INFORMATICA, 1991, Vol.2, No.2, 278–310

HEURISTIC REASONING IN MATHEMATICAL PROGRAMMING

Klaus SCHITTKOWSKI

Mathematisches Institut, Universität Bayreuth,
8580 Bayreuth, Germany

Abstract. In general terms some situations are described which require the exploitation of heuristics either to solve a mathematical optimization problem or to analyse results. A possibility to implement heuristic knowledge for selecting a suitable algorithm depending on available problem data and information retrieved from the user, is investigated in detail. We describe some inference strategies and knowledge representations that can be used in this case, and the rule-based implementation within the EMP system for nonlinear programming. Case studies are presented which outline on the one hand the heuristic recommendation of an optimization code and the achieved numerical results on the other hand.

Key words: Mathematical programming, nonlinear optimization, heuristic reasoning, expert systems.

1. Introduction. Whenever a practical optimization problem is to be solved, heuristic decisions must be made in many different situations. Among them are the questions, how to model a given real-life problem, how to select and execute a mathematical programming algorithm and how to analyse and interpret the results.

Whenever the decision maker possesses sufficient expe-

rience on the decision to be made, i.e. on the mathematical model, the numerical algorithms and their performance, there is certainly no need to assist him in form of special software tools. On the other hand, we often observe that mathematical optimization algorithms are in the hands of unexperienced or occasional users who do not posses the knowledge and practical experience about the model and the algorithms, to be able to find the best compromise between all decisions that must be made.

Thus the main intention of this paper is to discuss possible heuristic decisions and to present a way to implement them in form of a special software code. Although we will not investigate the structure of numerical algorithms in detail, we concentrate all conlusions on the general nonlinear programming problem of the following form:

$$\begin{aligned} & min f(x) \\ & g_j(x) = 0, \quad j = 1, ..., m_e \\ x \in \Re^n : \; & g_j(x) \geq 0, \quad j = m_e + 1, ..., m \\ & x_l \leq x \leq x_u \end{aligned}$$

Certainly part of the approach can be applied to other areas as well, in particular the inference mechanism presented.

In the above mathematical description various special problem types are hidden which can be solved by numerical algorithms that were entirely developed for them, e.g. least squares problems, global optimization problems, smooth or nonsmooth problems. In other words, there does not exist one 'black box' algorithm that is capable to solve the general mathematical programming problem without any further restrictive assumptions.

Neverthless we will try to discuss the mathematical programming problem as general as possible by describing heuristic reasoning as implemented in the EMP-system, cf. Schitt-

kowski (1987a). This interactive software system allows input of general nonlinear functions in form of FORTRAN-statements, the automatic generation of FORTRAN-programs with respect to a selected algorithm, their execution and investigation of results which are kept in a data base. Heuristics are exploited to assist a user when choosing a suitable optimization algorithm and when analysing possible error messages to recommend remedies.

Due to the general structure of the optimization problems that can be treated by EMP, the recommendations either of algorithms or remedies in failure situations must be quite vague. Much more precise exploitation of human experience by means of a software implementation can be attained when concentrating the decisions to be made, on a special area, e.g. structural mechanical optimzation. We mention here without going into details, that in a very similar way, heuristic reasoning systems were implemented by the author to assist non-experienced users of the structural optimization system LAGRANGE, cf. Schittkowski (1989) and Kneppe, Krammer (1987), to select a suitable algorithm and to interprete failures. Since the problem structure is more restrictive, specialized data and knowledge is available in this case, so that the resulting conclusions are more precise.

The subsequent section of this paper describes the necessity for applying heuristics in mathematical programming. To implement heuristic reasoning, various inference algorithms and knowledge representations are known from artificial intelligence. In Section 3, two traditional approaches are presented which are based on forward and backward reasoning and a suitable corresponding knowledge base. Both are part the language SUSY, see Schittkowski (1988), which can be considered as an expert system shell at least in the framework discussed in this paper. A rough outline of SUSY is also given in Section 3.

In Section 4, the expert system EMP is introduced. In particular the evaluation of experience factors and the implementation of the heuristic reasoning subprocesses for proposing a suitable algorithm and remedies in failure situations are described. Moreover some case studies are evaluated to show the feasibility of the approach. We compare the heuristic decisions of the system with the numerical results obtained. The results are summarized in Section 5.

2. Heuristics in mathematical programming. First we have to realize that in this paper, we do not consider any possibilities to include heuristic decisions into a numerical algorithm itself e.g. by introducing a hybrid structure to switch to another variant or algorithm based on interaction with the user. The basic idea is to accept available optimization algorithms as they were developed by numerical analysts and to facilitate or automize only access to these algorithms or the interpretation of results.
Thus we consider some situations in which heuristic knowledge must be exploited by a decision maker. One of them will be analysed in subsequent sections of this paper in more detail.

a) Model selection: Here we assume that one has to select a mathematical model depending on available data. Typical example is parameter estimation, where a suitable model function given either explicitely or in form of a differential equation, must be determined depending on the structure of the data and the application. In such a case, the acceptance of a model is often more important than the achieved numerical results. In other words a qualitative answer is desired and not a quantitive one. To make the forecast which model is the most acceptable one, is however often difficult and can be made only by experts.

b) Algorithm selection: Let us assume that, as in many

real situations, more than one optimization algorithm is available, and that a decision must be made to select not only an algorithm, but also a suitable combination of input parameters that control the execution. The more specialized the class of optimization problems under consideration is, the more human experience is available and can be evaluated. Any decision will always depend also on available results that were ontained in the past and that are evaluated whenever the new problem is somewhat related to an old one. This situation is analysed in the subsequent sections in more detail.

