Optimization Knowledge Center

A Decision Support System for Heuristic Optimization

Andreas Beham [1,2]
andreas.beham@fh-ooe.at

Stefan Wagner [1]
stefan.wagner@fh-ooe.at

Michael Affenzeller [1,2]
michael.affenzeller@fh-ooe.at

[1] Heuristic and Evolutionary Algorithms Laboratory
University of Applied Sciences Upper Austria, Softwarepark 11, Hagenberg, Austria

[2] Institute for Formal Models and Verification
Johannes Kepler University Linz, Altenberger Straße 69, Linz, Austria

ABSTRACT

The task of selecting an appropriate algorithm instance for a given optimization problem instance often requires significant experience. Efficient optimization requires a different set of parameters or an entirely different algorithmic approach for some characteristics of problem instances. Obtaining such experience takes significant amount of time and requires an in-depth analysis of the algorithms' performance. In addition to these difficulties, published results only provide a summary, the obtained raw performance data is often not reused later on. In this work we want to give such data more value and more publicity by storing it in a database and reusing it when solving new problem instances. We describe the information that the data should contain in order to maximize reusability. Furthermore, we discuss three use cases that supports optimization experts in their decisions and allows them to perform a manual exploration of the search space using available algorithm instances and the possibility to decide on the starting solutions and thus bias the search in a certain sub-space of the solution space.

Keywords

decision-support-system; knowledge base; heuristic optimization

1. INTRODUCTION

Metaheuristic optimization experts draw their knowledge from personal experience and published research work when applying suitable algorithm instances to new problem instances. Often comparisons are performed against results that have been previously published. However, it can be said that the results sections of most publications are only the tip of an ice-berg and many of the raw data is never reused again. Results that consist only of the final solution quality after x seconds or x function evaluations fall short of a more thorough analysis of the anytime behavior [13]. The final obtained quality distribution accompanied with the standard deviation in tabular form or box-plots are not addressing what would have happened if only half the time was available or double the effort could be made and similar questions. In a benchmark test of algorithm instances the choice of a time limit is often arbitrary and cannot be justified. The goal of heuristic optimization algorithms is to deliver as fast as possible solutions that are as good as possible. This is reflected better in studying the anytime behavior, that is the search quality as a function of the effort. Babai has coined the term *Las Vegas Algorithms* to describe algorithms where the runtime to obtain a solution with a certain quality is a random variable [2]. The performance can thus be described in form of a probability distribution which can be compared quite well with other algorithms, this is discussed further in Section 3.

If it wasn't already difficult enough to obtain a complete picture of an algorithm instance's performance, it is severely more complex when looking at the whole configuration space and alternative algorithm instances that may or must be considered. It is a priori unknown whether the increase of the population size by one hundred leads to better results than a different crossover. In order to tackle these issues meta-optimization techniques have been developed that will perform an optimization in the configuration space of algorithms. F-Race and its corresponding implementation as an R package [12, 22] have shown good potential in identifying suitable algorithm instances. Nevertheless, meta-optimization is a "design-time" tool that allows to identify a range of different configurations that will solve a certain benchmark set very well. In "solve-time" one should choose among a set of alternative instances in order to arrive at good solutions quickly and without the need to search the metaheuristic configuration space. It is thus in the interest of optimization experts to have a collection of well suited algorithm instances that provide a sample of different algorithm configurations.

Another approach in this context has been pursued with fitness-landscape analysis (FLA) [18, 19]. It is commonly believed that the topology of the fitness landscape strongly influences the performance of algorithms. If the topology could be described well enough a discriminating function could be created that would allow predicting the best performing algorithm before applying it. In practice it is however difficult to identify topological properties without sam-

GECCO'16 Companion, July 20–24, 2016, Denver, CO, USA.
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-4323-7/16/07...$15.00
DOI: http://dx.doi.org/10.1145/2908961.2931724

pling from the solution space and evaluating the solutions with respect to similarity and quality. Thus the study of the landscape is not different from the application of an optimization algorithm only that the algorithms are different. This is a valid critique, but targets the application of FLA to algorithm selection only. The study of landscapes itself is necessary basic research in order to derive new theories for optimization. We will look at FLA in this context in Section 2.

In the light of these developments it can be understood that the application of heuristic optimization is a complex task and that decision making and theory forming processes should receive better tool support. Problem understanding and algorithm benchmarking are complex topics that can generate a lot of data that needs to be processed. In solving problems, knowledge can be gained that needs to be presented in a more explicit way and with visual analysis support in order to aid the user in making good algorithmic choices.

