SIAM J. Optim.
Vol. 21, No. 4, pp. 1371–1391

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 a 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 a 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 goal 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

DOI. 10.1137/100793530

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 toward 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 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 (B&B) 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 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

[*]Received by the editors April 27, 2010; accepted for publication (in revised form) August 29, 2011; published electronically December 1, 2011. This work was supported by FWF (Austrian Science Fund) grants P18704-N13 and P22239-N13.
http://www.siam.org/journals/siopt/21-4/79353.html

[†]Faculty of Mathematics, University of Vienna, Nordbergstr. 15, A-1090 Vienna, Austria (Mihaly.Markot@univie.ac.at, Hermann.Schichl@esi.ac.at).

by interval arithmetic libraries (see, e.g., [1, 16]) 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 actual range) even when the calculations are done with finite-precision floating-point numbers. With the exception of some simple cases, however, the calculated interval result is not the precise function 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), calculations 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 refer the reader to the interval textbooks such as [14, 18], and many others.

It is very natural to employ interval analysis in a rectangular B&B global optimization framework, and it is the basic part in the design of solvers for a 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 on 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 the update of 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 that certainly contains a feasible (or even better, a locally optimal) point. Only then can the upper bound of the interval evaluation of the objective function over this box 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 that measures the frequency of getting the verified answer from the point returned by the local solver.

1.2. The COCONUT Environment.

The COCONUT (COntinuous CONstraints – Updating the Technology) Environment [9] 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 program of the European Community (IST-200-26063, 2000–2004) and has been continued since then on a smaller scale.

The COCONUT Environment (called COCONUT in what follows) 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++. The API contains, among other things, 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 B&B 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 perform 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 that 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 are created in the COCONUT interface, and then the problem class is passed to the solver for solving.
   Besides 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 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. *Callback communication.* The main routine needs to be called only once, and the evaluation routines are predeclared *callback* functions that should be defined by the user.
   - 2b. *Reverse communication.* 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 the main routine is called again.

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 be thread-safe—in particular, it does not use global variables—for the interface to stay 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 commonly passed to the solvers consists of

– the number of variables and constraints,
– the bounds on the variables and constraints,
– the starting point (currently just 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 derivatives are more versatile: in some cases it is mandatory that they be supplied; in others they can be approximated internally. Since first 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 that 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 automatic differentiation. The number of necessary Hessian-vector product evaluations needed to construct the whole Hessian is reduced by a Hessian-coloring algorithm, namely, Algorithm 4.1 of [11].)

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

Many solvers can utilize special problem structure information, e.g., whether the problem 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 make use of it).

Next we give a short description of the solver variants used in the study regarding their methodology and availability. In addition, in Table 1 we list some (comparable) characteristics of the solvers, including software version number, implementation language, licensing, sparsity utilization, type of interfacing with COCONUT, and need for supplying Hessians externally.

3.2. DONLP3.

Methodology. The DONLP3 solver of Spellucci [24, 25] implements a variant of the sequential quadratic programming (SQP) method. At each iteration, a correction to the current iterate is computed by solving a convex quadratic model of the problem. After a search direction for the next iterate is computed, an appropriate stepsize is determined by using an ℓ_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.

Table 1
Summary of the local optimization solvers interfaced to COCONUT.

Name	Version	Language	Licensing	Sparse?	Interface	Hessian supplied?
DONLP3	3	C++	free for research	no	2a	no
IPOPT-H	3.5.1	C++	CPL (free)	yes	1	yes
IPOPT-qN						no
L-BFGS-B	2.1	F77	GPL (free)	no	2b	no
LOQO	6.07	C	proprietary	yes	2a	yes
KNITRO-H						yes
KNITRO-FD	5.1.2z	C/C++	proprietary	yes	2b	no
KNITRO-Hv						as $H_L \cdot v$
SNOPT	6	F77	proprietary	yes	2a	no

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 Wächter and coworkers [19, 28, 30, 29, 31]. The primal-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 [10] in the line search, which means, in very simplified terms, that a trial point during the 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. We used the MUMPS multifrontal solver (http://graal.ens-lyon.fr/MUMPS/) version 4.7.3 for this purpose.

Second-order derivatives (the Hessian of the Lagrangian) can be either supplied explicitly or 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: Hessians internally approximated with a quasi-Newton method.

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 (http://www.coin-or.org) under the Common Public License.

