Constrained Black Box Optimization with Data Analysis

Revision 3

Kevin Kofler, Hermann Schichl, Arnold Neumaier
University of Vienna, Austria
Faculty of Mathematics
kevin.kofler@chello.at, Hermann.Schichl@esi.ac.at,
Arnold.Neumaier@univie.ac.at

February 18, 2010

Abstract

This paper presents the design of and test results for an algorithm solving constrained black box optimization problems globally using mainly methods from data analysis. A particular focus is put on constraints: in addition to bound constraints, we also handle black box inequality and equality constraints. In particular, our algorithm is able to handle equality constraints given in implicit form $f(x) = 0$ where f is a black box function and x a vector of one or more variables. We achieve this by approximating our black box functions by quadratic covariance models, using Gaussian mixture models to locate holes to fill with sample points and bounding implicit equality constraints by quadratic approximations. Our algorithm does not require gradients or gradient approximations, making it fit for problems where function evaluations are expensive and no derivative information is available.
Keywords: global optimization, black box optimization, derivative-free optimization, expensive functions, data analysis, black box constraints, implicit equality constraints, covariance models, Gaussian mixture models

Acknowledgements

Support by the Austrian Science Fund (FWF) under contract number P18704-N13 is gratefully acknowledged.

1 Introduction

The goal of our paper is to present both an algorithm and a reference implementation to solve optimization problems where both the objective function and the constraints may be black box functions, we do not have any gradient or Hessian information for those black box functions and the functions are assumed to be expensive to compute, thus the number of function evaluations

shall be kept as small as possible, using covariance models as models for the black box behavior, Pareto filter techniques for estimating the position of the optimum, the ratio-reject test and cubic regression to approximate the Pareto front and extrapolate an optimum and Gaussian mixture models (GMMs) and the Expectation-Maximization (EM) iteration for global search.

The implementation (available at `http://www.tigen.org/kevin.kofler/bbowda/`) is licensed under the GNU General Public License, version 3 [15] or later, with special exceptions allowing to link with the third-party optimizers used.

Our implementation has also been integrated [17] into the Diana framework [28], though this version is not contained in the public releases of Diana as of January 2010.

In section 2, we present an algorithm applying methods from data analysis to the solution of black box optimization problems. Using quadratic covariance models for heuristic local function enclosure and Gaussian mixture models for global density estimation, we construct an optimization algorithm which can handle both objective functions and constraints without any gradient or Hessian information. As we need not compute any difference quotients, the number of required function evaluations is low, so the algorithm lends itself particularly well to functions which are expensive to evaluate. Our algorithm is an incomplete global optimization algorithm: completeness cannot be achieved due to the lack of global information and any asymptotical properties are of limited use because we expect our optimizer to be used with expensive functions and thus a low cap on the number of function evaluations. Therefore, we do not pursue asymptotical completeness, but instead focus on density estimation aiming at filling the gaps in the search space in an optimal way given a finite number of points, and taking the points found by the local search into account.

We also present a way to handle black box implicit equality constraints: our local covariance models are formulated in a way naturally accounting for them; for our global density models, we solve linear programs to find heuristic quadratic over- and underestimators for each implicit equality constraint, which provide enclosures for the feasible set. As the enclosures cannot be made rigorous due to the lack of global information, we adjust them for each newly computed point, ensuring that all known points are always within the current enclosures. As black box implicit equality constraints are still an active field of research, this is a significant result.

We document a working implementation of the above concepts, written in the ISO C99 language [21, 22] and licensed under the GNU General Public License, version 3 [15] or later, with special exceptions allowing to link with the third-party optimizers used. The required third-party libraries are `lp_solve` for linear programs and either `DONLP2` or `Ipopt`, which requires `MUMPS` and implementations of `BLAS` and `LAPACK` (we recommend `ATLAS`, which not only performs better than the reference implementations, but also gave more accurate results in our tests), for the non-linear surrogate and density models.

