COMPARISON AND AUTOMATED SELECTION OF LOCAL OPTIMIZATION SOLVERS FOR INTERVAL GLOBAL OPTIMIZATION METHODS∗

MIHÁLY CSABA MARKÓT† AND HERMANN SCHICHL†

Abstract. We compare six state-of-the-art local optimization solvers with focus on their efficiency when invoked within an interval-based global optimization algorithm. For comparison purposes we design three special performance indicators: a solution check indicator (measuring whether the local minimizers found are good candidates for near-optimal verified feasible points), a function value indicator (measuring the contribution to the progress of the global search), and the running time indicator (estimating the computational cost of the local search within the global search). The solvers are compared on the COCONUT Environment test set consisting of 1307 problems. Our main target is to predict the behavior of the solvers in terms of the three performance indicators on a new problem. For this we introduce a $k$-nearest neighbor method applied over a feature space consisting of several categorical and numerical features of the optimization problems. The quality and robustness of the prediction is demonstrated by various quality measurements with detailed comparative tests. In particular, we found that on the test set we are able to pick a ‘best’ solver in 66–89% of the cases and avoid picking all ‘useless’ solvers in 95–99% of the cases (when a useful alternative exists). The resulting automated solver selection method is implemented as an inference engine of the COCONUT Environment.

Key words. local optimization, global optimization, classification, $k$-nearest neighbor fit, automated solver selection, interval arithmetic

AMS subject classifications. 65K05, 90C26, 90C30

1. Introduction. In this study we investigate a set of local optimization solvers in terms of their efficiency for interval global optimization. That is, we are interested in the behavior of the local solvers when invoked successively within the global search (from different starting points). Therefore we measure their success towards finding the global minimizers in terms of special aggregate features. Our goal is well beyond a purely comparative analysis of the solvers: we intend to explain solver performance from the particular problem characteristics and to predict it for new problems. This enables us to develop an automated local solver selection method for the global search.

Since the present study is part of the long-term extensive development of the COCONUT Environment, we need a somewhat longer than usual introduction to give an insight into the overall framework and the applied methodologies. Therefore, in the rest of this section we give an introduction to interval branch–and–bound methods for global optimization (including the role of local search within it), to the COCONUT Environment, and to the COCONUT interfaces to local search engines.

1.1. Interval branch–and–bound (B&B) methods. Interval analysis is an efficient and convenient tool for bounding ranges of functions over rectangular domains. Interval versions of the arithmetic operations and elementary functions are implemented by interval arithmetic libraries (see, e.g. [1, 15]) in a careful way with directed (outward) rounding. This ensures that the computed bounds to the range of the function are mathematically correct (i.e., they enclose the real range) even when the calculations are done with finite precision floating point numbers. With the exception of some simple cases, however, the calculated interval range is not the precise

∗This work was supported by the FWF (Austrian Science Fund) Grant Nr. P18704-N13.
†Faculty of Mathematics, University of Vienna, Nordbergstr. 15, A-1090 Vienna, Austria. Email: Mihaly.Markot@univie.ac.at, Hermann.Schichl@esi.ac.at
real range, but a wider interval containing some overestimation. Since the analysis of certain properties of functions often reduces to the range bounding problem of some related function (e.g., monotonicity or convexity checking) these can also be done with mathematical rigor using interval tools. Interval calculations will not be directly used in the present paper, so for more details we only refer to the interval textbooks like [13, 17], and many others.

Interval analysis is very natural to be employed in a rectangular branch-and-bound global optimization framework, and is the basic part in the design of solvers for complete search: with interval B&B, mathematically rigorous enclosures of all global minimizers and the global minimum can be determined. In interval B&B methods the original search box is split iteratively and a bound to the objective function range is determined with interval calculations over the actual box. When an interval-based tool verifies that a box or a part of it certainly does not contain any global minimizer, then the box or the respective part can be pruned from the search tree.

The role of local optimization. A key ingredient of the interval B&B method is to update the currently best-known upper bound for the global minimum. This value is used as the pruning threshold for suboptimality, but it can also improve constraint propagation methods (which are targeted to eliminate infeasible parts of the search domain), therefore, updating it efficiently is indispensable to the efficiency of the whole method. When local (usually not interval-based) solvers are available, they can be employed to produce unverified local optimizers. For an interval method, however, such a raw solution is not of immediate use: it must be verified that there is a box in the neighborhood of the provided solution which certainly contains a feasible (or even better, a locally optimal) point. Only then the upper bound of the interval evaluation of the objective function over this box can serve as a guaranteed upper bound for the minimum. The acquisition of such guaranteed information from an unverified point is an intensively researched field in the interval community. One of the performance indicators used in the current study – the solution check indicator – will be a heuristic indicator which measures the frequency of getting the verified answer from the point returned by the local solver.

1.2. The COCONUT Environment. The COCONUT Environment (Continuous Constraints – Updating the Technology) is an open-source software platform for complete global optimization and constraint satisfaction problems. The COCONUT project, started in 2000, was initially funded by an IST programme of the European Community (IST-200-26063, 2000–2004), and has been continued since then on a smaller scale.

The COCONUT Environment (called shortly COCONUT in the sequel) is designed to be modular and extensible with newly developed or existing solver components. Its central component is the COCONUT API (application programmer’s interface), mostly written in C++. Among others, the API contains tools for building an internal representation (a model) of the global optimization problem. The model is given as a directed acyclic graph (DAG) consisting of expression nodes for the various mathematical operations, and additional information on the constraints, the objective function, the current variable bounds, etc.

Other API components are the search graph, the evaluators, management modules, report modules, and inference engines and graph analyzers. The search graph is a DAG with a model in each node, representing the branch-and-bound procedure. Evaluators evaluate problem functions and their higher order derivatives (both with real and interval data). Management modules are used to initialize and manipulate
the search graph and the respective models (e.g., doing branching by splitting a node of the search graph). The report modules provide output related to the respective problem (problem specification, structural information, solution, etc.) in various formats. The graph analyzers and the inference engines do most of the work while solving a problem: graph analyzers work on the search graph (e.g., they cut off leaves if the respective model provides suboptimal solutions only), while inference engines work on a specific model (e.g., they do local optimization or generate the respective Karush-John conditions). In particular, external optimization solvers (including the local solvers used in this paper) are connected to the environment as inference engines.

2. Methods for interfacing solvers with the COCONUT API. The communication between the solver and COCONUT is implemented in one of the following ways:

1. Object-oriented interface with problem instance objects. Solvers written in an object-oriented language (e.g., in C++) usually provide object-oriented interfaces by having a (base) class of problem descriptions. The software which calls the solver (in our case, the COCONUT environment itself) needs to have a class derived from this base class, where the necessary problem information is evaluated by the caller (in a COCONUT routine) and is specified by overloading the base class member functions (i.e., initialization, evaluation and postprocessing routines). Upon execution, a problem instance and a solver instance is created in the COCONUT interface and then the problem class is passed to the solver for solving. Beside its transparency, the main advantage of this method is that, since the information is passed between the solver and the environment through the member functions, the solver calls the respective routines only when it is necessary, without causing overhead or using additional control parameters.

2. Procedural interface. The other type of COCONUT interface (usually provided by solvers implemented in C or Fortran) is procedure oriented. There is typically one main solver routine to be called (perhaps together with initialization and postprocessing routines). There are two principal ways of communication between the solver and the evaluation functions:

2a. The main routine needs to be called only once, and the evaluation routines are pre-declared callback functions which should be defined by the user. This way of communication is often denoted as callback communication.

2b. The main routine is called in a loop, and its requests and status are communicated through the call parameters and a returned status code. E.g. if a call to the main routine returns with the status “request a constraint gradient evaluation”, then the slots in the parameter list corresponding to the gradient should be filled before calling the main routine again. This way of information exchange is often called reverse communication.

In principle, any external solver supporting one or more of the above object-oriented, callback, or reverse communication modes can be interfaced with COCONUT. It is, however, also recommended that the solver is re-entrant — e.g., it does not use global variables — in order that the interface stays valid in future distributed COCONUT applications.

3. The local optimization solvers.
3.1. Common properties and requirements. Unless noted otherwise, all investigated solvers are designed for solving general nonlinear constrained problems.

The problem information passed commonly to the solvers consists of
- the number of variables and constraints,
- the bounds on the variables and constraints,
- the starting point (we specify starting values only for the primal variables).

