Amax, Amin: maximum and minimum allowed cross section area of the rods; E-Mod: Young’s modulus; σ_{max} : maximum allowed stress in rod; d_{max} : maximum allowed displacement of nodes

Input and Results of the Test Examples

Tables 3.2, 3.3 and 3.4 list the inputs of the test runs used. Table 3.5 and Fig. 3.2 (individuals per loop) summarize the results of the test runs. The most important data are the number of individuals analyzed to find a sufficient good design labelled as ‘Individuals [1000]’. The number given multiplied by 1000 gives the total number of individuals required to find the proposed design. *mean* and *stddev* (standard deviation) and *best* are descriptions of the results of the 20 runs. The ratio of the difference between the best and the average result divided by the standard-deviation (*reldev*) gives an idea of the stability of the strategy.

Table 3.2 Optimization settings used for EVO

Model	Parents	Kids	Mut. rad. max	Mut. rad. min	Generations
F1	10	20	0.5	0.05	60
F2	5	10	0.5	0.05	40
F3	5	10	0.5	0.05	50
F4	50	100	0.5	0.05	100
F5	100	200	0.5	0.05	200

Table 3.3 Optimization settings used for FS

Model	Parents	Kids/parent	Mut. rad. max	Mut. rad. min	Generations
F1	10	5	0.5	0.05	100
F2	10	4	0.5	0.05	50
F3	20	5	0.5	0.05	100
F4	100	5	0.5	0.05	200
F5	200	5	0.5	0.05	200

Mutation radius reduced for EVO and FS: 0–25 % of generations: $r_{mut} = 0.50$, 25–50 % of generations: $r_{mut} = 0.20$, 50–75 % of generations: $r_{mut} = 0.10$, 75–100 % of generations: $r_{mut} = 0.05$

Table 3.4 Optimization settings used for PSO

Model	Particles	Generations
F1	10	100
F2	10	50
F3	20	100
F4	100	200
F5	200	200

Weighting factors: $c_1 = 0.08$, $c_2 = 0.005$, $c_3 = 2.0$

Interpretation of the Results

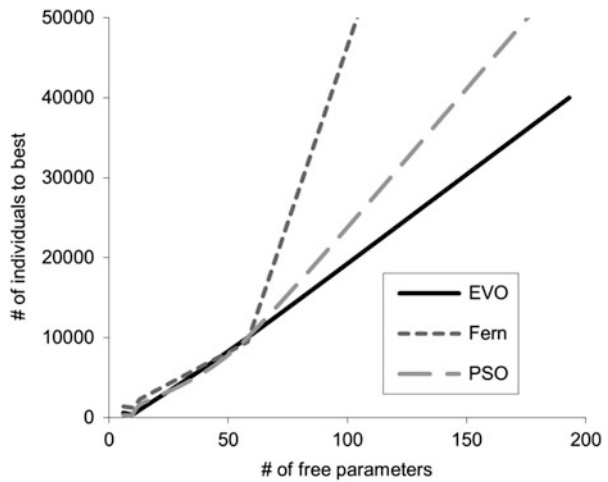
EVO, FS and PSO prove to be of a comparable efficiency when applied to the four smaller problems (F1, F2, F3, F4). Figure 3.2 indicates that there might be a nearly linear relation between the number of optimization variables and the individuals required to find good proposals. For the largest problem, F5, FS displays a performance that is essentially weaker than EVO and PSO. EVO and PSO seem to be of comparable power when applied to the problem class which we discuss here. FS shows promising results if the number of free parameters is not too large, but is less successful in random search in high dimensional spaces. The scatter indicator *reldev* proposes that PSO has a more stable tendency to find solutions near the best, while EVO and FS show a larger range after the 20 runs.

Some knowledge may be gleaned from the results of these series of studies. Foremost, that optimization, especially Bionic Optimization, is a process that consumes large amounts of time and computing power. Furthermore the results

Table 3.5 Results of 20 optimization runs per problem

Strategy	Model	Mean	Stddev	Best	Reldev	Individuals [1000]
EVO	F1	1.62e6	0.716e3	1.62e6	1.50	12
	F2	6.33e4	4.50e3	5.47e4	1.90	8
	F3	2.56e6	6.99e4	2.48e6	1.11	20
	F4	1.03e7	4.18e5	8.65e6	3.92	200
	F5	1.98e7	1.07e6	1.58e7	3.65	800
FS	F1	1.66e6	4.49e4	1.62e6	0.81	28
	F2	6.39e4	4.43e3	5.45e4	2.09	25
	F3	2.50e6	2.29e4	2.47e6	1.19	46
	F4	9.91e6	2.77e5	9.39e6	1.86	189
	F5	2.33e7	4.18e5	2.25e7	2.15	2570
PSO	F1	1.65e6	1.71e4	1.62e6	1.61	6
	F2	5.87e4	5.61e3	5.15e4	1.27	8
	F3	2.50e6	2.53e4	2.48e6	1.02	32
	F4	8.90e6	1.68e5	8.68e6	1.22	200
	F5	1.54e7	0.18e4	1.53e7	1.70	1120

Fig. 3.2 Efficiency of the three Bionic Optimization strategies tested



presented in this section would not have been found without a large number of preliminary studies providing experience in the field of frame optimization.

