

HeuristicLab: A Generic and Extensible Optimization Environment

S. Wagner, M. Affenzeller
Institute for Formal Models and Verification
Johannes Kepler University, Linz, Austria
E-mail: stefan@heuristiclab.com, michael.affenzeller@jku.at

Abstract

Today numerous variants of heuristic optimization algorithms are used to solve different kinds of optimization problems. This huge variety makes it very difficult to reuse already implemented algorithms or problems. In this paper the authors describe a generic, extensible, and paradigm-independent optimization environment that strongly abstracts the process of heuristic optimization. By providing a well organized and strictly separated class structure and by introducing a generic operator concept for the interaction between algorithms and problems, HeuristicLab makes it possible to reuse an algorithm implementation for the attacking of lots of different kinds of problems and vice versa. Consequently HeuristicLab is very well suited for rapid prototyping of new algorithms and is also useful for educational support due to its state-of-the-art user interface, its self-explanatory API and the use of modern programming concepts.

1 Introduction

Naturally inspired optimization heuristics have already been used to attack various kinds of problems. They were applied to e.g. routing problems, scheduling problems, function optimization, or graph problems. As a basis for the different optimization algorithms different natural archetypes were considered: E.g. the process of natural evolution inspired Evolutionary Computation, the foraging behavior of ants provided a basis for Ant Colony Optimization or the movement and social behavior of bird flocks or fish schools led to Particle Swarm Optimization. Due to their generality a lot of these heuristics are able to attack very different kinds of problems. However, to obtain very high quality results, various problem specific adaptations and hybrid variants were also developed. This leads to the situation that not only the number of problems but also the variety of different algorithms is enormous today.

So scientists working in the area of heuristic optimization have to deal with this variety of optimization al-

gorithms and problems. On the one hand it is necessary to test newly developed algorithms with different problems and on the other hand different optimization algorithms have to be evaluated concerning their applicability for an optimization problem. Consequently a paradigm-independent optimization environment is favorable that separates the algorithm and problem implementation from each other, making it possible to exchange both of them freely. Furthermore, it is preferable to have an environment at hand that supports the users not only concerning the implementation of algorithms and problems but also offers a very high degree of usability by providing e.g. a state-of-the-art user interface, a self-explanatory installer, and a concise documentation, making it also useful for educational purposes in student courses.

In this paper the authors present a new generic, flexible, and extensible optimization environment named HeuristicLab that is designed to deal with very different kinds of optimization algorithms and problems. Due to the strict separation of algorithms and problems (see section 4) one of the most important aspects is the development of a new generic concept to define abstract operators for all interactions between algorithms and problems (described in section 5).

2 Requirements Analysis

One of the basic features of each generic optimization environment is to enable the user to add new optimization algorithms as well as new optimization problem types in a very fast and easy way. Furthermore, the program should not be focused on a specific kind of optimization problem or algorithm. The environment should provide a framework with a well structured and self-explanatory application programmer's interface (API) that enables the user to implement all different kinds of algorithms and problems. This framework should especially take care of all not optimization specific parts like threading, saving and loading settings, reading and writing files, or the graphical user interface (GUI). Moreover,

it should also provide a plug-in architecture that allows adding, removing and using algorithm and problem implementations at any time.

Concerning the implementation of various different kinds of algorithms and problems the most important aspect should be to follow one of the fundamental principles of object oriented programming: the exchangeability of modules. Consequently it should be possible to attack all different kinds of implemented problems with *one* single algorithm implementation (e.g. a generic Genetic Algorithm (GA)) without the need to change any code of the algorithm plug-in. Vice versa *one* problem implementation (e.g. an implementation of the Traveling Salesman Problem (TSP)) should be useable for all available algorithms. So it would be of crucial importance to provide an abstract way of communication between algorithms and problems to be able to exchange both parts independently.

Additionally to an usable API the whole program itself should be very intuitional in order to use it not only for algorithm development and problem solving but also for educational aspects. It should provide a state-of-the-art installer and a GUI to enable students or other interested users to experiment with different optimization techniques and problems out of the box.