Finally, in section 3, we test our algorithm on a few low-dimensional example programs from the COCONUT benchmark and perform some scalability tests. We also compare our algorithm on a representative testcase with the existing gradient-free solvers DDE [32], DMS-PSO [31] and HOPSPACK [27]. The other gradient-free solvers available to us that work with nonlinear constraints were entirely unable to solve this type of problem:

- NOMAD [30] does not support implicit equality constraints. The model only supports inequalities and there is no special handling for matched pairs of inequalities.
- CONDOR [48, 49] does not support expensive or black-box constraints at all (neither equalities nor inequalities). One of the assumptions quoted in its manual is that the non-linear constraints are "cheap" to evaluate. In addition, the evaluation function is expected to return gradients for the constraints. Only the objective function can be an expensive black-box function.
- The OpenOpt package [29] also does not provide a solver which handles implicit black-box equality constraints. Its solvers either do not support any constraints (ShorEllipsoid, scipy_fmin), support only bound constraints (galileo), only linear constraints (pswarm) or only nonlinear inequalities (de), or require gradients (ralg). In addition, the "de" solver is a differential evolution implementation requiring an extremely high number of function evaluations (the main loop does 150000000 iterations!), so its usefulness even for black-box inequalities is limited.

A multitude of algorithms has been proposed to solve constrained global optimization problems efficiently, but most of them expect to have information about gradients or at least to be able to approximate them using difference quotients. Only a few algorithms have been successful at handling black box constraints where no gradients are available and which are too expensive to evaluate to consider difference quotients.

Simulated Annealing. Simulated annealing algorithms [24, 19, 47] employ a heuristic based on the observation of the natural process of cristallization: a cristalline structure is one which globally minimizes a certain potential. This global minimum can be obtained by heating the material and slowly cooling it down. However, no ready-made implementation dealing with general constraints is publicly available.

Genetic Algorithms. Genetic algorithms [18, 14, 33] are also inspired by natural processes: evolution and natural selection. They begin with a set of N starting points, the *population*. At each step, pN ($p \in]0, 1[$) points are retained (*selected*), the others are killed. Points with smaller function value have a higher probability of survival. The population is then filled up by the reproduction of the remaining points through a problem-dependent *crossover* procedure and/or random *mutations*.

A multitude of variations, which can handle black box constraints, of this flexible approach have been proposed at the CEC 2006 Special Session on Constrained Real-Parameter Optimization [26, 44].

DIRECT. Unlike the above Nature-inspired methods, the DIRECT (DIviding RECTangles) algorithm [23] is based on mathematical observations. It is a complete algorithm using no global information. Therefore, it has to produce a sequence of points which is dense in the feasible domain, as proven by Törn and Žilinskas in [46]. DIRECT starts from a box (i.e. bound constraints) and

constructs such a sequence by repeatedly splitting the box. A balance between local and global search is given by a domination relation.

The original DIRECT algorithm only handles bound constraints. Extensions have been proposed to handle black box constraints, e.g. in [6]. A convergence analysis is provided in [9].

EGO. The EGO (Efficient Global Optimization) algorithm [20] works by approximating the black box functions by response surfaces using DACE (Design and Analysis of Computer Experiments) stochastic process models [37]. An estimate for the expected improvement is then computed using a branch and bound approach. The approximation is refined and the process iterated until the expected improvement is less than 1% of the current best function value.

The main limitation of the above algorithms is that they cannot handle black box equality constraints. In fact, some of them already show their limitations with inequality constraints other than bound constraints. Adding support for general contraints to algorithms designed for no or only bound constraints through tricks such as penalty functions or artificial function values generally leads to much slower convergence, or even none at all. This paper develops an algorithm designed to tackle this problem, supporting inequality constraints as well as equality constraints in both explicit $y = f(x)$ and implicit $f(x) = 0$ forms. In this paper, we only describe the version that worked best; for a discussion of alternatives, see [25].

In Section 2, we will define the exact **model** we operate on and describe our **algorithm**. In Section 3, we will summarize the **results** obtained with our implementation, both in terms of speed and quality, and compare them with state-of-the-art algorithms on a representative testcase. Finally, Section 4 will **conclude** the paper with an outlook on possible future improvements.