The input characteristics used in the test runs is derived from these preliminary studies. For example the selection of the three weighting factors $\{c_1, c_2, c_3\}$ for the PSO required 100,000 runs. The proposal of the reduction of the mutation range for EVO and FS is the result of many studies as well. The proposal to use a number of initial parents or individuals in the size of the number of free variables for EVO and PSO is based on many studies, as well as the idea to use a large number of initial parents and a small number of children in FS.

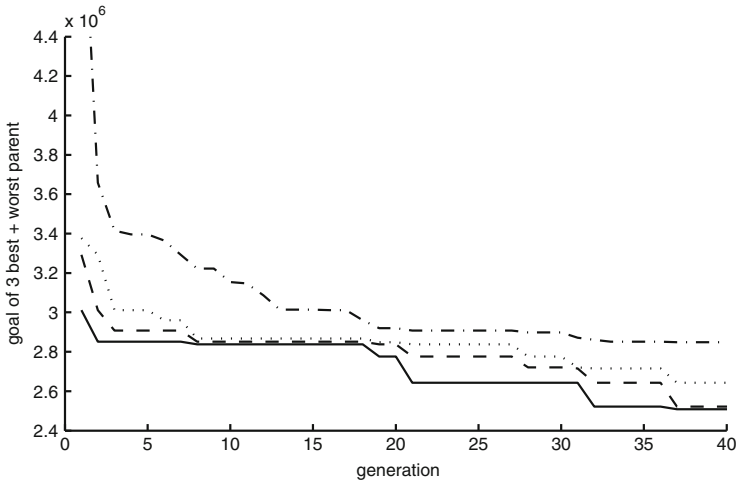


Fig. 3.3 History of an Evolutionary Optimization for example F3 (Fig. 3.1)

One central fact about all optimization may be learned from Fig. 3.3. If there is a good initial design, the number of optimization runs to be done may decrease significantly. If an experienced engineer proposes an initial design with a goal of e.g. 2.7×10^6 , we need only 20 generations or 50 % of the workload required to solve the task with a random initial design.

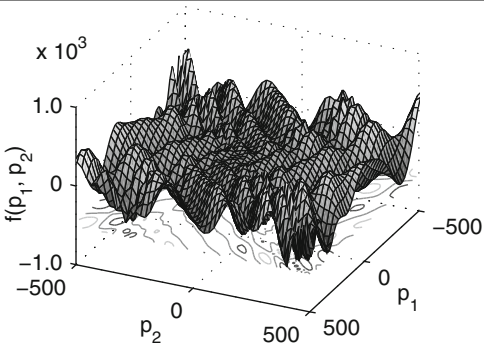
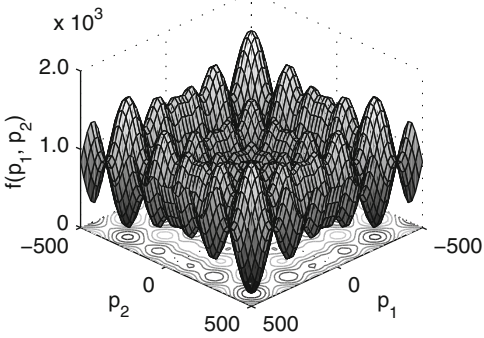
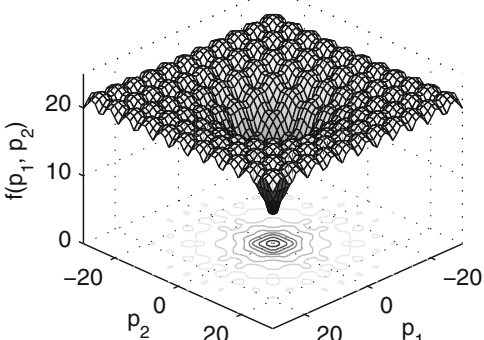
Further Test Examples

In addition to the trusses (Fig. 3.1) representing static structural optimization problems, there are many other test examples available (Surjanovic and Bingham 2015). Most of them are defined by mathematic functions and because of their characteristics and the known data of global and local optima, they are most suitable for algorithm testing. A selection of such benchmark problems is listed in Table 3.6. Some of them can be expanded to an arbitrary number of dimensions d . Especially in field of optimization they are useful in algorithm development or to improve existing procedures. Furthermore, with awareness of these problems, users can gain experience in optimization strategies, check their efficiency and learn how to choose proper optimization settings.