3.4. L-BFGS-B.

Methodology. L-BFGS-B is a limited memory quasi-Newton solver by Byrd et al. [6], applicable for bound constrained problems. L-BFGS-B proceeds by updating a quadratic model to the problem by a limited-memory BFGS approximation of the Hessian of the objective. The matrices 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 determining 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 [17] is performed.

Availability. L-BFGS-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 developed by Vanderbei [22, 26, 27]. The successive search directions are obtained by solving the appropriate primal-dual KKT system with sparse indefinite LDL^T factorization. After the search direction is evaluated, a line search is performed to achieve sufficient decrease of an ℓ_2 merit function. LOQO requires derivative information up to second order.

LOQO is connected to COCONUT through a type 2a 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 employs a C structure instead of a C++ class.

Availability. The most recent versions of LOQO are available for download from their author at http://www.princeton.edu/˜rvdb/. The full version of LOQO is licensed (although there exist free, student versions of the software with limitations on the problem size).

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 (named Interior/Direct) that solves the primal-dual KKT system with direct linear algebra methods [32], a similar interior point method (named Interior/CG) with a projected conjugate gradient method for the KKT system [5, 2], and an active-set method (named 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 options (parameters that can be given to a computer program) 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 options are made in accordance with the KNITRO manual, which recommends supplying exact Hessians or Hessian-vector products for best general performance and suggests the finite differences approximation as a possible further choice.

Remark 3.1. In the present study, we let KNITRO decide internally which of the basic algorithms discussed above to invoke (or when to switch from one algorithm to another). The relation of this internal decision process to our prediction methodology (e.g., that the latter can improve the decision of KNITRO) can be a subject of future research.

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

3.7. SNOPT.

Methodology. SNOPT uses the sparse SQP method of Gill, Murray, and Saunders [12].

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 quadratic program is solved with the active-set method SQOPT [13]. The solution gives a search direction for both primal and dual variables for the next iterate, which is determined by a line search using 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 a moderate number of degrees of freedom. In general it requests relatively few function and gradient 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 COCONUT environment we chose the `snOptA` interface, which is the generally recommended, easy-to-use option starting from Version 6.

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

4. The testing environment. We run all local solver variants on the COCONUT test set [23] consisting of a total of 1307 optimization and constraint satisfaction problems (CSPs) (http://www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html). The CSPs were also specified as optimization problems with a constant objective function. With the exception of L-BFGS-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 20 times by each solver, from the same set of starting points. The solvers were reinvoked for each problem instance (no warm-start information was considered). For bounded variables, the starting points were generated with equal probability from their domain. For unbounded variables, the starting points were created in such a way that smaller absolute values were picked with bigger probability. This was done with the aid of a special probability density function with quadratic decay toward $\pm\infty$. For precise details on the random point generation and on the used density function we refer to the `r_random` routine of the COCONUT API.

The problems were classified by their dimension (the number of variables) 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. The limit is meant individually for each problem instance, i.e., for each starting point. 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 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. The tests were run on a cluster of 32 PCs with 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 differ somewhat from the ones traditionally used in benchmarking studies, because we consider the use and efficiency of local optimization *within* interval global optimization algorithms.

In the first place, we expect the solution to be "useful" in the sense that it can be used as a candidate for a verified feasible solution. (Algorithms for the automatic feasibility verification of approximate solutions, e.g., ones based on [15, 21], are currently under development for COCONUT.) Namely, the reported search point 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` test of COCONUT. In addition, for the usefulness we also expect that the reported solution is classified by the solver as at least a rough approximation of a local minimizer; that is, the search point is obtained after successful termination (convergence), or after a partial progress when some running limit was exhausted, or when some stopping condition could not be satisfied.

In the present study we say 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. The relative constraint violation is obtained by evaluating the constraint at the reported optimizers and (in case of constraint violation) taking the relative error between the constraint value and the constraint bound. Note that 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 so that they will not interfere with these criteria. That is, when a solver 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* $\hat{S}_p^s$ regarding the performance of solver s on test problem p is defined by the number of times the delivered solution passes the solcheck test. Thus, $0 \le \hat{S}_p^s \le 20$.

Second, two local optimization methods can be compared by their efforts toward finding the *global* minimum, which can be measured by the *best reported local minimum* $\hat{F}_p^s \in \mathbb{R} \cup \{-\infty\}$ passing the solcheck test during the multistart testing. If $\hat{S}_p^s = 0$, we set $\hat{F}_p^s = \infty$. For brevity we also call this indicator the *function value indicator*.

