Optimization Knowledge Base: An Open Database for Algorithm and Problem Characteristics and Optimization Results

Andreas Scheibenpflug, Stefan Wagner, Erik Pitzer, Michael Affenzeller
University of Applied Sciences Upper Austria
School of Informatics, Communication and Media
Softwarepark 11, 4232 Hagenberg, AUSTRIA
{ascheibe, swagner, epitzer, maffenze}@heuristiclab.com

ABSTRACT

This paper describes the optimization knowledge base (OKB), a database for storing information about algorithms and problems. The optimization knowledge base allows to save results of algorithm executions as well as problem-specific information of fitness landscape analyses. This database can be queried and gives researchers a tool for gaining a better understanding of problems and algorithms and their behavior. Therefore the OKB supports parameter tuning and keeping track of tested algorithm and parameter settings as well as their results. Furthermore, the OKB and fitness landscape analysis can be used to not only explain the behavior of algorithms but to calculate similarities between problem instances and algorithms. Based on similarities and already captured knowledge, parameter settings can be extracted that could work well for new problem instances. Additionally, the OKB can be used to publish results of experiments for a broader audience, which advocates transparency of scientific work in the area of metaheuristics.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Experimentation

Keywords

parameter control, parameter tuning, knowledge base, metaheuristics, fitness landscape analysis, HeuristicLab

1. INTRODUCTION

Metaheuristics are a group of algorithms that are applied to hard optimization problems. Metaheuristics provide an acceptable ratio between achievable quality and effort. The generated solutions are in most cases not the best solutions, but are good enough for practical applications and therefore metaheuristics are frequently a promising way to generate feasible solutions in reasonable time. When applying metaheuristics to a problem, the first step is to choose a suitable algorithm, as metaheuristics are problem-independent and some algorithms may work better on a certain problem then others. The second step is to find good parameters to obtain a satisfying quality as metaheuristics typically have a set of parameters that can be configured. These parameters are used to tailor the behavior of an algorithm to a problem (parameter tuning [5]). As good enough solutions can always be improved and algorithm selection as well as parameter tuning is a time consuming process, the last decades have seen researchers trying to improve metaheuristics. These efforts can be organized in two categories:

- Methods for generating better solutions.
- Techniques to reduce or eliminate algorithm selection and/or manual parameter tuning.

In the following a short overview of previous works which fall in one of the two categories is given.

1.1 New Algorithms

One of the most popular areas of ongoing research in the field of metaheuristics is the development of new algorithms. The idea is to find algorithms that are superior to the existing ones and therefore lead to better results. Besides creating completely new algorithms, there are also attempts to combine existing algorithms to use the advantages that each algorithm offers (hybrid metaheuristics [20]).

As mentioned in [6] finding a suitable algorithm and an appropriate configuration for a problem or class of problems is regarded by many computer science researchers and practitioners as the holy grail. This problem is called the algorithm selection problem [17]. Hyperheuristics [14, 2] try to solve this problem by combining simpler heuristics into one heuristic that is able to solve a class of problems. The information on how to combine heuristics is derived from the performance of previously combined heuristics and their parametrization.

The parameters of a metaheuristic can have a big influence on the behavior of an algorithm and have to be tuned to the problem which requires an expert with knowledge of the algorithm and the problem at hand. Therefore parameterless

GECCO'12 Companion, July 7-11, 2012, Philadelphia, PA, USA.

algorithms emerged which need little or no parameters to be configured [1, 11, 15, 7]. The idea is to eliminate parameter tuning which would result in huge time savings. There is of course a quality trade-off between generating robust results over multiple problem instances and tuning parameters for a certain problem instance. As parameterless algorithms and hyperheuristics fall in the first category, it is difficult to achieve similarly good results with parameterless algorithms as compared to manually tune a metaheuristic for a specific problem instance.

1.2 Parameter Tuning and Parameter Control

Depending on their parameter configuration, metaheuristics can drastically change their behavior. This led to the idea of defining rules for finding suitable parameters for problems or problem classes [8, 10]. It turned out that it is impossible to define parameter tuning rules which work well on a broad range of different problems. Another idea is to find a technique to tune parameters while executing the algorithm (parameter control [18, 4, 13]). Like with parameterless algorithms, a user does not have to configure the parameters of the algorithm. But in the contrast to a parameterless algorithm, which has no parameters, an algorithm which utilizes parameter control exhibits parameters, but automatically changes the parameters based on the behavior of the algorithm during execution.