3.1.4 Conclusions

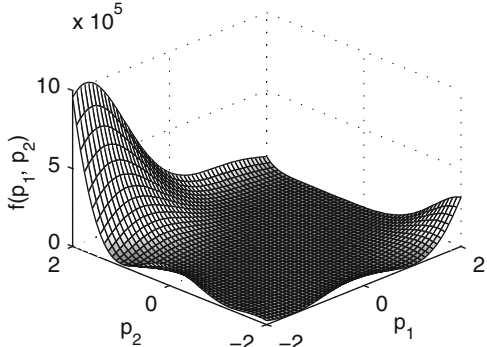
The quality of the initial proposals is the most important component of any optimization. If experienced and motivated engineers propose designs that are close to the optimal ones, there is a good chance that at least a local optimum

Table 3.6 Common test functions used for testing optimization algorithms

<i>Eggholder function</i>	
$f(p_1, p_2) = -(p_2 + 47) \sin\left(\sqrt{\left p_2 + \frac{p_1}{2} + 47\right }\right) - p_1 \sin\left(\sqrt{\left p_1 - (p_2 + 47)\right }\right)$	
	Free parameters: $i = 1, 2$ Search domain: $p_i \in [-512, 512]$ Global optimum (min): $f(512, 404.2319) = -959.6407$
<i>Schwefel function</i>	
$f(\mathbf{p}) = 418.9829 * d - \sum_{i=1}^d p_i * \sin(\sqrt{ p_i })$	
	Free parameters: $i = 1, \dots, d$ Search domain: $p_i \in [-500, 500]$ Global optimum (min): $f(p_i = 420.9687) = 0$
<i>Ackley function</i>	
$f(\mathbf{p}) = -20 \exp\left(-0.2 \sqrt{\frac{1}{d} \sum_{i=1}^d p_i^2}\right) - \exp\left(\frac{1}{d} \sum_{i=1}^d \cos(2\pi p_i)\right) + 20 + \exp(1)$	
	Free parameters: $i = 1, \dots, d$ Search domain: $p_i \in [-32.768, 32.768]$ Global optimum (min): $f(p_i = 0) = 0$

(continued)

Table 3.6 (continued)

<i>Goldstein-Price function</i>	
$f(p_1, p_2) = \left[1 + (p_1 + p_2 + 1)^2 (19 - 14p_1 + 3p_1^2 - 14p_2 + 6p_1p_2 + 3p_2^2) \right]$ $* \left[30 + (2p_1 - 3p_2)^2 (18 - 31p_1 + 12p_1^2 + 48p_2 - 36p_1p_2 + 27p_2^2) \right]$	
	Free parameters:
	$i = 1, 2$
	Search domain:
	$p_i \in [-2, 2]$
	Global optimum (min):
	$f(0, -1) = 3$

will be found which is not too far away from the best solution possible. If we are close to good proposals, gradient methods will improve the free parameters in a short time and with reasonable effort.

As soon as we doubt that our initial designs are close to the optimal ones, EVO or PSO have the capacity to propose better designs. Nevertheless, the number of function evaluations may be large. Which of the two is preferred for a particular problem must be decided with some preliminary test. Often, the particle swarm shows a faster tendency towards the assumed best values, but some examples indicate that the swarm might have the tendency to stick to local maxima, just as do gradient methods.

Switching to Gradient Optimization if approaching a maximum closely is always an interesting option. But experience of the problem and methodology is required there as well.

In every case, the optimization of large problems consumes time and resources. There is no way to avoid the evaluation of many individual solutions and there is no guarantee that the absolute best solution will be found at all.

3.2 The Curse of Dimensions

Rolf Steinbuch

One of the most problematic properties of optimization tasks with higher numbers of free parameters is “the curse of dimensions”. It appears to be one of the most governing drawbacks and sets the strongest limits on any attempt to accelerate the progress of all optimization strategies when dealing with higher numbers of free

parameters. We may compare it to the search for a needle in a haystack. If the needle has a length of $l_n = 6$ cm and the haystack is one-dimensional with a length on $l_h = 1$ m = 100 cm, it is relatively easy to find the needle. We subdivide the length l_h in intervals $l_{i1} = 5$ cm $< l_h$. Now we check at each end of the intervals if there is a needle traversing the interval border. After 19 checks at maximum, we found the needle. For a 2D haystack covering a square of 1 m² the procedure becomes more expensive. We need a mesh of width $l_{2i} = 4$ cm $< 6/\sqrt{2}$ cm to cover the area of the haystack and have now to check 625 mesh edges. Correspondingly, we need a 3D mesh with a side length of $l_{3i} = 3.333$ cm $< 6/\sqrt{3}$ cm to cover the 3D haystack and need to check 36,000 faces of the cubes defining the mesh. We may continue to higher dimensions even if we fail to imagine higher dimensional haystacks. Evidently, the higher the dimension is, then the larger the effort to find the needle.

Example 3.1 If we return to optimization, we may assume a local optimum in 1D given by one function such as

$$goal(p_1) = \frac{1}{2}(1 + \cos(\pi * p_1))$$

visualized in Fig. 3.4a. In the interval $-1 < p_1 < 1$ the 2D area below the function is

$$V_1 = 1 = 0,5 * V_{0,1}, \quad (3.1)$$

where $V_{0,1} = 2$ the area of the surrounding rectangle. If we step to a 2D problem, the corresponding function becomes

$$goal(p_1, p_2) = \frac{1}{4}(1 + \cos(\pi * p_1))(1 + \cos(\pi * p_2)) \quad (3.2)$$

(Fig. 3.4b) and the volume below the hill is

$$V_2 = 1 = 0,25 * V_{0,2}, \quad (3.3)$$

where $V_{0,2} = 4$ the volume of the surrounding cube.

