SIMULATION OPTIMIZATION WITH HEURISTICLAB

Andreas Beham[(a)], Michael Affenzeller[(b)], Stefan Wagner[(c)], Gabriel K. Kronberger[(d)]

[(a)(b)(c)(d)]Upper Austria University of Applied Sciences, Campus Hagenberg
School of Informatics, Communication and Media
Heuristic and Evolutionary Algorithms Laboratory
Softwarepark 11, A-4232 Hagenberg, Austria

[(a)]andreas.beham@heuristiclab.com, [(b)]michael.affenzeller@heuristiclab.com
[(c)]stefan.wagner@heuristiclab.com, [(d)]gabriel.kronberger@heuristiclab.com

ABSTRACT
Simulation optimization today is an important branch in the field of heuristic optimization problems. Several simulators include built-in optimization and several companies have emerged that offer optimization strategies for different simulators. Often the optimization strategy is a secret and only sparse information is known about its inner workings. In this paper we want to demonstrate how the general and open optimization environment HeuristicLab in its latest version can be used to optimize simulation models.

Keywords: simulation-based optimization, evolutionary algorithms, metaoptimization

1. INTRODUCTION

In Ólafsson and Kim (2002) simulation optimization is defined as "the process of finding the best values of some decision variables for a system where the performance is evaluated based on the output of a simulation model of this system". From the point of view of optimization a parameter vector is to be optimized where the components may stem from different domains such as integers, real or binary values or even items from an enumeration, strings or any array in general. Additionally there is usually a feasible region that restricts the possibilities of the parameter vector. To find the optimum to those models, there are exact algorithms, heuristics and especially metaheuristics in the toolbox of an optimization engineer: Genetic Algorithms, Tabu Search, Ant Colony Optimization, Simulated Annealing, Particle Swarm Optimization, Variable Neighborhood Search and Scatter Search to name just a few.

HeuristicLab (Wagner and Affenzeller 2005, Wagner et al. 2007) is a framework that allows users to build upon these optimization strategies or use predefined strategies that are adopted from published works. It was designed such that parameterization as well as customization of any strategy can be done via a graphical user interface (GUI). So the user does not work on source code files, but creates and modifies what is called a "workbench" file. This contains the structure and the parameters of the designed optimization strategy and can be executed in the HeuristicLab optimization environment. After the execution has terminated or was aborted the results or respectively intermediate results are saved along the workbench. So in addition to the structure and parameters, the results can also be saved in a single document, reopened at any time and examined.

2. DESCRIPTION OF THE FRAMEWORK

The user interface to an evolution strategy (ES) is shown in Figure 1. The methods defined by this interface are common to many optimization strategies and reusing them is one idea of the optimization environment.

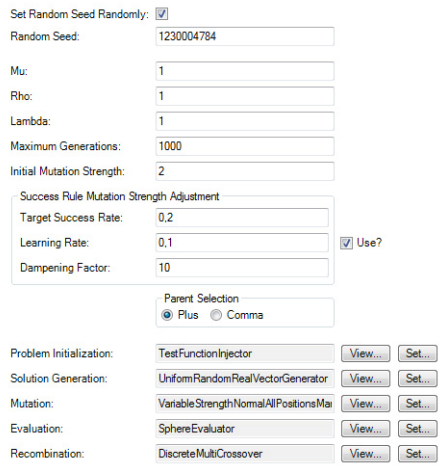

Figure 1: Interface to configure an evolution strategy

To apply simulation-based optimization HeuristicLab needs to know about the parameter vector and any constraints that are imposed on the parameters. This is called injecting the problem. An operator provides a user interface for this task with which the parameter vector is defined and a number of constraints which are specified directly on the parameters or on the vector itself when e.g. comparisons between two parameters need to satisfy a certain criterion.

Additionally for every parameter an initialization as well as manipulation operator is defined which will perform the respective tasks during the optimization. It is also possible to add a default operator to a certain parameter which will not perform any manipulation and it is even possible to manually set a certain parameter to some value. This makes it easy to setup a number of variables for optimization, but optimize only a subset of them leaving other variables fixed. A simulation expert then is able to try to optimize only certain aspects of the simulation model while retaining the possibility to quickly include more parameters. This flexibility is likely welcomed by those who already have some ideas about the optimum of a model which they want to explore more thoroughly. It is also possible to let the optimizer use all parameters and get a general idea of achievable quality.

