Towards automated optimization

Andreas Junghanns[a,*], Dirk Petzoldt[b], Jonas Dageförde[c], Marcus Meyer[a]

[a] *DaimlerChrysler, Research and Technology, Alt-Moabit 96a, HPC U118, 10559 Berlin, Germany*
[b] *SAP Deutschland AG & Co. KG, Neurottstrasse 15, 69190 Walldorf, Germany*
[c] *DaimlerChrysler, 800 Chrysler Drive, Auburn Hills, MI 48326, USA*

Abstract

The optimization literature has traditionally concentrated more on the power of optimization algorithms and not so much on their accessibility. In this paper we develop a methodology to automate optimization and thus to ease access to the multitude of optimization methods and their specific parameter settings. This will help engineers to solve more of their non-trivial optimization problems without (expensive) mathematically skilled help.

We show a prototypical system with this 'scheduling' functionality: it analyses a user-specified optimization problem, finds its optimizer-relevant properties, selects the 'best' of the available methods, sends the problem to the selected optimizer, and starts it with the appropriate parameters.

Preliminary, yet promising, results show that the basic idea works in principle and that future research in this area has high potential.

Keywords: Automated optimization; Optimization problem classification; Optimization scheduling

1. Introduction

1.1. The optimization process

Solving optimization problems is considered as something like an art: it requires considerable knowledge, skill and intuition about the domain, the methods, and the tools in order to find a satisfactory solution to the original problem under the resource constraints imposed by the real world environment.

Theoretically, optimization is a sequential process of abstracting, modeling and running optimization software. In practice, however, information gained later in the process may be relevant to earlier decisions. For instance, inspecting the performance, e.g. the convergence rate of an optimizer, might lead to the change of a certain parameter value of the algorithm or even the selection of a different method. In some cases, it might even lead to remodeling of the problem, because modeling an optimization problem allows for different abstraction levels, modeling styles, modeling languages, etc., as depicted in Fig. 1.

When considering the research efforts of the past decades, much effort has gone into the advancement of optimization algorithms. However, every time a new powerful alternative becomes available, it gets harder to choose from the increasing array of methods and tools, and harder still to adequately operate them.

1.2. Application scenarios

When we started our research efforts, we had the following three application scenarios in mind:

1. **Decision support for the engineer** We would like to make optimization a standard application 'at the fingertips' of the engineer. The necessary knowledge required for the user should be as minimal as possible, even at the expense of losing some efficiency. Ideally, the user is only responsible for the problem description and, as long as this description is syntactically correct and a solution exists, a solution is presented. The goal is to describe only **what** and not **how** to solve it.
2. **Automatically composed optimization problems** Large knowledge bases, such as product documentation systems, are filled with information that can be used to build optimization problems automatically. For example, one could use customer preferences to

* Corresponding author. Tel.: + 49(30) 399 82 478; Fax: +49(0) 7 11 3052 111 819; E-mail: andreas.junghanns@dcx.com

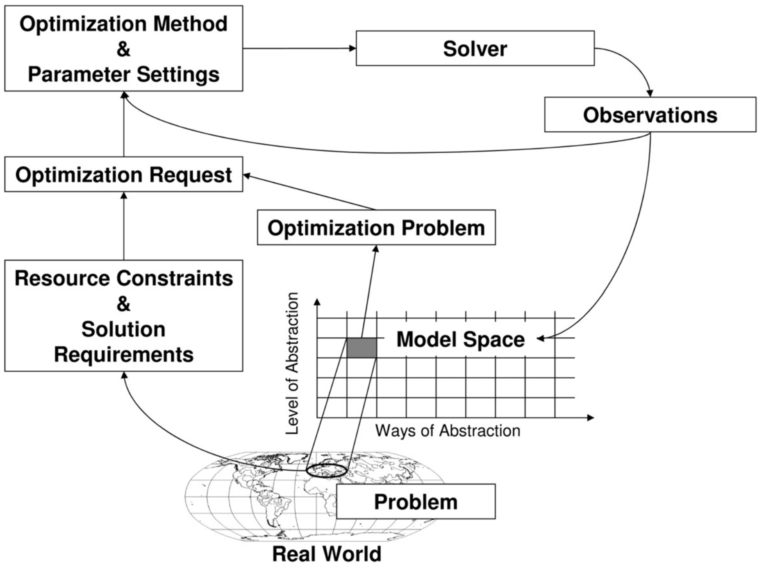

Fig. 1. The optimization process.

exclude certain product alternatives, restrict parameters and adapt the objective function in a product configuration tool. The resulting optimization problem can vary widely, even if it is created from the same knowledge base.