The third criterion of comparison is the amount of computational effort needed 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 COCONUT 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 orders (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* is denoted by $\hat{T}_p^s \ge 0$ and is computed as the average of the running times on the respective 20 problem instances. If a solver passes the hard time limit for a given problem instance, we declare that the solver is not advised to run at all on that problem w.r.t. the time requirement and set $\hat{T}_p^s = 10^{10}$.

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 values, but in a shorter time than another. 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 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 most frequently in the paper) are denoted by the respective letter without an accent. A *prediction of a performance score* is 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 not only because the different solvers should be compared on each problem, but because it should be possible to compare the individual solver behaviors from 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 $S_p^s := 5\,\hat{S}_p^s$. The case of the local minimum indicator $\hat{F}_p^s$ is a bit more tricky: for each problem p, if at first $\hat{F}_p^s = \infty$ for all s, then we set $F_p^s = 0$ for all 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 by S' the set of such solvers, and let $f_{p,m} = \min\{\hat{F}_p^s \mid s \in S'\}$. For $s \notin S'$, we set $F_p^s = 0$. 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 that 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 $\hat{F}_p^s, s \in P$ 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 $F_p^s = 15 + 85\,(1 - r)$. That is, in the latter case we introduce a linear growth in the score between 15 and 100. This way we have $F_p^s = 0$ whenever $\hat{F}_p^s = \infty$ and have $15 \leq F_p^s \leq 100$ when $\hat{F}_p^s$ is finite. (The choice of the value 15 is based on experiments; we found that it is a reasonable positive value to represent the lowest possible score for a finite $\hat{F}_p^s$.)

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 as well for S_p^s and 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

$\hat{T}_p^s$ varies between 0 and approximately 10^5 with the resolution of 0.01 when the hard time limit is not passed, and $\hat{T}_p^s$ 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 nine variants of six different state-of-the-art local optimization solvers on an especially wide set of test problems, we believe that publishing the aggregate comparative results of the tests is itself of value. Therefore, in Tables 2 and 3 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 in section 5.1.)

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 that the ranking of the dense DONLP3 solver drops while moving from smaller to bigger—thus, usually sparser—problems or that SNOPT has difficulties with the obviously less well suited large/huge bound-constrained problems with many degrees of freedom at the solutions), most of them cannot be explained from such a traditional numerical summary. It is worth emphasizing again that the present paper is targeted to explaining these nonobvious 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 properties of the optimization problems, and 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 for extracting 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 quickly 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 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 colloquial fit can be created. The grouping described below is therefore a somewhat subjective compromise between these two goals, designed for the current set of problems.

Table 2

Aggregate results on the bound-constrained problems (253 problem instances). The abbreviations S, F, and T stand for the solcheck, function value, and runtime performance indicators, respectively. For each problem group and performance indicator we list the solver rankings based on the average scores. The respective scores are given in parentheses after the rank numbers. The sizes of the problem groups are given next to the group names in the header lines.

	Tiny (137)			Small (24)		
	S	F	T	S	F	T
DONLP3	1 (96.7)	3 (91.4)	1 (93.6)	2 (99.2)	2 (92.7)	3 (82.6)
IPOPT-H	6 (93.6)	1 (93.5)	8 (84.0)	6 (91.0)	7 (84.9)	9 (72.5)
IPOPT-qN	3 (94.9)	8 (83.5)	9 (82.1)	5 (91.3)	5 (88.0)	5 (77.6)
KNITRO-FD	8 (91.9)	9 (81.9)	4 (91.3)	8 (87.1)	8 (83.6)	4 (80.0)
KNITRO-H	4 (94.5)	4 (91.3)	5 (89.7)	4 (91.5)	4 (88.7)	7 (75.8)
KNITRO-Hv	5 (94.1)	7 (84.7)	6 (89.0)	7 (89.4)	6 (86.1)	6 (76.1)
LOQO	9 (77.7)	5 (90.7)	7 (86.8)	9 (86.5)	9 (75.7)	8 (72.8)
SNOPT	7 (92.8)	2 (93.3)	3 (93.3)	3 (94.6)	1 (92.9)	2 (83.7)
L-BFGS-B	2 (95.4)	6 (87.5)	2 (93.5)	1 (99.4)	3 (91.8)	1 (85.5)