After the parameter vector is defined it is necessary to specify the communication between optimizer and simulator. Azadivar (1999) already mentions that interfacing simulation and optimization is not always an easy task. The generic concepts that HeuristicLab provides for designing optimization strategies also influenced the decision to build on a generic and customizable interface to communicate with external applications. The backbone to this interface is the design of the communication protocol. This protocol is described as a state-machine, a powerful tool to model algorithmic behavior and the communication between two peers. In the protocol editor users define states, which objects should be communicated in which states, state transitions and their condition. Finally the protocol is woven into the optimization strategy in the form of operators which can be put into any place during the execution of the optimization algorithm. See Figure 2 for an example interface to create the protocol.

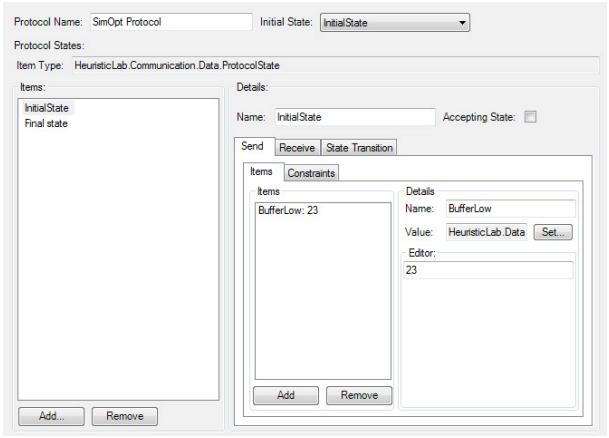

Figure 2: Communication protocol interface

On the simulation side the protocol needs to be implemented as well, but because the language of choice for many simulators is C/C++ or Java and HeuristicLab builds on C# and Microsoft .NET an automated way of including the protocol into the simulation is not yet available. It is however possible to implement libraries for different programming languages that will take care of the protocol handling. Basically it is a good idea to start designing the simulation, then think about the communication with the optimizer and model the protocol using the editor in HeuristicLab.

The generality of this protocol specification does not yet forestall the actual communication topology that will be used in the communication process. As is mentioned above, transmission is implemented in operators which can be interwoven with the optimization strategy and generally part into two levels: Serialization and transmission. The more complex, but more general serialization is an XML based serialization of the objects. This is useful when the partner that deserializes the information does know about XML and can reconstruct the HeuristicLab datatypes. Another possibility is to use a simpler serialization which just outputs the variable types, names and values each on a new line and is quick to implement in different programming languages. It is also possible to add any other type of serialization by implementing new operators as plugins. Once the data has been serialized it can be transmitted and again several operators will take care of that. There are operators for StdIn/StdOut communication and operators for sending the data over a TCP network. Again, extension to any underlying network transport protocol is possible.

During the execution of the algorithm communication will be performed by a specific object which derives from the interface *IDataStream*. This interface has five methods: *Initialize*, *Connect*, *Close*, *Write* and *Read*. This object is placed within the scope tree and available to a number of operators which then call these methods.

After the communication with the simulator has been fixed the next step is to choose an optimization strategy, appropriate parameters and start optimizing.

3. EVOLUTION STRATEGY

The first ES (Rechenberg 1973) described used a population of one parent and one offspring typically denoted as (1+1)-ES. Starting from a randomly initialized parent, mutation adds Gaussian distributed noise to each value of the parameter vector. The mutated parameter-vector is evaluated and replaces the old parent if it is better than the parent; if not, the new solution is discarded. Further investigations and developments resulted in the (μ+λ)-ES (Schwefel 1987) where λ new individuals are generated for the next replacement phase. This (μ+λ)-ES is effectively an elitist algorithm and an alternative scheme called ‘comma-strategy’ is also defined. In this the μ best individuals are used only from the offspring not considering the old parents. This is denoted by (μ,λ)-ES. In this scheme λ needs to be significantly greater than μ or otherwise the search performs a random walk instead of an optimization.

The random noise added to each element of the parameter vector is normally distributed with parameters (0, σ). Rechenberg observed for the (1+1)-ES that adapting σ in the course of the evolution leads to better results; based on this observation he formulated the "1/5 success rule" that adapts σ based on the average success of the last n mutations with the goal of generating on average 1/5 successful mutations. When the ratio of successful mutations is greater than 1/5, the standard deviation should be increased while it should be decreased when the ratio of successful mutations is less than 1/5.