Seeing the problem of finding parameters for an algorithm as a problem itself which can be optimized is the main idea of metaoptimization [9, 12, 19]. As any metaheuristic can be used as meta-level optimizer, metaoptimization does not use any information about the problem but measures obtained from the evaluated solution candidates (e.g. performance, effort or robustness). Because solution candidates are metaheuristics with a certain, generated parameterization, metaoptimization often exhibits enormous runtime requirements. Additionally, even though metaoptimization optimizes the parameters of metaheuristics, the meta-level algorithm still has to be parametrized and tuned by an expert.

1.3 Fitness Landscape Analysis

The techniques discussed in the previous sections focused on improving solution quality by designing new algorithms or reducing the amount of tuning needed for their parameters without making use of problem characteristics. If it is not possible to find good algorithms and parameter configuration strategies, another possibility is to analyse the problem which should be optimized. Fitness landscape analysis (FLA) [16] tries to develop metrics for determining certain aspects of a problem and gather significant information. Having more meaningful information about the problem at hand can provide crucial hints about which algorithm with which configuration can lead to good results. Of course the ultimate goal of predicting good parameter settings for metaheuristics based on these metrics has yet to be reached and is the goal of ongoing research in this area.

In our opinion there already exists a large number of sophisticated metaheuristics which lead to good results, if configured properly. Wolpert and Macready state in [23] that "These [No Free Lunch theorems] establish the equivalent performance of all optimization algorithms when averaged across all possible problems." The No Free Lunch Theorem [24] therefore restricts the idea of developing a general purpose algorithm that works very well on all problems. And even if such an algorithm could be found, the parameters for this algorithm would still have to be tuned manually. The above mentioned techniques for eliminating manual parameter tuning often don't lead to good results for multiple problems or exhibit enormous runtimes (e.g. metaoptimization). On the other hand manual parameter tuning can lead to good results for a certain problem instance but often lack applicability for other instances. Figure 1 shows this conflict between robust results and the achievable quality for some of the above mentioned techniques.

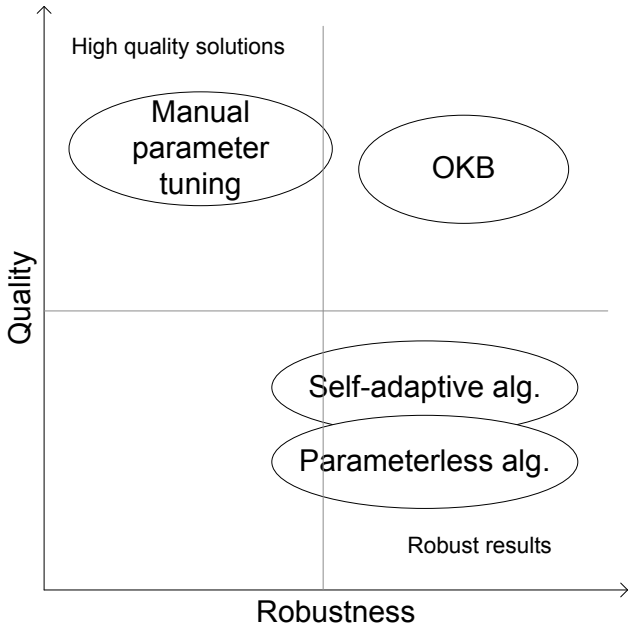


Figure 1: Conflict between robustness and achievable quality

We therefore propose a system that simulates manual parameter tuning by using FLA and knowledge collected from previous experience to automatically discover suitable algorithms with good parameter settings for new problem instances.

The remainder of the paper is organized as follows: Section 2 states the goals and requirements of the optimization knowledge base. Section 3 gives an overview of the optimization knowledge base and its architecture. It details on how information for the OKB is collected and discusses in Section 3.2 ideas on how to automate this process as well as how this information can be used for predicting algorithm performance. Section 3.3 presents a generic and extensible data scheme capable of representing the collected information and Section 3.4 gives an overview of the tools for querying the OKB.

2. GOALS AND REQUIREMENTS FOR THE OKB

The primary goal of the OKB is to establish an open platform for storing information about problems and their relationship with different algorithms and parameter settings.

Information about problems is generated by applying fitness landscape analysis to give a deeper insight into the problem and it's structure as well as a prediction of how difficult the problem is to solve. The information of the relationship between algorithms and problems is obtained by running different algorithms with different parameter configurations. The information that is stored is the parameter configuration as well as the results of the runs. This information is crucial in understanding how well an algorithm with a certain parameter configuration works on a specific problem. The more information there is about a certain problem and the behavior of algorithms applied to this problem, the better an understanding can be developed for the problem and the algorithms. The goal is to generate an algorithm-problem mapping showing which algorithms work well on which problem(-instance)s as shown in Figure 2.