The following sections will discuss methods that are relevant in this context and their application in a decision support system that is described in this paper. We want to present three use cases that should be supported, describe their implementation in the Optimization Knowledge Center (OKC), and discuss the methods' application.

2. FITNESS LANDSCAPE ANALYSIS

The goal of performing FLA is to describe the topology of the solution space. Ruggedness, deception, and neutrality are among the landscape properties that are believed to have a high impact on algorithm performance [14]. These properties, are often not "either-or", but can be observed to a certain degree.

Ruggedness is described as the correlation of neighboring points in the search space and the number of local optima [16, 18]. Neutrality describes the presence of contiguous areas where a change in the solution within a certain neighborhood does not result in a change in quality. Deceptiveness may be regarded as a property of a landscape [14], however it has to be stated that it also depends on the search method. An algorithm such as random search is not deceptible. The degree of greediness of an algorithm instance may be correlated with the observed deceptiveness. We suggest to describe the presence of deceptiveness as a decrease of the probability of sampling solutions which are globally optimal over the course of the search.

Several of these properties are correlated with each other, for instance deceptiveness implies ruggedness while ruggedness contradicts neutrality. The distinction between ruggedness and deceptiveness is not entirely clear and both are highly influenced by the neighborhood function. In continuous search spaces, the choice of the sampling distance may highly influence the perceived ruggedness of a landscape [14]. Further correlations arise with respect to other properties of the problem instances. It may be observed, that the number of local optima increases with problem dimension [6]. At the same time it may be concluded, that ruggedness decreases with problem dimension, as measured for example by the autocorrelation length [7]. This can be explained in that many neighborhoods only alter a fixed number of components of a solution, for instance a $2-opt$ move changes 2 edges and leaves $N-2$ edges intact. It follows that with an increase of N the relative change in fitness becomes smaller. At the

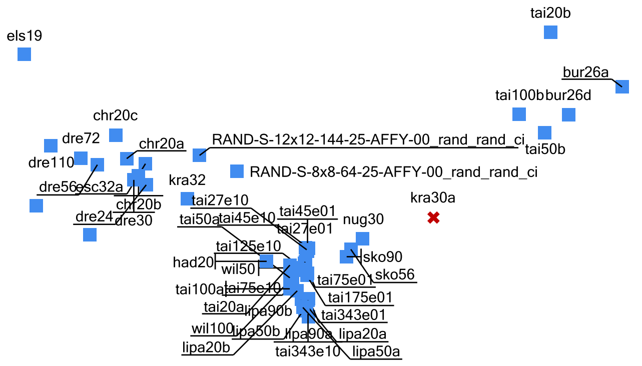

Figure 1: PCA projection of problem instances with respect to their similarity in several characteristic values. Lines indicate which point represents which instance.

same time, the relative change to the solution is smaller and thus the steepness of the gradient should remain identical, but methods that are based on autocorrelation abstract the change in solution to a unit of 1 in a time-series. Thus, it is suggested that autocorrelation length should be studied in combination with problem size in order to draw conclusions on problem hardness [7].

The methods to perform fitness landscape analysis often involve exploratory search algorithms. For example, in real-valued search spaces one could sample three points along a linear trajectory and evaluate the goodness of fit of a quadratic model on the quality of those solutions [15]. There are also several types of walks, e.g. a random walk that will randomly sample solutions from the neighborhood advancing without bias. An analysis of the quality trajectory of this random walk could reveal some properties of the fitness landscape. Additional types of walks include adaptive walks and up/down walks. The advantage of such walks is that they are applicable to almost any problem unlike other analysis methods that are meaningful only in a smaller context of problems. There are still many open questions in the theory of fitness landscapes which remains an active field for studies. Several good overviews exist [18, 19].

When multiple such characteristics are collected for problem instances one can identify similarities among them and create map-like plots by dimensionality reduction techniques (e.g. principal components analysis (PCA) [21], t-stochastic neighbor embedding (t-SNE) [25], self-organizing maps (SOM) [10], or multi-dimensional scaling (MDS) [11]) or even supervised techniques such as neighborhood components analysis (NCA) [8] if class labels can be assigned to the data. In the present software system we have implemented PCA, MDS, and SOM which provide a similar, but nonetheless different picture. It may be useful for the expert to not have only one projection method, but to compare different projections in order to draw conclusions on the proximity of problem instances. In addition, using the OKC, experts may change the characteristics that are actually used to create the maps in the software system. Thus the problem instance space can be explored with any subset of characteristics with immediate results on the projection. Figure 1 shows such a map for instances of the quadratic assignment problem. Missing values in the characteristics,