2 Model and Algorithm

This section proposes an algorithm to solve the above model. First, we give a rough overview of the algorithm, then we detail each step. We start by describing our method for starting point generation, then the local search technique, then our global search method which looks for unexplored regions, and finally we present a way to deal with implicit equality constraints in the global search. We will go into depth over the mathematical description of the algorithm, we will however leave the details of the concrete implementation for the next section. We will also motivate our decisions by mentioning some alternative approaches we tried without success.

2.1 Model

Our algorithm solves optimization problems of the form

$$\begin{array}{lll} \min & \gamma + c^T \begin{pmatrix} x \\ y \end{pmatrix} & \\ \text{s.t.} & y = F_1(x) & \text{(explicit equality constraints)} \\ & F_2(x) = 0 & \text{(implicit equality constraints)} \\ & x_l \leq x \leq x_u & \\ & F_l \leq y \leq F_u & \end{array}$$

where x and y are variable vectors, γ is a constant, c, x_l, x_u, F_l, and F_u are constant vectors (the bounds on x should be finite and not too wide to allow for global search, the bounds on y can be infinite), inequalities are component-wise, and

$$F(x) = \begin{pmatrix} F_1(x) \\ F_2(x) \end{pmatrix}$$

is assumed to be an **expensive**, **black box** function. By this we mean that no closed-form algebraic expression for the function is known, no gradients are available, and the bulk of the runtime of the algorithm on a real-world problem is expected to be given by the function evaluations. These assumptions are central to the design of the algorithm. We perform an **incomplete global** optimization on this model; this means we attempt to find a global solution for our model, but are unable to guarantee globality. In fact, we cannot even guarantee always finding a local optimum, due to the lack of gradients and any sort of global information. Despite this lack of guarantees, the algorithm performs well in practice, see Section 3.

The above form was chosen very carefully to avoid inefficiencies that resulted from earlier attempted formulations (see [25, Section 3.1]).

2.2 Overview of the Algorithm

The outline of our algorithm is the following:

1. If we do not have enough starting points, generate these.
2. As long as the maximum number of function evaluations is not reached, we do after the k^{th} function evaluation:
   - for even k a local search:
     (a) pick a "best" point using a Pareto filter method (see [10, 11]),
     (b) compute a regularized weighted nonlinear covariance model around the point,
     (c) optimize the model using a third-party local optimization method;
   - for odd k a global search:
     (a) compute a Gaussian mixture model (see [54]) approximating the point density using Expectation-Maximization Iteration (see [8]),
     (b) optimize the model (minimize the density, in order to search in unexplored regions) using a third-party local optimization method.

Then evaluate the black box function at the minimizer found.

3. If we have implicit equality constraints, try extrapolating a feasible point by:

   (a) picking the points which are Pareto-optimal (within the set of computed points) for the simultaneous minimization of the objective f and the constraint violation cv,

   (b) throwing out those which are either already feasible to the desired tolerance (cv too small) or too far from feasibility (cv too large),

   (c) throwing out possible outliers using the method developed by Tax and Duin in [45] (coined "ratio-reject" by [12]),

   (d) using cubic regression to extrapolate cv to 0 from the remaining points (which are assumed to be a good approximation for the Pareto front, i.e. the set of theoretically Pareto-optimal points),

   (e) computing the actual constraint violation to verify actual feasibility.

   Then evaluate the black box function at the extrapolated point.

The global search ignores inequality constraints, i.e. the bounds F_l and F_u for the explicit equality constraints $y = F_1(x)$, by design, because we only have local information for the constraints, so we can't reliably tell which points are feasible and which aren't, and in addition evaluating at infeasible points can give us information important to find further feasible regions. (The explicit equality constraints themselves are irrelevant because the global search only searches in the x coordinates.) However, it has proven impractical to completely ignore the implicit equality constraints $F_2(x) = 0$ and blindly search everywhere, therefore we approximate them with quadratics, and actually optimize the GMM twice: first without the equality constraints, then, using the result from this optimization as a starting point, again with the approximations for the constraints. (Optimizing the model directly with the constraints from a generic starting point has proven too hard for the local optimization methods, thus the two-stage approach.) Only the result with the constraints is retained.