The evolution strategy is designed with respect to continuous parameter adjustment. To apply it on simulation-based optimization problems, however does not always rely on continuous parameters alone (Affenzeller et al. 2007). Some models and also the one we used for testing in this paper make use of discrete parameters, e.g. integers as well. The mutation operators need to adapt to these mixed-integer optimization problems. Newer mutation strategies are proposed that take into account a mixture of discrete and continuous parameters. Even more data representations, such as unordered discrete parameters, arrays, boolean values and others may be encountered in simulation-based optimization which requires the definition of further extensions to the evolution strategy. Common to all these manipulation concepts are several properties (Bäck and Schütz 1995):

- Smaller changes should be more likely than larger ones
- Change in any direction of the domain should be equally likely

4. RESULTS

4.1. Optimization of a Supply Chain Simulation Model

Results have been computed with one of the simulation models that ships with AnyLogic 6, namely the supply chain simulation model. This model consists of three buffers at each location in a supply chain: Retailer, Wholesaler and Factory. The retailer has to hold items in stock to sell to customers and orders new items from the wholesaler which in turn orders from the factory which produces items from an infinite source of instantly available raw materials. Each buffer is accompanied with two decision variables representing an upper and lower limit of the buffer. New items are ordered/produced when the stock falls below the lower limit in such quantity that it would fill the buffer to the upper limit. Orders at the factory and the wholesaler are shipped at the start of each day only. The model has constrained input parameters: The decision variables have to lie in the interval [1;200] and the lower limit must not be greater than the upper limit. The goal of the model is to reduce the total costs in the supply chain by minimizing buffer sizes in each location. Naturally, the optimum to this goal would be not to have any buffers and produce only on demand. This would however have customers wait for days to get their items and so reducing the mean customer waiting time (E[waitingTime]) is a second goal which stands in opposition to the first. A two goal approach however would require the application of multiobjective optimization techniques, so in this model the second goal is considered as constraint. The solution is described as feasible only when the mean customer waiting time is ≤ 0.001. In our example we allowed infeasible solutions during the search, but added a penalty to their quality value. For a feasible child with E[waitingTime] ≤ 0.001 the fitness function is E[DailyCosts], in case of an infeasible child, this value becomes: E[DailyCosts] * (100 + E[waitingTime]).

The optimum is to set the decision variables such that the buffer size becomes as small as possible while maintaining a low waiting time. Given these characteristics it is clear to assume that the "global best feasible" solution borders the infeasible region in the search space. It is also valid to assume that the feasible region covers a coherent space of reasonable size. It seems unlikely to find an optimal buffer configuration deep within the territory of infeasibility, e.g. with low costs (small buffers), but also low waiting time.

The size of the search space is reasonably large such that exact calculation or enumeration is not viable anymore. For each buffer there are about 20,000 possible combinations of lower and upper bounds, which amount to about 8 trillion possible combinations when considering all three buffers.

Another challenge in this model is its stochastic behavior. Customer demand is modeled by a random variable, so that a single simulation run usually is not enough to test for the feasibility of a solution or give a reliable estimate for its quality. The implications to the fitness landscape are that the region surrounding the global optimal solution(s) is highly disturbed by noise. There is even the question remaining if there can be a global optimal solution at all without a given confidence level for the feasibility of a solution. Because arrival is random even the safest solution can seem infeasible if the retailer is stormed on an unlucky day. The chance for such an event is extremely low though and thus we can only find feasible optimal solutions with regard to a certain probability.

We have performed two tests with HeuristicLab to optimize this model: In the first test we treated the simulation as deterministic model and initialized it with the same random seed in each replication, while in the second test we used a different random seed and optimized a stochastic model.

The results are compared to the commercial optimizer OptQuest which is already implemented within this sample simulation model. To each optimizer about the same amount of simulation replications is given as termination criterion. OptQuest applies a mixture of Scatter Search and Tabu Search and combines it with Artifical Neural Networks to learn about the response curve during the search (Glover et al. 1999).