c) **Failure analysis:** We cannot expect that an optimization algorithm is always capable to solve a problem in the class for which the algorithm was designed. There may be many instabilities in the problem data, e.g. round-off errors, ill-conditioning, bad scaling, that prevent a successful numerical solution. On the other hand, we have to expect also failures in the problem data itself, e.g. empty feasible domain, or even programming errors either in function or, more frequently, in gradient evaluation. However a numerical algorithm will break down with an error message that is a consequence of the error, e.g. by reporting that a line search could not terminated successfully. The conclusion which reason caused the failure, is often difficult and requires detailed knowledge on the numerical algorithm and at least some fundamental mathematical optimization theory to understand the error message. Moreover any conclusion based only on the error message and some data, must be very vague without further investigations.

d) **Acceptance of a solution:** Even if an optimization method reports that an optimal solution was found, the solution might be unacceptable. One reason is that the algorithm is only capable to check some stopping condi-

tions which can be met too early e.g. due to badly scaled problem functions. Another reason is the possibility to achieve only a local solution which must be rejected in special situations. Worst of all, even a known global optimal solution is sometimes unacceptable in the sense that the underlying model turns out to be incorrect, e.g. as in parameter estimation. Thus one has to analyse an achieved answer of an optimization algorithm very carefully to decide whether the solution is acceptable or not. Experience and knowledge is necessary to get a decision which does depend heavily on heuristics.

In the following, we consider only the selection of a suitable optimization code whithin the EMP-system in detail to give an example how heuristics can be embedded in an interactive system. The application of the proposed methodology to the other si tuations is straightforward. For understanding why the infe rence procedures introduced in the subsequent section, work in these cases, we have to know that all conclusions are very vague in general. Without deeper information on the model structure, the data and in particular available experience we are unable to get more precise answers.

Again it must be mentioned that by no means the software tools presented, can replace a human expert. The basic idea is to exploit the available experience of a human expert and to im plement it so that other users who do not have direct contact with him, are assisted at least by a software tool. It is hoped that the user will get then his own expertise about the mathematical optimization model he is investigating, so that he might become able after some experimentation to solve his problem without any external help.

3. Inference mechanisms. The interactive optimization system EMP that incorporates some of the heuristics mentioned in the previous section, is written in the SUSY (support system) language, see Schittkowski (1988). The ba-

sic structure of SUSY is procedural and commands are interpreted. By the following lines, a brief summary of the main features of the procedural part of SUSY is to be given first.

The programming language SUSY was designed to develop interactive software systems like data management systems, interactive user interfaces, intelligent software systems (expert systems), or integrated problem solving systems. Since a SUSY program can interact directly with the operating system and therefore with existing programs written in any other language, possible applications are:

- Interactive processing of problem data and results in a data base
- Construction of formatted input files for an existing system
- Problem dependent selection and linking of existing modules
- Automatic start of an external solution method
- Information retrieval of performed solution attempts
- Interpretation of results
- Report writers

Since SUSY supports the generation of include-files which are interpreted, even large software systems can be implemented with minimum core storage needed. The main important facilities of SUSY are:

- Permanent and temporary variables of various data types (CHAR, NAME,INTEGER, DECIMAL, STRING, LINE, TEXT, TABLE, FILE)
- Structured data types (RECORD, STRUCTURE)
- Assignment, compound and goto-statements

- Logical and arithmetic expressions
- Brief in- and output commands (':' - in- and output from terminal, '<' - output to file, '>' - input from file, '?' - display of help-text)
- Support of monitor representation (COLOUR, SCROLL, ENLARGE, ...)
- Windows with input mascs, scrolling table columns, hand calculator and help windows, overlapping windows (WINDOW FROM ... TO ..., ENDWINDOW)
- Data base commands (NEW, LAST, FIRST, NEXT, DELETE, SEARCH, SORT, ...)
- Table calculation (MIN, MAX, AVERAGE, SUM, ...)
- File management (RESET, REWRITE, SEARCH, SORT, ...)
- Interactive input of SUSY commands (INTERACTIVE)
- Include files and macros (INCLUDE, MACRO)
- Direct input and execution of arbitrary operating system commands ($ <command>, GO, EXECUTE)
- Interactive help and system documentation (HELP, DOCUMENTATION)
- Editor and programming environment (EDIT)
- Text formatter and automatic hyphenation (FORMAT, HYPHENATE)

An interactive installation programm facilitates the generation of a new software system to be written in SUSY. Maintenance and debugging is supported by a programming environment. The SUSY language was also used to implement larger practical integrated software systems, e.g. an

interface for the mechanical structural optimization system LAGRANGE, see Schittkowski (1989), or an interactive modelling and parameter estimation system GAUSS, see Schittkowski (1990). Also in these cases, the systems are self-learning, heuristic rules were used for selecting a suitable algorithm or model in dependance on the problem to be solved, and to analyse failures for proposing suitable remedies.

SUSY has the additional option to execute two different inference strategies to process heuristic knowledge represented in form of rules. The corresponding knowledge base consisting of different types of rules, of actions, goals and other constructs, is defined separateley in a SUSY program, i.e. either in the declaration part or in include files. Several rule systems of the same or different types can be part of one program and are executed whenever needed by special commands (REASONING). One rule system is allowed to call other rule subsystems, arbitrary SUSY commands or even operating system commands.