	Large (34)			Huge (58)		
	S	F	T	S	F	T
DONLP3	1 (99.9)	1 (75.7)	9 (62.1)	6 (81.1)	7 (33.6)	9 (44.9)
IPOPT-H	8 (81.3)	5 (54.7)	8 (64.1)	7 (77.5)	8 (33.0)	3 (57.3)
IPOPT-qN	7 (87.2)	8 (38.5)	3 (66.9)	8 (77.0)	6 (35.3)	4 (57.2)
KNITRO-FD	3 (92.1)	6 (52.2)	4 (66.1)	5 (87.0)	5 (44.5)	6 (54.9)
KNITRO-H	4 (90.7)	4 (64.3)	6 (64.3)	2 (92.0)	3 (50.3)	7 (54.0)
KNITRO-Hv	4 (90.7)	7 (48.1)	5 (64.4)	3 (91.1)	4 (49.6)	8 (52.2)
LOQO	6 (87.5)	3 (64.7)	7 (64.2)	4 (89.7)	2 (59.6)	5 (56.8)
SNOPT	9 (39.3)	9 (28.7)	1 (74.6)	9 (14.1)	9 (13.7)	1 (71.6)
L-BFGS-B	2 (97.2)	2 (67.9)	2 (70.5)	1 (92.6)	1 (68.8)	2 (60.9)

Table 3

Aggregate results on the equality/inequality constrained problems (1050 problem instances). The table is organized in the same way as Table 2.

	Tiny (551)			Small (208)		
	S	F	T	S	F	T
DONLP3	2 (83.3)	1 (94.7)	2 (93.1)	4 (71.6)	5 (82.8)	2 (81.3)
IPOPT-H	3 (81.4)	3 (93.0)	7 (84.6)	3 (78.1)	2 (83.4)	5 (79.5)
IPOPT-qN	1 (84.0)	2 (93.1)	8 (84.4)	1 (83.9)	1 (86.2)	3 (80.2)
KNITRO-FD	7 (69.3)	7 (86.8)	3 (88.4)	7 (65.7)	7 (80.6)	4 (79.9)
KNITRO-H	5 (72.9)	4 (90.4)	4 (87.9)	5 (70.4)	4 (82.9)	6 (79.0)
KNITRO-Hv	6 (69.8)	6 (87.3)	6 (86.1)	6 (66.3)	6 (81.7)	8 (76.6)
LOQO	8 (68.6)	8 (84.6)	5 (87.0)	8 (40.8)	8 (57.7)	7 (77.8)
SNOPT	4 (78.5)	5 (89.3)	1 (94.4)	2 (79.7)	3 (83.3)	1 (88.5)

	Large (138)			Huge (153)		
	S	F	T	S	F	T
DONLP3	7 (45.5)	7 (48.3)	8 (56.7)	8 (4.2)	8 (5.5)	8 (11.9)
IPOPT-H	3 (67.2)	3 (71.4)	3 (66.2)	2 (60.7)	4 (63.9)	3 (55.0)
IPOPT-qN	1 (73.0)	2 (75.5)	2 (68.7)	1 (78.7)	1 (84.8)	2 (60.5)
KNITRO-FD	4 (57.6)	5 (65.5)	3 (66.2)	4 (50.4)	2 (69.3)	4 (53.1)
KNITRO-H	5 (56.7)	4 (66.2)	6 (65.1)	3 (54.5)	5 (60.4)	5 (53.0)
KNITRO-Hv	6 (53.7)	6 (63.6)	7 (63.3)	5 (49.6)	3 (64.8)	5 (53.0)
LOQO	8 (12.9)	8 (26.9)	5 (65.3)	7 (9.5)	7 (23.2)	7 (52.0)
SNOPT	2 (69.0)	1 (76.3)	1 (72.1)	6 (37.7)	6 (45.3)	1 (61.2)

Table 4
Grouping the test problems by the categorical features.

Obj\cstr	no	bnd	lin	q-cvx	o-cvx	q-ncvx	o-ncvx
const	0	0	0	6	6	6	6
lin	0	0	0	5	5	1	2
q-cvx	0	0	0	5	5	1	2
o-cvx	4	4	5	5	5	2	2
q-ncvx	0	4	3	3	3	1	2
o-ncvx	4	4	3	3	3	2	2

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 at most second-order function information, the polynomial degree feature will be differentiated up to degree 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 [20] returns "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 4.

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 seven groups are those 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 can be solved as a system of linear equations, a linear program, or a convex quadratic program. This can be done efficiently by standard linear programming software, such as CPLEX or FICO Xpress (139 problem instances);

1: nonconvex general convexly constrained quadratic programs (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 what follows, $G(p)$ 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 eight 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 constraints that are not 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 scale the feature values nonlinearly in the same 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 are denoted by the respective letter of the unscaled feature without the "hat" accent.