The values of the objective and constraint functions evaluated at the specified input points must be provided externally for all solvers. The requests for first and second order derivatives are more versatile: in some cases they are mandatory to be supplied, in others they can be approximated internally. Since first order derivatives are evaluated in COCONUT efficiently with backward automatic differentiation, in this study we always provide them externally, even if they are optional.

For second order information, however, the choice between explicitly supplied values and internal approximations can lead to substantial differences in efficiency, and the right selection can obviously depend on the particular problem properties. Therefore, for solvers which offer a choice between externally supplied and internally approximated Hessians, we consider both possibilities and use different solver subversions in the comparison test based on this distinction. The details of this are given in the solver introductory sections below. (In COCONUT, Hessians are evaluated as compositions of Hessian-vector products with backward AD. The number of necessary Hessian-vector product evaluations needed to construct the whole Hessian are reduced by a Hessian-coloring algorithm, namely, Algorithm 4.1 of [10].)

With the exception of DONLP3 and LBFGS-B, all of our tested solvers strongly utilize sparsity information. Therefore COCONUT also passes this information for the Jacobian, and, in cases second order derivatives are supplied, for the Hessian of the Lagrangian. The sparsity information passed always consists of the number and the position of the nonzeros of the respective matrix.

Many solvers can strongly utilize if the problem has some special structure, most typically, if it is known to be linear, linearly constrained, or quadratic. Since this can be easily determined by COCONUT, we always supply this extra information whenever the solver can use it.

Next we give a short description of the solver variants used in the study regarding their methodology and availability. In addition, in Table 3.1 we list some (comparable) characteristics of the solvers, including software version number, implementation language, licensing, sparsity utilization, the type of interfacing with COCONUT, and the need for supplying Hessians externally.

3.2. DONLP3. Methodology. The DONLP3 solver, written by P. Spellucci, implements a variant of the sequential quadratic programming (SQP) method [23, 24]. At each iteration, a correction to the current iterate is computed by solving a convex quadratic model of the problem. After a line search direction for the next iterate is determined, an appropriate stepsize is determined by using an $\ell_1$ exact penalty merit function. To avoid the evaluation of all constraint gradients at each iterate, DONLP3 maintains a set of working constraints. DONLP3 does not utilize sparsity information and there is no need to supply second order derivative information to it. The software is originated from Spellucci's F77 DONLP2 algorithm.

Availability. DONLP3 is freely available for research purposes from P. Spellucci or from the COCONUT development team.

3.3. IPOPT. Methodology. IPOPT is a primal-dual interior point algorithm with a line search filter method by A. Wächter et. al. [18, 27, 29, 28, 30]. The primal-
Table 3.1
Summary table of the local optimization solvers interfaced to COCONUT.

<table>
<thead>
<tr>
<th>name</th>
<th>version</th>
<th>lang.</th>
<th>licensing</th>
<th>sparse?</th>
<th>interf.</th>
<th>Hessian supplied?</th>
</tr>
</thead>
<tbody>
<tr>
<td>DONLP3</td>
<td>3</td>
<td>C++</td>
<td>free for research</td>
<td>no</td>
<td>2a.</td>
<td>no</td>
</tr>
<tr>
<td>IPOPT-H</td>
<td>3.5.1</td>
<td>C++</td>
<td>CPL (free)</td>
<td>yes</td>
<td>1.</td>
<td>yes</td>
</tr>
<tr>
<td>IPOPT-qN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LBFGS-B</td>
<td>2.1</td>
<td>F77</td>
<td>free</td>
<td>no</td>
<td>2b.</td>
<td>no</td>
</tr>
<tr>
<td>LOQO</td>
<td>6.07</td>
<td>C</td>
<td>proprietary</td>
<td>yes</td>
<td>2a.</td>
<td>yes</td>
</tr>
<tr>
<td>KNITRO-H</td>
<td>5.1.2z</td>
<td>C/C++</td>
<td>proprietary</td>
<td>yes</td>
<td>2b.</td>
<td>as $H_L \cdot v$</td>
</tr>
<tr>
<td>KNITRO-FD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>SNOPT</td>
<td>6</td>
<td>F77</td>
<td>proprietary</td>
<td>yes</td>
<td>2a.</td>
<td>no</td>
</tr>
</tbody>
</table>

dual KKT equations are solved by a damped Newton method; the resulting search
direction is then passed to a line search algorithm. IPOPT uses a filter method [9]
in the line search, which means that a trial point during search is considered acceptable
when it improves either the (barrier) objective function value or the constraint
violation (in contrast to the more traditional merit functions combining these two
measurements into one quantity).

IPOPT requires at least one sparse symmetric linear solver as third party software.
In our study we used MUMPS (a multifrontal sparse direct solver) version 4.7.3 from
ENS Lyon, France for this purpose.

Second order derivatives (the Hessian of the Lagrangian) can be either supplied
explicitly or can be approximated by IPOPT internally with a limited-memory quasi-
Newton method. In the present study we employ and compare two versions of IPOPT:
- IPOPT-H: Hessians evaluated by COCONUT,
- IPOPT-qN: internally approximated Hessians with quasi-Newton.

In the interface routines we also supply IPOPT with the number and list of
nonlinear variables (this is only optionally required by the solver, but is useful when
the quasi-Newton approximation is selected).

Availability. IPOPT is freely available together with its source code from COIN-OR
(www.coin-or.org) under the Common Public License.

3.4. LBFGS-B. Methodology. LBFGS-B is a limited memory quasi-Newton
solver by Byrd et. al. [6], applicable for bound constrained problems only in its pre-
sented form. LBFGS-B proceeds by updating a quadratic model to the problem by
a limited memory BFGS approximation of the Hessian of the objective. The ma-
trices are represented in a special compact form [7], which is well-suited for bound
constrained problems. In each iteration the search direction is obtained by first deter-
mining the active variables with a gradient projection method and then minimizing
the quadratic model in the free variables. Then a line search algorithm presented in
[16] is performed.

Availability. LBFGS-B is freely available under the General Public License from J.
Nocedal (http://www.ece.northwestern.edu/~nocedal/lbfgsb.html).

3.5. LOQO. Methodology. LOQO is a primal-dual interior point solver devel-
oped by R. J. Vanderbei [21, 25, 26]. The successive search directions are obtained
by solving the appropriate primal-dual KKT system with sparse $LDL'$ factorization.
After the search direction is evaluated, a line search is performed (to achieve sufficient decrease of an $\ell_2$ merit function) to get the next iterate. LOQO mandatorily requires derivative information up to second order.

LOQO is connected to COCONUT through a type 2.a interface. Note, however, that LOQO provides access to its whole internal problem representation (a structure called LOQO) through a pointer, therefore, the interface is programmed similarly to those of Type 1., but employing a C structure instead of a C++ class.

Availability. The most recent versions of LOQO are available for download from their author at www.princeton.edu/~rvdb/. The full version of LOQO is licensed.

3.6. KNITRO. Methodology. KNITRO is a commercial package designed primarily for large-scale nonlinear optimization. For nonlinear problems KNITRO employs three different optimization algorithms: an interior point method (referred as Interior/Direct) which solves the primal-dual KKT-system with direct linear algebra methods [31], a similar interior point method (referred as Interior/CG) with a projected conjugate gradient method for the KKT-system [5, 2], and an active set method (referred as Active Set) with a sequential linear-quadratic programming (SLQP) algorithm [3, 4]. A summary of the algorithms implemented in KNITRO is given in [8]. KNITRO has built-in strategies to automatically select the probably best-suited algorithm (depending on the characteristics of the problem) and to switch from Interior/Direct to Interior/CG if the direct steps behave poorly.

There are many choices for the KNITRO user to specify the evaluation of the gradients and Hessians. As mentioned above, the gradients are always supplied by COCONUT. For the Hessian evaluations/approximations we selected three different options, which determine the KNITRO variants of the present study:

- KNITRO-H: Hessians evaluated by COCONUT,
- KNITRO-FD: internal Hessian-vector product approximations with finite differences, and
- KNITRO-Hv: Hessian-vector products (with vectors specified by KNITRO) evaluated by COCONUT.

These choices are made in accordance with the KNITRO manual: it recommends the exactly supplied Hessians or Hessian-vector products for best general performance, otherwise, when exact gradients are available, the finite differences approximation is suggested.