3 Other Available Optimization Software Packages

Of course the idea of a generic and extensible optimization environment is not new. Especially in the area of Evolutionary Computation there are numerous more or less mature libraries (a comprehensive list can be found e.g. on the EvoWeb homepage¹). Among them are a lot of small projects which have just been developed to solve a specific optimization problem, some libraries which are a little bit outdated or have not yet reached a state of broad application (like GALib [1], OpenBEAGLE [2], TEA, GENOM, JDEAL), and a few highly-developed frameworks (like EO [3], JEO [4], ParadiseEO [5], ECJ). Especially the Evolving Objects (EO) project and its further developments JEO and the DREAM project [6] have to be mentioned as very ambitious and successful approaches towards a generic and flexible framework for (parallel) Evolutionary Computation. However, all these libraries suffer from some weaknesses concerning the requirements stated in the previous section:

Almost all of the existing libraries are dedicated to a specific optimization paradigm like Evolutionary Algorithms, Local Search or Swarm Algorithms. It is a very hard task to break up the libraries in such a way that optimization algorithms of another paradigm can be equally used. However, for practical applications the ability to

compare results of different paradigms is very essential.

All the existing libraries are - as the name suggests - libraries. Therefore, their usability cannot really be compared with state-of-the-art software products. If at all GUIs are only rudimentally available, self-explanatory installers are missing altogether. None of the existing packages can be used out of the box without deeper programming knowledge. Consequently, the usability for e.g. university courses is rather limited as the students should not spend their time with installing but with using the software.

So it can be said that there definitively exist some powerful optimization libraries, but all of them seem not to be able to fulfill the desires of the authors: a generic and extensible optimization environment, open for all different kinds of optimization paradigms, comparable to state-of-the-art software packages, equipped with a GUI and a self-explanatory installer, usable out of the box not only for rapid prototyping and comparison of new optimization algorithms but also for educational purposes.

4 Modeling the Optimization Process

After having thought of all the different requirements for the project the first point to start with is the modeling of the optimization process itself. As already stated in the introduction there are manifold natural archetypes that inspired heuristic optimization techniques. Consequently the resulting algorithms have almost no structural similarities. So due to the strict demand for generality and openness the process has to be abstracted to a very high level to assure that all different kinds of algorithms and problems can be integrated into the framework.

Primarily a heuristic optimization process can be stated in the following way: An *algorithm* iteratively modifies one or more *solutions* of a *problem* in a specific way in order to increase their quality and generates *results* to inform others about the progress of the optimization. Obviously the four modules Algorithm, Problem, Solution, and Result form the main actors and consequently they have been modeled as base classes in the HeuristicLab framework. Furthermore, also the EvaluationOperator delegate is included as the calculation of the quality value of the different solutions is one of the main issues in every heuristic optimization algorithm. More details on the operator concept are given in section 5. In Fig. 1 an UML class diagram shows these classes and their most important properties and methods.

Due to the very high level of abstraction the base classes only provide very rudimental functionality. The three classes Algorithm, Problem, and Result contain basic parameter properties (e.g. Rounds, RandomSeed, Is-

¹<http://evonet.lri.fr/evoweb/resources/software>

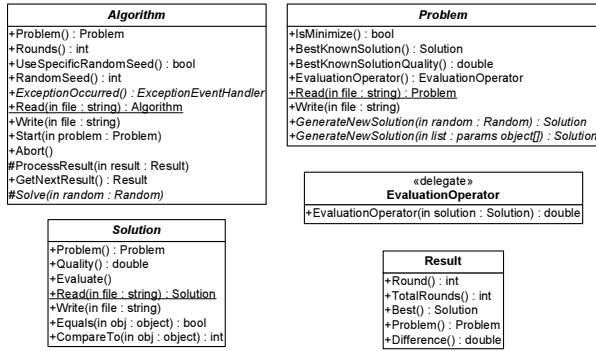


Fig. 1. Class diagram of the main HeuristicLab framework classes.