Given a set of problems Q that belong to the same category according to section 7.1, the numerical features of all $p \in Q$ are scaled as follows: for $\hat{n}_p$, $\hat{d}_p$, $\hat{b}_p$, $\hat{e}_p$, $\hat{i}_p$, and $\hat{v}_p$ first we apply a logarithmic scaling, and then we scale the results into $[0, 1]$, dividing by the maximal feature value *within the respective problem category*. That is, for the numerical feature $\hat{f} \in \{\hat{n}, \hat{d}, \hat{b}, \hat{e}, \hat{i}, \hat{v}\}$ 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 $\hat{f} \in \{\hat{b}, \hat{e}, \hat{i}, \hat{v}\}$ and $s_f = 0$ for $\hat{f} \in \{\hat{n}, \hat{d}\}$). 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 $\hat{f} \in \{\hat{J}, \hat{H}\}$ and $\forall\, p \in Q$ we set $f_p := \hat{f}_p^{1/2}$.

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 next.

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, because 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 $f_p := \min\{\log_{10}(\hat{f}_p + s_f)/m_f, 1\}$ if $m_f \neq 0$, and $f_p := 1$ otherwise.

Next we find the k problems in P_g that are 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 $e_p = i_p \equiv 0$, $J_p \equiv 1$, and $v_p \equiv 0$, so the features actually vary in a 4-dimensional subspace) and $w = (3, 3, 2, 2, 2, 3, 3, 6)^T/24$ 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 $d_{q_i} = (\sum_{l=1}^{8} w^{(l)}(f_p^{(l)} - f_{q_i}^{(l)})^2)^{1/2}$ weighted distance values. (The upper index l denotes the lth component of the respective vectors.) The predicted performance score will be the weighted sum of the performance scores of q_i, with weights W_{q_i} defined to be proportional to the inverse of the distances (with some correction to avoid division by zero): we set $\tilde{d}_{q_i} := 1/\max\{d_{q_i}, 10^{-3}\}$ and $W_{q_i} := \tilde{d}_{q_i}/\sum_j \tilde{d}_{q_j}$. Thus, the predicted performance scores are formally given as $\bar{I}_t^s := \sum_i W_{q_i} I_{q_i}^s$.

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 preknown 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 were 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 conclusion is that in the k-nearest neighbor prediction method, most information on the effect of the numerical features on 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 compared to other prediction methods utilizing less problem information. Since the local solver predictions are needed only for problems that 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 what follows, P denotes this problem set. With the methodology of section 5, we first compute the performance indicator scores for all problems $p \in P$ and available solvers. These will be called the *preknown* scores. Then the *predicted scores* are computed for all problems $p \in P$ and solvers using the preknown performance scores *of all other available problems* $P \setminus \{p\}$. (This models the scenario of predicting solver performance on a new problem instance.) In other words, we create the prediction for all p by using $P_0 = P \setminus \{p\}$ as the training sample.

TABLE 5

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 preknown and predicted scores), and ER (mean absolute error between the preknown and predicted ranks). Larger values for b_1 and w_1, and smaller values for ES and ER, mean better prediction power. The proposed k-nearest neighbor fit is compared with the trivial, the purely categorical-based, and the combined fit.

Measurememt	Performance indicator	Trivial	Categorical	Combined	k-nearest
b_1	S	0.82	0.82	0.83	0.84
	F	0.83	0.87	0.89	0.89
	T	0.55	0.55	0.60	0.66
w_1	S	0.95	0.96	0.96	0.97
	F	0.94	0.96	0.96	0.95
	T	0.99	0.99	1.00	0.99
ES	S	34.33	29.99	24.28	15.94
	F	34.02	32.62	24.55	15.79
	T	13.32	12.92	6.63	4.76
ER	S	1.87	1.72	1.67	1.32
	F	2.05	1.92	1.85	1.05
	T	2.08	1.86	1.44	1.23

Another way of testing the fit, namely 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 does the prediction of the solver performance change from problem to problem). For our current purpose, the first type of measurement is 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. Given a performance indicator and problem, a solver s is considered best with tolerance t if its indicator score differs by at most t from that of the best performing one (as in section 6). The set of such best solvers can thus be computed for both the predicted and preknown scores. Given these two sets and assuming that we pick a solver among the *predicted* best solvers with equal probability, we can, for each indicator and problem, determine the probability of choosing a solver from the set of *preknown* best ones. In Table 5 we display the average of these probabilities over all problems; that is, we have an aggregate measurement for each performance indicator.