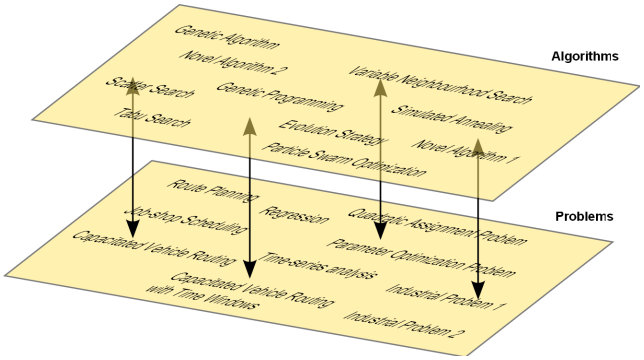


Figure 2: Algorithm-problem mapping

This mapping shows which algorithms work well on which problems, but not why this is the case. FLA can answer such questions as it delivers information about the characteristics of a problem which can be used to explain the performance and behavior of an algorithm and parameter setting.

Besides giving researchers and practitioners a very comprehensive view of the problem-algorithm landscape, the information stored in the OKB can also be used by programs. Metaoptimization algorithms can use the OKB as a cache for storing already evaluated solution candidates. Metaheuristics utilizing parameter control and agents (see Section 3.2) can use the stored information for adjusting the configuration of their parameters. Based on already acquired information it's possible for algorithms to pursue a much more deliberate and informed optimization process in contrast to classical metaheuristics that rely on repeated trial and error or hyperheuristics/metaoptimization which do not take any problem characteristics into account. Such enhanced methods can solve a problem much more like a experienced researcher who can learn from mistakes and interpret intermediate results.

The goal of the OKB is to enable predicting suitable algorithms and their parameter settings for new problem instances. Because the similarity of a new problem instance to existing problems can be calculated, parameter settings of these similar problems can be applied to the new problem and should therefore lead to reasonably good results. In this way the OKB acts like an expert who has done extensive parameter tuning. It simulates the expert by having knowledge about previous problems and what algorithms and settings worked well on them. It can therefore, like a human does, categorize the problem based on it's characteristics and, having a memory, know which algorithms and parameter settings should work well. The advantage of this approach is that having a knowledge, finding good parameter settings for new problem instances could be done fairly fast in contrast to metaoptimization or hyperheuristics, which do not have a memory they can build on.

Another aspect of the OKB is that such a system, if publicly available, allows research to be more transparent and reproducible. Currently researchers often use custom implementations of well known algorithms which are not available publicly. Furthermore experimentation and tests often lack a detailed description of the used parameter settings which makes it very hard to reproduce the results described in papers. By having an open database for storing such information, algorithms and parameter settings could be very well retraced and verified by other researchers. The OKB also offers a platform for cooperation as researchers can exchange their experiences through such an open system more easily.

2.1 Requirements

The proposed goals and features described in Section 2 lead to the following requirements:

- Central and open database: There has to be a central database where the information collected from different sources can be stored and queried. Because the OKB should be available to all researchers interested in the information the OKB contains, it has to be open to everybody. Access should be as easy as possible as researchers should be encourage to not only retrieve but also publish their results.

- A general purpose, extensible, and machine readable data schema: The schema has to be designed in a way so that it can be used for any optimization and FLA algorithm. If a new algorithm has new properties that have to be stored in the OKB, the data schema must be able to cope with the new data. Additionally the information must be represented in a way that programs written in different languages can access and interpret the information.

- Querying and filtering: As information grows, there has to be a mechanism which not only allows to query data but also to constrain the returned information so that only relevant data can be accessed easily.

- A well-defined, standards-based Web service interface: Because researchers should be encourage to use the OKB, different clients (metaheuristic frameworks) must be able to store and retrieve information, independent of the used programming language and platform.

- Scalability: Because the system is open, it has to be able to cope with large amounts of queries and a growing amount of data.

In the next section we describe the optimization knowledge base as well as how the above mentioned requirements have been implemented.