2.3 Starting Point Generation

To generate starting points, we use the following heuristic. Let N be the total number of starting points to generate. If we already have at least N user-provided starting points, we can skip this step. Otherwise, let m be the dimension of the points we want to generate, i.e. we want to construct $x_1, \ldots, x_N \in \mathbb{R}^m$. We construct a grid of $(2N)^m$ points $x[1, \ldots, 1], \ldots, x[2N, \ldots, 2N]$, equidistant along each dimension, such that $x[i_1, \ldots, i_{j-1}, 1, i_{j+1}, \ldots, i_m]_j = x_{lj}$ and $x[i_1, \ldots, i_{j-1}, 2N, i_{j+1}, \ldots, i_m]_j = x_{uj}$. (Note that this construction is purely theoretical, in practice we will not compute the coordinates of these exponentially many points!) We then proceed to fill this grid semi-randomly, but taking care not to generate more than one point with a given x_j coordinate, and preferring points farther away from existing ones to closer points. This heuristic is designed to prevent the formation of random clusters of starting points, with other areas remaining unexplored.

The way we obtain such a filling is:

1. If we are given starting points by the user, we round these starting points to the closest points on the grid (just for the purpose of the starting point generation step; later in the algorithm, the actual user-provided points will be used), and proceed as if these points had been generated by the automated heuristic.

2. For each dimension $j = 1, \ldots, m$, we pick an index i_j which has not been used yet. Given that there are $2N$ total possibilities for i_j and only N starting points are to be generated, there are always at least $N + 1$ such possibilities. Each unused i_j is given the same probability. The resulting point is the point $x[i_1, \ldots, i_m]$ on the grid.

3. We repeat the above procedure for a total of 10 points, then retain the one the farthest away from the existing points, i.e. with the largest euclidean distance to the closest existing point. (If there are several points with the same distance to existing points, any of them can be picked.) The other 9 points generated in this step are discarded.

4. We repeat the above two steps until we have N retained points.

It shall be noted that the rounded points from step 1 might not have been valid for automated generation, in particular there might be several points with the same j^{th} index i_j. However, this does not impact the algorithm in any way, and therefore we do not attempt to "correct" user-provided starting points.

2.4 Local Search

Our local search starts at a previously-found point, constructs a local surrogate model approximating the black box optimization problem around the point, then optimizes that problem using a third-party local optimizer.

2.4.1 Choice of Best Point

The point to start the local search at is picked using a Pareto filter method as described in [10, 11]. In the presence of constraints, there is (in general) no single best point, as one must take into account both the objective function value and the constraint violation. We chose a filter approach because penalty approaches proved too sensitive to arbitrary problem-specific parameters.

As in the general Pareto filter method, our best point is a point from the set P of x_j for which there is no x_k with $c^T \binom{x_k}{y_k} \leq c^T \binom{x_j}{y_j}$ and $e(x_k) < e(x_j)$ or $c^T \binom{x_k}{y_k} < c^T \binom{x_j}{y_j}$ and $e(x_k) \leq e(x_j)$. For $e(x)$, we did not pick the raw constraint violation $\mathrm{cv}(x) = \mathrm{cv}_1(x) + \|F_2(x)\|_1$, but the weighted penalty term $e(x) = N\mathrm{cv}_1(x) + \kappa(N)\|F_2(x)\|_1$ (which was tuned for a previous penalty approach), where

$$\mathrm{cv}_1(x) = \sum_{x_i < x_{li}} (x_{li} - x_i) + \sum_{x_i > x_{ui}} (x_i - x_{ui}) + \sum_{y_i < F_{li}} (F_{li} - y_i) + \sum_{y_i > F_{ui}} (y_i - F_{ui}),$$

$$\kappa(N) = \begin{cases} \sqrt{\frac{N}{18}}, & N < 18 \\ \frac{N}{18}, & 18 \leq N < 77 \\ \frac{N\sqrt{N}}{158}, & N \geq 77 \end{cases}$$

and N is the number of already-evaluated points, though we do not expect this to make a lot of difference in practice. We then pick a "best" point from P using the following recipe:

- If the most recent point x_r in P has never been used as the starting point for a local search, we pick it with a probability of $1 - \frac{1}{|P|}$.
- Otherwise, i.e. with a probability of $\frac{1}{|P|}$ if x_r was never used and always if it was, we pick a random point out of P (which can also be x_r) with equal probabilities (i.e. each point is picked with probability $\frac{1}{|P|}$).