Minimize, etc.) which are reasonable in every kind of optimization task. Furthermore, they provide a reading and writing mechanism to store and retrieve the current object (binary serialization). Additionally Algorithm also takes care about threading by performing the algorithm execution in an own thread, separating this computational heavy task from the representation and administration issues.

Based upon these four base classes the HeuristicLab framework can be used to implement very different kinds of optimization algorithms and problems by inheriting from them and by implementing their abstract methods. However, HeuristicLab is not just one more class library offering some more assistance for solving optimization problems. HeuristicLab is more thought of as a fully functional optimization environment which can be extended by own algorithm and problem plug-ins. So a layer is needed above the framework base classes that implements a GUI front end and is therefore responsible for all user interactions, for the presentation of various results generated by the algorithms and for administrative tasks like installing or removing plug-ins or changing global settings. The propagation of results from the Algorithm to the front end is done via Result objects. Fig. 2 visualizes the interplay between the different HeuristicLab classes.

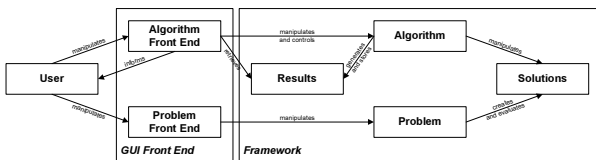


Fig. 2. Diagram showing the interaction between the various HeuristicLab classes.

5 Communication between Algorithms and Problems – The Operator Concept

Due to the high demand for generality and extensibility a fundamental problem occurs concerning the interaction between algorithms and problems. It has to be possible to use a specific algorithm with any optimization problem that is available and the other way round to attack a concrete problem implementation with any algorithm. Consequently both sides can only use the base classes Algorithm, Problem, and Solution to work with each other as they are the least common denominator of all algorithms and problems. However, e.g. in the case of GAs the algorithm has to manipulate the solution candidates via the genetic operators crossover and mutation. These operators are problem specific as they depend on the problem encoding. But the algorithm itself has no idea about the encoding used for the problem implementation and furthermore also cannot provide any crossover or mutation operator on its own.

This general dilemma occurs not only in the case of GAs. In some way any heuristic optimization algorithm has to manipulate solution candidates in order to find better solutions. So there has to be a way that the algorithm can perform problem specific operations without having to know details about the problem's implementation.

To handle this problem the operator concept was developed. This programming technique is based on a main idea of object oriented programming (OOP): the separation of the declaration and the implementation of objects as it is done in OOP by interfaces and classes. However, in the case of HeuristicLab this separation doesn't take place on the level of classes but on the level of methods. An algorithm declares the parameters and the return type of the operators it needs (i.e. the method interface) and a problem has to provide concrete methods that match exactly with the declared interfaces and represent those operators. So in the case of GAs the algorithm declares e.g. a crossover operator as a method that takes two solutions as parameters (the parents) and returns a new solution (the child).

From a more software development oriented point of view this can be realized by using delegates. A delegate is a type that represents methods. When a delegate is defined the parameter types and the return type of the method are specified. Then any concrete method that has the same parameter types and return type can be assigned to a variable of the delegate type. This variable can be used like any other method. So e.g. the Algorithm class in a GA implementation has to have two properties that are used to store the two genetic operators crossover and mutation. When a problem instance is selected for optimization a value is assigned to these properties that

references to the methods implemented by the problem that should be used as crossover or mutation operators. Furthermore, as the algorithm and problem implementations are located in different HeuristicLab plug-ins meta data is used to label those methods that implement specific operators. So an algorithm can read this meta data via reflection at runtime to initialize the needed delegates properly.