Algorithm 1 ECDF calculation from convergence graphs.

```
1: procedure GetEcdf(↓ $cgraphs[][]$, ↓ $target$, ↑ $ecdf$)
2:     Calculate(↓ $cgraphs$, ↓ $target$, ↑ $h$, ↑ $m$)
3:     Sort($h$); Sort($m$)      ▷ sorted array by key ($\hat{=}$ effort)
4:     $total \leftarrow 0$
5:     $sum \leftarrow$ length($cgraphs$)      ▷ number of runs
6:     $j \leftarrow 1$
7:     $ecdf \leftarrow$ graph()
8:     for $i \leftarrow 1$ to length($h$) do
9:         $total \leftarrow total + h[i]$.Value
10:        while $j \leq$ length($m$) $\wedge$ $m[j]$.Key $<$ $h[i]$.Key do
11:            $sum \leftarrow sum - m[j]$.Value
12:            $j \leftarrow j + 1$
13:        end while
14:        $ecdf$.Add($h[i]$.Key, $\frac{total}{sum}$)
15:    end for
16:    return $ecdf$
17: end procedure
```

i.e. because a certain characteristic has not been calculated for a given problem instance, are replaced with the median value of that characteristic.

Problem instance similarity is an important first task in the understanding of a new problem instance. In obtaining several characteristics it should be possible to draw from past experience by examining the performance of algorithm instances that have been applied to similar problems.

3. ALGORITHM PERFORMANCE

It is important to measure algorithm performance in a way that it remains useful after the publication of the research work in which it is summarized. [3] describes a total of 18 performance measures among them success ratio, bootstrapped best, mean best (MBST), run-length distribution (RLD) and many more.

In order to maximize reusability of results, final quality values do not adequately represent performance for use in expert-systems. These results always depend on the maximum amount of effort a certain algorithm instance was given to compute them. So, quality-based measures lack information on the convergence speed and do not depict the time that is required to obtain them. On the other hand, measures on the convergence speed such as efficiency rates, quality-effort relationship, convergence velocity, and others do not capture the quality that was achieved. It becomes obvious that a single value that describes the performance of an algorithm instance applied to a certain problem instance is hardly feasible.

Among the better choices for performance measures are run-length distributions (RLD). [9] describes metaheuristic algorithm performance as a random variable with a certain distribution. With a given effort there is a probability of achieving a solution with a certain quality. For a range of different quality-targets these RLDs are computed and the expected run time (ERT) can be computed [9, 1]. The empirical cumulative distribution function (ECDF) enables a graphical comparison between algorithm instances, see Figure 3. The calculation base for ECDFs are convergence graphs. These are monotonic functions that store the best quality along with the amount of effort that has been invested in achieving it. In order to save space when storing

Algorithm 2 Aggregating the hits and misses in convergence graph data of multiple runs under the assumption of minimizing the objective function.

```
18: procedure Calculate(↓ $cgraphs[][]$, ↓ $target$, ↑ $hits$,
        ↑ $misses$)
19:     $hits \leftarrow$ dictionary()
20:     $misses \leftarrow$ dictionary()
21:     for $i \leftarrow 1$ to length($cgraphs$) do
22:         $missed \leftarrow true$
23:         $last \leftarrow$ length($cgraphs[i]$)
24:         for $v \leftarrow 1$ to $last$ do
25:             if $cgraphs[i][v]$.Quality $\leq target$ then
26:                 $e \leftarrow cgraphs[i][v]$.Effort
27:                 $hits[e] = hits[e] + 1$
28:                 $missed \leftarrow false$
29:                 break
30:             end if
31:         end for
32:         if $missed$ then
33:             $le \leftarrow cgraphs[i][last]$.Effort
34:             $misses[le] = misses[le] + 1$
35:         end if
36:     end for
37:     return $hits, misses$
38: end procedure
```

these graphs only those points should be provided where an improvement is made including an additional point with the final quality and the final effort. Recording the maximum effort spent is necessary as we want to take into account runs with different amount of maximum effort.

In the described software system it is possible to study ECDF comparisons by grouping the runs according to certain parameters and problem instances. If no grouping is performed the performance of all algorithm instances is combined to see how the set of available instances may tackle a single or multiple problem instances. If runs are grouped by algorithm name, the performance of each instance is displayed as can be seen in Figure 3. Furthermore, the optimization knowledge center allows defining a range of target values as relative deviations from the best-known quality. Additionally, expected-runtimes are printed out to tables in order to copy and paste the results to other applications.

Our method for computing ECDFs from convergence graphs is described in Algorithms 1 and 2. In these pseudocode descriptions down arrows represent inputs and up arrows denote outputs of the procedures.

