Parameter Meta-Optimization of Metaheuristic Optimization Algorithms

Christoph Neumüller, Stefan Wagner, Gabriel Kronberger, Michael Affenzeller

Heuristic and Evolutionary Algorithms Laboratory
School of Informatics, Communications and Media
Upper Austria University of Applied Sciences
Softwarepark 11, A-4232 Hagenberg, Austria
{cneumuel,swagner,gkronber,maffenze}@heuristiclab.com

Abstract. The quality of a heuristic optimization algorithm is strongly dependent on its parameter values. Finding the optimal parameter values is a laborious task which requires expertise and knowledge about the algorithm, its parameters and the problem. This paper describes, how the optimization of parameters can be automated by using another optimization algorithm on a meta-level. To demonstrate this, a meta-optimization problem which is algorithm independent and allows any kind of algorithm on the meta- and base-level is implemented for the open source optimization environment HeuristicLab. Experimental results of the optimization of a genetic algorithm for different sets of base-level problems with different complexities are shown.

1 Introduction

Metaheuristic optimization algorithms have proven to be very good problem solvers for real world optimization problems. Since the beginning of the 1990s lots of variations of metaheuristic optimization algorithms have been developed, yet none of them has been identified as the jack of all trades in optimization. According to the *no free lunch* theorem (NFL) in heuristic search [6, 18] this is in fact impossible, as there exists no algorithm which performs better than all other algorithms on all problems. Furthermore, even the comparison of heuristic algorithms for a certain problem is difficult, because such a comparison would only be valid, if each algorithm is optimally parameterized.

2 Parameter Optimization

Finding good parameter values for an optimization algorithm is not a trivial task. A high number of parameters with dependencies on each another and mutual influences makes finding good parameter values a complex problem.

The work described in this paper was done within the Josef Ressel Centre for Heuristic Optimization *Heureka!* (http://heureka.heuristiclab.com/) sponsored by the Austrian Research Promotion Agency (FFG).

The most common approach to find optimal parameter settings is manual experimentation. However, this approach requires human expertise and interaction which are both rare and expensive resources. *Deterministic* parameter control in evolutionary algorithms uses deterministic and usually time dependent rules to adapt parameters [5]. *Adaptive* parameter control in evolutionary algorithms dynamically adapts parameter values based on feedback from the search. A well-known representative of adaptive optimization algorithms are evolution strategies [13].

Another method of parameter optimization is called *parameter meta-optimization* (PMO) which employs an optimization algorithm again (meta-level) to optimize the parameters of another algorithm (base-level). The conceptual structure of a meta-optimization approach is shown in Figure 1. Early research was done by Mercer and Sampson [9] who used a genetic algorithm (GA) on the meta-level called Meta-GA. Later Grefenstette [7] used a GA as meta-optimization algorithm to optimize several test function problems. In this work binary encoding was used for solution encoding. The biggest problem in those early works was the lack of computational power, so only small problems could be solved.

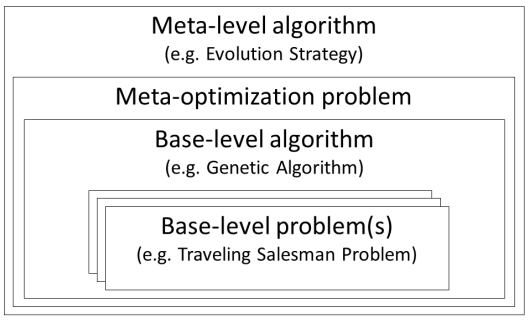

Fig. 1. Meta-optimization concept

The algorithm on the base-level needs to be executed for evaluating the fitness of a single individual. To explore a realistic search space, this needs to be repeated thousands of times which makes PMO very runtime-consuming. This is the reason why these early researches used tiny problem dimensions and small populations. Since then, powerful hardware, cloud- and grid-computing approaches enabled research in the area of PMO algorithms with realistic problem dimensions. More recently Pedersen [12] developed a specialized algorithm (Local Unimodal Sampling) for meta-optimization. In this work only the average solution quality of the base-level problems was used in the fitness evaluation.

3 Parameter Meta-Optimization for HeuristicLab

The goal of this paper was to implement parameter meta-optimization (PMO) for HeuristicLab and to overcome several of the shortcomings of approaches in the past such as:

- fitness evaluation is only based on solution quality
- single base-level problems with tiny dimensions
- no parallelization
- binary encoding
- specialized meta-optimization algorithms which are not easily exchangeable

3.1 HeuristicLab

HeuristicLab[1] is an open source environment for heuristic and evolutionary algorithms [16]. Despite a sophisticated graphical user interface, it supports a number of well known optimization algorithms and problems. A significant benefit of HeuristicLab is its clean separation of algorithms and problems. In order to implement a *meta-optimization problem* for HeuristicLab, it is necessary to define the problem-encoding (solution representation), the solution evaluator, as well as mutation and crossover operators. Any evolutionary algorithm in HeuristicLab can then be used to solve the meta-optimization problem.