Availability. KNITRO is a licensed product of Ziena Optimization, Inc. General information is provided at www.ziena.com/knitro.html.

3.7. SNOPT. Methodology. SNOPT is a software using a sparse sequential quadratic programming (SQP) method of Gill et. al. [11].

In the general nonlinear case, SNOPT considers a special modified Lagrangian (based on constraint linearization) and at each iteration it builds a convex quadratic model of this Lagrangian at the current iterate. The resulting QP is solved with the active set method SQOPT [12]. The solution gives a search direction for the next iterate, which is determined by a line search on an augmented Lagrangian merit function.

SNOPT approximates the Hessians of the modified Lagrangian with a limited-memory quasi-Newton (BFGS) method, therefore, no second order information on the problem needs to be provided externally.

The solver is designed especially for large problems with moderate degrees of freedom. In general it performs a relatively small number of function evaluations,
therefore it is particularly suitable when the problem functions are expensive to evaluate.

The SNOPT 6 package offers various interfaces to invoke the solver and specify the problem details. For the COCONUT Environment we chose the snOptA interface, which is the generally recommended easy-to-use option starting from Version 6.

SNOPT does not require any second order information to be supplied.

Availability. SNOPT is a proprietary software available from Stanford Business Software Inc. ([http://www.sbsi-sol-optimize.com/asp/sol_products_snopt.htm](http://www.sbsi-sol-optimize.com/asp/sol_products_snopt.htm)).

4. The testing environment. We run all local solver variants on the COCONUT test set [22] consisting of a total of 1307 optimization and constraint satisfaction problems. With the exception of LBFGS-B, which was run only on the bound-constrained and unconstrained problems (a total of 253 instances), all solvers were tested on all problems.

We generated 20 random starting points for each problem, that is, every problem was attempted to be solved 20 times per solver. The starting points were generated in a way that for unbounded variables starting values with a smaller absolute value were picked up with a bigger probability. Let us denote the problem bounds by \([x^l_i, x^u_i]\), \(i = 1, \ldots, n\), where \(x^l_i \leq x^u_i, x^l_i, x^u_i \in \mathbb{R} \cup \{-\infty, \infty\}\) and \(n\) is the problem dimension. The starting points were obtained as follows: we first evaluated \([v^l_i, v^u_i] := [x^l_i, x^u_i] \cap [-10^8, 10^8]\), then we determined a symmetric finite interval \([-\beta, \beta]\) by

\[
\beta = \begin{cases} 
10 & \text{if } [v^l_i, v^u_i] = [-10^8, 10^8] \\
\max\{b \in \mathbb{R} : [-b, b] \subseteq [v^l_i, v^u_i]\} & \text{otherwise},
\end{cases}
\]

and finally, with a probability of 0.9 we chose a random point with uniform distribution from \([-\beta, \beta]\) and with a probability of 0.1 we chose a point from \([v^l_i, v^u_i] \setminus [-\beta, \beta]\) according to a special probability density function with quadratic decay toward \(\pm \infty\).

For the precise details on the random point generation and on the aforementioned density function we refer to the \texttt{random} routine of the COCONUT API.

The problems were classified by their dimension \(n\) as ‘tiny’ (1 \(\leq n \leq 10\), ‘small’ (11 \(\leq n \leq 100\), ‘large’ (101 \(\leq n \leq 1000\), and ‘huge’ (1001 \(\leq n\)) problems, and according to this, the maximal allowed running time was set to 10, 60, 300, and 600 seconds, respectively. In addition, we set up a hard time limit of 1000 seconds; if this amount of time elapsed and the solver was still running, we externally killed the optimization process. (This happened mostly when large sparse problems were tried to be solved with the obviously not suitable dense DONLP3 solver, and in a few other cases when the problem was so big that program control did not reach the first time-checkpoint of the solver code.)

The solvers were executed with their default parameters except for the time limit and those additional settings specified in Section 3. Due to the excessive computational requirements, the tests were run on a cluster of 32 PCs, each with two dual core AMD Opteron 2.4 GHz CPUs and 16 GB of RAM.

5. Methodology of solver comparison.

5.1. The performance indicators. In this section we detail the three basic criteria we use to measure the quality of the solutions reported by the local solvers. It is important to emphasize that these criteria somewhat differ from the ones traditionally used in benchmarking studies, since we consider the use and efficiency of local optimization \textit{within} interval global optimization algorithms.
In the first place, we expect that the solution is ‘useful’ in the sense that it can be used as a candidate for a verified feasible solution. Namely, the reported local minimizer is expected to be at least ‘almost’ feasible, and the objective function value attained at the minimizer is expected to be ‘consistent’ with the reported minimum value. These expectations are checked by the solcheck algorithm of COCONUT. Thus, here we use the simplifying assumption that a reported solution passes the solcheck test if and only if a verified feasible solution can be acquired from it by an interval algorithm. (Such automatic feasibility verification algorithms, e.g. ones based on [14, 20] are currently under development in the COCONUT Environment.) In the present case the tolerances were set so that a solution passes the solcheck test if (i) the relative error between the reported optimum and the objective function value evaluated at the reported optimizers is at most $10^{-3}$, and (ii) the relative constraint violation evaluated at the reported optimizer is at most $10^{-3}$ for all constraints. (For CSPs, condition (i) holds by definition.) Since the solvers have a wide variety of sometimes incomparable built-in stopping criteria, the solcheck tolerance values are set intentionally big enough to expect that it will not interfere with these criteria, that is, when a solvers stops and the claimed solution is indeed an approximate optimizer, then it will not be rejected by the solcheck test purely because of an expected higher precision. The solcheck indicator $S^s_p$ regarding the performance of a solver $s$ on a test problem $p$ is defined by the number of times the delivered solution passes the solcheck test, i.e. $0 \leq S^s_p \leq 20$.

Secondly, two local optimization methods can be compared by their contribution to the progress of the global search. Two sub-criteria may be considered here: the first is the ability to deliver verifiable local optimizers at different regions of the search space and different phases of the global algorithm – this is basically equivalent with the solcheck indicator above. The second is the effort towards finding the global minimum, which can be measured by the best reported local minimum $\hat{F}^s_p \in \mathbb{R} \cup \{-\infty\}$ passing the solcheck test during the multistart testing. If $S^s_p = 0$, we set $\hat{F}^s_p = \infty$. For brevity we also call this indicator the function value indicator.

The third criterion of comparison is the amount of computational effort to obtain (approximate) local minimizers, which is simply represented by the solver running time including pre- and postprocessing (in seconds). This can obviously be measured within the COCONUT Environment in a uniform way for all solvers. Note that other indicators such as the number of evaluations are less suitable for comparison reasons in the present case: different solvers use evaluations up to different order (which are hard to compare), and, perhaps more importantly, the amount of computation spent between evaluations is also important to be estimated. The running time indicator will be denoted by $T^s_p \geq 0$. If a solver passes the hard time limit for a given problem instance, we declare the solver is not advised to run at all on that problem w.r.t. the time requirement and set $T^s_p = 10^{10}$.

There is, of course, a natural trade-off between quantifying or ranking by the usefulness, the function value and the time criteria: a solver can, for instance, produce worse objective function value, but in a shorter time than an other. Indeed, it is neither easy nor practical to create a good general combined measurement of these criteria. Therefore, during the performance analysis and the prediction of the expected solver behavior we consider all three indicators separately. When the prediction system is in use for a new problem, the predicted indicators can be combined into one single measurement – depending e.g. on the particular problem characteristics and the user’s expectations – to determine the final solver selection.
Notation. The symbols for the raw performance indicator values are marked with a ‘hat’ accent, as above; the transformed values (scores) introduced below (and used the most frequently in the paper) will be denoted with the respective letter without accent. A prediction of a performance score will be denoted by a ‘bar’ accent.

5.2. Transforming the performance indicators into scores. In the next step we transform the raw indicators above into a score between 0 and 100 for each problem and solver variant. This is done because not only the different solvers should be compared on each problem, but the individual solver behaviors should be comparable problem to problem in order to create a prediction (a fit) of the indicators.