3. OPTIMIZATION KNOWLEDGE BASE

The OKB consists of a central database that contains the collected information as well as tools for producing information (Section 3.1) and querying information (Section 3.4). The following image gives an overview of the OKB:

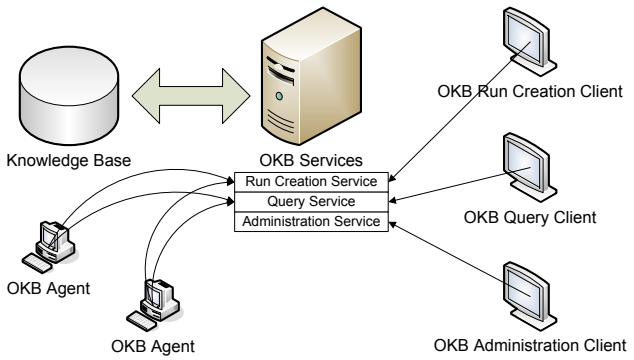


Figure 3: Overview of the OKB software system

The knowledge base is a relational database that stores information about algorithms, problems, and results. The OKB services are Web services that are used to access the information stored in the knowledge base:

- Run creation service: The run creation service is used for uploading runs to the OKB. A run contains the parameters and results generated by applying an algorithm to a problem. The service therefore offers a Web service method `AddRun` which takes a `Run` object and saves it in the database. Additionally the service contains methods to obtain a list of algorithms and problems so that a client can match the algorithm and problem from which the run was obtained to an algorithm and problem from the OKB.

- Query service: The query service is used to retrieve data from the OKB. It allows to query filters which a client can use to constrain the data a query returns. Additionally to restrict the returned runs, the service can return complete runs or partial runs which only contain a subset of the information of a run. Therefore `GetFilters` returns all available filters. These filters are then used when calling the `GetRunIds` method which returns the ids of the runs which match the filter criteria. With these ids the runs can be queried by calling `GetRuns` for retrieving complete runs or `GetRunsWithValues` for retrieving partial runs. The second method requires that a list of values is passed which defines the data that should be returned.

- Administration service: The administration service provides create, read, update and delete (CRUD) methods for algorithms and problems. Algorithms and problems have to be defined first in the OKB and can then be used to associate runs with them.

The OKB Run Creation Client is the front-end for storing information in the database. It collects information of an algorithm run and sends this information to the run creation service which stores it in the knowledge base. The OKB Query Client is used to search the OKB for information. It uses the filters provided by the query service to restrict the returned results of a query. OKB Agents interact with the OKB through the query and run creation service and perform various tasks to broaden and improve the information stored in the OKB. OKB Agents are discussed in more detail in Section 3.2.

The clients for creating and querying information are exemplarily implemented for HeuristicLab[1] [21]. HeuristicLab (HL) is a software system for heuristic and evolutionary algorithms. It contains a wide range of already implemented algorithms and problems as well as support for implementing new algorithms either by graphically designing them [22] or by writing code. Because the OKB services are open, other clients for the OKB can be written (there exists an experimental OKB Connector for the ECJ framework). This enables researchers to use their favourite tools and frameworks for executing experiments but at the same time still allowing them to store their results into the OKB.

3.1 Collecting information

Filling the optimization knowledge base with information is done by executing algorithms and using the OKB run creation service to store the gathered information in the database. Even if there exists no information about a problem in the OKB it still supports the process of parameter tuning as already tested configurations are recorded and can be analysed again at a later time. Of course if the OKB already contains information, parameter configurations can be chosen based on the knowledge of already tested configurations. When testing new parameter settings the OKB automatically receives new information and therefore the knowledge about a certain problem grows. Seeing what parameter settings have already been applied to an algorithm and problem prevents running the same configuration over and over again. It can show what configurations other researchers have previously tried leading to time savings when doing parameter tuning. Another way of collecting information is through metaoptimization or grid searches. These methods lead to different parameter combinations which are automatically generated, executed and the results can be automatically stored in the optimization knowledge base. In the next section ideas for automatic collection and utilization of information are discussed in detail.

3.2 Automatic information collection and use for algorithm performance prediction

To support the process of collecting data we introduce agents that take away most of the burden of manually performing parameter tuning. In the following three agents for gathering information are described:

- Diversification agents: This type of agent scans the OKB for missing algorithms and parameter configurations. If the agent finds that there are algorithms or configurations which haven't been tried, it runs these configurations and saves the results back to the knowledge base. This type of agent extends already existing information by doing parameter grid searches.

- Meta-information agents: Meta-information agents run special algorithms that generate meta-information about problems using fitness landscape analyses.

[1] `http://dev.heuristiclab.com`

- Intensification agents: Intensification agents take the information gathered by the first two types of agents and try to actively find better solutions for poorly covered and new problem instances. While the first two agents only broaden the gathered information, this type of agent actually tries to find better parameter settings based on the already known information. The ultimate goal of intensification agents is to emulate an expert doing parameter tuning. By learning from a knowledge base, containing not only results of algorithms and parametrizations but also meta-information of problems, the agent should choose new parameter settings intelligently based on this information with the concrete goal of finding better configurations and achieving better results.