4.1.1. Deterministic Simulation

Here we used a (5+10)-ES with self adaptive mutation strength adjustment as described in (Igel et al. 2006). The initial population is generated by a uniform random initialization of the decision variables within their bounds, manipulation adds a normal distributed variable N(0,1) multiplied with the current mutation strength and rounds the result to the next integer. This is not optimal as (Bäck and Schütz 1995) notes, however we are using a simpler self adaptive technique with just one adaptive parameter for the whole vector and no recombination. We found that as simple as this approach is, the results we could achieve are of good quality. The parameters for the 1/5-success rule self adaptation were set as follows: Initial mutation strength: 10, learning rate: 0.1 and damping factor: 50. It was run for a maximum of 1000 generations which amounts to about 10,000 evaluations. Because the model is deterministic no replications are made.

The best found solution in 5 runs with different random seeds has a quality value of 505.595, the parameter vector contained 43-55, 44-51 and 36-77 as the lower and upper bounds for the retailer, wholesaler and factory respectively. This has been the lowest feasible quality that we encountered during the test and it is obvious that it is likely to be infeasible in a majority of the situations. Indeed, evaluation of this setting with 100 independent replications showed that none of them achieved a feasible result. This is likely a solution which is feasible in very few cases. Other solutions obtained from these tests had slightly worse quality, but were only slightly more likely to be feasible when tested on the stochastic model. Thus the optimizer, in its best run, moved past a global optimal solution and into the region where a solution is likely to be infeasible, except that for the one random seed that the model has been initialized with it was still feasible. This shows that the optimizer will find highly specific solutions which work only under the fixed random setting. To obtain a real result of the performance, we will need to optimize the stochastic model.

4.1.2. Stochastic Simulation

As interesting and well performing as the results from the deterministic case are, the stochastic case is much closer to the real world. Its optimization however raises a new challenge. If we would evaluate a given configuration only a single time we may not have enough confidence about the feasibility of the solution to accept it into the next generation. So we need to have some confidence which configurations and their qualities to accept.

It was tried first with just 5 replications per individual and the final solutions were found to have moved into a region where solutions have a higher probability of being infeasible. Replications were then raised to 10 per evaluation and the results showed more confidence. Naturally, the more replications are made the more computational effort is necessary and we were still not satisfied fully with the confidence given this amount of replications, but then thought of a different approach instead of raising it yet another time. A new child is evaluated and judged by 10 replications, but as it becomes parent, producing offsprings over possibly many generations and thus influencing the search trajectory, it is evaluated again once in each generation and the average is computed anew. This requires us to keep track of the previous qualities and shows once again the strength of HeuristicLab where this could be prototyped easily within the running application. For the search it means that the longer a parent survives, the more evaluations it will collect and the more confidence we could gain in its quality.

To compute the results we used the same (5+10)-ES as in the deterministic case, with 10 replications per evaluation and adding another replication for each parent in each generation. The stop criterion was 50 generations at which about 5300 simulation runs have been counted. One test was set to run to 100 generations producing 10,550 simulation runs. The parameters for the 1/5-success rule self adaptation were set as follows: Initial mutation strength: 10, learning rate: 0.2 and damping factor: 10 to allow for quicker adaption given the small number of generations. OptQuest was also applied with the stopping criterion of 5300, 10,550 and one test with a maximum of 100,000 runs to see what is possible. Table 1 shows a summary of the results.

Table 1: Comparison of HeuristicLab (HL) with OptQuest (OQ) on the supply chain simulation model in AnyLogic 6

	Evaluations	Average	Best	Worst
HL	5300	550.03	540.17	564.51
	10550	537.39		
OQ	5300	578.63	571.34	585.04
	10550	567.01	559.73	580.21
	100000	548.88		

The best found solution with the (5+10)-ES in HeuristicLab was finally given more replications later and has a confidence of more than 90% in 100 additional replications. The solution had 14 replications counted by the algorithm. The parameters found were 57-60, 57-65 and 45-92 for the bounds of the retailer, wholesaler and factory respectively. The oldest solution in this run's parent population came to 28 replications, which means it survived 18 generations. Its quality was 538.81 and not much worse than the best. Its confidence was also above 80%. The best solution found by HeuristicLab with 5300 replications had a quality of 540.17 stored 17 replications during the algorithm run. It has a confidence of about 87%. The parameters are 66-66, 63-66 and 34-78. One problem that we saw was that feasibility is hard to evaluate. The optimization strategy used is very keen to walk past an optimal solution and into the region of infeasibility. Elitism may partly be blamed for this kind of behavior: Once the

"right" 10 replications have been found for a good configuration it is accepted into the parent generation. The approach using reevaluations could lessen this effect somewhat. An alternative strategy would be to use less replications in the beginning, and increase this number as the search progresses.