Transforming the solcheck indicator \( \hat{S}_p^s \) is obvious; we just set \( S_p^s := 5 \cdot \hat{S}_p^s \). The case of the local minimum indicator \( \hat{F}_p^s \) is a bit more tricky: for each problem \( p \), if, at first, \( \hat{F}_p^s = \infty \), \( \forall s \), then we set \( \hat{F}_p^s = 0 \), \( \forall s \) (none of the solver results passed the solcheck test). Otherwise, there is at least one solver for which \( \hat{F}_p^s \) is finite; let us denote \( S' \) the set of such solvers, and let us denote \( f_{p,m} = \min \{ \hat{F}_p^s \mid s \in S' \} \). For \( s \not\in S' \), we set \( \hat{F}_p^s = 0 \). In order to avoid unfair scoring due to the different stopping criteria and tolerances of the solvers, we consider the quality of \( \hat{F}_p^s \) and \( \hat{F}_p^t \), \( s, t \in S' \) to be the same, if either the absolute or the relative difference of the two values is at most \( 10^{-3} \). Clearly, the above relation of “having the same quality” is reflexive and symmetric, but not transitive. By making the transitive closure we turn the relation into an equivalence relation which gives a partitioning of \( S' \). (As experience shows, taking the transitive closure does not significantly change the relation; indeed, the absolute/relative difference between any two elements within a partition never exceeded \( 2 \cdot 10^{-3} \).) Then, if \( s \in S' \) belongs to the partition \( P \subseteq S' \), we replace \( \hat{F}_p^s, s \in P, P \subseteq S' \) with \( \min \{ \hat{F}_p^s \mid s \in P \} \).

Now we set \( F_p^s, s \in S' \) relative to \( f_{p,m} \). If \( |f_{p,m}| \leq 10^{-3} \), then we decide by the absolute difference of the (modified) local minimum indicators: for each \( s \in S' \), if \( |\hat{F}_p^s - f_{p,m}| \leq 10^{-3} \), then let \( F_p^s = 100 \), otherwise let \( F_p^s = 15 \). If \( |f_{p,m}| > 10^{-3} \), then we decide by the relative difference of the (modified) local minimum indicators: for each \( s \in S' \), if \( r := (|\hat{F}_p^s - f_{p,m}|)/f_{p,m} > 1 \), let \( F_p^s = 15 \), otherwise let \( F_p^s = 15 + 85 \cdot (1 - r) \). That is, in the latter case we introduce a linear decay in the score between 15 and 100. This way we have \( F_p^s = 0 \) whenever \( \hat{F}_p^s = \infty \) and \( 15 \leq F_p^s \leq 100 \) when \( \hat{F}_p^s \) is finite.

When transforming the running time indicators \( \hat{T}_p^s \) we run into another difficulty: since the fitting is done by taking linear combinations of the scores, the score must be linear in the sense that if the differences between two pairs of running times are ‘empirically’ the same, then they should be mapped into a score where the respective numerical differences are the same as well. (This criterion holds well for \( \hat{S}_p^s \) and \( \hat{F}_p^s \).) For instance, the differences between 0.09 and 0.1 seconds and between 100.09 and 100.1 seconds are empirically not the same, hence a nonlinear scaling is required.

We found that a logarithmic transformation is appropriate for this purpose: since \( T_p^s \) varies between 0 and around \( 10^5 \) with the resolution of 0.01 when the hard time limit is not passed, and is set to \( 10^{10} \) otherwise, we define \( T_p^s \) as \( (10 - \log_{10}(\min(\hat{T}_p^s + 0.01, 10^{10})))) \cdot 100/12 \). (The additive term and the multiplicative factor outside the log are used to scale into \([0, 100]\).)

6. A summary of the numerical results. Since we run altogether 9 variants of 6 different state-of-the-art local optimization solvers on a uniquely wide set of test problems, we believe that publishing the aggregate comparative results of the
Aggregate results on the bound constrained problems (253 problem instances). For each problem group and performance indicator we list the solver rankings based on the average scores. The respective scores are given in parenthesis after the rank numbers. The sizes of the problem groups are given next to the group names in the header lines.

<table>
<thead>
<tr>
<th></th>
<th>tiny (137)</th>
<th></th>
<th>small (24)</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DONLP3</td>
<td>1 (96.7)</td>
<td>3 (91.4)</td>
<td>1 (93.6)</td>
<td>2 (99.2)</td>
<td>2 (92.7)</td>
<td>3 (82.6)</td>
</tr>
<tr>
<td>IPOPT-H</td>
<td>6 (93.6)</td>
<td>1 (93.5)</td>
<td>8 (84.0)</td>
<td>6 (91.0)</td>
<td>7 (84.9)</td>
<td>9 (72.5)</td>
</tr>
<tr>
<td>IPOPT-qN</td>
<td>3 (94.9)</td>
<td>8 (83.5)</td>
<td>9 (82.1)</td>
<td>5 (91.3)</td>
<td>5 (88.0)</td>
<td>5 (77.6)</td>
</tr>
<tr>
<td>KNTIRO-FD</td>
<td>8 (91.9)</td>
<td>9 (81.9)</td>
<td>4 (91.3)</td>
<td>8 (87.1)</td>
<td>8 (83.6)</td>
<td>4 (80.0)</td>
</tr>
<tr>
<td>KNTIRO-H</td>
<td>4 (94.5)</td>
<td>4 (91.3)</td>
<td>5 (89.7)</td>
<td>4 (91.5)</td>
<td>4 (88.7)</td>
<td>7 (75.8)</td>
</tr>
<tr>
<td>KNTIRO-Hv</td>
<td>5 (94.1)</td>
<td>7 (84.7)</td>
<td>6 (89.0)</td>
<td>7 (89.4)</td>
<td>6 (86.1)</td>
<td>6 (76.1)</td>
</tr>
<tr>
<td>LOQO</td>
<td>9 (77.7)</td>
<td>5 (90.7)</td>
<td>7 (86.8)</td>
<td>9 (86.5)</td>
<td>9 (75.7)</td>
<td>8 (72.8)</td>
</tr>
<tr>
<td>SNOPT</td>
<td>7 (92.8)</td>
<td>2 (83.3)</td>
<td>3 (93.3)</td>
<td>3 (94.6)</td>
<td>1 (92.9)</td>
<td>2 (83.7)</td>
</tr>
<tr>
<td>LBFGS-B</td>
<td>2 (95.4)</td>
<td>6 (87.5)</td>
<td>2 (93.5)</td>
<td>1 (99.4)</td>
<td>3 (91.8)</td>
<td>1 (85.5)</td>
</tr>
<tr>
<td>S</td>
<td>1 (99.9)</td>
<td>1 (75.7)</td>
<td>9 (62.1)</td>
<td>6 (81.1)</td>
<td>7 (33.6)</td>
<td>9 (44.9)</td>
</tr>
<tr>
<td>F</td>
<td>8 (81.3)</td>
<td>5 (54.7)</td>
<td>8 (64.1)</td>
<td>7 (77.5)</td>
<td>8 (33.0)</td>
<td>3 (57.3)</td>
</tr>
<tr>
<td>T</td>
<td>7 (87.2)</td>
<td>8 (38.5)</td>
<td>3 (66.9)</td>
<td>8 (77.0)</td>
<td>6 (35.3)</td>
<td>4 (57.2)</td>
</tr>
<tr>
<td>S</td>
<td>3 (92.1)</td>
<td>6 (52.2)</td>
<td>4 (66.1)</td>
<td>5 (87.0)</td>
<td>5 (44.5)</td>
<td>6 (54.9)</td>
</tr>
<tr>
<td>F</td>
<td>4 (90.7)</td>
<td>4 (64.3)</td>
<td>6 (64.3)</td>
<td>2 (92.0)</td>
<td>3 (50.3)</td>
<td>7 (54.0)</td>
</tr>
<tr>
<td>T</td>
<td>4 (90.7)</td>
<td>7 (48.1)</td>
<td>5 (64.4)</td>
<td>3 (91.1)</td>
<td>4 (49.6)</td>
<td>8 (52.2)</td>
</tr>
<tr>
<td>S</td>
<td>6 (87.5)</td>
<td>3 (64.7)</td>
<td>7 (64.2)</td>
<td>4 (89.7)</td>
<td>2 (59.6)</td>
<td>5 (56.8)</td>
</tr>
<tr>
<td>F</td>
<td>9 (39.3)</td>
<td>9 (28.7)</td>
<td>1 (74.6)</td>
<td>9 (14.1)</td>
<td>9 (13.7)</td>
<td>1 (71.6)</td>
</tr>
<tr>
<td>T</td>
<td>2 (97.2)</td>
<td>2 (67.9)</td>
<td>2 (70.5)</td>
<td>1 (92.6)</td>
<td>1 (68.8)</td>
<td>2 (60.9)</td>
</tr>
</tbody>
</table>

Tests is itself a value. Therefore, in Tables 6.1 and 6.2 we give a summary of these results. The test set was split into bound constrained optimization problems and equality/inequality constrained optimization problems/CSPs, and both groups were further divided by the problem size according to Section 4. For the eight resulting problem groups we ranked the solvers by their average scores for all three indicators. (In the tables the three performance indicators are abbreviated by S, F, and T, according to their formal notation above.)