Contrary, to controlled experiments with pre-defined parameters the OKB (see Section 4.1) may contain a large set of runs from different experiments with varying maximum effort. But, ECDF describes the cumulative ratio of successful to total runs. In our case, total runs must be seen as a function of the effort. Therefore, we have taken into account that there are runs that are shorter than others and also unsuccessful (called a "miss") in reaching the target. Consider the scenario where a previous experiment is repeated, but this time with a higher maximum effort. It would not be correct to count all runs to the total as we do not know if a miss would turn into a hit given more runtime. We thus compute the cumulative success probability at effort x as the ratio of successful runs at effort x to all runs where at least effort x was observed. In Algorithm 1 variables *total* and *sum* represent the successful and total runs at a certain effort. In Algorithm 2 the convergence graph data, which is a simple list of tuples, is converted into a dictionary of hits and misses. In this dictionary, effort is the key, while the number of times a hit or miss was observed at a certain effort is the value. To the best of our knowledge no such algorithm has been previously published.

In calculating the ERT with equation (1) (see [1]) determining the success probability p_s as the ratio between successful and total number of runs also requires adaptations. Unsuccessful runs that are conducted with too little effort cannot be considered as actual unsuccessful attempts. For example, when a certain algorithm instance requires an observed effort of about 10000 ± 1000 then a run with a maximum effort of e.g. 2000 can hardly be considered an unsuccessful attempt. In our pruning strategy we have chosen to remove all runs where the run-length is less than two standard deviations from the mean of all successful runs. Among the remaining set of runs R we then compute the average run-length of all successful runs $\mathbb{E}(T^s)$ and the success probability $p_s = \frac{|\{r \in R | r \text{ is successful}\}|}{|R|}$ in order to obtain the expected runtime according to equation (1).

$$\text{ERT} = \frac{\mathbb{E}(T^s)}{p_s} \quad (1)$$

While RLDs and ERT are well suited to describe algorithm performance the difficulty lies in defining "effort". Two options have been used in the past, the execution time on the one hand and the number of calls to the fitness function (function evaluations or FEs) on the other hand. Both options have their own advantages and disadvantages.

3.1 Execution Time

The amount of wall clock time that has passed since the algorithm was started is a directly relevant measure for anyone seeking to apply a certain algorithm instance. Often time constraints based on wall-clock time are easier to define and in real-world applications there may be hard limits regarding the maximum time until a good answer is found. However, there are a number of disadvantages which the reader might have already on her mind:

- Comparing execution time requires a reference machine
- Execution time is implementation specific (e.g. slow implementation of an algorithm, different platform)
- Parallelization affects execution time
- Examining algorithm results while the algorithm is running (for instance console prints or GUI updates) may affect execution time

Often the wall clock performance of an algorithm depends on how well it is implemented. An implementation may be changed with respect to run time later on and while the results based on quality remain the same it is now several times faster. This requires to retest the performance repeatedly.

3.2 Function Evaluations

Effort depends to a large part on the complexity and runtime of evaluating the fitness function. Thus the number of times this function is called is a good indicator of algorithm performance. However, for lower dimensional problems or for algorithms that make intensive computations in-between function evaluations (e.g. learning meta-models) this measurement of effort is not correct. Additionally, for some problems the fitness can be computed faster through so called delta-evaluations. In these problems the fitness of a solution needs to be re-evaluated only for those components that changed. For instance, in the quadratic assignment problem, a full solution evaluation has complexity of $O(N^2)$, but swap moves can be computed with a complexity of $O(N)$ and even $O(1)$ [23] if the quality of previous moves is retained between iterations. Delta-evaluations typically exist for white-box problems and used in fast local search algorithms in order to compute the qualities in a neighborhood efficiently. For black-box problems where the actual fitness function is not known or too complex (i.e. simulation) delta-evaluation cannot be used. A trade-off is to compute delta-evaluations in terms of full solution equivalents.

Summarizing, execution time is too closely linked with the hardware environment that is used to perform the tests which is changing rather fast. We think that function evaluation (FE) is the more promising description of effort. It is in our interests that we obtain measurements which are useful for a longer period of time. Execution time is invalidated quickly by the introduction of new hardware, change to the implementation detail of an algorithm instance, or the change to a different platform (e.g. .Net on Windows vs Mono on Linux).