The diversification agents need to be able to calculate the distance between parameter configurations. Having a distance metric for configurations enables agents to do parameter clustering. Based on the clustering agents can choose regions of the parameter landscape which haven't been thoroughly explored and start new runs to explore these regions.

The most advanced agent is the intensification agent. It can use the data collected by the FLA to determine similar problem instances. If two problem instances are very similar, the parameter setting which worked good on the first problem instance may also work good on the similar problem instance. This of course assumes that the problem was thoroughly searched and there are enough tested parameter configurations which led to good results. The advantage of this approach is that FLA is computationally far less expansive then parameter grid searches. The following pseudo code summarizes this idea:

Algorithm 1 Predicting parameter settings by problem instance distance

Calculate FLA for new problem instance $p1$
for all $p \in P$ **do**
Calculate distance to $p1$
end for
Select $p2 \in P$ with minimal distance to $p1$
for all $c \in C_n(p2)$ **do**
Execute $p1$ with c
end for
return Best solution obtained by applying c to $p1$

P is the list of all problems stored in the OKB. $C_n(p2)$ returns the n best configurations of the most similar problem instance to the new problem instance.

Of course parameter grid searches are computational expensive. To improve runtime, instead of calculating a more or less complete parameter grid, only a partial set could be generated and learned by a machine learning algorithm. This requires a technique that can learn collected parameter configurations and use the achieved quality as the target variable. If configurations are learnt, the quality of new configuration settings could be predicted. This results in significant runtime improvements as only parameter settings which look promising should be evaluated.

To reduce the amount of parameter configurations that have to be tested, the impact of parameters on the quality can be analysed (variable impact analysis). This means that it can be evaluated which parameter contributes how much to the overall quality of a solution. This allows to evaluate only parameter configurations with parameters that are predicted to improve the overall quality. Of course when one parameter is changed, the other parameters could change their behavior which also needs to be considered.

Automating parameter tuning leads to generating a lot of parameter configurations that have to be run on problems which are missing sufficient coverage or satisfying results. To speed up the generation and execution of parameter configurations, a system is needed that can distribute these tasks to multiple computers. Distribution of work is essential for obtaining an initial stock of information which can then be used by the intensification agent. HeuristicLab offers such a system (HeuristicLab Hive) that executes metaheuristics in parallel by distributing them to multiple computers. Through the OKB Services these systems can interact and provide the needed computing power for exploring algorithm configurations and problems.

3.3 Data model

All major metaheuristic frameworks support serialization of results. Results are often stored in a custom, binary format which can only be read and interpreted by the framework that generated it. This has the disadvantage that binary values are not human readable and also require applications handling the data to understand the binary format. Binary data can be stored in the OKB but it makes it difficult for agents and other frameworks to read. Additionally, the OKB needs values that it can understand to provide the filtering functionality to the query clients. Therefore the OKB supports saving values according to their data types which can be read by humans and used across different frameworks. HeuristicLab's OKB Run Creation Client for example checks the results and parameters of runs for known data types. If it finds a known data type this variable is stored as an OKB value. If there is no known OKB data type for the data type of a result or parameter value, the value is stored in a binary format. This means that the value can be retrieved by a framework which can interpret it, but it can't be used as a filter criteria by the OKB. The OKB supports data types based on the data types provided by the MS SQL Server 2008R2[2]. Table 1 gives an overview of OKB data types as well as an exemplarily mapping to HeuristicLab's data types.

OKB data type	HL data type
Int	int
BigInt	long
Bit	bool
Float	double
Real	float
NVarChar(MAX)	string
VarBinary(MAX)	byte[]

Table 1: Mapping of HeuristicLab data types to OKB data types

Additionally, more advanced data types can be represented as basic data types. For example, the HeuristicLab