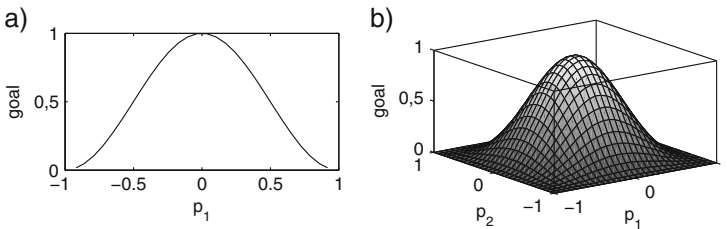


Fig. 3.4 Decreasing volume of hills covered by cos functions. (a) 1D cos function. (b) 2D cos function

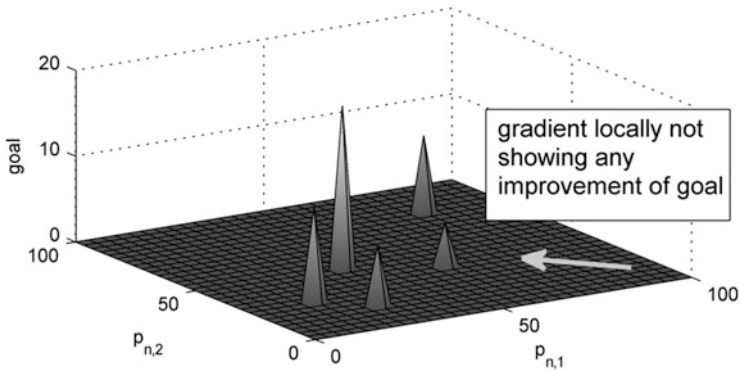


Fig. 3.5 Local optima as steep needles in higher dimensional spaces

We realize that the higher the number of dimensions n , the smaller the ratio

$$\frac{V_n}{V_{0,n}} = \left(\frac{1}{2}\right)^n. \quad (3.4)$$

The probability to find the hill gets smaller and smaller as the number of dimensions increases. The hills of local optima degenerate to needles, as Fig. 3.5 tries to demonstrate. Even worse is the fact that the gradient of the goal on the plane between the needles is close to zero; there is no strong force driving the optimization process in the direction of the needles. So we are bound to do many studies and repeat many searches to come close to promising designs. This problem is often called the curse of dimensions. It limits the maximum achievable velocity to first find a promising region and then converge to a solution. The only work-around is a reduction of the problem's dimension, which reduces the power of the optimization range or the reduction of the search space, which may exclude interesting regions. Therefore we have to live with the curse of dimensions if we are dealing with the optimization of problems with many free parameters.

3.3 Acceleration of Bionic Optimization Processes

Tatiana Popova

Optimization today is a very promising area and is used as a standard method to decrease production costs or the weight of a part or assembly. From the point of mechanical calculation, optimization methods enable designers to choose the best variant of a design with the best allocation of resources, reduction of the cost of materials, energy, and etc. All parts of the optimization procedure are important, from identification of variables and initial algorithm parameters to identification of

the correct fitness function. Optimization as a subject receives serious attention from engineers, scientists, managers and anyone else involved with manufacturing, design, or business. This focus on optimization is driven by competition in quality assurance, cost of production and, finally, in success or failure of businesses. Throughout the past century, optimization has developed into a mature field that includes many specialized branches, such as linear conic optimization, convex optimization, global optimization, discrete optimization, etc. Each of these methods has a sound theoretical foundation and is supported by an extensive collection of sophisticated algorithms and software. With rapidly advancing computer technology, computers are becoming more powerful, and correspondingly, the size and the complexity of the problems being solved using optimization techniques are also increasing.

The requirements for optimization is the possibility of achieving good results within a short processing time. Gradient methods can be sufficient, but the increase of complexity of optimized components often leads gradient methods to wrong proposals at local optima. Gradient methods are sufficient when the task is to find a local optimum. Gradient methods are not applicable for global optima search. The methods stop at one hill of the goal function without investigating the others, which could possibly contain better results. This situation necessitates the investigation and research into new optimization methods that could deal with complicated optimization problems.

Traditional optimization algorithms often depend on the quality of the objective function, but many objective functions are usually highly non-linear, steep, multi-peak, non-differential or even discontinuous, and have many continuous or discrete parameters. Almost all problems need vast amounts of computation. Traditional optimization techniques are incapable of solving these problems. Bionic engineering copies living systems with the intention of applying their principles to the design of engineering systems. In recent years, bionic engineering has been actively developed globally. Much bionic scientific research has been conducted, and new products have been designed and developed. Biomimetic structural optimization methods, for example, aim at the improvement of design and evaluation of load-bearing structures.

The quality of algorithms depends not only on the problem's complexity, but also on the individual adjustment of the parameters of the corresponding optimization methods. Incorrect choices for algorithm parameters could lead to a decrease in search time or even to false results. This shows the importance of collecting the results of an algorithm's parameters variation to understand their influence. The investigation's goal is to find sets of parameters that could provide stable results for the wide problem range.

The efficiency of the algorithms should be proven. Optimization testing functions are used to estimate the quality of algorithms. These functions (e.g. in Table 3.6) were chosen because they represent the common difficulties seen in isolated optimization problems. By comparing and contrasting these functions, a user can make judgments about the strengths and weaknesses of particular algorithms.