The most important consequence of the results is their diversity: There is no unique best choice for all problem groups and indicators. Each solver variant performs in an excellent way for one or more groups or indicators (ranked in the top three), but occasionally performs worse on others. Although there are a few performance trends that follow easily from the basic solver properties (such as how the ranking of the dense DONLP3 solver drops while moving from smaller to bigger — thus, usually sparser — problems), most of them cannot be explained from such a traditional numerical summary. It is worth to emphasize again that the present paper is targeted to explain these non-obvious performance facts, and therefore its ultimate goal is far beyond reporting such aggregate results: in the next sections we introduce a much more sophisticated analysis of the problem set (by categorical and numerical features) and a method to predict future solver behavior from existing performance results.

7. Prediction methods. Predicting the most suitable solver for a problem w.r.t. a performance indicator can be considered as a classification problem: we first determine a feature space consisting of numerous categorical and numerical proper-
### Table 6.2
Aggregate results on the equality/inequality constrained problems (1050 problem instances). The table is organized the same way as Table 6.1.

<table>
<thead>
<tr>
<th>Solver</th>
<th>tiny (551)</th>
<th></th>
<th></th>
<th>small (208)</th>
<th></th>
<th></th>
<th>huge (153)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S</td>
<td>F</td>
<td>T</td>
<td>S</td>
<td>F</td>
<td>T</td>
<td>S</td>
</tr>
<tr>
<td>DONLP3</td>
<td>2 (83.3)</td>
<td>1 (94.7)</td>
<td>2 (93.1)</td>
<td>4 (71.6)</td>
<td>5 (82.8)</td>
<td>2 (81.3)</td>
<td></td>
</tr>
<tr>
<td>IPOPT-H</td>
<td>3 (81.4)</td>
<td>3 (93.0)</td>
<td>7 (84.6)</td>
<td>3 (78.1)</td>
<td>2 (83.4)</td>
<td>5 (79.5)</td>
<td></td>
</tr>
<tr>
<td>IPOPT-qN</td>
<td>1 (84.0)</td>
<td>2 (93.1)</td>
<td>8 (84.4)</td>
<td>1 (83.9)</td>
<td>1 (86.2)</td>
<td>3 (80.2)</td>
<td></td>
</tr>
<tr>
<td>KNITRO-FD</td>
<td>7 (69.3)</td>
<td>7 (86.8)</td>
<td>3 (88.4)</td>
<td>7 (65.7)</td>
<td>7 (80.6)</td>
<td>4 (79.9)</td>
<td></td>
</tr>
<tr>
<td>KNITRO-H</td>
<td>5 (72.9)</td>
<td>4 (90.4)</td>
<td>4 (87.9)</td>
<td>5 (70.4)</td>
<td>4 (82.9)</td>
<td>6 (79.0)</td>
<td></td>
</tr>
<tr>
<td>KNITRO-Hv</td>
<td>6 (69.8)</td>
<td>6 (87.3)</td>
<td>6 (86.1)</td>
<td>6 (66.3)</td>
<td>6 (81.7)</td>
<td>8 (76.6)</td>
<td></td>
</tr>
<tr>
<td>LOQO</td>
<td>8 (68.6)</td>
<td>8 (84.6)</td>
<td>5 (87.0)</td>
<td>8 (40.8)</td>
<td>8 (57.7)</td>
<td>7 (77.8)</td>
<td></td>
</tr>
<tr>
<td>SNOPT</td>
<td>4 (78.5)</td>
<td>5 (89.3)</td>
<td>1 (94.4)</td>
<td>2 (79.7)</td>
<td>3 (83.3)</td>
<td>1 (88.5)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>large (138)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>F</td>
<td>T</td>
<td>S</td>
<td>F</td>
<td>T</td>
<td>S</td>
</tr>
<tr>
<td>DONLP3</td>
<td>7 (45.5)</td>
<td>7 (48.3)</td>
<td>8 (56.7)</td>
<td>8 (4.2)</td>
<td>8 (5.5)</td>
<td>8 (11.9)</td>
<td></td>
</tr>
<tr>
<td>IPOPT-H</td>
<td>3 (67.2)</td>
<td>3 (71.4)</td>
<td>3 (66.2)</td>
<td>2 (60.7)</td>
<td>4 (63.9)</td>
<td>3 (55.0)</td>
<td></td>
</tr>
<tr>
<td>IPOPT-qN</td>
<td>1 (73.0)</td>
<td>2 (75.5)</td>
<td>2 (68.7)</td>
<td>1 (78.7)</td>
<td>1 (84.8)</td>
<td>2 (60.5)</td>
<td></td>
</tr>
<tr>
<td>KNITRO-FD</td>
<td>4 (57.6)</td>
<td>5 (65.5)</td>
<td>3 (66.2)</td>
<td>4 (50.4)</td>
<td>2 (69.3)</td>
<td>4 (53.1)</td>
<td></td>
</tr>
<tr>
<td>KNITRO-H</td>
<td>5 (56.7)</td>
<td>4 (66.2)</td>
<td>6 (65.1)</td>
<td>3 (54.5)</td>
<td>5 (60.4)</td>
<td>5 (53.0)</td>
<td></td>
</tr>
<tr>
<td>KNITRO-Hv</td>
<td>6 (53.7)</td>
<td>6 (63.6)</td>
<td>7 (63.3)</td>
<td>5 (49.6)</td>
<td>3 (64.8)</td>
<td>5 (53.0)</td>
<td></td>
</tr>
<tr>
<td>LOQO</td>
<td>8 (12.9)</td>
<td>8 (26.9)</td>
<td>5 (65.3)</td>
<td>7 (9.5)</td>
<td>7 (23.2)</td>
<td>7 (52.0)</td>
<td></td>
</tr>
<tr>
<td>SNOPT</td>
<td>2 (69.0)</td>
<td>1 (76.3)</td>
<td>1 (72.1)</td>
<td>6 (37.7)</td>
<td>6 (45.3)</td>
<td>1 (61.2)</td>
<td></td>
</tr>
</tbody>
</table>

ties of the optimization problems, then we create a mapping (the prediction) from the points of the feature space to the set of applicable solvers. The problem is then classified by finding the solver with the best predicted value.

COCONUT provides many tools to extract categorical and numerical properties of optimization problems. These, like evaluators, propagate the respective pieces of information through the model DAG. All problem information we use in this study can be extracted fast by COCONUT (with an effort proportional to a small constant multiple of the number of nodes).

Our strategy consists of two phases: first, we group the problems by taking into account the categorical features of the problems only. In the second phase, for each such group we create the prediction by an appropriate fitting method using the numerical features.

#### 7.1. Dealing with the categorical features
The categorical grouping should be descriptive enough to be able to catch the different behaviors of the local solvers, but on the other hand, the size of each group should be large enough so that a decent fit can be created. The grouping described below is therefore a somewhat subjective compromise between these two goals, applied specially to the current set of problems.

The two most important categorical features are the polynomial degree and the convexity information about both the objective and the constraints. Since all local solvers use function information up to second order, the polynomial degree feature will be differentiated up to degree of two; higher order polynomial degrees and non-polynomial functions will be referred to as ‘other’ w.r.t. this feature. The analyzer for the convexity information [19] returns either with verified convexity, verified concavity, or unproven convexity (i.e. uncertainty w.r.t. convexity). We refer to the last two types of outputs as ‘nonconvex’ below. After some experimentation, we ended up with the categorization displayed in the row and column headers of Table 7.1.
Table 7.1
Grouping the test problems by the categorical features.