3. **Optimizing black-box functions** Analyzing engineering design alternatives today means using complex, expensive special purpose functions, such as simulations, as part of the objective. It is in general impossible to analyze black-box functions a priori, especially when they are applied to different kinds of user data. This has led to a recent increase of interest concerning so-called direct optimization methods, which use only the value of the objective function itself, and do not need any additional information, i.e. the Jacobian or Hessian of the objective function. Examples are the algorithms [1,2,3,4], which differ in the way they build an internal model of the objective function. For further details the reader is referred to [1,5]; a comparison of different direct methods in the industrial context can be found in [6].

Of the three scenarios that we have in mind, the last is the most advanced. Here many of the optimizers implementing these algorithms will require additional features to allow sharing of results produced by the previous optimization run(s), which is vital for efficiency when considering expensive black-box functions.

2. The BBO project

We started developing a proof-of-concept software tool in 2002, called 'Black-Box-Optimizer' (BBO). Since this paper leaves little room to describe all technical details, the reader is referred to [7,8] for further details about the BBO project.

Figure 2 illustrates the architecture of the BBO tool: a central manager module receives an optimization request, i.e. an optimization problem plus resource constraints and solution requirements. The scheduler uses static classification information and a knowledge base, and possibly existing dynamic classification information, in order to produce a script (a schedule) that can be executed by the manager.

Note that the grey boxes contain all the functionality that is currently performed by a human optimizer using standard optimization modeling and solving environments (the white boxes).

In the following we briefly describe the functionality of the individual modules as seen in Fig. 2 and give some details regarding their implementation.

2.1. Classification module

Classification is the determination of properties of an object. These properties are then used to reason about the next action to be taken. Here we need to classify three kinds of objects: problem descriptions, optimizers, and status information of optimization runs.

Problem descriptions are usually classified using symbolic algebra tools, such as Mathematica or Maple. These tools can determine properties like 'linear' or 'non-linear'. However, the user can also supply classification information as annotation to the problem

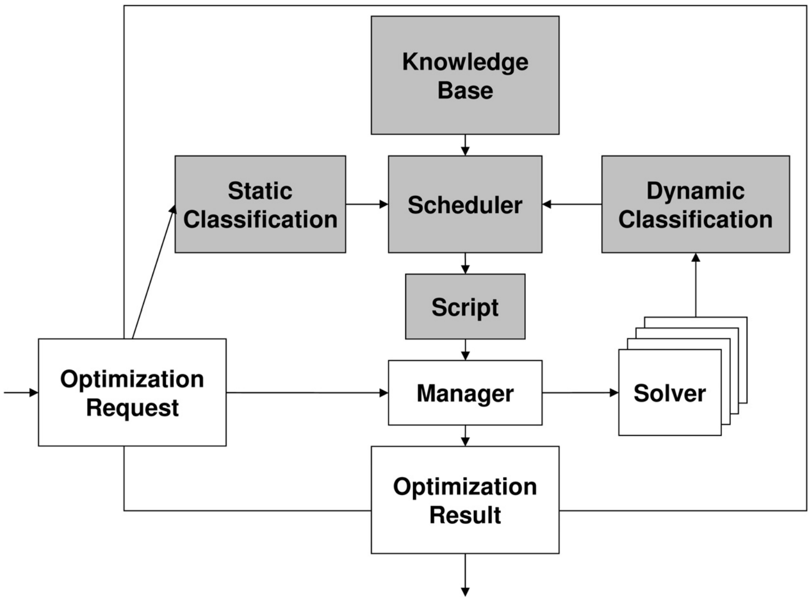

Fig. 2. Architecture of the BBO tool.

description. This will short-cut the call to the classification algorithm determining this property.

Properties of the optimization methods are currently supplied by the user implementing the BBO optimizer APIs. Future optimizer APIs should incorporate inquiry features that allow determination of important properties automatically.

2.2. Scheduling module

Scheduling is matching a specific situation to a specific set of actions. For example, after analyzing an optimization problem description, one can use the properties of the problem to determine which optimization method to use.

Coarse and heuristic scheduling

When scheduling an optimization problem, the first task is to determine the class of optimization problem, e.g. is the problem continously linear or nonlinear constrained? We call this step 'coarse scheduling'. Coarse scheduling will then establish a set of feasible optimizers – those that are able to solve problems of the given class under the resource constraints and solution requirements given in the optimization request.