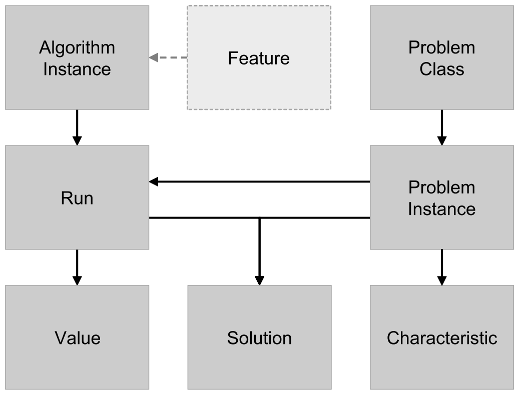

Figure 3: An abstracted version of the OKB data model, still unrealized parts are highlighted. The arrows model dependencies similar to foreign-key relationships. Here Run is an entity that depends on an algorithm and a problem instance.

4. EXPERT KNOWLEDGE

Experts accumulate knowledge over time in that they remember algorithm configurations that have been repeatedly proven to be successful on some problem instances. Thus, an expert has a handful of configurations at hand that she would test on a new problem instance. In the described system it should be possible to store those algorithm instances in a database and thus share them with other experts. By performing runs and storing them in the database as well we can visualize and compare the performance of algorithm instances and thus make some part of the knowledge of the

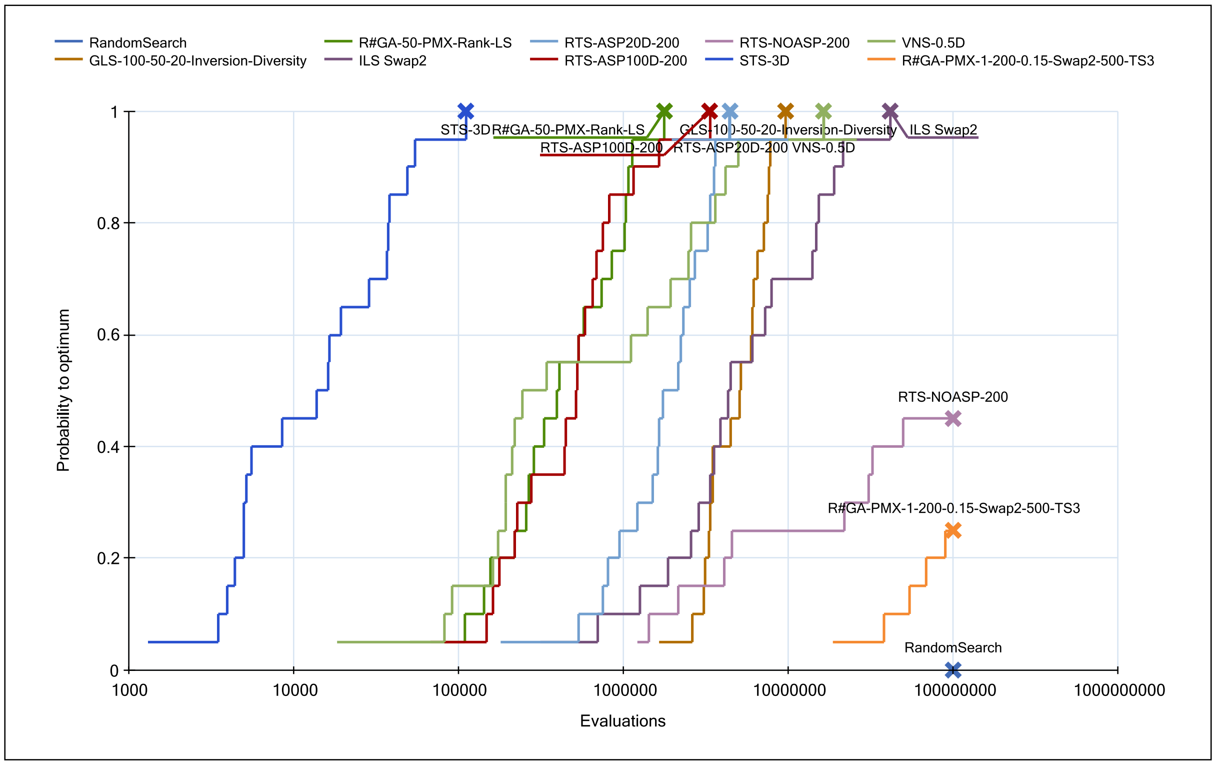

Figure 2: Comparison of several empirical cumulative distribution functions (ECDF) for algorithm instances seeking to find the optimal solution of the QAPLIB's esc32a instance.

experts more explicit. Finally, experts usually have a mental framework that they apply to optimization problems and which they often try to implement in form of a new algorithm. However, such a mental framework is highly adaptive to the current situation and the description of self-adaptive algorithms is both difficult and highly empirical. We want to describe a manual approach where experts are able to apply their mental model by repeatedly executing algorithm instances in the solution space. In this process experts have the possibility of reusing already found solutions as starting points. We think that this may be a helpful first step to implementing that model in an algorithm. Additionally, with such results stored in the database, it may become possible to detect that it is beneficial to apply algorithm instance B with the resulting solution of algorithm instance A using machine learning techniques.