By this operator concept no specific information about the problem implementation is needed and so all problems that provide the necessary operators can be attacked by an algorithm. On the other hand a problem just has to provide some methods implementing certain operators and to label them correctly. So e.g. if a problem implements a neighborhood operator, it can be solved with any neighborhood-based optimization technique that only needs to compute the neighborhood of a solution (like Simulated Annealing, etc.).

6 Current State of Development

As reflection, meta data and delegates are needed to realize the above discussed architecture and concepts, the Microsoft .NET Framework 1.1 and C# were used for the implementation of HeuristicLab. At the moment HeuristicLab 1.0.0 is freely available for non-commercial and educational use. More details are available at <http://www.heuristiclab.com>.

Up to now several different optimization algorithms and problems have been implemented for HeuristicLab and lots of them are already included in the actual version. The following plug-ins are currently available (plug-ins printed in *italics* are currently still under development): GA (Generic Genetic Algorithm), SSGA (Steady-State Genetic Algorithm), IslandGA (Coarse Grained Parallel Genetic Algorithm), *SASEGASA* [7], GP (Genetic Programming), ES (Evolution Strategy), SA (Simulated Annealing), PSO (Particle Swarm Optimization), *ACO* (Ant Colony Optimization), TS (Tabu Search), TSP (Traveling Salesman Problem), JSSP (Job Shop Scheduling Problem), MPSP (Multiprocessor Scheduling Problem), *GCP* (Graph Coloring Problem), *BP* (Bin Packing Problem), TestFunctionsND (n-Dimensional Real-Valued Test Functions). The architecture of the HeuristicLab framework as well as the operator concept have proven themselves to be extremely flexible and very well suited for all different kinds of algorithms and problems. Besides its potential for the rapid prototyping of new optimization heuristics HeuristicLab was also already used as supporting tool in university courses and is also used as development environment for real-world applications ([8],[9]).

7 Conclusion

In this paper we have presented the basic architecture of HeuristicLab - a generic, extensible, and paradigm-independent optimization environment. A clear and well organized class structure, the operator concept, a self-explanatory API and the use of modern programming concepts make it possible to easily use HeuristicLab for the development of different kinds of heuristic optimization algorithms and problems as well as to exchange the different implementations freely. Consequently especially for rapid prototyping of new optimization algorithms HeuristicLab turned out to be a powerful tool. Moreover, HeuristicLab is not just an optimization class library. It also provides an easy to use and state-of-the-art GUI and installer and is therefore also suitable as an educational tool for university courses.

8 References

- [1] Wall, M. (1996) GALib: A C++ Library of Genetic Algorithm Components. Mechanical Engineering Dept. MIT
- [2] Gagné, C., Parizeau, M. (2002) Open BEAGLE: A New Versatile C++ Framework for Evolutionary Computations. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 2002
- [3] Keijzer, M. et al. (2001) Evolving Objects: A General Purpose Evolutionary Computation Library. Proceedings of the 5th International Conference on Evolutionary Algorithms (EA 01)
- [4] Arenas, M. G. et al. (2002) JEO: Java Evolving Objects. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 2002
- [5] Cahon, S. et al. (2004) ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. Journal of Heuristics – Special Issue on New Advances on Parallel Meta-Heuristics for Complex Problems 10:357–380
- [6] Arenas, M. G. et al. (2002) A Framework for Distributed Evolutionary Algorithms. Lecture Notes in Computer Science 2439:665–675
- [7] Affenzeller, M., Wagner, S. (2004) SASEGASA: A New Generic Parallel Evolutionary Algorithm for Achieving Highest Quality Results. Journal of Heuristics – Special Issue on New Advances on Parallel Meta-Heuristics for Complex Problems 10:239–263
- [8] Winkler, S., Affenzeller, M., Wagner, S. (2004) Identifying Nonlinear Structures Using Genetic Programming Techniques. Cybernetics and Systems 2004 689–694
- [9] Braune, R., Wagner, S., Affenzeller, M. (2004) Applying Genetic Algorithms to the Optimization of Production Planning in a Real-World Manufacturing Environment. Cybernetics and Systems 2004 79–84