Note that the set P is never empty by construction, therefore it is always possible to pick such a point. The local search is then started from this point.

2.4.2 Surrogate Model

Let x_{best} be the point constructed above. We compute a weighted covariance ellipsoid [53] around this point considering the data points

$$X_i = \begin{pmatrix} x_i \\ z_i \\ y_i \end{pmatrix}$$

where x_i are our iterates, $y_i = \binom{y_{1i}}{y_{2i}} = F(x_i) = \binom{F_1(x_i)}{F_2(x_i)}$ and z_i is the vector formed by the columns of the upper half of the symmetric matrix $x_i x_i^T$:

$$z_i = (x_{i11}\ x_{i12}\ x_{i22}\ x_{i13}\ x_{i23}\ x_{i33}\ \dots\ x_{i1m} \dots x_{imm})^T,$$

where $x_{ijk} = x_{ij} x_{ik}$, i.e. the product of the j^{th} and k^{th} component of x_i (i.e. our model is partially quadratic), and the weights

$$w_i = \frac{1}{\|x_i - x_{\text{best}}\|_2^6 \sqrt{p(x_i) - p_{\min} + \frac{1}{10}}}$$

with $p(x) = c^T \binom{x}{y} + e(x)$, the $e(x)$ from above and $p_{\min} = \min_i p(x_i)$, i.e. the closer to x_{best}, the higher the weight (guaranteeing locality) and the smaller $p(x)$ (i.e. the better the point), the higher the weight (guaranteeing a better fit in the area most likely to contain the optimum), but priority is given to locality (while not discarding global information completely). The best point itself (which would have infinite weight) is ignored, it is instead used to center the covariance model. The precise formula is the result of empirical experimentation.

In the presence of implicit equality constraints, there is an additional heuristic: if we have enough points (we used a hardcoded lower limit of 28 points in our implementation), we consider only the half with the lowest equality constraint violation $\|F_2(x)\|_1$ and discard the other half. We force the mean of the covariance model to X_{best} (i.e. the X_i with $x_i = x_{best}$), compute the weighted covariance matrix C_X of the finite sequence $(X_i)_i$ with weights w_i and build a model

$$k_{\text{low}} \leq (X - X_{best})^T C_X^{-1} (X - X_{best}) \leq k_{\text{up}}.$$

The inversion C_X^{-1} is implemented in practice by computing a Cholesky factorization $C_X = LL^T$ (remember that C_X is always positive semidefinite), which

gives us an easy way to handle singular or near-singular C_X by regularizing the Cholesky factorization: we replace zeros or near-zeros in the diagonal of the Cholesky factor, i.e. $L_{ii} \approx 0$, by εC_{Xii} with a small constant ε if $C_{Xii} \neq 0$, and 1 otherwise. More precisely, we regularize the diagonal elements with $L_{ii} \leq \varepsilon C_{Xii}$ with the above ε, the non-strict inequality ensures the case $L_{ii} = 0$ is always taken into account too. In higher dimensions, we add an additional εC_{Xii} to all L_{ii} independently of the value of L_{ii} because this proved beneficial in our tests, the models in higher dimensions tended to be too close to degeneracy without this tweak; we do not do this in lower dimensions because our tests showed the stronger regularization to be counterproductive in low dimensions. Let L be the regularized Cholesky factor and $M := L^{-T}L^{-1}$. Then the model we consider is $k_{\text{low}} \leq k(X) \leq k_{\text{up}}$ with