4.1 Optimization Knowledge Base

The optimization knowledge base (OKB) is a database and a set of web service methods to query and manipulate the data [20]. An abstract view on the OKB data model is depicted in Figure 3. The database is designed to store algorithm and problem instances as well as the runs that have been observed and the solutions that have been achieved. Each run is split up into several *values* which may either be a parameter or a result. Values that cannot be stored as integral data types have to be serialized to binary format. In addition algorithm and problem instances and solutions are also stored in binary format. This is not optimal as binary data can be read only by the software that uploaded the information. Nevertheless, complex types are difficult to describe in a relational database. The use of document-oriented databases may be explored in the future.

Currently, the OKB does not store inference rules or recommendation models and these have to be calculated on the client. This information shall be included in the OKB at a later date.

4.2 Use Cases

We want to describe three use cases that we think are important for experts in heuristic optimization. In general we assume that the expert approaches the knowledge center with a new and previously unseen problem instance.

1. Understanding of the problem instance
2. Solving the problem instance
3. Documenting and learning from the process

These use cases are overlapping to some degree as the understanding of a problem instance increases when performing several attempts to solve it. When analyzing the results of a certain solver with known properties, for instance population-based exploratory search or randomized local search, the user can formulate hypothesis on the structure of the problem instance. Most algorithm instances are created with the idea that it can compute a solution efficiently for some fitness landscape model. If a better performance can be observed with this solver than with others it can be assumed that this landscape has the respective properties. We think that methods such as fitness landscape analysis should be combined with performance data to create better pictures on problem instance similarity.

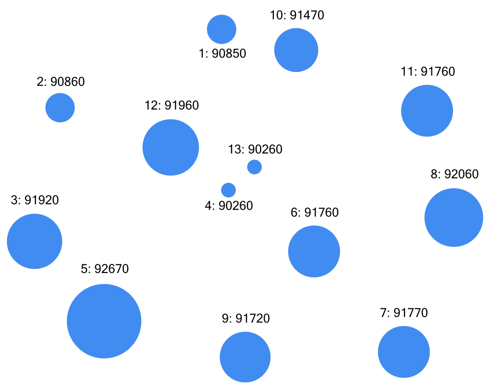

Figure 4: Some obtained solutions to the kra30a QAPLIB instance layed out according to multi-dimensional scaling of the distance matrix in solution space. Note that this does not represent a LON, the text represents the solution id followed by the quality. It may be a helpful tool when deciding if a certain algorithm instance should be started from an existing solution.

Understanding

Users would probably like to identify similar problem instances and see whether there are clusters or neighboring instances. In the optimization knowledge center we support several projection methods to visualize these clusters such as PCA, MDS, and SOM. An important part is the selection of the characteristics that are used in the projection method. This may result in entirely different maps. Highly experienced users probably want the possibility to choose freely among the characteristics and explore alternate pictures, while less experienced users would certainly like a reasonable default. Figure 1 displays a map of problem instances of the quadratic assignment problem (QAP) from various benchmark libraries that are projected using PCA. The data basis has been calculated from properties of the QAP (problem dimension and coefficient of variation, symmetry, sparsity, skewness, and disparity of both the flow and the distance matrix). Such simple properties are fast to calculate for a new QAP instance.

Another level of understanding can be obtained in analyzing the solutions from a certain problem instance. Networks may be formed among solutions, such as for instance local optima networks (LON) [17] where the landscape is represented as a graph. Nodes in a LON are local optima and edge weights represent transition probabilities between their associated basins of attraction. In visualizing a LON nodes are displayed in different sizes in relation associated basin's size, while the node's color is related to its quality. A visual inspection of such a network may easily reveal local optima that are highly attractive, but of worse quality. The disadvantage in analyzing LONs is that they are very expensive to compute and require a thorough exploration of the solution space. In the OKC we have thus implemented a visualization of solutions regarding their similarity only. Figure 4 shows identified solutions to the kra30a problem instance where the similarity matrix was projected to two dimensions using MDS.