<table>
<thead>
<tr>
<th>obj\cstr</th>
<th>no</th>
<th>bnd</th>
<th>lin</th>
<th>q-cvx</th>
<th>o-cvx</th>
<th>q-ncvx</th>
<th>o-ncvx</th>
</tr>
</thead>
<tbody>
<tr>
<td>const</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>lin</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>q-cvx</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>o-cvx</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>q-ncvx</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>o-ncvx</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

For the objective function (table row headers), we have `const` for constant (or missing) objective, `lin` for linear, `q-cvx` for quadratic convex, `o-cvx` for other convex, `q-ncvx` for quadratic nonconvex, and `o-ncvx` for other nonconvex functions, respectively.

For the constraints and the feasible set (table column headers), we have `no` for unconstrained problems, `bnd` for bound constrained problems, `lin` for linearly constrained problems, `q-cvx` for quadratically constrained problems with convex feasible set, `o-cvx` for other problems with convex feasible set, `q-ncvx` for quadratically constrained problems with nonconvex feasible set, and `o-ncvx` for other problems with nonconvex feasible set.

The above categorization gives a good resolution according to the possible solver behaviors, however, many cells of the table do not contain enough problems to create a fit. To satisfy the second goal mentioned above, we merged some cells, as described by the numbers in the table. The resulting 7 groups will be the ones used throughout the analysis of the numerical features. These are:

0: the problems that are treated as ‘trivial’ from the global optimization point of view; these problems either have no solution or they can be solved as a system of linear equations, an LP, or a convex QP. This can be done efficiently by standard LP software, such as CPLEX or FICO Xpress (139 problem instances).
1: Nonconvex general QCQPs (137),
2: nonconvex, non-quadratic problems (290),
3: convexly constrained general nonconvex optimization problems (132),
4: unconstrained/bound constrained problems (218),
5: general convex optimization problems (39), and
6: nonlinear CSPs (352).

In the sequel $G(p)$ denotes the categorization of a given problem $p$; that is, $G$ maps $p$ into the respective categorical group.

7.2. Dealing with the numerical features. Throughout the study we use 8 different numerical features to represent the problem characteristics. These are:
- the number of variables (denoted by $\hat{n}_p$ for problem $p$),
- the number of nodes in the DAG representation ($\hat{d}_p$), which is one way to represent the complexity of evaluating the objective and the constraints,
- the number of bound constraints ($\hat{b}_p$),
- the number of equality constraints ($\hat{e}_p$),
- the number of inequality (non-bound) constraints ($\hat{i}_p$),
- the sparsity ratio of the extended Jacobian matrix (the Jacobian extended with the objective gradient), i.e. the ratio of the number of nonzeros and the total number
of elements ($\hat{J}_p$),
- the sparsity ratio of the Hessian of the Lagrangian ($\hat{H}_p$), and
- the average $l_1$ norm of the constraint violation vector of the 20 starting points, as a rough indicator of the shape and size of the feasible set ($\hat{v}_p$).

**Scaling the feature values.** For each problem group of Section 7.1, the feature space in which we create the fit consists of the above features. To be able to use scaled Euclidean distances to measure the distance of two problems in the feature space, we need to nonlinearly scale the feature values in a way we did for the performance scores in Section 5.2. That is, if the intuitive difference of the feature values between a pair of problems is the same as the intuitive difference of the feature values between another pair, then the respective numerical difference of the transformed feature values should also be approximately the same. For simplicity we scale all feature values into $[0,1]$. The scaled feature values will be denoted by the respective letter of the unscaled feature without the ‘hat’ accent.

Given a set of problems $Q$ which belong to the same category according to Section 7.1, the numerical features of all $p \in Q$ are scaled as follows: For $\hat{a}_p, \hat{d}_p, \hat{b}_p, \hat{e}_p$, and $\hat{i}_p$ we apply a logarithmic scaling first, then scale the results into $[0,1]$, dividing by the maximal feature value within the respective problem category. That is, for the numerical feature $f \in \{\hat{a}, \hat{d}, \hat{b}, \hat{e}, \hat{i}\}$ we first evaluate $m_f := \max\{\log_{10}(\hat{f}_q + s_f) | q \in Q\}$, where $s_f$ is a small shift constant introduced to avoid domain errors for the logarithm ($s_f = 1$ for $\hat{f} \in \{\hat{b}, \hat{e}, \hat{i}\}$ and $s_f = 0$ for $\hat{f} \in \{\hat{a}, \hat{d}\}$). Then for all $p \in Q$ we set $f_p := \log_{10}(\hat{f}_p + s_f)/m_f$, if $m_f \neq 0$, and $f_p := 1$ otherwise. (The latter case occurs only if $\hat{f}_q$ is constant over $Q$, and thus all scaled feature values can be considered equal.)

For $\hat{J}_p$ and $\hat{H}_p$, we apply a scaling by taking their square roots: for $\hat{J}, \hat{H} \in \{\hat{J}, \hat{H}\}$ and for all $p \in Q$ we set $f_p := f_p^{1/2}$.

8. **Creating the fit.** Up to this point, we have defined the three performance scores for all solvers and problems, we have grouped the problems by their categorical features, and we have evaluated an 8-dimensional (scaled) numerical feature vector for all problems. Given a new optimization problem instance $p$ (as a trial problem) and a set $P_0$ of problems with known performance and feature values (as the training set), we predict the performance scores of all available solvers on $p$ by a $k$-nearest neighbor method, as described below:

First, we analyze the problem structure with the COCONUT tools and determine the categorical and numerical feature values described in Sections 7.1 and 7.2. Then we determine the respective problem category $G(p) = g$ by Section 7.1, together with the problem group $P_g := \{q \in P_0 | G(q) = g\}$, and scale the numerical features to get $f_p \in [0,1]^8$. The logarithmic scaling needs a small adjustment as compared to those given in Section 7.2, since $p$ is now not in $P_g$: we keep $m_f$ as above (i.e., do not rescale the features of the training set), but evaluate $f_p$ as $f_p := \min\{\log_{10}(\hat{f}_p + s_f)/m_f, 1\}$, if $m_f \neq 0$, and $f_p := 1$ otherwise.

Next we find those $k$ problems in $P_g$ which are the closest to $f_p$ in the numerical feature space. After some experimentation we found that $k = 5$ is a suitable value. The distances in the feature space are defined as weighted Euclidean distances.

In this study we use an easy to obtain, uniform weighting of the numerical features. The weights are set in the following way: we group the numerical features into smaller sets; the set of features reflecting problem dimensionality ($n, d$), constraints ($b, e, i$), sparsity ($J, H$), and infeasibility ($v$), respectively, and give the same weights to these
feature sets, dividing these weights equally among the relevant individual features of that set. Formally, we have \( w = (1, 1, 2, 0, 0, 0, 2, 0)^T/6 \) for problem group 4 (since for unconstrained/bound constrained problems \( e_p = i_p = 0, J_p = 1, \) and \( v_p = 0, \) so the features actually vary in a 4-dimensional subspace), and \( w = (3, 3, 2, 2, 3, 3, 6)^T/24 \) for all other problem groups.

Finally, to create the \( k \)-nearest neighbor fit for the fixed \( p, \) for all \( I \in \{ S, T, F \} \), and for all applicable solvers \( s, \) we look for the \( k \) problems \( q_i, i = 1, \ldots, k, q_i \in P_g \) with the smallest \( d_{qi} = w \cdot ||f_p - f_q||_2 \) values. The predicted performance score will be the weighted sum of the performance scores of \( q_i, \) with weights \( W_{qi} \) defined to be proportional to the inverse of the distances (with some correction to avoid division by zero): we set \( \hat{d}_q := 1/\max\{d_{qi}, 10^{-3}\} \) and \( W_{qi} := \hat{d}_q / \sum_i \hat{d}_q. \) Thus, the predicted performance scores are formally given as \( \hat{I}_s := \sum_i W_{qi} I_{qi}. \)

**Remark 8.1.** During our experiments we also tried seemingly more sophisticated weighting of the numerical features, based on the idea that the weights reflect the effect of the numerical features on the pre-known performance scores on \( P_g. \) This was done by two different methods: in the first we computed the linear correlation between the feature values and the scores on \( P_g \) and set the weights on the basis of the absolute values of the correlation coefficients. In the second method we scaled the scores into \([0, 1]\) and computed the mean absolute differences between the features and scores (as if only one feature value was at hand to create the prediction) and set the weights of the features on the basis of these error estimates. Both of these methods produce different weights for all training samples. However, somewhat surprisingly, we found that in terms of the overall quality-of-fit measurements of Section 9 these dynamic weighting methods did not give any significant improvements when compared to the fixed scaling. Our consequence is that in the \( k \)-nearest neighbor prediction method most information on the effect of the numerical features to the performance scores can be acquired by the fixed feature weighting; further (tuned) weighting methods would lead only to minor (if any) improvements.