Essential for optimization problems is the identification of the point when convergence is reached, so as not to lose time on future unnecessary investigations. This is especially valuable for optimizations with a high number of variables and relatively long fitness function evaluation time.

To find an acceptable optimization calculation time, accelerating strategies must be found. Some are based on the fact that not all parameters have the same impact on the object. Decreasing the number of parameters handled may help to reach areas with good objective values. Other strategies use statistical predictions to estimate the best values of the objective function in early stages. From those estimations, a decision on whether the reached objective value is promising could be derived. Unfortunately these predictions, again, need many runs to yield reliable data. Nevertheless, accelerated optimization may be a tool to find reliable results at acceptable time and cost.

3.3.1 Selecting Efficient Optimization Settings

Tatiana Popova

We have been discussing different ways to perform Bionic Optimizations. However, we are missing some guidelines for running a real task, i.e. how many parents, kids, individuals to use, how to define crossing and mutation, which weighting factors will perform well, and which will be less effective.

As there are unlimited possible problems and different strategies tend to perform differently in different applications, general rules are hard to propose. Nevertheless, here we try to give some hints for starting a process. Motivated users will soon learn how to accelerate their studies. They will optimize the optimization process, also known as meta-optimization. We discuss specific strategies for each of the three most important Bionic Optimization methods respectively.

Evolutionary Optimization

The optimization data we may access are:

- Number of parents
- Number of kids
- Survival of parents
- Way of crossing
- Mutation radius
- Number of generations

As a general rule, we realize, the larger the number of individuals, the greater the probability to find promising results. But the time required for analysis limits these numbers, so we need some starting data. From our previous experiences, we

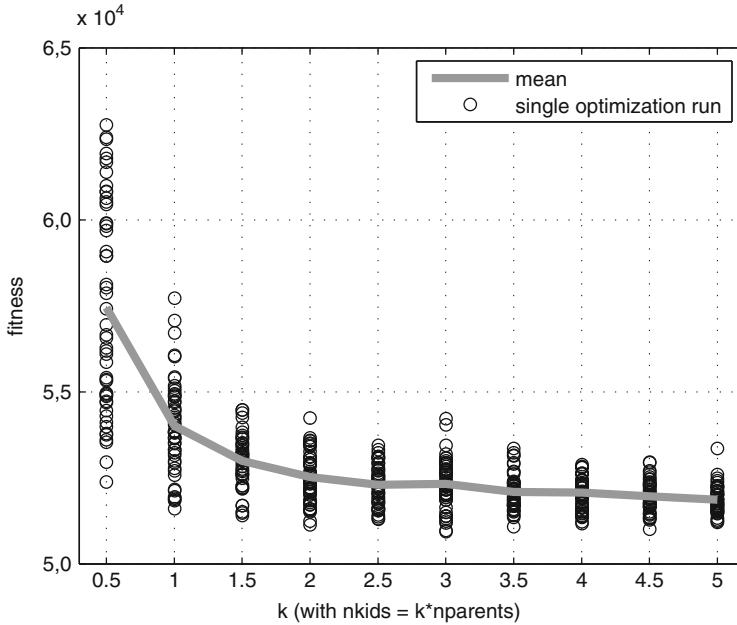


Fig. 3.6 Minimization of goal with multiple of number of kids related to # of parents. Taking # of kids = $2 \times$ # of parents seems a good guess

propose to use a *number of parents* in the range of 0.5–2 times the number of free parameters. The higher the number of parents, the better the performance will be. From Fig. 3.6 we learn that the *number of kids* should be about double the number of parents, while less is not efficient, higher values seem also not to be as efficient because the computational effort increases with the rising number of individuals, without achieving much better results. In this study we used the 10 rods frame problem (Fig. 3.1b) and did multiple optimization runs with different number of kids, related to the number of parents, and fixed values for the other optimization settings ($nparents = 15$, $ngen = 40$, $mutrad = 25\%$, parents survive). For each variation, we repeated in a loop of 50 runs.

Survival of parents is generally recommended, as it removes the danger of deleting some good proposed designs. It might be a good idea to limit the number of generations an individual may survive, e.g., to a maximum of five generations, but this does not increase the performance of the process in a very powerful way.

Crossing might be accomplished by using the ideas outlined in Sect. 2.1. From our experience, averaging parents' properties is quite a good idea.

Mutation might be done to each parameter individually. Using large mutation radii leads to a pure random search, where the properties of the parents no longer influence those of the kids. Too small mutation radii correspond to a local search around the parents' values. If a local search is intended, it is better to use a gradient method. We generally recommend starting with a mutation radius in the range of 15–25 % of a parameter's range. After a certain number of generations, these

mutation radii might be reduced to values of 5–10 %. A switch to a gradient method could be a good idea as well.

The *number of generations* should not be less than the number of free parameters. To avoid unnecessary repetitions, it is recommended to monitor the process and to kill it if no further progress is observed. We realize that the method does not often show any further improvement, so it is appropriate to stop the run. If restarting the process at any generation is possible, it might save time and computing power.