[2] `http://msdn.microsoft.com/en-us/library/ms187752.aspx`

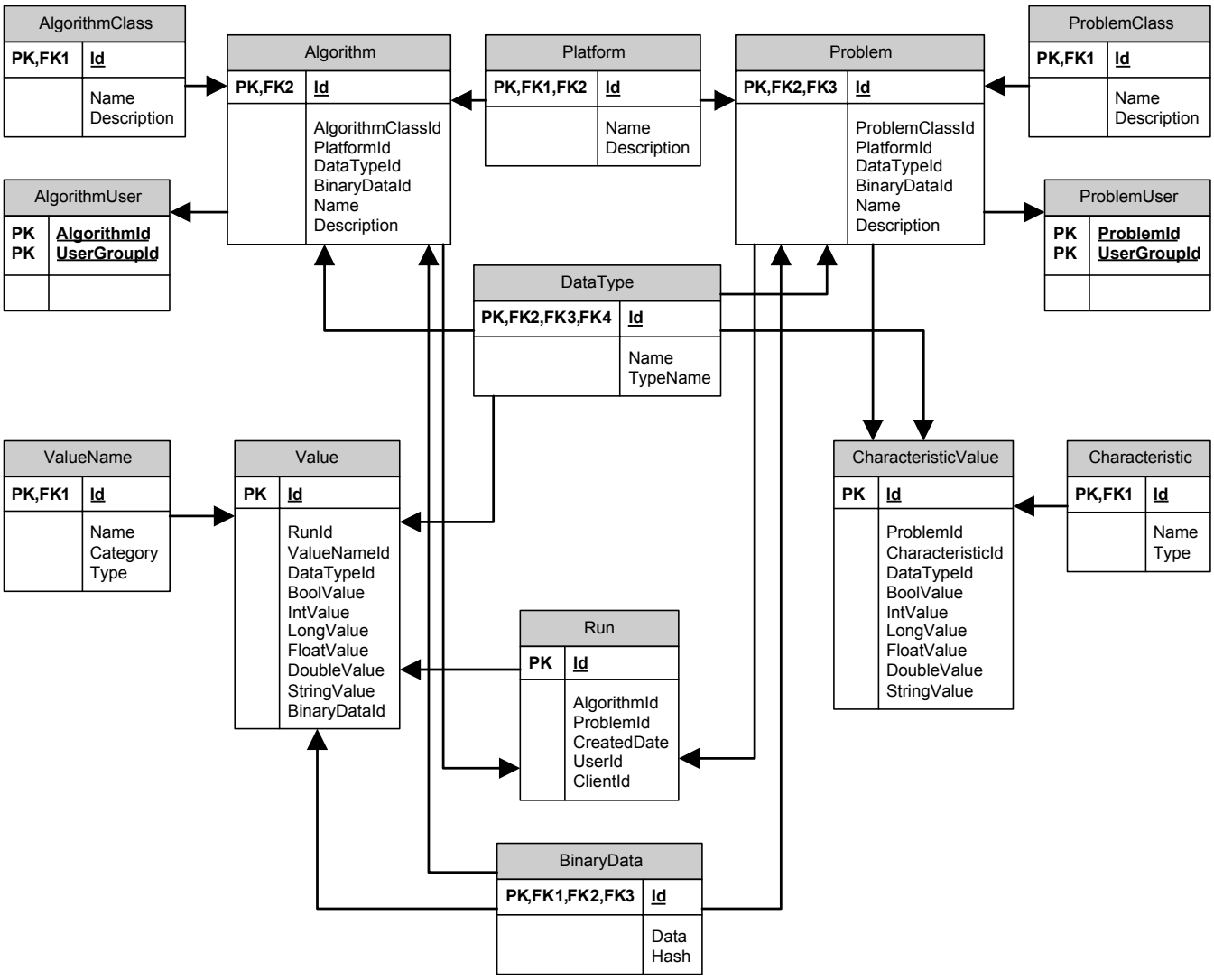
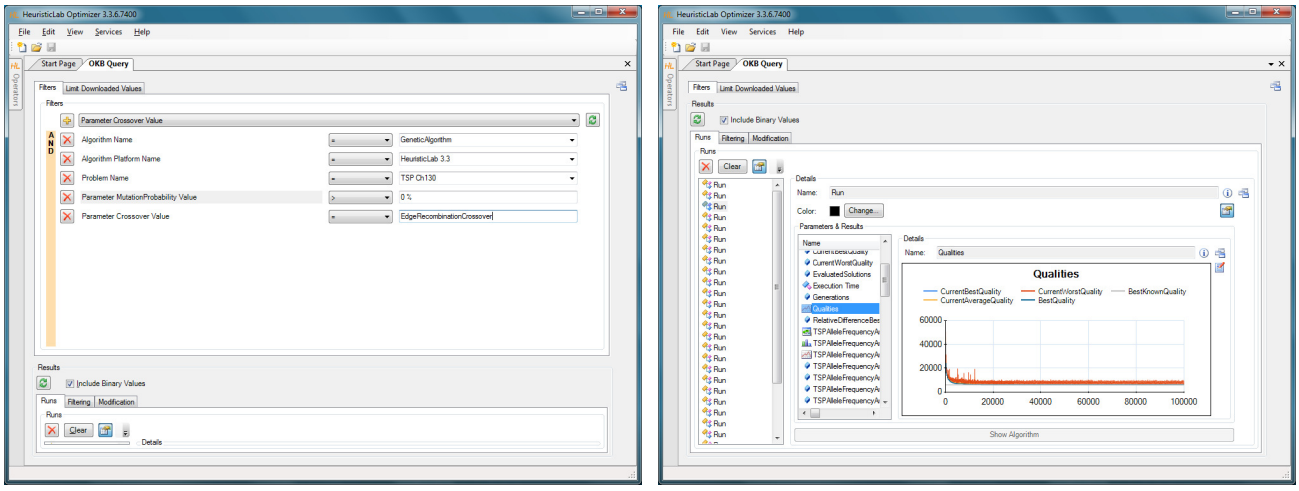


Figure 4: The data model of the OKB

OKB Clients additionally support HeuristicLab's *PercentValue* type which is mapped to a *Float* or .NET's *TimeSpan* data type which is mapped to a *BigInt* data type.