HeuristicLab features a parallelization concept which allows to parallelize the solution evaluation of population based algorithms. Either threading on a local machine or distributed computing can be used. Solution evaluation in meta-optimization is highly runtime intense, but all evaluations are independent from each other, so it is perfectly suited for parallelization. For the experiments in this paper the distributed computing infrastructure of HeuristicLab (*HeuristicLab Hive*) with up to 108 CPUs (6 high performance blade server nodes and up to 20 desktop computers) was used.

3.2 Solution Encoding

Many meta-optimization approaches used a binary [7] or real-valued encoding to represent parameter values. In HeuristicLab the parameters of an algorithm are represented as a composite tree structure. Nodes can either be leafes which contain values (of type integer, double or bool), or contain child parameters. An example would be a tournament selection operator which itself has a group size as parameter.

In PMO for HeuristicLab, the solution encoding fulfills two purposes. First, it is a configuration of the parameter tree which should be optimized. It is possible to configure if and how a parameter should be optimized. Different parameter types have different configuration options. Integer and double parameters for example can define search ranges, whereas parameterizable parameters can

[1] HeuristicLab can be downloaded from `http://dev.heuristiclab.com`.

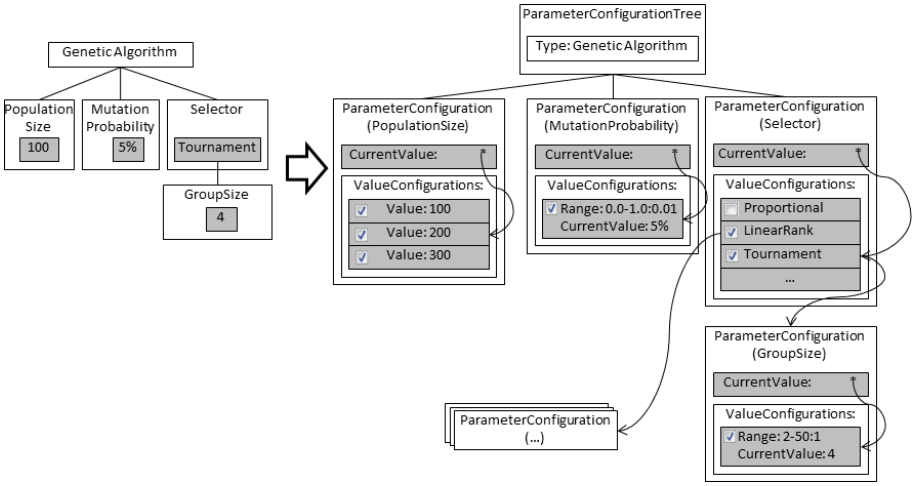

Fig. 2. The tree on the left represents the parameters of a GA (simplified), the right side shows an example of a parameter configuration tree which defines how each parameter should be optimized.

configure which child-parameters should be optimized. For operators, a list of candidates can be selected. Second, the solution encoding stores the current value of each parameter. Figure 2 illustrates the solution encoding.

3.3 Fitness Function

The goal of parameter optimization is to find parameter values for an algorithm which result in optimal

- **Solution Quality** (q): The average achieved quality of n algorithm runs.
- **Robustness** (r): The standard deviation of the quality of n algorithm runs.
- **Effort** (e): The average number of evaluated solutions of n algorihtm runs.

Since the result of an optimization algorithm underlies a stochastical distribution, it is necessary to repeat the evaluation of a parameterization n times. For multiple problem instances, the evaluation has to be repeated for each instance.

Different problem instances might yield quality values in different dimensions. An example would be to optimize a genetic algorithm for three Griewank [8] test functions in the dimensions 5, 50 and 500. A given parameterization might result in the quality values 0.17, 4.64 and 170.84. Using a simple arithmetic mean for the fitness function would overweight the third problem. It is the goal to find parameter values which solve each problem equally well. To tackle this issue normalization has to be applied on all results (q, r, e). The reference values for normalization are the best values from the first generation (rq, rr, re). Furthermore each objective needs to be weighted (wq, wr, we). The quality of a solution

candidate for minimization problems is thereby defined as:

$$((q/rq * wq) + (r/rr * wr) + (e/re * we))/(wq + wr + we)$$

3.4 Operators

For evolutionary algorithms in HeuristicLab only mutation and crossover operators needed to be implemented. To mutate a parameter tree, one parameter is chosen randomly. Depending on the datatype of the parameter a different manipulation is applied:

- **Boolean**: Randomly choose true or false.
- **Integer/Double**: A new value is sampled from a normal distribution, where μ is the current value and σ is 10% of the search range.
- **Operator**: One of the available operators is randomly chosen.