**9. Comparative results.** In this section we investigate the quality of the proposed \( k \)-nearest neighbor fit-based prediction as compared to other prediction methods utilizing less problem information. Since the local solver predictions are needed only for problems which indeed require global optimization (i.e. are not in the trivial category 0), we make the comparisons only over the test problems from the categories 1 to 6 (a total of 1168 problems). In the sequel \( P \) denotes this problem set. The comparisons are done in a way that the predictions are computed for all problems \( p \in P \) using the performance information on all other available problems \( P \setminus \{ p \}. \) In other words, we create the prediction for all \( p \) by using \( P_0 = P \setminus \{ p \} \) as the training sample. Another way of testing the fit, namely a cross validation, is done in Section 10, where we gradually restrict the training sample to smaller and smaller sets.

The quality of the prediction can be measured either per optimization problem (i.e. what are the predicted performance indicators of all available solvers for a given problem) or per solver (i.e. how the prediction of the solver performance changes from problem to problem). For our current purpose, the first type of measurements are more relevant. The per solver predictions can be useful for studies related to the ‘sensitivity’ of a solver to the changes of the features. For the per optimization type of analysis we introduce the following measurements:

*Probability of picking one of the best solvers (\( b_t \)): let \( 0 \leq t \leq 100 \) be a tolerance value. For a given (pre-known) performance indicator and problem, a solver \( s \) is considered best with tolerance \( t \) if its indicator score differs by at most \( t \) from that
Quality of predictions for the three performance indicators. The four quality measurements are $b_1$ (probability of picking a best solver, with score tolerance $t = 1$), $w_1$ (probability of avoiding all useless solvers, with score tolerance $t = 1$), ES (mean absolute error between the pre-known and predicted scores) and ER (mean absolute error between the pre-known and predicted ranks). For $b_1$ and $w_1$ larger values, for ES and ER smaller values mean better prediction power. The proposed $k$-nearest neighbor fit is compared with the trivial, the purely categorical-based, and the combined fit.

<table>
<thead>
<tr>
<th>meas.</th>
<th>perf. ind.</th>
<th>trivial</th>
<th>categorical</th>
<th>combined</th>
<th>$k$-nearest</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_1$</td>
<td>S</td>
<td>0.82</td>
<td>0.82</td>
<td>0.83</td>
<td>0.84</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>0.83</td>
<td>0.87</td>
<td>0.89</td>
<td>0.89</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>0.55</td>
<td>0.55</td>
<td>0.60</td>
<td>0.66</td>
</tr>
<tr>
<td>$w_1$</td>
<td>S</td>
<td>0.95</td>
<td>0.96</td>
<td>0.96</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>0.94</td>
<td>0.96</td>
<td>0.96</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>0.99</td>
<td>0.99</td>
<td>1.00</td>
<td>0.99</td>
</tr>
<tr>
<td>ES</td>
<td>S</td>
<td>34.33</td>
<td>29.99</td>
<td>24.28</td>
<td>15.94</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>34.02</td>
<td>32.62</td>
<td>24.55</td>
<td>15.79</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>13.32</td>
<td>12.92</td>
<td>6.63</td>
<td>4.76</td>
</tr>
<tr>
<td>ER</td>
<td>S</td>
<td>1.87</td>
<td>1.72</td>
<td>1.67</td>
<td>1.32</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>2.05</td>
<td>1.92</td>
<td>1.85</td>
<td>1.05</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>2.08</td>
<td>1.86</td>
<td>1.44</td>
<td>1.23</td>
</tr>
</tbody>
</table>

of the best performing one (just like in Section 6). Similarly, the set of predicted best solvers with tolerance $t$ can be defined on the basis of the respective predictions for the same indicator and problem. Given these two sets and assuming that we pick a solver among the predicted best solvers with equal probability, we can determine the probability of choosing a solver from the set of (pre-known) best ones. In Table 9.1 we display the average of these probabilities over all problems, that is, we have an aggregate measurement for each of the performance indicators.

Probability of avoiding all ‘useless’ solvers ($w_1$): similarly to the best pick discussed above, for each performance indicator and solver we can define the set of solvers which are considered the most poorly performing (shortly, ‘useless’) with tolerance value $t$, as those with performance indicator scores at most $t$. Given the set of pre-known ‘useless’ solvers and the set of predicted best solvers (both with tolerance $t$), we can evaluate the probability of not selecting a useless solver. In the rare case when all solvers are known to be useless, we do not consider that problem in the analysis. In Table 9.1 we display the average of the probabilities over all problems (with at least one non-useless choice).

Mean absolute error between the pre-known and predicted scores (ES): while the above two measurements describe the predicting power related to the best pick, this measurement gives an insight into the relations among all the solvers (which is somewhat more informative in cases when the user does not have all local solvers of our test). For each problem and performance indicator we consider the vectors of pre-known and predicted scores (with lengths equal to the number of available solvers) and evaluate the mean of the componentwise absolute difference between them. These values are displayed as averages over all problems.

Mean absolute error between the pre-known and predicted rank numbers (ER): this measurement is similar to the previous one except that we rank the solvers by their pre-known and predicted scores, respectively, and we evaluate the mean absolute difference between these two vectors. (In case there is a tie in positions $p, \ldots, p + i$, we award the average ‘rank’ $p + i/2$ for all tied solvers.) The mean absolute rank error
values are also given as averages over all problems.

Remark 9.1. One might guess that the linear and rank correlation between the pre-known and predicted scores/ranks could also be an appropriate measurement. However, it often happens that either the pre-known or the predicted scores/ranks have uniform values (most often 100) — in around 20 and 40% of the problems for the solcheck and the function value indicators, respectively —, so that the correlation coefficient cannot be interpreted. This makes correlation-based quality indicators inappropriate when presenting aggregated results.

To demonstrate the power of the fit created in Section 8, we compare its results with three other ones involving more and more information on the features. First, we consider the ‘trivial fit’. For each problem \( p \), each (applicable) solver \( s \), and each performance indicator \( I \in \{S,T,F\} \), the prediction of the indicator value is defined as the average of the respective indicator scores over all training problems: \( I_p^s := (\sum_{q \in P_0} I_q^s)/|P_0| \). That is, for a given \( s,I \), the prediction is constant over all \( p \). Second, we consider the ‘categorical fit’. For each \( p \) with \( G(p) = g \), let \( P_g := \{q \in P_0 | G(q) = g\} \), the set of training problems in the same category as \( p \). Then for each \( p,s,I \) the categorical-based prediction of the indicator value is given as the average of the respective indicator scores over all elements of \( P_g \): \( \bar{I}_p^s := (\sum_{q \in P_g} I_q^s)/|P_g| \). Third, we introduce the ‘combined fit’. Here we involve the most descriptive numerical feature as well: the problems within each category are further refined by their dimensionality (just like in Section 6), resulting in \( 4 \cdot 6 = 24 \) problem groups. Defining \( P' \subseteq P_0 \) as the set of test problems in the same refined categorical group as \( p \), for each \( p,s,I \) the combined prediction of the indicator is given as the average of the respective indicator scores over all elements of \( P' \): \( \bar{I}_p^s := (\sum_{q \in P'} I_q^s)/|P'| \).

Summarizing the above discussion, for the comparison analysis we determined the predictions with the \( k \)-nearest neighbor fit and the above three reference methods for all test problems \( p \in P \) using the training set \( P_0 = P \setminus \{p\} \). Table 9.1 contains the comparative results for the four fitting methods and the four quality measurements, with \( t = 1 \) for \( b_t \) and \( w_t \). According to the transformations of Section 4, the interpretation of the tolerance value of 1 in the context of the raw indicators is the following: a solver just passed the solcheck test the same number of times as the best one (since the solcheck scores are nonnegative multiples of 5), the best found function value differs from the overall best one by at most \( 10^{-3} \) (for absolute difference) or around 1% (for relative difference), and the solver running time is at most around 30% worse than the fastest one.