Solving

In solving problem instances experts seek to identify good solutions given a randomly initialized starting solution. However, they may also want to reuse existing solutions in that they aim to search certain parts of the solution space more thoroughly. Seeding of algorithms is a way for the expert to have more control over the exploration or exploitation of the search space. Experts may choose certain solutions as starting points for the algorithm instances. There are two kinds of seeding strategies:

- Cloning
- Sampling

When *cloning* the solutions are directly included in the starting configuration of the algorithm as they are. This is a useful strategy for single-solution metaheuristics that aim on intensifying the search space around a certain solution. When *sampling* only some components of the seeded solutions are used and solutions may either be a mix of several seeded solutions or they may contain random components. Mixing occurs in a process that is similar to crossover. Such a strategy may be more suitable for population-based metaheuristics where components of seeding solutions are introduced into the starting population, but could also be used for single-solution metaheuristics.

In the presented software system, experts have the possibility to draw on a set of algorithm instances that they may apply. As has been discussed, the whole parameter space of metaheuristic algorithms is quite large leading to an unsupportable large amount of algorithm instances. The idea is that algorithm designers provide only a limited "hand-selected" set of algorithm instances. Each of these is then treated as an independent entity in the tests. Potentially, these instances should provide some meaningful, but basic self-adaption such as changing parameters that are influenced by the size of the solution space. For instance, a tabu search approach may require a longer tabu list when the solution space becomes larger.

The OKC software system also provides recommendation algorithms to experts that can be used to propose a ranking of algorithm instances for a new problem instance. The expected runtime of the ranked algorithm instances is also estimated giving a hint to experts on the effort that should be spent. Recommendation algorithms, such as k-nearest neighbor, can be tested by leave one out crossvalidation on the results of previously benchmarked problem instances.

Documenting and Learning

The presented software system allows uploading FLA characteristics, runs and solutions to the OKB. Uploading this information may improve algorithm instance recommendation in the future when a similar problem instance should be solved.

It is not yet possible to store or retrieve learning models in the OKB. Such information may be interesting as it would make the system more attractive for users with less expertise. Similar to how search engines on the web crawl sites to update their index, a server process alongside the OKB could periodically learn new and improved recommendation models.

5. OUTLOOK

In creating and discussing the use of such an expert system many topics arise that may become important for other researchers and algorithm designers. Here we want to select two concrete topics and present some thoughts as outlook to this paper.

5.1 Design of Heuristic Algorithms Instances

Algorithm instances that shall be used within the optimization knowledge center should be designed in order to meet a few important requirements:

- Stopping criteria that algorithm instances must provide
  - Maximum number of function evaluations
  - Maximum execution time
  - (optional) Target quality
- Results that algorithm instances must provide
  - Best found quality and solution
  - Convergence graph
  - (optional) Local optima

The optimization knowledge center does not know the specific algorithm instances. It assumes that the common properties described above are defined. If, for example, maximum evaluation is not available, the software system cannot provide support to limit the runtime of the algorithm instance. We have implemented a workaround such that the OKC monitors the EvaluatedSolutions result and stops the algorithm instance when it has surpassed the defined budget.

5.2 Feature Model for Heuristic Algorithms

We think that it is important to tag algorithm instances with certain features. This allows to know more about the algorithm designer's intentions and what can be expected from an algorithm instance. Following could be a list of features that may be included with one or the other algorithm instances. This list is highly influenced by the categories of search algorithms as described in [4, 5, 24]:

- Population-based vs single-solution search
- Memory-usage vs memory-less methods
- Single- vs multi-neighborhood
- Trajectory vs discontinuous search
- Deterministic vs stochastic search
- Iterative improvement
- Greedy construction
- Converging search

Of course, this list must be seen as only a starting point to enabling a more thorough analyis and study of the behavior of algorithm instances. Still, it already raises some questions regarding the value of these features. An algorithm instance may both feature a trajectory and a discontinous search behavior. For example, in variable neighborhood search, the hill climber typically follows a trajectory to a nearby local optimum while the perturbation with multiple neighborhoods can be seen as a discontinuous approach. Thus features may be present in an algorithm instance to a certain degree. One could allow an additional parameter to rate the presence of a feature for a given algorithm instance in form of a percentage. However, such a rating would be difficult to pinpoint to exact terms and would complicate the description. Additionally, different algorithm designers would judge the presence of a feature differently even if they would design the same algorithm instance. Another possibility would be to create two categories one to describe major features and another to describe minor features and not allow an overlap in these categories.

The goal of having such a feature model would be to use it in learning about the performance. If there are correlations in the data with respect to the presence or absence of these features and the performance characteristics one could draw additional conclusions for a certain problem class and give new ideas for the creation of new algorithm instances. Without such a feature model these conclusions are difficult to make as algorithm instances are otherwise treated mostly as black boxes.