For a crossover of two parameter trees, every parameter configuration node is crossed with the corresponding parameter configuration node of the second tree until basic datatypes are reached:

- **Boolean**: Randomly choose one of the parent values.
- **Integer/Double**: A new value is sampled from a normal distribution, where μ is the value of the parent with the better quality and σ is the distance to the worse parent divided by 3. This crossover operator ensures that the search process is guided towards the better parent, yet diversity is kept through the randomness of the normal distribution.
- **Operator**: Randomly choose one of the parent values.

4 Experimental Results

This section is dedicated to the experimental results of a GA as meta-level algorithm which optimizes a GA as base-level algorithm. The parameters of the meta-level algorithm are: *maximum generations=100, elites=1, mutation probability=15%, population size=100, selection operator=proportional, repetitions=6, quality weight=1.0, robustness weight=0.0, effort weight=0.0.* The experiments in this paper are only focus on optimal solution quality so the weights for robustness and effort are set to zero.

The optimization was tested using different base-level problems to show how the problem influences the choice of parameters. The parameters of the base-level genetic algorithm and the search ranges are shown in Table 1. The *BestSelector* considers a single double quality value and selects the best, likewise the *WorstSelector* selects the worst.

As base-level problems five different instances of the Griewank function with different problem dimensions were used: *f1=[griewank(500)], f2=[griewank(1000)], f3=[griewank(1500)], f4=[griewank(2000)], f5=[griewank(500), griewank(1000), griewank(1500), griewank(2000)].* The goal of this experiment was to show, if

Maximum generations	1000 (fixed)
Population size	100 (fixed)
Elites	0-100
Mutation probability	0-100%
Crossover operator	Average [1], BlendAlphaBeta [15], BlendAlpha [15], Discrete [1], Heuristic [19], Local [4], RandomConvex [4], SimulatedBinary [2], SinglePoint [11], UniformSomePositionsArithmetic [11]
Mutation operator	BreederManipulator [10], MichalewiczNonUniformAllPositions [11] (IterationDependency: 2-10), MichalewiczNonUniformOnePosition [11] (IterationDependency: 2-10), SelfAdaptiveNormalAllPositions [1], PolynomialAllPosition [3], PolynomialOnePosition [3], UniformOnePosition [11]
Selection operator	Best, GenderSpecific [17], LinearRank, Proportional, Random, Tournament (GroupSize: 2-100), Worst

Table 1. Parameter configuration of the base-level genetic algorithm.

different problem dimensions would cause the algorithm to be parameterized differently.

Table 2 shows the results of five PMO runs. Interestingly the parameter sets $p(f2)$ to $p(f5)$ seem to be very similar. Since Best- and WorstSelector behave almost the same, the only significant outlier is the 46% value in $p(f3)$. In this case, the optimization process could have done further improvement as the quality value is worse than the one for $p(f4)$. To test if each parameterization is optimal for the test function, we cross-tested the all results. Figure 3 shows that $p(f1)$ performs significantly better on $f1$ than on the other problems, while $p(f2)$ to $p(f5)$ perform almost equally well on $f2$ to $f5$, but not so well on $f1$. Each of the test runs consumed an average CPU-time of 32 days. All runs were executed with *HeuristicLab Hive* at the same time and finished after 3 days.

	Elites	Crossover operator	Mutation operator	Mut. prob.	Selection operator	Average qualities
$p(f1)$	1	BlendAlpha	SelfAdaptiveNormalAllPos	27%	Tournament (GroupSize: 5)	1.3952
$p(f2)$	1	Average	MichalewiczNonUniformAllPos (IterationDependency: 10)	30%	Worst	12.5546
$p(f3)$	1	Average	MichalewiczNonUniformAllPos (IterationDependency: 2)	46%	Best	25.3463
$p(f4)$	1	Average	MichalewiczNonUniformAllPos (IterationDependency: 2)	25%	Best	20.18
$p(f5)$	1	Average	MichalewiczNonUniformAllPos (IterationDependency: 10)	25%	Worst	5.86, 11.79, 17.51, 23.58

Table 2. Best genetic algorithm parameter values found for different problem sets.