$$k(X) = (X - X_{best})^T M (X - X_{best}).$$

To put the directional covariance information into practical use, we pick sensible values for the bounds k_{low} and k_{up} as follows: Let m be the dimension of the iterates, i.e. $x_i \in \mathbb{R}^m$. Let $\mathcal{N}$ be the set formed by the $2m+1$ iterates X_i (out of those used to build the covariance model, so points discarded because of an excessive equality constraint violation are not considered) closest to x_{best}. If we don't have $2m+1$ such iterates, let $\mathcal{N}$ be the set of all applicable iterates. In case of points with equal distance, all points with the same distance from x_{best} as the $(2m+1)^{\text{st}}$ closest point are added to $\mathcal{N}$. The distance considered is the Euclidean distance in the x component $\|x_i - x_{\text{best}}\|_2$. We define

$$k_{\text{low}} = \min_{X \in \mathcal{N}} k(X), \quad k_{\text{up}} = 2^{m-1} \max_{X \in \mathcal{N}} k(X).$$

Finally, we obtain the surrogate problem

$$\begin{aligned} \min \quad & c^T \begin{pmatrix} x \\ y_1 \end{pmatrix} \\ \text{s.t.} \quad & k_{\text{low}} \leq k(X) \leq k_{\text{up}} \\ & y_2 = 0 \\ & z = (x_{11}\ x_{12}\ x_{22}\ x_{13}\ x_{23}\ x_{33}\ \dots\ x_{1m} \dots x_{mm})^T \\ & x_l \leq x \leq x_u \\ & F_l \leq y \leq F_u. \end{aligned}$$

We eliminate the redundant variables y_2 and add bounds z_l and z_u for z which are the result of interval multiplications (without rounding control) on the matching components of x_l and x_u and optimize the resulting problem

$$\begin{aligned} \min \quad & c^T \begin{pmatrix} x \\ y_1 \end{pmatrix} \\ \text{s.t.} \quad & k_{\text{low}} \leq k \begin{pmatrix} x \\ z \\ y_1 \\ 0 \end{pmatrix} \leq k_{\text{up}} \\ & z = (x_{11}\ x_{12}\ x_{22}\ x_{13}\ x_{23}\ x_{33}\ \dots\ x_{1m} \dots x_{mm})^T \\ & x_l \leq x \leq x_u \\ & F_l \leq y \leq F_u \\ & z_l \leq z \leq z_u \end{aligned}$$

using a local optimization method, with

$$X_{\text{best}} = \begin{pmatrix} x_{\text{best}} \\ z_{\text{best}} \\ y_{\text{best}} \end{pmatrix}$$

as the starting point. We accept the optimum $\hat{x}$ as our next iterate.

2.5 Global Search

In this section we will assume that there are no implicit equality constraints, the next section will present a way to deal with those. The goal of our global search is to "fill the gaps" in the search space, to improve the likelihood that the surrogate problem closely matches the original problem. Mathematically, this means we want to find points in areas where the density of already existing points is low. Therefore, our global search works by minimizing a density estimator. The motivation for this is given by the observation that neglecting certain areas can lead to missing the global optimum (whenever it happens to be located in the neglected area), an empirical observation which has been formalized and proven by Törn and Žilinskas in [46]. Unfortunately, their theoretical result is not of immediate use to us: they proved that a global optimization algorithm using only local information at the iterates can be complete only if the iterates lie dense in the search space, but both denseness of the iterates and completeness of the algorithm are asymptotical concepts, they only apply if we allow an infinite number of iterates. In practice however, the number of iterates is finite, and in our case, function evaluations (of the black box constraints) are assumed to be expensive, and each new iterate implies a new function evaluation, which forces us to stop after a relatively small number of iterates. Therefore, asymptotical results are only of limited use, and thus we do not attempt to construct a sequence which lies dense in the search space when an infinite number of points are constructed, because this will not be useful in practice anyway. Instead, we take a more heuristic approach, using methods from data analysis to define a concept of density for a finite number of points, which can then be optimized.

As the density estimator we want to minimize, we use a Gaussian mixture model (GMM, see [54]), over the x coordinates only, as those are the only ones we can control. Let N be the number of points we have already computed, then we consider a sum of $\lfloor \frac{N}{4} \rfloor$ Gaussians. As for the local search, we regularize the covariance matrices of the Gaussians during the Cholesky factorization. In this case, if the diagonal element L_{ii} of the cholesky factor is smaller than a small ε, we simply set $L_{ii} := \varepsilon$. In our implementation, ε was taken to be the square root of the machine epsilon `DBL_EPSILON`, but in principle any $\varepsilon > 0$ will do. (In practice, however, a too small ε will cause numerical difficulties, a too large ε will lose too much information.)