Fern Optimization

Fern Optimization is, from our experience, recommended for problems with small numbers of free parameters. Figure 3.2 using the models of frames introduced in Sect. 3.1 indicates that, for more than 50 parameters, PSO and EVO are essentially more efficient. Using Fern Optimization, we are concerned with:

- Numbers of parents
- Numbers of kids per parent
- Mutation radius
- Number of generations
- Killing underperforming families (cf. Fig. 2.6)

Again, the *number of parents* should not be essentially smaller than the number of free parameters, as this number supports the coverage of the parameter space.

The *number of kids* should be about 3–5 per parent; more is better, but increases computing time. These kids are generated by the mutation of one parent’s properties. To reiterate, a not too large *mutation radius* of 15–25 % is recommended at the onset of the study. Keep in mind: as soon as good designs have been approached, the mutation radius can be decreased or the optimization could switch to a gradient method.

Additionally, it is important that a sufficient *number of generations* should be calculated. The process can be terminated if it approaches saturated levels, which can be monitored.

The decision to remove a family, the offspring of one initial parent from the process, is possible, but should not start too early as indicated by Fig. 2.6.

Particle Swarm Optimization

PSO generally tends to give very promising results as long as some basic criteria are met. The definition of the optimization parameters covers:

- Number of individuals
- Number of time steps
- Initial velocities
- Weighting of contributions to velocity update

The *number of individuals* seems to be the most important input for PSO. At least, the number of free parameters should be covered. Again, the larger the number, the better the results, but the larger the computational effort as well. On the other hand, we often observe the particle swarm sticking to local optima. This might be dependent on the number of individuals, so again, larger numbers are to be preferred. The *number of time steps* must not be too large, as often 10–20 generations yield interesting designs, and a restart option allows continuing processes that have not yet converged. The *initial velocities* should be in the range of 10–25 % of the parameter range for each of the optimization parameters.

Many discussions deal with the definition of the *weighting factors* in the equation (cf. Eq. (2.1) and the associated declaration)

$$\mathbf{v}_j(t+1) = c_1 \mathbf{v}_j(t) + c_2 \mathbf{r}_1 \circ (\mathbf{p}_{Pb,j} - \mathbf{p}_j(t)) + c_3 \mathbf{r}_2 \circ (\mathbf{p}_{Gb} - \mathbf{p}_j(t)).$$

Generally, a large value of c_1 (inertia) yields a broad search of the parameter space, but suppresses convergence to the best values. Large values of c_3 (social) accelerate the convergence, but might stick to early found local optima. The weighting of c_2 (cognitive) has a smaller effect on the performance. As a coarse rule we often use

- $0.4 < c_1 < 1$
- $0 < c_2 < 0.5$
- $0.5 < c_3 < 2$.

These proposals might help to get started more effectively using Bionic Optimization methods, but are not guaranteed to be the best ones for every problem.

3.3.2 Parallelization and Hardware Acceleration

Simon Gekeler

Using Bionic Optimization procedures requires computing and evaluating many different design variants. It is a time consuming process, but offers the chance to find the global optimum or at least a good solution, even in a large and complex design space with many local optima. If a single design evaluation, e.g., a non-linear FEM simulation, takes an excessive amount of computation time, it is no longer efficient to integrate a population-based optimization method, like PSO or EVO, into the usual design development process. The number of evaluated variants is limited by schedules and processing capabilities. We need strategies for applying these methods in an acceptable period of time.

To reduce process times, first check the model being optimized and its computation time, before trying to accelerate the optimization algorithm itself. In FEM simulations, for example, we should verify if the model is adequately simplified to be calculated quickly with sufficient result accuracy. Additionally, it is worthwhile

to allot some time for an accurate parameterization of the design. Especially in case of geometry parameters, good parameterization is important for the prevention of inconsistent situations during automated variance of the parameters in the optimization process, which leads to aborted processes and needless waste of development time.

To perform a quick and efficient optimization run, which means to achieve satisfying results with fewer design computations, choosing the right strategy is critical, including the choice of optimization method and algorithm settings (see Sect. 3.1.1) for the specific type of problem. To reach that goal, there are different ways to proceed, e.g., run the optimization in several stages, maybe using different methods, limited parameter ranges, or just using the most significant parameters for an exploration phase first, and following with an detailed phase in a localized area but increased parameter space. Furthermore, hybrid optimization methods, such as PSO with an automated switch to the Gradient method in the final stage (Plevris and Papadrakakis 2011) or meta models (cf. Sect. 2.7), can greatly reduce optimization time. Preliminary investigations, such as sensitivity analysis, can help to reduce computation effort with more information about the problem, choosing the important parameters and neglecting the insignificant ones. For highly complex problems, where the computation time is expected to be large to find the global optimum, the most efficient method could be finding an already acceptable local optimum by using the relatively fast gradient method with a good starting position.

If the optimization job is well prepared and ready for efficient execution, we can possibly gain an additional and drastic reduction of processing time by accelerating the computation time with using faster computer hardware or with using further computer resources.