We can observe that, as expected, the quality measurements are getting better when the fitting involves more and more information about the problem. (The two exceptions are \( w_1 \) for \( F,T \), where the measurement values are almost constant.) The useless solvers are always avoided with very high probability (whenever there is a non-useless alternative), even for the simplest fit. The \( k \)-nearest neighbor fit method picks a best solver in 84%, 89%, and 66% of all cases, for \( S,F \), and \( T \), respectively, which is very convincing. The increase in the quality of the fit is even more substantial for the mean error measurements. In particular, the \( k \)-nearest neighbor fit is 54–64% better for ES and 29–49% better for ER than the trivial fit, and it is 28–36% better for ES and 15–43% better for ER than the combined fit. This shows that the investigated categorical and numerical features indeed influence the performance behavior, and a good part of their effect can be acquired by the proposed \( k \)-nearest neighbor fit. The \( k \)-nearest neighbor fit approximates the scores in average by around 16 for \( S,F \) and by around 5 for \( T \), and it approximates the ranks in average by slightly more than 1.
for all three indicators.

The per solver quality of the fit is measured by the (cumulative) frequencies of the differences between the pre-known and predicted scores, called prediction error, aggregated for each solver separately over each problem. (Due to size limitations of the current analysis, we do not present comparative results with the reference fits above — but for this type of measurement one can observe the same kind of improvements on the quality of the fits as we have given for the per problem measurements above.)

For each solver \( s \), performance indicator \( I \in \{ S, T, F \} \) and threshold \( t, 0 \leq t \leq 100 \) we count the frequencies \(|I^s_p - I^p|\leq t\) over all \( p \in P \). The relative frequencies (given as percentages rounded to the nearest integer) are displayed in Table 9.2 for \( t = 1, 5, 10, 20, \) and 50. That is, for instance, for the solcheck indicator \( S \) and the solver DONLP3, the pre-known and the predicted scores differ by at most 1 in about 36% of the problems and they differ by at most 5 in around 49% of the problems.

Concerning the solcheck indicator \( S \), the prediction error is at most 1 in 25–37% of the problems for the first 8 solver variants and in 60% for LBFGS-B. It is at most 50 in 87–98% of the problems and is best again for LBFGS-B. (Thus, one may say that for the indicator \( S \) LBFGS-B is the ‘most predictable’ solver.) For the function value indicator \( F \), the prediction error is within 1 and 50 in 44–62% and 85–90% of the problems, respectively. Here DONLP3 is the ‘most predictable’ variant, closely followed by IPOPT-qN and SNOPT. Finally, for the time indicator \( T \), the prediction error is at most 1 (i.e. within \( \approx 30\% \) w.r.t. the real running time) in 18–32% of the problems, with SNOPT as the ‘most predictable solver’, and it is at most 50 for virtually all problems.

Thus, with the \( k \)-nearest neighbor method we can predict all performance score values for all problems and all applicable solvers. This enables us to define a new ‘solver’: on each problem and solver we aggregate the three performance scores by taking their weighted sum (we just now simply use the weights \( 1/3 \)), and then to each problem we assign a solver with the smallest respective aggregated score. Aggregating the individual solver scores the same way enables us to compare the new prediction-based ‘solver’ with the others, like we did in Tables 6.1 and 6.2 (but now with one aggregated score value for each problem group). We found that the new strategy was ranked first in 7 out of the 8 groups, and it was ranked second once (by a score difference of 2 behind IPOPT-qN for huge equality/inequality problems), thus it was performing well uniformly. Furthermore, the average placing was also far the best: it was 1.00 for the bound constrained problems, followed by LBFGS-B (2.75), and DONLP3 (4.00) and 1.25 for the equality/inequality constrained problems, followed by IPOPT-qN (2.75) and SNOPT (3.5). When aggregated over all bound constrained problems, the prediction based solver had the best average aggregate score of 88.8, followed by LBFGS-B (86.2) and KNITRO-H (82.6). The same numbers for equality/inequality constrained problems were 85.7 for the prediction-based solver, followed by IPOPT-qN (82.7) and SNOPT (79.0). These facts indicate that the prediction-based solver selection method results in far the most efficient local optimization ‘solver’ for our purposes.

10. Cross-validating the fit. Finally, we demonstrate the robustness of the \( k \)-nearest neighbor fit with cross-validation. Here a given percentage \( p \) of the whole problem set \( P \) is taken as the training set. Then the scores for the test set (i.e., for the remaining problems) is predicted. The pre-known and predicted scores are compared by the same four measurements as above. For \( p = 80, 60, 40, 20 \), we picked
Table 9.2
Cumulative relative frequencies of the differences between the pre-known and predicted scores. For each given \( t \), the respective column contains the relative frequencies of the problems for which the predicted and pre-known scores differ by at most \( t \).

<table>
<thead>
<tr>
<th></th>
<th>( t = 1 )</th>
<th>( t = 5 )</th>
<th>( t = 10 )</th>
<th>( t = 20 )</th>
<th>( t = 50 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S</td>
<td>F</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DONLP3</td>
<td>36 49 59 72 90</td>
<td>62 64 68 78 90</td>
<td>24 67 86 96 99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPOPT-H</td>
<td>35 45 55 69 88</td>
<td>56 60 63 75 87</td>
<td>25 70 90 99 100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPOPT-qN</td>
<td>37 53 63 77 91</td>
<td>61 64 67 79 90</td>
<td>26 72 90 99 100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KNITRO-FD</td>
<td>27 43 54 73 93</td>
<td>53 57 61 74 85</td>
<td>18 59 82 97 99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KNITRO-H</td>
<td>33 47 57 74 93</td>
<td>55 59 62 72 87</td>
<td>18 60 83 97 100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KNITRO-Hv</td>
<td>30 45 57 73 94</td>
<td>55 59 62 74 86</td>
<td>19 62 83 97 100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOQO</td>
<td>25 37 46 61 87</td>
<td>48 50 54 66 85</td>
<td>28 75 93 98 99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SNOPT</td>
<td>31 47 57 72 91</td>
<td>59 63 67 79 89</td>
<td>32 82 93 99 99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LBFGS-B</td>
<td>60 79 87 88 98</td>
<td>44 50 53 75 87</td>
<td>25 78 93 100 100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

20 different random training samples; the results displayed in Table 10.1 are the means of the respective measurements (which are themselves averages as well) over the different samples. The line \( p = 100 \) is included for reference and corresponds to the case studies in Section 9, i.e., when the whole problem set (except the problem to be predicted) is taken as the training set.

As Table 10.1 shows, the prediction quality drops continuously for all performance indicators and quality measurements, but the loss of prediction power is relatively small: e.g., the quality decrease is only 2–9% for \( b_1 \), 0–2% for \( w_1 \), 24–32% for ES, and 15–23% for ER, respectively, when the size of the training set is reduced from 100 to 20 percent. Thus, with the proposed \( k \)-nearest neighbor fit we avoid overfitting, and the method is robust enough for predicting the solver behavior on future problems.

11. Summary. We designed a method for predicting the performance of local solvers when used in interval global optimization algorithms. The method consists of two stages: first, the problem to be solved is classified by its categorical features, with respect to the polynomial order and convexity of its objective function and con-
strains. Then the $k$-nearest neighbor predictions of the performance indicators are calculated by using the indicators on the pre-known problem instances from the respective problem class. The predictions are made on an 8-dimensional feature space consisting of the numerical features of the problems. We demonstrated the superiority of the proposed prediction method to other ones that use less information on the problems, and showed its robustness with a cross-validation. The prediction can be made for each available and applicable local solvers, therefore its ultimate use is to help selecting the best possible local solver. We showed that this new solver selecting strategy greatly improves the performance of the individual solvers, thus, it mixes their advantages on a wide set of test problems. The solver selection strategy (including the data tables of the pre-known solver performances on the test problems) is implemented in the locoopt_chooser inference engine of the COCONUT Environment and is available there for public use since December 2009.

Acknowledgments. The authors thank Arnold Neumaier for his valuable suggestions to improve the paper.

REFERENCES