Figure 4 shows the data model of the optimization knowledge base. A `Run` is the container for the parameters and results obtained by executing an algorithm. Therefore a run contains the algorithm and the problem from which it was obtained. The algorithm and the problem contain information about themselves as well as the possibility to store the actual algorithm or problem in the database (referenced with `BinaryDataId`). Every algorithm and problem is categorized and belongs to a certain algorithm and problem class. For every algorithm and problem at minimum one user or user group has to be assigned. This can be used to make certain algorithms or problems only available to a selected group of people. Though the default case is that all data is available to the public, this feature is needed if e.g. confidential problem data is saved in the OKB. Algorithms and problems are also bound to a `Platform` which is the metaheuristic framework they originate from. Additionally, the user and the computer (client) on which the run was executed is stored. Because the OKB stores the runtime of the algorithm, there has to be additional information about the computer to make the execution time interpretable. Therefore every computer that is used to execute and upload runs has to be registered in the OKB. In the process of registering a client, a benchmark algorithm (LINPACK [3]) is executed. A benchmarking algorithm measures the performance of a computer and therefore allows to interpret the execution time of runs in context to the actual performance of the computer.

Every run contains parameter settings and results. This information is stored in the `Value` table. A value is one of the above mentioned allowed data types. The `ValueName` table contains the type of the value as well as it's category. The category is either that the value was obtained from the parameters or the results. The `Name` field contains the name of the parameter. If the value is a binary value, then the value is stored in the `BinaryData` table. In addition to the data a SHA-1 hash function of the data is computed and saved as well. This allows to prohibit duplicate storing of binary



(a) A sample query

(b) Results returned from the query service

Figure 5: HeuristicLab's OKB query client

data. The `DataType` table contains the source data types of the values from the metaheuristic framework it was obtained from. This makes it possible for the host framework to interpret binary data. Because every algorithm and problem is assigned to a metaheuristic framework, these frameworks can additionally make use of the binary data of their algorithms and problems by using the information stored in the `DataType` table.

Because the `Value` and `ValueName` tables allow storing arbitrary data, this structure is extensible and new parameters and results can be added at any time. Of course this generic structure has the disadvantage that any name can be chosen for parameters and interpretation of the actual value is not described. Therefore a common guideline for names and values is needed so that cross-framework comparisons are possible. Additionally, there is a similar data store as the `Value` table that is called `CharacteristicValue`. This table is used to store meta-information to problems and is used for the results obtained by fitness landscape analysis.

3.4 Querying information

The OKB provides a service for querying the information stored in the database. All information that is represented as values can further be used to filter the results returned by a query. Information that can be used for filtering includes for example the algorithm and problem type, parameters of algorithms and result values. The OKB offers a range of filters for the supported data types. These filters can be combined by the boolean operations `AND` and `OR`. This leads to a very flexible query language allowing expressions such as e.g.: Query all genetic algorithm runs performed with HeuristicLab where the problem is the Travelling Salesman Problem of type ch130 with edge recombination crossover and a mutation rate higher then zero percent.

HeuristicLab provides an exemparily implementation of a client for the OKB query service. Figure 5 shows a screenshot of the above query on the left and on the right the results of the query. The results are displayed in HeuristicLab's run view allowing to use HeuristicLab's tools for analysing and exporting the results. Additionally it can be defined which values of the runs should be returned to speed up the download of the information.

4. CONCLUSION AND FUTURE WORK

We presented the optimization knowledge base, an open, extensible and scalable database for storing information about algorithms, parameter settings and problems. It offers a well-defined, standards-based Web service interface, allowing researchers and programs to easily interact with the data stored in the knowledge base. The OKB can be used to support the parameter tuning process, provide an algorithm-problem mapping and give researchers a better understanding of the problem structure and algorithm behavior. The OKB fills the gap between achieving high quality results obtained by manual parameter tuning and generating robust results using e.g. parameter control. An important part for achieving this goal and subject of future work is automatic information collection. Missing parameter configurations and fitness landscape analyses have to be automatically detected and executed. Based on the collected knowledge good parameter configurations for new problem instances can be predicted. Because filling the OKB with information is runtime intensive, machine learning algorithms can be used to predict the results of algorithm executions based on already available results and problem characteristics. Therefore the described system greatly reduces the time to find suitable algorithms and parameters for new problem instances and thus lifts the burden of manual parameter tuning from researchers and experts. Furthermore, because the OKB is an open system, it encourages researchers to publish their experiments and make them available to a broader audience. This promotes transparency and allows researchers to work more efficiently together.