One of the two inference procedures in SUSY is based of a knowledge representation in form of subject-object-attribute-value-tupels and uses backward-chaining, see Kummert (1989). Goals must be defined which are then tried to be satisfied recursively by available rules. If a fact in the antecedent of a rule is unknown, the user will be asked. In this situation an explanation component can be executed to get information on the local status and the subgoal that is to be fullfilled. Certainty factors are used to express uncertain facts and conclusions and are evaluated in a way similar to MYCIN, cf. Shortliffe (1976). Further components are automated knowledge aquisition, tracing, meta-rules and variable facts, goals, rules and meta-rules.

The heuristic reasoning processes implemented in EMP, however, use forward chaining because of the relatively simple structure of decision trees. Thus we describe this SUSY option

a bit more detailed.

The reasoning process allows the storage and processing of heuristic knowledge based on rules with certainty factors and actions. A typical rule is of the form

$$IF \ < antecedent >$$
$$THEN \ < consequent > \ WITH \ < cf >$$

Antecedents can be logical expressions or actions, where the consequent must be an action, i.e. an arbitrary sequence of SUSY commands. The reasoning process allows additional bounds for the antecedent certainty factors, logical combination of antecedents, repeated execution of rules, automatic alteration of certainty factors in case of executing a rule, modifying certainty factors 'by hand', and an explanation component. The system one is only capable to give local information or information on decision history, but not on a goal to be attained.

All actions will get an internal certainty factor when starting the inference procedure. If an antecedent is satisfied, the certainty factor of the action in the consequent part is updated according to some formulas similar to the MYCIN implementation, see Shortliffe (1976). An antecedent is satisfied, if either the logical expression is true or if the certainty factors of all actions are between the predetermined bounds.

The SUSY commands of the action that got the largest certainty factor within one loop, are then executed. If necessary, the certainty factor of a rule is updated according to the arithmetic expression defined by the WITH-part of the consequent, if any. This allows also repeated executions of rules.

Since all certainty factors can be provided in form of system variables, they can be modified within actions. In addition it is possible to use variable expressions in defining rules and actions, which are replaced symbolically.

4. Heuristic reasoning in EMP. EMP is an interactive programming system that supports model building, numerical solution and data processing of constrained mathematical programming problems, cf. Schittkowski (1987a). Various options are available in EMP to facilitate the formulation of problem functions. The objective function e.g. may be a linear or quadratic function, a data fitting function, a sum or maximum of functions, or a general function without a structure that could be exploited. More precisely the following mathematical models are available for facilitating the formulation of objective or constraint functions and exploiting special structures mathematically whenever possible:

- Linear programming
- Linear regression
- Quadratic programming
- Nonlinear L_1 and L_∞-data fitting
- Nonlinear L_2- or least squares data fitting
- Multicriteria optimization
- Min-max optimization
- Non-smooth optimization
- Global optimization
- General nonlinear programming

All problems may have bounds for the variables and linear or nonlinear constraints. In both cases it is possible to proceed from two-sided bounds for the restrictions. Data fitting problems are either composed of a sequence of arbitrary nonlinear functions or of one model function, where the experimental data are provided separately.

For most optimization problems, several different algorithms are available, which were either developed by the author or taken over from libraries and other authors in original form, i.e. without any adaptions to the EMP system. The present version of EMP contains mathematical methods of the following type:

- Sequential quadratic programming methods for nonlinear programming
- Bundle methods for non-smooth problems
- Stochastic global optimization methods
- Levenberg-Marquardt, Gauss-Newton, Newton, and quasi-Newton methods for nonlinear least squares problems
- Ellipsoid method for smooth and non-smooth optimization
- Dual and primal methods for quadratic programming
- Adapted sequential quadratic programming methods for constrained L_1-, L_2-, L_∞-norm and min-max problems

EMP includes program generators for codes of the frequently used optimization libraries NAG and IMSL, and the algorithm base is extended steadily.

For objective function and constraints, the input of quadratic or linear functions reduces to definition of some vectors and matrices, respectively, where sparsity can be exploited. Gradients of nonlinear and nonquadratic functions are approximated numerically, but can also be provided by the user in analytical form. For nonlinear problem functions that can be represented either by analytical expressions or statements of a simple modelling language, gradients can be evaluated automatically, i.e. exactly without differentiation 'by hand', see

Liepelt (1990). The input of sequences of similar objective or constraint functions and variables is facilitated, if they differ at most by an index.

Only the problem relevant data need to be provided by a user in an interactive way. General functions must be defined by sequences of FORTRAN statements addressing a numerical value to a user pro- vided function name. All generated problems are stored in form of a data base system, so that they are easily retrieved, modified, or deleted on request. EMP proposes a suitable mathematical algorithm and writes a complete FORTRAN source program. The system executes this program and stores the numerical results in the data base, so that they are available for further processing. Since individual names for functions and variables can be provided by a user, it is possible to get a problem dependable output of the achieved solution.

The user will be asked whether he wants to link the generated FORTRAN program with some of his own files or whether he wants to insert additional subroutines, declaration and executable statements to formulate the problem. It is possible to generate the same programs automatically, that must be generated by 'hand' otherwise.

All actions of EMP are controlled by self-explained commands which are displayed in form of menues. Step by step the user will be informed how to supply new data. Whenever problem data are generated or altered, the corresponding information will be saved on a user provided file. Besides commands to generate, solve or edit a problem, there are others to transfer data from one problem to another, to delete a problem, to sort problems, to get a report on problem or solution data, to halt the system and to get some information on the system, the mathematical models and the available algorithms. It is even possible to insert arbitrary operating system commands without leaving EMP.