It is the task of 'heuristic scheduling' to determine the 'best' method among the set of feasible optimizers and its appropriate parameter settings. The term 'heuristic' was chosen due to the nature of the knowledge that is necessary for this step.

Static and dynamic scheduling

We consider an orthogonal split among the scheduling tasks depending on whether it considers information gathered from optimizer status messages or not. While 'static scheduling' relies solely on what can be determined before an optimization is started, 'dynamic scheduling' uses performance feedback from the optimization process in order to adapt algorithm parameters or even switch optimization methods.

2.3. Current status of the BBO project

Currently, the BBO tool can read optimization problems defined in XML or OL. OL is a superset of AMPL [9] where we added scripting ability to

- control the optimization procedure;
- model algorithm in- and output, and
- to include runtime state and meta information.

BBO comes with an extensive test framework, is connected to NEOS [10] to make the NEOS optimization methods available, has access to internal optimization algorithms, e.g. the Nelder-Mead Simplex Method [11], uses PROTÉGÉ [12] as a knowledge base, ALGERNON-J [13] for reasoning capabilies, and Mathematica to answer classification requests using symbolic algebra.

The central scheduling component analyzes incoming optimization requests and produces a solution strategy in the form of an OL script. The script contains the optimizer selection and setup. BBO is currently using a number of classifiers:

- For the objective and each constraint it analyses the contained variables, if external method calls exist, its 'complexity', if there are explicit discontinuities, if differentiable, if linear, if derivable, and the sparsity pattern.
- For constraints it checks equality or inequality and total number of constraints.
- Finally, for decision variables it analyzes the total number of variables, simple bound preprocessing, tighter upper and lower bound, and if a variable is fixed, bound, or free.

Coarse scheduling

Coarse scheduling is currently able to determine the following optimization problem classes:

- (continuous) linear programming;
- mixed integer programming, either linearly constrained, nonlinearly constrained, or completely nonlinear;
- nonlinear optimization, either unconstrained, bound constrained, nonlinearly constrained, or completely nonlinear;
- (nonlinear) complementarity problems; and
- global optimization.

3. First numerical results

The non-trivial part of scheduling is, not surprisingly, the heuristic scheduling. We are using the COPS test set [14,15], a collection of large-scale **C**onstrained **O**ptimization **P**roblem**S**, to test BBO's scheduling algorithms for the class of nonlinear constrained problems. It consists of 17 different AMPL models and 3 data sets, which come from fluid dynamics, population dynamics, optimal design, mesh smoothing, and optimal control.

The knowledge base of BBO currently contains about 50 facts and rules. An optimization method without heuristic scheduling adds 2 to 4 facts to the knowledge base, **N**onlinear **C**onstraint **O**ptimization (NCO) NEOS methods supporting heuristic scheduling add close to 10 facts.

We compare BBO's choice of optimization method to random selections. For this purpose we use the runtime results from [14] for 3 NCO NEOS optimizers that perform best on at least one problem (LOQO, MINOS, and KNITRO); using BBO's NEOS optimizers would have introduced uncontrollable network latencies. We evaluated our scheduler on those 10 problem instances that had the largest runtime difference between best and worst of the selected method.

The second column of Table 1 shows which optimizer the scheduler recommends to use respectively **not** to use if the optimizer's name is prefixed with an exclamation mark. The BBO knowledge base reports executed rules as justification for its results. In this particular case, only three rules are responsible (discriminating) for the relative evaluation:

- LOQO slows down when many free variables exist.
- KNITRO slows down when many bounds exist.
- MINOS slows down when nonlinear functions are expensive to evaluate.

Columns three to five list the free variable ratio n_F/n, the bound constraint ratio m_B/m and the asymptotic evaluation costs c for each problem. These properties are the conditions for the three rules above.

Column six contains the average solving time over all three optimization methods as measure for the performance a naive user without BBO scheduling could expect. Column seven lists BBO's solving time, which is either the time of the recommended method or the average of the two methods that have not been excluded.