6. CONCLUSIONS

We have described a system that is aimed at heuristic optimization experts in order to manually apply certain solvers to a new problem instance. We described three use cases that are interesting for experts. In applying fitness landscape analysis and comparing problem instances to their new ones they may obtain information on a priori suitable algorithm instances to solve them. In applying multiple algorithm instances to solve the problem instances solutions are recorded and analyzed regarding quality and similarity to gain additional knowledge. Through the ability to use seeding the expert can intensify the search in some regions of the solution space or explore a certain sub-space of the solution space with a higher probability. Finally, by uploading the information back to the OKB experts can continue at another time as well as provide additional information that is helpful in applying optimization to further problem instances.

7. ACKNOWLEDGMENTS

The work described in this paper was done within the COMET Project Heuristic Optimization in Production and Logistics (HOPL), #843532 funded by the Austrian Research Promotion Agency (FFG).

8. REFERENCES

[1] A. Auger and N. Hansen. Performance evaluation of an advanced local search evolutionary algorithm. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC)*, volume 2, pages 1777–1784, Sept 2005.

[2] L. Babai. Monte-carlo algorithms in graph isomorphism testing. Technical Report 79-10, Université tde Montréal, 1979.

[3] T. Bartz-Beielstein. *Experimental Research in Evolutionary Computation: The New Experimentalism.* Springer, 2006.

[4] M. Birattari, L. Paquete, T. Stützle, and K. Varrentrapp. Classification of metaheuristics and

design of experiments for the analysis of components. Tech. Rep. AIDA-01-05, Intellektik, Darmstadt University of Technology, 2001.

[5] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, September 2003.

[6] F. Chicano, F. Daolio, G. Ochoa, S. Vérel, M. Tomassini, and E. Alba. Local optima networks, landscape autocorrelation and heuristic search performance. In *Parallel Problem Solving from Nature-PPSN XII*, pages 337–347. Springer, 2012.

[7] F. Chicano, G. Luque, and E. Alba. Autocorrelation measures for the quadratic assignment problem. *Applied Mathematics Letters*, 25(4):698–705, 2012.

[8] J. Goldberger, G. E. Hinton, S. T. Roweis, and R. Salakhutdinov. Neighbourhood components analysis. In *Advances in neural information processing systems*, pages 513–520, 2004.

[9] H. H. Hoos and T. StÃijtzle. Evaluating las vegas algorithms - pitfalls and remedies. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 238–245, San Francisco, CA, 1998. Morgan Kaufmann.

[10] T. Kohonen and P. Somervuo. Self-organizing maps of symbol strings. *Neurocomputing*, 21(1):19–30, 1998.

[11] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[12] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011.

[13] M. López-Ibáñez, T. Liao, and T. Stützle. *Parallel Problem Solving from Nature - PPSN XII: 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part I*, chapter On the Anytime Behavior of IPOP-CMA-ES, pages 357–366. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[14] K. M. Malan and A. P. Engelbrecht. Fitness landscape analysis for metaheuristic performance prediction. In *Recent advances in the theory and application of fitness landscapes*, pages 103–132. Springer, 2014.

[15] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, and G. Rudolph. Exploratory landscape analysis. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 829–836, New York, NY, USA, 2011. ACM.

[16] P. Merz and B. Freisleben. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *Evolutionary Computation, IEEE Transactions on*, 4(4):337–352, 2000.

[17] G. Ochoa, S. Verel, F. Daolio, and M. Tomassini. Local optima networks: A new model of combinatorial fitness landscapes. In *Recent Advances in the Theory and Application of Fitness Landscapes*. Springer-Verlag Berlin Heidelberg, 2014.

[18] E. Pitzer and M. Affenzeller. *Recent Advances in Intelligent Engineering Systems*, chapter A Comprehensive Survey on Fitness Landscape Analysis, pages 161–191. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[19] H. Richter and A. Engelbrecht. *Recent advances in the theory and application of fitness landscapes*. Springer, 2014.

[20] A. Scheibenpflug, S. Wagner, E. Pitzer, and M. Affenzeller. Optimization knowledge base: An open database for algorithm and problem characteristics and optimization results. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '12, pages 141–148, New York, NY, USA, 2012. ACM.

[21] K. Smith-Miles, D. Baatar, B. Wreford, and R. Lewis. Towards objective measures of algorithm performance across instance space. *Computers & Operations Research*, 45:12–24, 2014.

[22] T. Stützle and M. López-Ibáñez. Automatic (offline) configuration of algorithms. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '13 Companion, pages 893–918, New York, NY, USA, 2013. ACM.

[23] E. D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17:443–455, 1991.

[24] E.-G. Talbi. *Metaheuristics: From Design to Implementation*. John Wiley & Sons, 2009.

[25] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.