The main intention of EMP is to prevent the organisational 'ballast' otherwise required to solve a nonlinear programming problem with a special algorithm. Once the system is implemented, it is not necessary

- to define special in- or output files for each problem to be solved,
- to select a suitable mathematical algorithm 'by hand',
- to read any documentation of the used mathematical programming algorithm,
- to write long lists of declaration statements, e.g. for dimensioning auxiliary arrays required by the algorithm, or to call the mathematical programming code with a long list of parameters that are to be defined initially,
- to provide the problem functions and their gradients in a special form required by the mathematical programming algorithm,
- to make the results readable for a decision maker.

Thus the domain of application of EMP is summarized as follows:

(a) Programming neighbourhood for developing a first executable program version solving a specific practical problem (or class of problems).

(b) Investigation of different model variants fitting best to a given real world situation.

(c) Testing certain types or modifications of mathematical programming algorithms for solving a class of problems.

(d) Collecting numerical experience on solution methods for optimization problems.

(e) Teaching students on model building (e.g. structural optimization courses in engineering science) or on numerical behavior of optimization algorithms (e.g. optimization courses in mathematics).

EMP allows a user to concentrate all his efforts on the problem he wants to solve and takes over the additional work to select a suitable algorithm and to organize the data, the execution of the problem functions and the program structure. It should be possible to solve optimization problems of the class under consideration within a minimal fraction of time needed otherwise.

EMP is implemented in the SUSY language described briefly in the previous section, cf. Schittkowski (1988). The system is running at present on VAX/VMS, HP-UNIX and MS-DOS computing environments.

In various ways the system is capable to learn and to store its own experience on the success of solution attempts. The proposals offered by EMP, will therefore become better and better with increasing knowledge on the numerical structure of the user provided models. A rule-based failure analysis explains some reasons for possible false terminations and proposes remedies to overcome numerical difficulties.

The solution of an optimization problem by EMP is to be explained more precisly now, in particular the evaluation of learning factors and the heuristic proposal of numerical algorithms. After input of the problem name, a user has the option to require the display of all available codes that could solve his optimization problem. The list contains a certainty factor for each proposed program which indicates a very rough approximation of a measure for the numerical performance of an algorithm. A value of 100 is the maximum attainable degree of belief, whereas a value of 0 indicates a very poor performance of the algorithm on previous runs. The numerical values of the

certainty factors are defined and updated in three different ways:

1. Initially every code obtains a certain permanent default value of the system author which is based on his own subjective experience.

2. Whenever a code could solve a problem succcessfully, so that the stopping criteria are satisfied subject to the provided tolerances, the corresponding permanent certainty factor of the code is increased. If, on the other hand, a failure is reported, then the permanent certainty factor is decreased. The factor is not altered at all if the iteration was interrupted due to the fact that the maximum number of iterations was attained.

3. When starting the solution of a problem, a local certainty factor is defined which gets the value of the permanent one, and all previous numerical solution attempts for this problem are investigated. If the problem could not be solved by a specific algorithm, then the local certainty factor is decreased significantly, and enlarged otherwise.

The local certainty factors are displayed to support the selection of a suitable code. It is hoped that the choice of these factors reflects the influence of special model structures and that EMP is capable to learn, i.e. to propose better and better solution methods in dependance on the problem to be solved.

Moreover the user is asked whether he wants to select a code 'by hand', where he may exploit the experience reflected by the certainty factors, or whether he prefers to initiate a rule-based code selection by the system. In this case, some additional questions must be answered and the outcome is the display of a list of codes in the order of their certainty. The

evaluation of the certainty factors is based on the given experience factors described above, some internal problem data like number of variables or constraints, numerical differentiation etc., and on the answers of the user. More precisely the following data and information are imbedded in the decision process:

- structure of the model (e.g. least squares)
- number of variables
- number of constraints
- type of constraints (e.g. bounds, linear)
- calculation type of (sub-)gradients (e.g. numerically)
- smooth problem functions
- noise in evaluating problem functions
- expected number active constraints
- ill-conditioned problem
- approximation of global solution
- location of starting point
- expensive function evaluations

Rules are evaluated by forward chaining as described in the previous section. To give an example, a few rules supporting or rejecting the sequential quadratic programming routine NLPQL of Schittkowski (1985/86) are listed in somewhat simplified form:

if active with 50 to 100
then nlpql with 30

if nonoise with 50 to 100
then nlpql with 10

if degene with 70 to 100
then nlpql with 30

if nonglobal with 50 to 100
then nlpql with 20

if smooth with 50 to 100
then nlpql with 10

if expensiv with 50 to 100
then nlpql with 40

if stclose with 50 to 100
then nlpql with 20

if large with 50 to 100
then nlpql with 10

if noise with 30 to 70
then not nlpql with 40

if noise with 70 to 100
then not nlpql with 80

if nonsmooth with 50 to 100
then not nlpql with 20

if manyrest with 50 to 100
then not nlpql with 50

There must be some other rules to initiate the ruling procedure and to set the actions used above, in an appropriate way by retrieving the information either from the available data base or from the user.

It is selfevident that a user may reject the algorithm which got the largest certainty value, and to choose another one. The described evaluation of heuristic knowledge is available only for the general nonlinear programming or the nonlinear least squares model, since only in these situations, a larger number of different codes is available to solve the problem.

Subsequently some additional decisions must be made by a user, e.g. the choice of a suitable output flag. It is

possible that a previously computed and eventually inaccurate approximation of the solution is available. Then the user is asked whether he wants to accept this guess as a starting point for the actual run or not. If some additional output from the underlying mathematical programming algorithm is required, then the information is displayed on the terminal in original form, in particular without individual user-provided names for functions or variables.