5. ACKNOWLEDGEMENTS

The work described in this paper was done within the Josef Ressel-Centre HEUREKA! for Heuristic Optimization sponsored by the Austrian Research Promotion Agency (FFG). HeuristicLab, Hive and OKB are developed by the Heuristic

and Evolutionary Algorithm Laboratory[3] and can be downloaded from the official HeuristicLab homepage[4]. The software described in this paper is licensed under the GNU General Public License[5].

6. REFERENCES

[1] T. Bäck, A. E. Eiben, and N. A. L. v. d. Vaart. An empirical study on gas "without parameters". In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, PPSN VI, pages 315–324. Springer, 2000.

[2] E. K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, chapter 16, pages 457–474. Kluwer, 2003.

[3] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart. *LINPACK User's Guide*. SIAM, 1979.

[4] A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. Smith. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3:124–141, 1999.

[5] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computation*. Natural Computing Series. Springer-Verlag, 2003.

[6] Y. Hamadi, E. Monfroy, and F. Saubion. *Autonomous Search*. Springer, 2012.

[7] G. R. Harik and F. G. Lobo. A parameter-less genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 258–265. Morgan Kaufmann, 1999.

[8] M. d. l. Maza and B. Tidor. An analysis of selection procedures with particular attention paid to proportional and boltzmann selection. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 124–131. Morgan Kaufmann, 1993.

[9] R. Mercer and J. Sampson. Adaptive search using a reproductive metaplan. *Kybernetes*, 7:215–228, 1978.

[10] R. Myers and E. R. Hancock. Empirical modelling of genetic algorithms. *Evol. Comput.*, 9(4):461–493, 2001.

[11] F. Nadi and A. T. Khader. A parameter-less genetic algorithm with customized crossover and mutation operators. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 901–908. ACM, 2011.

[12] V. Nannen and A. Eiben. A method for parameter calibration and relevance estimation in evolutionary algorithms. *Genetic And Evolutionary Computation Conference*, pages 183–190, 2006.

[13] V. Nannen and A. E. Eiben. Relevance estimation and value calibration of evolutionary algorithm parameters. In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, pages 975–980. Morgan Kaufmann, 2007.

[14] E. Özcan, B. Bilgin, and E. E. Korkmaz. A comprehensive analysis of hyper-heuristics. *Intell. Data Anal.*, 12(1):3–23, 2008.

[15] G. Papa. Parameter-less evolutionary search. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 1133–1134. ACM, 2008.

[16] E. Pitzer and M. Affenzeller. A comprehensive survey on fitness landscape analysis. In *Recent Advances in Intelligent Engineering Systems*, volume 378 of *Studies in Computational Intelligence*, pages 161–191. Springer, 2012.

[17] M. Preuss. Adaptability of algorithms for real-valued optimization. In *Proceedings of the EvoWorkshops 2009 on Applications of Evolutionary Computing*, EvoWorkshops '09, pages 665–674. Springer, 2009.

[18] I. Rechenberg. *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973.

[19] S. K. Smit and A. E. Eiben. Comparing parameter tuning methods for evolutionary algorithms. In *IEEE Congress on Evolutionary Computation*, pages 399–406, 2009.

[20] E.-G. Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8(5):541–564, 2002.

[21] S. Wagner. *Heuristic Optimization Software Systems: Modeling of Heuristic Optimization Algorithms in the HeuristicLab Software Environment*. PhD thesis, Johannes Kepler Universität Linz, 2009.

[22] S. Wagner, G. Kronberger, A. Beham, S. Winkler, and M. Affenzeller. Model driven rapid prototyping of heuristic optimization algorithms. computer aided systems theory. In *EUROCAST 2009*, volume 5717, pages 729–736. Springer, 2009.

[23] Wolpert and Macready. Coevolutionary Free Lunches. *IEEE Transactions on Evolutionary Computation*, 9:721–735, 2005.

[24] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[3] `http://heal.heuristiclab.com/`

[4] `http://dev.heuristiclab.com/`

[5] `http://www.gnu.org/licenses/gpl.txt`