Table 1
BBO COPS schedule. The last column is the BBO time divided by the average time

Problem	Schedule	n_F/n %	m_B/m %	c	Avg.	BBO	Ratio %
Steering	!LOQO	79.8	20.2	$O(1)$	668.11	2.17	**0.3**
Robot	MINOS	33.3	50.0	$O(1)$	669.12	6.38	**1.0**
Rocket	!KNITRO	0.0	57.2	$O(1)$	669.72	1000.78	149.4
Minsurf	LOQO	0.0	100.0	$O(n^2)$	693.93	3.99	**0.6**
Torsion	LOQO	0.0	100.0	$O(n^2)$	106.60	1.02	**1.0**
Bearing	LOQO	0.0	100.0	$O(n^2)$	59.59	0.85	**1.4**
Chain	!LOQO	100.0	0.0	$O(1)$	2.84	4.08	143.5
Glider	MINOS	39.7	42.9	$O(1)$	1377.48	2000.00	145.2
Elec	KNITRO	100.0	0.0	$O(n^2)$	4.34	0.54	**12.4**
Polygon	!MINOS	0.0	7.1	$O(n)$	7.13	7.90	110.7
					4258.86	3027.7	56.6

Considering the simplicity of the rules, runtime savings of over 25% (3000 instead of 4250 seconds) are encouraging results.

The reader should keep in mind that BBO already found the correct problem class. This is non-trivial for a typical engineer – even if, or maybe because, the problem description was self-created. BBO also handles much of the logistics: how to access the optimizer, how to send the data in the correct format, and how to set which parameter. We believe this to be a big step towards our ultimate goal of automated optimization.

4. Summary and future research

We showed that automated optimization is possible in principle and pointed out areas of further work and research. We identified optimization knowledge as one of the primary bottlenecks for scheduling quality. Furthermore, connecting optimization methods to our environment is a time consuming task, here improvements of the optimizer APIs could help reduce this obstacle. The next step of the project will be to integrate dynamic scheduling capabilities in order to e.g. optimize expensive black-box cost functions efficiently. Finally, we need to extend and fine tune the entire scheduling module: the classification and the knowledge base. We are convinced that our efforts will lead to an easier access to optimization, thus assisting engineers to improve their designs in an automated manner, and enabling the automatic generation and solution of optimization problems from knowledge bases.

References

[1] Kolda TG, Lewis RM, Torczon V. Optimization by direct search: new perspectives on some classical and modern methods. SIAM Rev 2003;45:3:385–482.

[2] Jones DR, Schonlau M, Welch WJ. Efficient global optimization of expensive black box function. J Global Optimization, 1998;13:455–492.

[3] Powell MJD. On the use of quadratic models in unconstrained minimization. Technical report, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, 2003.

[4] Colson B, Toint Ph.L. A derivative-free algorithm for sparse unconstrained optimization problems. In: AH Siddiqi and M. Kocvara, editors, Trends in Industrial and Applied Mathematics. Dordrecht: Kluwer Academic Publishers, 2002, pp. 131–147.

[5] Bockholt M. Optimization methods for expensive computer simulations. Master's thesis, TU Braunschweig, 2004.

[6] Booker AJ, Dennis JE, Jr, Frank PD, Serafini DB, Torczon V. Optimization using surrogate objectives on a helicopter test example. In: J Borggard et al., editors, Computational methods for optimal design and control. Proceedings of the 2nd AFOSR Workshop on Optimal Design and Control. Arlington, VA, 30 September–3 October, 1997. Boston: Birkhäuser. Prog Syst Control Theory 1998;24:49–58.

[7] Petzoldt D. Design and implementation of a software system for automated optimization. Master's thesis, Technische Universität Berlin, 2004.

[8] Dageförde J. Design and validation of an open and modular architecture for mathematical optimization in industrial engineering. Master's thesis, Christian-Albrechts-Universität zu Kiel, 2004.

[9] AMPL: A mathematical programming language. http:// www.ampl.com

[10] Neos server for optimization. www-neos.mcs.anl.gov/ neos/

[11] Nelder JA, Mead R. A simplex method for function minimization. Comput J 1965;7:308–313.

[12] The protege ontology editor and knowledge acquisition system. http:// protege.stanford.edu/

[13] Algernon-j: Rule-based programming. http://algernon-j. sourceforge.net

[14] Dolan ED, Mor JJ, Munson TS. Benchmarking optimization software with cops 3.0. Technical Report ANL/ MCS-TM-273, Argonne National Laboratory, 9700 South Cass Avenue Argonne, Illinois 60439, February 2004.

[15] Cops: Large-scale optimization problems. http://www-unix.mcs.anl.gov/ ~more/cops/