The generated FORTRAN code is written on a text file with the name 'EMPCODE.FOR'. The code consists of a main program and, if the problem functions are nonlinear, two subroutines for evaluating problem functions and corresponding gradients in a form required by the selected optimization method. EMP compiles the object code, links it with a chosen nonlinear programming algorithm and eventually some object files of the user, and executes the resulting program. All numerical results, performance data and termination reasons are stored automatically in the underlying data base and are available for further processing. Afterwards the main-menue of EMP is displayed again and the user may select any additional actions, e.g. to investigate the obtained results. Note that after leaving the system, the last generated FORTRAN program is still available and could also be used furtheron independently from EMP. It is possible to direct the output of an optimization program to an existing file on request.

5. Numerical results. We want to test now the heuristic proposal process within the EMP system and to compare the recommendations of the system with the numerical results obtained. We consider only the class of general nonlinear programming problems, similar conclusions would be obtained for least squares problems. Since the decision process starts from available and eventually modified experience factors, we use always the initial default values. The EMP interface contains a corresponding reset option.

To evaluate certainty factors, 11 different test cases are constructed, see Table 1. The table contains in its first column an identification number, then the source, some dimensioning parameters and the answers to the questions presented by EMP. Source information beginning with 'TP' indicates that the test problem was taken from the test problem collections Hock and Schittkowski (1981) or Schittkowski (1987b). The following abbreviations are used for identifying the situation:

n – number of variables
m – number of constraints (without bounds, equality and inequality ones)
l – number of equality constraints
S – smooth problem
N – noisy problem functions (N-no noise, W-weak noise, B-big noise)
D – degenerate problem functions, i.e. highly nonlinear at a solution
G – global solution desired
E – expensive function evaluation
C – starting point close to a solution
W – well-scaled problem functions and variables

Usually the table entry is 'Y' for yes or 'N' for no. Besides of the dimensioning parameters, there are no attempts to qualify the other problem features in a quantitative way. Since we do not want or are unable to evaluate them numerically, the information as obtained from the user, is processed symbolically. Of course, the real system has also the option to give the answer 'D' standing for 'do not know' or any intermediate classification.

There are some other items influencing the decision process, e.g. number of constraints expected to be active at an optimal solution. But in this paper, we want to give only an impression how heuristic reasoning can be used in mathe-

matical optimization, and investigate therefore only the data mentioned above, which are required to understand the subsequent tests.

Table 1. Test problem characteristics

No	Source	n	m	l	S	N	D	G	E	C	W
1	TP87	6	4	4	N	N	Y	N	N	N	Y
2	TP1	2	0	0	Y	N	N	N	N	N	Y
3	" /weak noise	2	0	0	Y	W	N	N	N	N	Y
4	" /big noise	2	0	0	Y	B	N	N	N	N	Y
5	Mifflin (1982)	2	0	0	N	N	N	N	N	N	Y
6	Branin (1972)	2	0	0	Y	N	N	Y	N	N	Y
7	" /expens.	2	0	0	Y	N	N	Y	Y	N	Y
8	TP299	100	0	0	Y	N	N	Y	N	N	Y
9	TP343	3	2	0	Y	N	N	N	N	Y	Y
10	" /scaled	3	2	0	Y	N	N	N	N	Y	N
11	TP106	8	6	0	Y	N	Y	N	N	N	N

In some cases, the original problem description was slightly modified or a corresponding situation was only simulated. Test problem TP87 is one of the famous Colville (1968) problems. The objective function is piecewise linear and not continuous, the constraints are highly nonlinear equality restrictions. TP87 is considered as a degenerate problem, since the third equality constraint possesses a very small multiplier value, cf. Hock and Schittkowski (1981).

The term 'weak noise' indicates, that a value of the form 'EPS*RAN' was added to all problem functions, where RAN is a randomly generated number between 0 and 1 and EPS is set to 1.E-7. Since all gradients are evaluated numerically by forward differences with a steplength of 1.E-7, the induced error should perturb the gradient heavily. For obtaining a 'big noise', we set EPS to 1.E-3. Both situations reflect the presence of round-off errors in function and, in particular, gradient

evaluations, which often occur in real life situations. It should be mentioned here that all numerical computations were carried out in double precision FORTRAN on a 32 bit machine (VAX 8600). We test the influence of noise for the Rosenbrock function TP1, which has a polynomial, banana-shaped objective function with a zero value at the optimal solution.

Test problem no. 5 of Mifflin (1982) is a typical nonsmooth test problem, since the objective function is composed of the maximum of smooth nonlinear functions.

No. 6 was often used to compare global optimization problmes, since it possesses 6 different local minima. It is the 'six-hump camel-back' problem introduced by Branin (1972). Algorithms for global minimization use a large number of function evaluations. To simulate expensive function evaluations, the problem was used to define test case no. 7. It was declared that the evaluation of problem functions would be expensive. In both cases, we define the starting point by (1,1), which differs significantly from the global optimal solution at (0.08983,-0.7126).

The Rosenbrock function TP1 can easily be extended to very many variables, as done for test problem no. 8. The problem has 100 variables. We assume in this case that the user gave the answer 'Y' when the system wanted to know whether a global solution is to be calculated, only for testing the reasoning process.

Test problem TP343 was used to simulate some another situation which arises quite often in practical situations. No. 9 is the original smooth and well-scaled problem. For defining no. 10, we omitted the scaling factor 1.E-7 in the objective function and scaled the second restriction by 1.E+7, to get a badly scaled test problem. Also we can suppose in these cases that the starting point is close to an optimal solution.

Test problem no. 11 is considered to be a degenerate one, since the multipliers differ significantly at an optimal solution.