Probability of avoiding all "useless" solvers (w_t). Similarly to the best pick discussed above, for each performance indicator and solver we can define the set of solvers that are considered the poorest performing ("useless") with tolerance value t as those with performance indicator scores at most t. Given the set of preknown "useless" solvers and the set of predicted best solvers (both with tolerance t), we can evaluate the probability of not selecting a useless solver. When all solvers are known to be useless, we do not consider that problem in the analysis (there are 43 and 44 such problems for the S and F indicators, respectively). In Table 5 we display the average of the probabilities over all problems (with at least one nonuseless choice).

Mean absolute error between the preknown and predicted scores (ES). While the above two measurements describe the predicting power related to the best pick, this measurement gives 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 preknown 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 preknown and predicted rank numbers (ER). This measurement is similar to the previous one except that we rank the solvers by their preknown 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" $p + i/2$ to 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 correlation and rank correlation between the preknown and predicted scores/ranks could also be appropriate measures. However, it often happens that either the preknown or the predicted scores/ranks have uniform values (most often 100)—in around 20% and 40% of the problems for the solcheck and function value indicators, respectively—so that the correlation coefficient cannot be interpreted. This makes correlation-based quality indicators inappropriate for presenting aggregated results.

To demonstrate the power of the fit created in section 8, we compare its results with three others 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*: $\bar{I}_p^s := (\sum_{q \in P_0} I_q^s)/|P_0|$. That is, for given s and 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 5 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 known 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.

We can observe that, as expected, the quality measurements become 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

TABLE 6

Quality of the k-nearest prediction for the three performance indicators, grouped by problem categories. The problem categories are the ones denoted by 1 *to* 6 *in section* 7.1. *The four quality measurements b_1, w_1, ES, and ER are the same as in Table* 5.

Measurement	Performance indicator	Problem categories					
		1	2	3	4	5	6
b_1	S	0.84	0.84	0.94	0.93	0.78	0.74
	F	0.90	0.87	0.94	0.83	0.91	0.93
	T	0.63	0.60	0.66	0.72	0.66	0.67
w_1	S	0.96	0.98	0.98	0.99	0.94	0.94
	F	0.96	0.97	0.98	0.98	0.92	0.92
	T	1.00	0.98	1.00	1.00	1.00	1.00
ES	S	13.29	18.41	10.68	11.31	17.72	19.59
	F	12.46	19.71	10.54	18.89	15.12	13.98
	T	3.74	5.53	3.92	3.98	6.29	5.15
ER	S	1.47	1.37	0.98	1.28	1.64	1.35
	F	1.01	1.12	0.86	1.35	0.88	0.91
	T	1.11	1.29	0.99	1.16	1.50	1.34

solvers are always avoided with very high probability (whenever there is a nonuseless 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 solver performance, 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 on average by around 16 for S, F and by around 5 for T, and it approximates the ranks on average by slightly more than 1 for all three indicators.

Table 5 contains the prediction measurements aggregated over all problems. However, it is also interesting to investigate the k-nearest prediction separately for each problem category, as given in Table 6. One can observe that in most cases there are significant deviations around the aggregated values of Table 5, which shows that on some problem categories the solver performance is more predictable than on others. In particular, the solcheck indicators can be significantly better predicted (for all four measurements) for categories 3 and 4 (convexly constrained general nonconvex and unconstrained/bound-constrained problems) than for the other four categories. The function value indicator can be in general the best predicted for categories 1 (general nonconvex QCQPs), 3, and 6 (nonlinear CSPs). The running time is the most predictable also for categories 3 and 4, and—except for the b_1 measurement—for category 1. The solver performance on category 2 (nonconvex, nonquadratic problems) is usually among the least predictable performance indicators for all measurements.

The per solver quality of the fit is measured by the (cumulative) frequencies of the differences between the preknown 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.)

Table 7

Cumulative relative frequencies of the differences between the preknown and predicted scores. For each given t, the respective column contains the relative frequencies of the problems for which the predicted and preknown scores differ by at most t.

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

For each solver s, performance indicator $I \in \{S, T, F\}$ and threshold $t,\ 0 \leq t \leq 100$ we count the frequencies $|I_p^s - \bar{I}_p^s| \leq t$ over all $p \in P$. The relative frequencies (given as percentages rounded to the nearest integer) are displayed in Table 7 for $t = 1, 5, 10, 20$, and 50. That is, for instance, for the solcheck indicator S and the solver DONLP3, the preknown and 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 L-BFGS-B. It is at most 50 in 87–98% of the problems and is best again for L-BFGS-B. (Thus, one may say that for the indicator S L-BFGS-B is the "most predictable" solver.) Note that L-BFGS-B was run on bound constrained problems only; that is, in Table 7 it appears with a different basis of aggregation. The frequencies show that, somewhat naturally, on bound-constrained problems it is relatively easy to find useful solutions, i.e., reach high solcheck scores.