Parallel Jobs for Speeding Up Optimization Processes

In Bionic Optimization procedures, such as EVO or PSO, in one generation or iteration we compute many different design variants, before the design's results are evaluated and the next loop starts with the computation of newly generated designs. Thus, in this section of the optimization algorithm, each job can be processed independently of each other. This allows for the possibility of running the jobs in parallel, enabling an enormous acceleration of these optimization procedures. The ability of optimization algorithms for parallelization depends on the ratio of the sequential workflow and tasks which can be done in parallel. For example, with EVO we can increase the number of parallel tasks by increasing the number of kids to be calculated in one generation. However, we rely on adequate optimization settings to improve calculation times.

When processing Bionic Optimization tasks in parallel, we are referring to parallel computation of design variants on different workstations connected by a Local Area Network (LAN). An architecture for the distribution of jobs to be run in parallel is depicted in Fig. 3.7. The optimization algorithm is running on one workstation (master), which distributes the tasks to be run in parallel to different

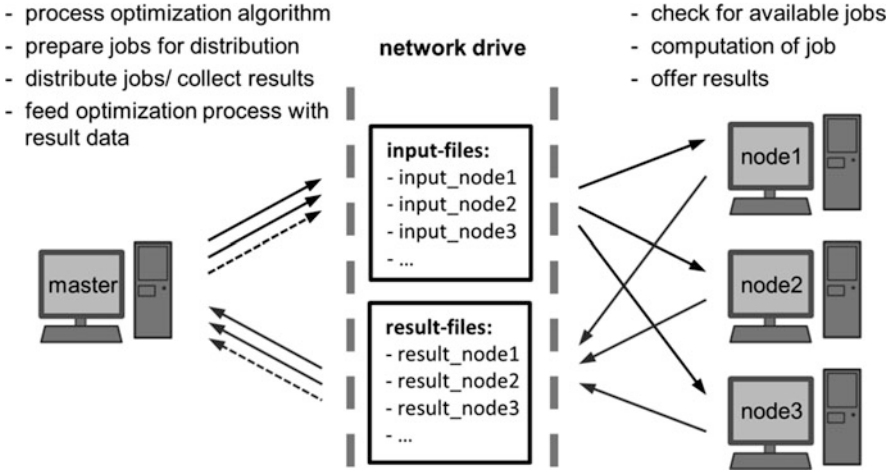


Fig. 3.7 Parallelization of optimization processes in a computer cluster

workstations (nodes) in the cluster. Afterward, the results of computed jobs are collected by the master computer and inserted into the optimization algorithm for further processing and generating new designs for the next optimization loop. In this example the exchange of input and output data is similar to outer loop optimization (cf. Sect. 4.1.2) by generating and reading files, which are stored on a network drive, accessible to all computers.

For the management of this distributed computing, appropriate software or generated code is required, which must fulfill the following functions:

- Write/modify input-files for the particular type of solver on node computers
- Identify status of nodes in cluster: busy, standby, results available, etc.
- Distribute and launch individual jobs in the cluster
- If necessary, copy and offer additional files required for solving jobs on nodes
- Collect available results and prepare for further processing in optimization algorithm
- Perform error management: error in result, no response of node (time out), handle loss of node in cluster, etc.

It should be clear that, with this organization (distributing jobs, writing/reading of data for the exchange, and waiting time) in addition to the optimization algorithm runtime and the computation of the designs, parallelization includes an extra effort. Compared to the time saved overall, this time cost, the so-called overhead, should be small. It is obvious that parallelization is more efficient as the computation time for one individual design increases. On the other hand, if individual designs can be computed quickly, it is possible that the overhead causes even slower process times when using parallelization than in a common sequential procedure on only one workstation. In Fig. 3.8 we can see the speed-up for problems with different computation times when using parallelization. For each problem (small, medium,

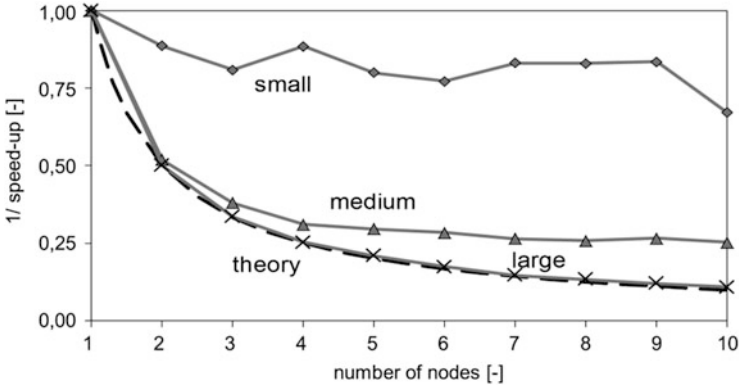


Fig. 3.8 Parallelization speed-up for problems with different computation time

large), we compute 50 variants with a various number of nodes. Furthermore, the theoretical maximum achievable speed-up is indicated, without the time cost for the distribution of jobs.

Efficient computer clusters have all nodes at full capacity and without unnecessary waiting times. For example, if there are 10 parallel tasks to be processed, it would be absurd to use nine nodes in the cluster with eight of them waiting 50 % of the time. With 5 nodes we can reach the same decrease of process time, but using 10 nodes would be ideal. We need to coordinate the number of nodes with the number of jobs we can compute in parallel. Furthermore, the performance of all nodes in the cluster should be comparable to prevent bottlenecks due to diverse computation times of design evaluation.