Also the problem seems to be badly scaled.

To understand the whole idea behind the tests presented, it is important to note that the whole decision procedure is an experimental one from the viewpoint of the author. We know that many rules may be added to describe further experiences that may influence the selection of an algorithm in the one or other case. However the underlying structure of an optimization problem is very general, since it is hidden behind some sequence of FORTRAN-statements which cannot be analysed by the system. Thus the basic idea is to outline the methodology and to develop the tools that can then be applied to more specialized areas in more detail.

As noted before, we investigate only the class of general mathematical programming problems without further assumptions on the model structure. Table 2 shows the algorithms that were implemented in the EMP system to solve this problem type, and gives a brief information on the mathematical method. More details can be retrieved from the literature or the EMP documentation.

Table 2. General mathematical programming algorithms in EMP

Code	CF	Source	Method
NLPQL	80	Schittkowski (1985/86)	sequential quadratic progr.
E04VDF	80	NAG/Gill e.al. (1983)	sequential quadratic progr.
ZXMWD	60	IMSL/Fletcher (1972)	penalty method
M1FC1	60	Lemarechal e.al. (1981)	bundle method
BT	60	Schramm, Zowe (1989)	bundle/trust region method
UNT	40	Törn, Žilinskas (1989)	stochastic global search
GLOPT	40	Törn, Žilinskas (1989)	random search
ELL	20	Schittkowski (1986)	ellipsoid method

The column headed by CF shows the default certainty factors which are set initially when starting EMP with an

empty data base. They must be taken into account when analysing the subsequent system recommendations, since the evaluation of certainty factors is started from the above values. They reflect the idea that when nothing is known about a problem, we may suppose that some algorithms are more preferable than others. In fact EMP contains also some modifications of the NLPQL-code for solving problems with very many constraints and for solving problems with automatic differentiation. They are omitted here since these modifications do not influence the basic algorithmic structure.

It must be noted here, that some of the algorithms are unable to take nonlinear constraints into account. In this case, a penalty function is formulated which, however, makes the problem eventually ill-conditioned. Only GLOPT and UNT are gradient-free, all other codes need the evaluation of gradients either numerically, as in our tests, or analytically. Since ZXMWD uses an internal numerical differentiation routine, the corresponding entries for NG are zero in the subsequent tables. The code ZXMWD uses different starting points to attempt to approximate a global solution.

The subsequent tables contain the results of the decision process on the one hand and the numerical data obtained by the optimization codes. Only the results of those algorithms are considered, that got a certainty factor greater than 40. The algorithms were executed with their default parameters as included in EMP with the exception, that the maximum number of iterations was increased in some cases. When comparing the results, we have to consider the achieved function values and to relate them to the number of function and gradient evaluations. Of course the proposals made by the system, are very temptative and vague, but reflect the situation that also a human expert would have difficulties to yield a better result based only on the available information. The tables use the following abbreviations for the columns:

Code	– optimization algorithm
CF	– certainty factor evaluated by EMP
IF	– termination code reported by the algorithm
NF	– number of objective function evaluations
NG	– number of gradient evaluations of objective function
OBJECTIVE	– objective function value
CON	– violation of constraints

A termination reason '0' indicates that the optimality conditions of the algorithm were satisfied, and '1' that the maximum number of iterations or function evaluations, respectively, was attained. All other failure codes are the original messages of the optimization algorithms.

Table 3. Test results for test case no. 1

Code	CF	IF	NF	NG	OBJECTIVE	CON
NLPQL	72	0	131	57	0.892759E+4	0.47E-9
E04VDF	72	0	45	45	0.892764E+4	0.66E-10
BT	47	1	267	267	0.923081E+4	0.47E-4
M1FC1	47	5	384	384	0.253764E+5	0.92E+0

Test case no. 1 (Table 3):
Although we declared, that the problem functions would be nonsmooth, the special purpose algorithms BT and M1FC1 got significantly lower certainty factors for several reasons. First we declared that the problem would be degenerate due to highly nonlinear equality constraints, which cannot be handled directly by the codes. Then we are using numerical approximations for gradient evaluations which is not very useful in this case. Also both algorithms are basically designed to solve

convex problems which should be at least continuous. The recommendation of EMP, however, was subsequently verified by the numerical results, since it turns out that the 'jump' in the objective function is close to an optimial solution, but fortunately not identical with it.

Table 4. Test results for test case no. 2

Code	CF	IF	NF	NG	OBJECTIVE	CON
NLPQL	88	0	41	33	0.566069E-7	0.0
E04VDF	88	0	33	33	0.839924E-9	0.0
ZXMWD	68	0	931	0	0.349837E-24	0.0
BT	64	0	79	79	0.519446E-9	0.0
M1FC1	64	0	113	113	0.499141E-5	0.0
GLOPT	46	1	3000	0	0.199429E-3	0.0
UNT	46	0	90	0	0.417657E+1	0.0

Test case no. 2 (Table 4):
The test problem TP1 is highly nonconvex making it impossible for BT to approach the solution. Also UNT has difficulties to get a better approximation.