The best result that OptQuest found had following parameters: 73-82, 60-60 and 41-74. In 100 additional tries it was found to be infeasible 10% of the time. The number of fixed replications per solution was also set to 10 similar to the ES in HeuristicLab.

Elapsed runtime is another important property of an optimization strategy and one where HeuristicLab is also competitive as optimization environment for simulation-based optimization. The workbench with 10550 replications took approximately 4 minutes to evaluate on an Intel Pentium 4 running at 3.0 Ghz. This comes to about 44 replications per second. On the same computer and using the same number of replications OptQuest, including user interface updates, finished in 3 minutes which is about 60 replications, though it is likely to be even faster when the GUI is turned off.

Nevertheless, to scale to more computational intensive simulations as well as to parallel hardware HeuristicLab also features several parallelization strategies such as threading and distributed computing and can thus talk to multiple simulation models running at once.

The results indicate that our evolution strategy performed quite successful and that it seemed adequate from the point of view of optimization complexity. One advantage with a general and open optimization environment such as HeuristicLab is that basically any optimization strategy can be tried, from the simplest greedy search to the most complex metaheuristic techniques.

4.2. Metaoptimization

Another possibility for simulation-based optimization is in metaoptimization. The goal is similar to that of simulation-based optimization, except that the simulation model is a given optimization strategy with several parameters that is applied on a given problem. A metaoptimization algorithm continuously feeds new parameters into another optimization strategy and receives a measure of its performance. The goal could be to tune the parameters so that the underlying optimization strategy finds better solutions and/or becomes more robust so that it finds them with higher probability. In our example we tried to optimize the self-adaptive parameters of an evolution strategy that is applied on a real valued optimization problem.

The learning parameters to be optimized are: The initial mutation strength, the learning rate and the damping factor. They control the mutation strength adjustment in a (1+1)-ES which is applied on a 50 dimensional sphere function. The ES is set to run for 2000 generations. The metaoptimizer in this case is a Genetic Algorithm with a population of 100, 5% mutation, 1-Elitism and Roulette-wheel selection. The crossover is a simple one point crossover that takes one part of the parameter vector from one parent and the other part from another parent.

The results in the five test runs performed indicated that several different settings have been found to achieve a good result within the 2000 generations. The best result had a very high learning rate of 0.97, an initial mutation strength of 4.70 and a damping factor of 4.28. Its quality averaged over 10 independent runs ranged from 0.0118 to 0.05800. Another metaoptimization run resulted in a best quality of 0.0139 with a learning rate of 0.58, an initial mutation strength of 6.62 and a damping factor of 4.04. Another one found a good solution with a quality ranging from 0.0147 to 0.0674 and a low learning rate of 0.21, initial mutation strength of 13.63 and damping factor of 6.37. It is difficult to draw conclusions about the direction of a best setting. One common thing that we observed while examining the found solutions is that the higher quality solutions overall had a lower damping factor than the worse quality solutions in the final population, whereas we found a damping factor too low, e.g. close to 1 also among the worse solutions in the final population. Such correlations however are not based on statistical evaluation. The amount of data generated during such metaoptimization tests could be used to gather a better understanding of the influence of the parameters on the metaheuristics applied, especially on more complex problems and strategies which do not lend well to analytical study. Metaoptimization could be used to explore the parameter space and statistical as well as machine learning methods could be used to find correlations, clusters or patterns in the data which could lead to more efficient parameter settings.

5. CONCLUSION AND FUTURE PERSPECTIVES

In this work we shared how the HeuristicLab framework can be used for simulation-based optimization and presented some results based on a basic simulation model as well as metaoptimization. The results show that metaheuristics built with this environment can be used effectively for these kind of tasks.

We also showed several interesting topics of research: Dealing with noisy optimization when optimizing stochastic simulation models on the one hand and the adaptation and application of well known heuristics for the purpose of simulation-based optimization on the other hand. Also interesting will be the topic of metaoptimization and how to combine it with simulation-based optimization as well as the topic of parallelization which goes beyond the scope of this paper.