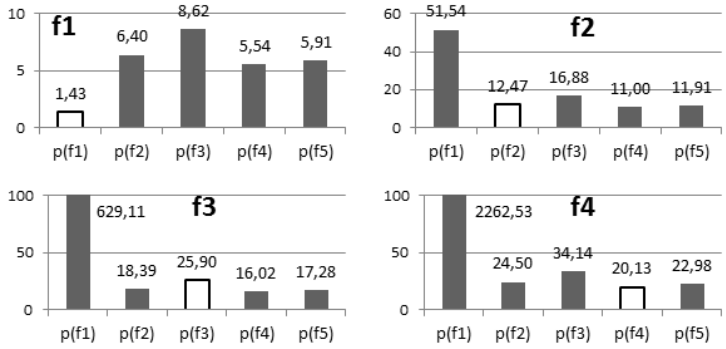

Fig. 3. Qualities achieved by different parameterizations for the problems $f1$ to $f4$

5 Conclusion and Outlook

This paper describes a meta-optimization approach to optimize parameters of metaheuristic optimization algorithms. In contrast to meta-optimization approaches in the past, the meta-level algorithm is abstracted from the problem in such a way that it is possible to use any existing evolutionary algorithm implemented in HeuristicLab on the meta-level. It is further possible to optimize any algorithm on the base-level as well as to optimize for multiple base-level problems. Experiments with multiple sets of test function problems have shown that different instances of the same problem may require completely different parameter values for the optimization algorithm. The massive runtime challenges were mastered by using the distributed computation infrastructure *HeuristicLab Hive*.

Meta-optimization has shown to be able to find interesting parameter values which are hard to find manually, yet it requires massive computing power. Two ways to optimize runtime would be to implement *Racing* and *Sharpening* [14]. When *Racing* is used, promising solution candidates are evaluated more often than bad solution candidates. With *Sharpening*, the number of repetitions is increased dependending on the current generation. In that way the solution evaluation is faster in the beginning of the optimization process and gets more and more precise towards the end.

References

1. Beyer, H.G., Schwefel, H.P.: Evolution strategies - A comprehensive introduction. Natural Computing 1(1), 3–52 (2002)
2. Deb, K., Agrawal, R.B.: Simulated binary crossover for continuous search space. Complex Systems 9, 115–148 (1995)
3. Deb, K., Goyal, M.: A combined genetic adaptive search (geneas) for engineering design. Computer Science and Informatics 26, 30–45 (1996)

4. Dumitrescu, D., Lazzerini, B., Jain, L.C., Dumitrescu, A.: Evolutionary Computation. CRC Press (2000)
5. Eiben, A.E., Michalewicz, Z., Schoenauer, M., Smith, J.E.: Parameter control in evolutionary algorithms. IEEE Transactions on Evolutionary Computation (1999)
6. English, T.M.: Evaluation of evolutionary and genetic optimizers: No free lunch. In: Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming. pp. 163–169. MIT Press (1996)
7. Grefenstette, J.: Optimization of control parameters for genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics 16(1), 122–128 (1986)
8. Griewank, A.O.: Generalized descent for global optimization. Journal of Optimization Theory and Applications 34, 11–39 (1981)
9. Mercer, R., Sampson, J.: Adaptive search using a reproductive metaplan. Kybernetes 7(3), 215–228 (1978)
10. Mühlenbein, H., Schlierkamp-Voosen, D.: Predictive models for the breeder genetic algorithm i. continuous parameter optimization. Evolutionary Computation 1(1), 25–49 (1993)
11. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer, 3rd edn. (1999)
12. Pedersen, E.M.H.: Tuning & Simplifying Heuristical Optimization. Ph.D. thesis, University of Southampton (2010)
13. Schwefel, H.P.P.: Evolution and Optimum Seeking: The Sixth Generation. John Wiley & Sons, Inc. (1993)
14. Smit, S.K., Eiben, A.E.: Comparing parameter tuning methods for evolutionary algorithms. Proceedings of the Eleventh conference on Congress on Evolutionary Computation pp. 399–406 (2009)
15. Takahashi, M., Kita, H.: A crossover operator using independent component analysis for real-coded genetic algorithms. In: Proceedings of the 2001 Congress on Evolutionary Computation. pp. 643–649 (2001)
16. Wagner, S.: Heuristic Optimization Software Systems - Modeling of Heuristic Optimization Algorithms in the HeuristicLab Software Environment. Ph.D. thesis, Johannes Kepler University, Linz, Austria (2009)
17. Wagner, S., Affenzeller, M.: SexualGA: Gender-specific selection for genetic algorithms. In: Proceedings of the 9th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI) 2005. vol. 4, pp. 76–81. International Institute of Informatics and Systemics (2005)
18. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1(1), 67–82 (1997)
19. Wright, A.H.: Genetic algorithms for real parameter optimization. In: Foundations of Genetic Algorithms. pp. 205–218. Morgan Kaufmann (1991)