Table 5. Test results for test case no. 3

Code	CF	IF	NF	NG	OBJECTIVE	CON
NLPQL	66	4	51	30	0.317379E+0	0.0
E04VDF	66	2	78	78	0.273508E-1	0.0
ZXMWD	59	0	1009	0	0.406197E-6	0.0
BT	56	0	76	76	0.783571E-1	0.0
M1FC1	56	0	106	106	0.151011E+0	0.0
GLOPT	55	1	3000	0	0.199494E-3	0.0
UNT	55	0	10	0	0.417657E+1	0.0
ELL	43	0	72	0	0.583132E-1	0.0

Test case no. 3 (Table 5):
All recommendation of EMP do not differ significantly. We observe that in particular the SQP-algorithms are very sensitive with respect to gradient errors, where the gra- dient-free algorithm UNT and GLOPT achieved approximately the same results. Also the ellipsoid algorithm ELL is recommended slightly in this case, since this code is not that much sensible with respect to the accuracy in the gradient evaluation.

Table 6. Test results for test case no. 4

Code	CF	IF	NF	NG	OBJECTIVE	CON
GLOPT	89	1	3000	0	0.401946E-2	0.0
UNT	89	0	110	0	0.417744E+1	0.0
BT	51	1	406	406	0.437104E+2	0.0
M1FC1	51	5	16	16	0.651401E+2	0.0
ELL	48	0	136	136	0.780423E+3	0.0

Table 7. Test results for test case no. 5

Code	CF	IF	NF	NG	OBJECTIVE	CON
NLPQL	70	4	131	25	-0.155514E+0	0.0
E04VDF	70	1	600	600	-0.800000E+0	0.0
ZXMWD	60	0	799	0	-0.999993E+0	0.0
BT	59	0	58	58	-1.000000E+0	0.0
M1FC1	59	1	329	329	-0.999971E+0	0.0
GLOPT	56	1	3000	0	-0.995626E+0	0.0
UNT	56	0	74	0	-0.922585E+0	0.0
ELL	42	0	81	80	-0.999998E+0	0.0

Test case no. 4 (Table 6):
The sequential quadratic programming codes NLPQL and E04VDF are not recommended by EMP. It turns out in fact.

that they break down after a few iterations. Due to extremely large round-off errors, only GLOPT is able to approximate a solution, as expected by EMP.

Test case no. 5 (Table 7/8):
Again it turns out that sequential quadratic programming methods are sensitive with respect to the gradient evaluation, whereas the ellipsoid method is surprisingly efficient. The non-smooth codes BT and M1FC1 got some lower certainty factors, since numerical differentiation was used. If we use analytical differentiation, we will get then the results shown in Table 8.

Table 8. Test results for test case no. 5 (anal. deriv.)

Code	CF	IF	NF	NG	OBJECTIVE	CON
BT	85	0	57	57	-1.000000E+0	0.0
M1FC1	85	1	316	316	-0.999957E+0	0.0
NLPQL	70	4	9	1	-0.778083E+0	0.0
E04VDF	70	4	12	12	-0.800000E+0	0.0
ZXMWD	60	0	717	0	-0.998893E+0	0.0
GLOPT	56	1	3000	0	-0.995626E+0	0.0
UNT	56	0	74	0	-0.922585E+0	0.0
ELL	42	0	49	48	-0.999999E+0	0.0

Table 9. Test results for test case no. 6

Code	CF	IF	NF	NG	OBJECTIVE	CON
GLOPT	89	1	3000	0	-0.103163E+1	0.0
UNT	89	2	106	0	-0.333832E+0	0.0
NLPQL	86	0	16	12	-0.103163E+0	0.0
E04VDF	86	0	12	12	-0.103163E+0	0.0
ZXMWD	84	0	462	0	-0.103163E+0	0.0
BT	60	0	28	28	-0.103163E+0	0.0
M1FC1	60	0	30	30	-0.103163E+0	0.0
ELL	55	0	53	52	-0.103162E+0	0.0

Test cases no. 6/7 (Table 9/10):
It is interesting to observe that also the local algorithms are able to approximate the global solution, but the problem is easy to solve. We obtain a quite different code recommendation of EMP if we would state that the problem functions are expensive to evaluate. In this case, Table 10 shows the corresponding certainty factors, where the numerical results are identical with those of Table 9. Obviously EMP does not recommend any other algorithm besides of the sequential quadratic programming methods, since it is assumed that the solution time, i.e. number of function evaluation, would be too large.

Table 10. Test results for test case no. 7

Code	CF	IF	NF	NG	OBJECTIVE	CON
NLPQL	91	0	16	12	-0.103163E+0	0.0
E04VDF	91	0	12	12	-0.103163E+0	0.0

Test case no. 8 (Table 11):
Table 11 contains the results for the large problem no. 8 with 100 variables, where we assume that the user wants to get a global solution. Nevertheless only the locally convergent algorithms are recommended by EMP, since special global codes are unable to solve problems with more than 10 or 20 variables, respectively.

Table 11. Test results for test case no. 8

Code	CF	IF	NF	NG	OBJECTIVE	CON
NLPQL	87	0	199	101	0.200413E-7	0.0
E04VDF	87	4	75	75	0.795567E+3	0.0
ZXMWD	78			error		
BT	42	1	413	413	0.715645E+2	0.0
M1FC1	42	1	1704	1704	0.762619E+2	0.0

Test case no. 9 (Table 12):
For smooth, well-scaled problems, where the starting point is close to a solution, the usage of sequential quadratic programming codes is highly recommended by EMP, see Table 12. All other algorithms have difficulties to find a solution that could be accepted.

Table 12. Test results for test case no. 9

Code	CF	IF	NF	NG	OBJECTIVE	CON
NLPQL	90	0	5	5	-0.568478E+1	0.31E-12
E04VDF	90	0	5	5	-0.568478E+1	0.31E-12
ZXMWD	66	130	1064	0	-0.494226E+1	0.0
BT	51	0	197	197	-0.568478E+1	0.80E-11
M1FC1	51	2	18	18	-0.119544E+108	0.13E+60

Test case no. 10 (Table 13): If the problem is badly scaled, also sequential quadratic programming problems might have some difficulties to find the optimal solution, but they are still more preferable than others, cf. Table 13. Global search methods got some certainty in this case, since they are usually quite independent from scaling of problem functions.