We plan to continue and extend the work described and further test optimization strategies for the purpose of simulation-based optimization as well as seek applications of our work to simulation models from the industry to evaluate the performance of a number of possible approaches.

REFERENCES

Affenzeller, M., Kronberger, G., Winkler, S., Ionescu, M., Wagner, S. 2007. Heuristic Optimization Methods for the Tuning of Input Parameters of Simulation Models. *Proceedings of I3M 2007*, pp. 278-283. October 4-6. Bergeggi (Italy).

Azadivar, F. 1999. Simulation Optimization Methodologies. *Proceedings of the 1999 Winter Simulation Conference*, pp. 93-100. December 5-8. Phoenix (Arizona, USA).

Bäck, T., Schütz, M. 1995. Evolution Strategies for Mixed-Integer Optimization of Optical Multilayer Systems. *Evolutionary Programming IV: Proceedings of the 4th Annual Conference on Evolutionary Programming*, pp. 33-51. March 1-3. San Diego (California, USA). MIT Press, Cambridge, MA (USA).

Glover, F., Kelly, J.P., Laguna, M. 1999. New Advances for Wedding Optimization and Simulation, *Proceedings of the 1999 Winter Simulation Conference*, pp. 255-260. December 5-8. Phoenix (Arizona, USA).

Igel, C., Suttorp, T., Hansen, N. 2006. A Computational Efficient Covariance Matrix Update and a (1+1)CMA for Evolution Strategies, *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO'06)*, pp. 453-460. July 8-12. Seattle (Washington, USA).

Ólafsson, S., Kim, J. 2002. Simulation Optimization. *Proceedings of Winter Simulation Conference*, pp. 79-84. December 8-11, San Diego (California, USA).

Rechenberg, I. 1973. *Evolutionsstrategie.* Friedrich Frommann.

Schwefel, H-P. 1987. Collective Phenomena in Evolutionary Systems. *Preprints of the 31st Annual Meeting of the International Society for General System Research*, pp. 1025-1033. June 1-5. Budapest (Hungary).

Wagner, S., Affenzeller, M. 2005. HeuristicLab: A Generic and Extensible Optimization Environment. In Ribeiro, B. Albrecht, R.F., Dobnikar, A., Pearson, D.W., Steele, N.C., eds. *Adaptive and Natural Computing Algorithms*. Springer-Verlag New York, Inc., pp. 538-541.

Wagner, S., Winkler, S., Braune, R., Kronberger, G., Beham, A., Affenzeller, M. 2007. Benefits of Plugin-Based Heuristic Optimization Software Systems. *Lecture Notes in Computer Science 4739*, pp. 747-754. Springer-Verlag.

AUTHORS BIOGRAPHY

ANDREAS BEHAM received his MSc in computer science in 2007 from Johannes Kepler University (JKU) Linz, Austria. His research interests include heuristic optimization methods and simulation-based as well as combinatorial optimization. Currently he is a research associate at the Research Center Hagenberg of the Upper Austria University of Applied Sciences (Campus Hagenberg).

MICHAEL AFFENZELLER has published several papers and journal articles dealing with theoretical aspects of genetic algorithms and evolutionary computation in general. In 1997 he received his MSc in Industrial Mathematics and in 2001 his PhD in Computer Science, both from Johannes Kepler University Linz, Austria. He is professor at the Upper Austria University of Applied Sciences (Campus Hagenberg) and associate professor at the Institute of Formal Models and Verification at the Johannes Kepler University Linz, Austria since his habilitation in 2004.

STEFAN WAGNER received his MSc in computer science in 2004 from Johannes Kepler University Linz, Austria. He currently holds the position of an associate professor at the Upper Austria University of Applied Sciences (Campus Hagenberg). His research interests include evolutionary computation, heuristic optimization, theory and application of genetic algorithms, machine learning and software development.

GABRIEL K. KRONBERGER received his MSc. in computer science in 2005 from Johannes Kepler University Linz, Austria. His research interests include parallel evolutionary algorithms, genetic programming, machine learning and data-mining. Currently he is a research associate at the Research Center Hagenberg of the Upper Austrian University of Applied Sciences (Campus Hagenberg).

The Web-pages of the authors as well as further information about HeuristicLab and related scientific work can be found at http://www.heuristiclab.com.