The relative speed-up S_p of parallel jobs can be calculated by

$$S_p = \frac{T_1}{T_p},$$

where T_1 is the time needed for the sequential process on one computer and T_p is the time when using p computers in a cluster to parallelize jobs.

How many nodes we use in a cluster depends on not only the number of available nodes, but also on the specific optimization task. When using commercial solvers, the number of available software licenses can also limit the number of nodes. Currently, most FEM software providers offer particular license packages for parallel computing.

Benefit of Hardware Raising

Another way to reduce computation time and accelerate the optimization process is the use of well-equipped workstations with appropriate hardware. Here we focus especially on the performance of the Central Processing Unit (CPU), the Random-

Access Memory (RAM), the hard disk, and (growing more and more important and very promising) the Graphics Processing Unit (GPU), which can be used in addition for computation, often in FEM and CFD (Ohlhorst 2012).

For quick computation, the most important component of a workstation is a powerful CPU, as it is responsible for the computer's overall performance. After architecture, the clock rate, which defines the speed of data processing, of the CPU is the most important. Further increases to the performance of a single CPU are limited by economic and mechanical concerns. Currently, the use of multi-core processors with several processing units is standard. Current FEM software is able to use multiple cores in parallel.

Also important, is sufficient RAM. The more RAM the better, as long as the FEM simulation consumes the presented memory and the operating system allocates properly. There are different types of RAM with different RAM speeds. If the system, especially the motherboard, can be upgraded to faster RAM, this will enhance performance, too.

The massive amount of data in an FEM simulation must be handled by large hard drives. To prevent a bottleneck when using high performance CPUs, it is important that the storage component offers sufficient read and write capability. Today Solid-State-Drives (SSD) provide excellent properties for fast processes.

The newest, very efficient strategy for hardware acceleration is the inclusion of high performance GPUs in general-purpose computing. CPUs are developed for universal use and sequential processing. In contrast, the architecture of GPUs is designed for massive parallel processing. In FEM, we can release the equation solving part, which takes about 70 % of the total evaluation time, to the highly efficient GPUs (Güttler 2014). To take advantage of increased process speed by using GPUs, there are special license packages offered by commercial software suppliers.

In general, it is important when buying or upgrading a workstation to verify all components of the system to avoid bottlenecks. All parts must interact well to guarantee a high efficiency and enable fast computation processes.

Conclusions

With the emergence of commercial software for optimization, sensitivity studies, or the evaluation of robustness and reliability, the information of many design variants can be considered and handled. Often computation time limits such detailed investigations, especially when we have complex multi-physics simulations. Currently, parallelization and hardware acceleration with GPUs is a common tool to accelerate such time consuming studies.

But when applying the optimization methods mentioned in this book, the correct strategy, previous experience in optimization, and good preparation are also important for a quick route to the optimal design. Optimization is not simply pressing the start button and waiting for a result. We need to study and understand the problem, find additional information about its parameters and reuse this for the next

optimization steps. Well thought-out action is more significant than computing a lot of different variants. Time saving also entails preventing failures or wrong definitions that lead to meaningless results in optimization. However, if an optimization run is canceled, it is beneficial to have access to the results obtained in the partial run to identify and resolve conflicts for further studies.

References

- Berke, L., Patnaik, S. N., & Murthy, P. L. N. (1993). Optimum design of aerospace structural components using neural networks. *Computers and Structures*, 48, 1001–1010.
- Coelho, L. D. S., & Mariani, V. C. (2006). Particle swarm optimization with Quasi-Newton local search for solving economic dispatch problem. *IEEE International Conference on Systems, Man and Cybernetics*, 4, 3109–3113.
- Güttler, H. (2014). *TFLP performance for ANSYS mechanical 2.0*. Nürnberg: ANSYS Conference 32. CADFEM Users Meeting.
- Lagaros, N. D., & Papadrakakis, M. (2004). Learning improvement of neural networks used in structural optimization. *Advances in Engineering Software*, 35, 9–25.
- Ohlhorst, F. J. (2012, December). Optimize workstations for faster simulations. In *Desktop Engineering – Technology for Design Engineering* (pp. 68–70). Retrieved June 11, 2015, from http://www.engineerquiz.com/issues/level5_desktopengineering_201212.pdf
- Plevris, V., & Papadrakakis, M. (2011). A hybrid particle swarm—gradient algorithm for global structural optimization. *Computer-Aided Civil and Infrastructure Engineering*, 26, 48–68.
- Rechenberg, I. (1994). *Evolutionsstrategie '94*. Stuttgart: Frommann-Holzboog.
- Steinbuch, R. (2010). Successful application of evolutionary algorithms in engineering design. *Journal of Bionic Engineering*, 7(Suppl), 199–211.
- Surjanovic, S., & Bingham, D. (2015). Virtual library of simulation experiments: Test functions and datasets. Retrieved May 12, 2015, from <http://www.sfu.ca/~ssurjano>
- Widmann, Ch. (2012). *Strukturoptimierung mit Neuronalen Netzen*. Master thesis, Reutlingen University.