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. A possible explanation of the predictability of SNOPT is that it is in general the fastest on all problems (see Tables 2 and 3), including problem classes that are less well suited for the solver.

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, as we did in Tables 2 and 3 (but now with one aggregated score value for each problem group). We found that the new strategy was *ranked first in seven out of the eight groups*, and it was ranked second once (by a score difference of two behind IPOPT-qN for huge equality/inequality problems). Thus it was performing well *uniformly.* Furthermore, the average placing was also by far the best: it was 1.00 for the bound-constrained problems, followed by L-BFGS-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 L-BFGS-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 by 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) are predicted. The preknown and predicted scores are compared by the same four measurements as above. For $p = 80, 60, 40, 20$, we picked 20 different random training samples; the results displayed in Table 8 are the means of the respective measurements (which are themselves averages as well) over the different samples. The column $p = 100$ 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 8 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 constraints. Then the k-nearest neighbor predictions of the performance indicators are calculated by using the indicators on the preknown problem instances from the respective problem class. The predictions are made on an 8-dimensional feature space

TABLE 8

Cross-validation for the k-nearest neighbor fit prediction for the three performance indicators. The four quality measurements are the same as in Table 5. The size of the training samples is p percent of the whole problem set.

Measurement	Performance indicator	$p = 100$	$p = 80$	$p = 60$	$p = 40$	$p = 20$
b_1	S	0.84	0.84	0.82	0.82	0.79
	F	0.89	0.89	0.88	0.88	0.87
	T	0.66	0.64	0.65	0.63	0.60
w_1	S	0.97	0.97	0.96	0.96	0.95
	F	0.95	0.95	0.95	0.95	0.94
	T	0.99	0.99	0.99	0.99	0.99
ES	S	15.94	17.09	17.22	17.87	19.76
	F	15.79	17.00	17.26	17.97	19.59
	T	4.76	5.25	5.27	5.54	6.26
ER	S	1.32	1.37	1.41	1.44	1.55
	F	1.05	1.13	1.14	1.19	1.29
	T	1.23	1.29	1.29	1.32	1.41

consisting of the numerical features of the problems. We demonstrated the superiority of the proposed prediction method to others that use less information on the problems, and showed its robustness with cross-validation. The prediction can be made for each available and applicable local solver; therefore its ultimate use is to help to select the best possible local solver. We showed that this new solver selecting strategy substantially 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 preknown solver performances on the test problems) is implemented in the `locopt_chooser` inference engine of the COCONUT Environment and has been available there for public use since December 2009.

Acknowledgment. The authors thank Arnold Neumaier for his valuable suggestions which improved the paper.

REFERENCES

[1] H. Brönnimann, G. Melquiond, and S. Pion, *The design of the boost interval arithmetic library*, J. Theoret. Comput. Sci., 351 (2006), pp. 111–118.

[2] R.H. Byrd, J.-Ch. Gilbert, and J. Nocedal, *A trust region method based on interior point techniques for nonlinear programming*, Math. Program., 89 (2000), pp. 149–185.

[3] R.H. Byrd, N.I.M. Gould, J. Nocedal, and R.A. Waltz, *An algorithm for nonlinear optimization using linear programming and equality constrained subproblems*, Math. Program. Ser. B, 100 (2004), pp. 27–48.

[4] R.H. Byrd, N.I.M. Gould, J. Nocedal, and R.A. Waltz, *On the convergence of successive linear-quadratic programming algorithms*, SIAM J. Optim., 16 (2005), pp. 471–489.

[5] R.H. Byrd, M.E. Hribar, and J. Nocedal, *An interior point algorithm for large-scale nonlinear programming*, SIAM J. Optim., 9 (1999), pp. 877–900.

[6] R.H. Byrd, P. Lu, J. Nocedal, and C. Zhu, *A limited memory algorithm for bound constrained optimization*, SIAM J. Sci. Comput., 16 (1995), pp. 1190–1208.

[7] R.H. Byrd, J. Nocedal, and R.B. Schnabel, *Representation of quasi-Newton matrices and their use in limited memory methods*, Math. Program., 63 (1994), pp. 129–156.