Table 13. Test results for test case no. 10

Code	CF	IF	NF	NG	OBJECTIVE	CON
NLPQL	70	0	6	6	-0.568478E+8	0.15E-7
E04VDF	70	4	16	7	-0.568478E+8	0.0
BT	51			error		
M1FC1	51	2	33	33	-0.886226E+182	0.46E+104
ZXMWD	48			error		
GLOPT	41	1	3000	0	-0.190509E+14	0.0
UNT	41	0	130	0	-0.179258E+14	0.0

Test case no. 11 (Table 14):
Similar to test case 10, EMP recommends to use sequential quadratic programming methods if the problem is degenerate. The highly nonlinear constraints prevent a solution by other available codes.

Table 14. Test results for test case no. 11

Code	CF	IF	NF	NG	OBJECTIVE	CON
NLPQL	72	0	65	65	0.704925E+4	0.98E-6
E04VDF	72	0	59	59	0.704925E+4	0.21E-6
BT	51	1	400	400	0.139328E+5	0.87E+10
M1FC1	51	5	33	33	0.147504E+5	0.0

Conclusions. It was shown how heuristic knowledge of mathematical programming experts can be implemented in form of a suitable software systems so that their knowledge becomes available for non-specialists or occasional users of optimization algorithms. The approach was demonstrated by introducing briefly an interactive optimization system called EMP, which contains a rule-based subsystem for proposing a suitable code depending on some problem characteristics. For a series of case studies, the recommendations of EMP were compared with the numerical results achieved.

REFERENCES

Branin, F.H. (1972). Widely convergent methods for finding multiple solutions of simultaneous nonlinear equations. *IBM Journal of Research Developments,* 504–522.

Colville, A.R. (1968). *A Comparative Study on Nonlinear Programming Codes.* IBM Scientific Center Report. No. 320–2949, New York.

Fletcher, R. (1972). *Fortran Subroutines for Minimization by Quasi-Newton Methods.* Report R7125, AERE, Harwell, England.

Gill, P.E., W.Murray, M.A.Saunders, M.H.Wright (1983). *User's guide for SOL/NPSOL: a FORTRAN Package for Nonlinear Programming.* Report SOL 83-12, Department of Operations Research, Stanford University, Stanford, USA.

Hock, W., and K.Schittkowski (1981). *Test Examples for Nonlinear Programming. Lecture Notes in Economics and Mathematical Systems,* Vol.187. Springer.

Kneppe, G., J.Krammer, E.Winkler (1987). *Structural Optimization of Large Scale Problems Using MBB-LAGRANGE.* Report MBB-S-PUB-305, Messerschmidt-Bölkow-Blohm, Munich , Germany F.R.

Kummert, A. (1989). *Rückwärtsverkettende Schlussverfahren für Regelsysteme.* Diplomarbeit, Mathematisches Institut, Universität Bayreuth.

Lemarechal, C., J.-J.Strodiot, and A.Bihain (1981). On a bundle algorithm for nonsmooth optimization. In Mangasarian, Meyer and Robinson (Eds.), *Nonlinear Programming.* Academic Press.

Liepelt, M. (1990). *Automatisches Differenzieren.* Diplomarbeit, Mathematisches Institut, Universität Bayreuth.

Mifflin, R. (1982). A modification and an extension of Lemarechal's algorithm. *Mathematical Programming Study,* **17**, 77-90.

Schittkowski, K. (1985/86)). NLPQL: A FORTRAN subroutine solving constrained nonlinear programming problems. *Annals of Operations Research,* **5**, 485-500.

Schittkowski, K. (1986). *ELL: A FORTRAN Implementation of an Ellipsoid Algorithm for Nonlinear Programming: User's Guide.* Report, Mathematisches Institut, Universität Bayreuth, FRG.

Schittkowski, K. (1987a). *EMP: An Expert System for Mathematical Programming.* Bericht, Mathematisches Institut, Universität Bayreuth, Bayreuth, Germany F.R.

Schittkowski, K. (1987b). *More Test Examples for Nonlinear Pro-*

gramming Codes. Lecture Notes in Economics and Mathematical Systems, Vol.282. Springer.

Schittkowski, K. (1989). Knowledge-based problem solving systems in structural optimization. In Eschenauer and Thierauf (Eds.), *Discretization Methods and Structural Optimization.* Vol.42. Springer.

Schittkowski, K. (1990). *GAUSS: Interactive Modelling and Parameter Estimation.* Report, Mathematisches Institut, Universität Bayreuth.

Schramm, H., and J.Zowe (1989). A version of the bundle idea for minimizing a nonsmooth function: Conceptual idea, convergence analysis, numerical results. (submitted for publication).

Törn A., and A.Žilinskas (1989). *Global Optimization. Lecture Notes in Computer Science.* Vol.350. Springer.

Received February 1991

K.Schittkowski received his 'Diplom' (1972), 'Dr. rer.nat.' (1974) and 'Dr.habil' (1979) in Mathematics all at the University of Würzburg, Department of Mathematics. From 1982 to 1984 he got a position of an associate professor ar the Computer Science Department of the University of Stuttgart. Since then he is professor for applied computer science at the University of Bayreuth, Department of Mathematics. His main research areas are numerical optimization, nonlinear programming application problems and the development of userfriendly interfaces particularly for mathematical optimization.