[8] R.H. Byrd, J. Nocedal, and R.A. Waltz, *KNITRO: An integrated package for nonlinear optimization*, in Large-Scale Nonlinear Optimization, G. di Pillo and M. Roma, eds., Springer, Berlin, New York, 2006, pp. 35–59.

[9] *The COCONUT Environment*, http://www.mat.univie.ac.at/coconut-environment (2010).

[10] R. Fletcher and S. Leyffer, *Nonlinear programming without a penalty function*, Math. Program., 91 (2002), pp. 239–269.

[11] A.H. Gebremedhin, F. Manne, and A. Pothen, *What color is your Jacobian? Graph coloring for computing derivatives*, SIAM Rev., 47 (2005), pp. 629–705.

[12] P.E. Gill, W. Murray, and M.A. Saunders, *SNOPT: An SQP algorithm for large-scale constrained optimization*, SIAM J. Optim., 12 (2002), pp. 979–1006.

[13] P.E. Gill, W. Murray, M.A. Saunders, and M.H. Wright, *Inertia-controlling methods for general quadratic programming*, SIAM Rev., 33 (1991), pp. 1–36.

[14] E. Hansen, *Global Optimization Using Interval Analysis*, Marcel Dekker, New York, 1992.

[15] R.B. Kearfott, *On Verifying Feasibility in Equality Constrained Optimization Problems*, Technical report, University of Southwestern Louisiana, 1996.

[16] M. Lerch, G. Tischler, J. Wolff von Gudenberg, W. Hofschuster, and W. Krämer, *FILIB++, a fast interval library supporting containment computations*, ACM Trans. Math. Software, 32 (2006), pp. 299–324.

[17] J.J. Moré and D.J. Thuente, *Line search algorithms with guaranteed sufficient decrease*, ACM Trans. Math. Software, 20 (1994), pp. 286–307.

[18] A. Neumaier, *Interval Methods for Systems of Equations*, Encyclopedia Math. Appl. 37, Cambridge University Press, Cambridge, UK, 1990.

[19] J. Nocedal, A. Wächter, and R. A. Waltz, *Adaptive barrier update strategies for nonlinear interior methods*, SIAM J. Optim., 19 (2008), pp. 1674–1693.

[20] D. Orban, R. Fourer, A. Neumaier, H. Schichl, and C. Maheshwari, *Convexity and concavity detection in computational graphs: Tree walks for convexity assessment*, INFORMS J. Comput., 22 (2010), pp. 26–43.

[21] H. Schichl and A. Neumaier, *Exclusion regions for systems of equations*, SIAM J. Numer. Anal., 42 (2004), pp. 383–408.

[22] D.F. Shanno and R.J. Vanderbei, *Interior-point methods for nonconvex nonlinear programming: Orderings and higher-order methods*, Math. Program., 87 (2000), pp. 303–316.

[23] O. Shcherbina, A. Neumaier, D. Sam-Haroud, X.-H. Vu, and T.-V. Nguyen, *Benchmarking global optimization and constraint satisfaction codes*, in Global Optimization and Constraint Satisfaction, Ch. Bliek et al., eds., Springer, Berlin, 2003, pp. 211–222.

[24] P. Spellucci, *A new technique for inconsistent problems in the SQP method*, Math. Methods Oper. Res., 47 (1998), pp. 355–400.

[25] P. Spellucci, *An SQP method for general nonlinear programs using only equality constrained subproblems*, Math. Program., 82 (1998), pp. 413–448.

[26] R.J. Vanderbei, *LOQO: An interior point code for quadratic programming*, Optim. Methods Softw., 12 (1999), pp. 451–484.

[27] R.J. Vanderbei and D.F. Shanno, *An interior-point algorithm for nonconvex nonlinear programming*, Comput. Optim. Appl., 13 (1999), pp. 231–252.

[28] A. Wächter, *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering*, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, 2002.

[29] A. Wächter and L. T. Biegler, *Line search filter methods for nonlinear programming: Local convergence*, SIAM J. Optim., 16 (2005), pp. 32–48.

[30] A. Wächter and L. T. Biegler, *Line search filter methods for nonlinear programming: Motivation and global convergence*, SIAM J. Optim., 16 (2005), pp. 1–31.

[31] A. Wächter and L. T. Biegler, *On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming*, Math. Program., 106 (2006), pp. 25–57.

[32] R.A. Waltz, J.L. Morales, J. Nocedal, and D. Orban, *An interior algorithm for nonlinear optimization that combines line search and trust region steps*, Math. Program., 107 (2006), pp. 391–408.