We compute the GMMs using the Expectation Maximization (EM) iteration as described in [8]. As our starting points for the Gaussians, we take every 4^{th} iterate. We rotate through our iterates, so the same point is used only once every 4 iterations. The Gaussians are implicitly regularized during the factorization at each step, ensuring that the algorithm does not crash in a singular configuration. We stop the iteration after a fixed 10 steps. This stopping criterion is arbitrary, but it is hard to give a better one, as the EM iteration tends to become numerically unstable when running too many iterations (at least this

was our observation on our test cases). In particular, in the case where the iteration converges to a singular model, more and more useful information is lost to regularization with each iteration step. In addition, stopping after a fixed number of steps also ensures reasonably bounded computation time.

As for the local search, we minimize this density estimator using a third-party local optimizer. In this case, we optimize over the x coordinates only. The optimization is unconstrained in the case without implicit equality constraints, because inequality constraints are ignored by design, both due to the inability to determine the feasible region reliably in the absence of global information and because evaluating the constraint in infeasible areas can give us useful information to approximate it within the feasible region, especially at the boundary. As our starting point, we pick the center of the box. We use explicitly computed derivative information: The gradient of the multi-dimensional Gaussian distribution

$$p(x|j) = \frac{e^{-\frac{(x-\mu)^T C^{-1}(x-\mu)}{2}}}{(2\pi)^{\frac{N}{2}}\sqrt{det(C)}},$$

is given by

$$g(x|j) = -\frac{e^{-\frac{(x-\mu)^T C^{-1}(x-\mu)}{2}} C^{-1}(x-\mu)}{(2\pi)^{\frac{N}{2}}\sqrt{det(C)}} = -p(x|j)C^{-1}(x-\mu)$$

and the Hessian by

$$\begin{aligned} H(x|j) &= -p(x|j)C^{-1} - g(x|j)\left(C^{-1}(x-\mu)\right)^T \\ &= -p(x|j)C^{-1} + p(x|j)C^{-1}(x-\mu)\left(C^{-1}(x-\mu)\right)^T \\ &= p(x|j)\left(C^{-1}(x-\mu)\left(C^{-1}(x-\mu)\right)^T - C^{-1}\right). \end{aligned}$$

As for the local search, we accept the optimum $\hat{x}$ as our next iterate.

The EM procedure gives both the clusters and the density estimator at the same time and the resulting GMM satisfies a provable optimality criterion (maximum expectation) at the fixed point.

2.6 Implicit Equality Constraints

A very hard problem in black box optimization is how to handle implicit equality constraints, i.e. our $F_2(x) = 0$, in an efficient way, the (ideal) goal being not to blow up the search space more than if the equality constraint was explicit. For our local search, this goal was easily reached by substituting $y_2 = 0$ in the resulting surrogate problem, which eliminates the variables y_2 created by the implicit constraints completely. When doing global search, however, the implicit equality constraints are a much more serious problem, and we can no longer reach our goal by a simple substitution. In fact, we cannot reach it fully at all, we can only approximate it. However, we have seen in our experiments that it is still essential to do this, as without this treatment, all the global search tends to be wasted on infeasible areas. Special handling of implicit equality constraints helps a lot with this.

During global search, we do not want to have to search in variables which are determined by constraints. This is because the goal of global search is

to fill the gaps in the search space we can control. Explicit constraints make this easy: we just search in the space spanned by the independent variables and evaluate at the resulting point to get the values of the dependent variables. With implicit constraints, however, we cannot simply do this, because a relation like $F(x, y) = 0$ doesn't give us **any** information on what y values, if any, are valid for a given x, especially in our black box algorithm where we do not have any idea about what F looks like. The only assumption we can reasonably make is that each implicit equality constraint corresponds to a hypersurface (i.e. a submanifold of dimension one less than the dimension of the containing space) or a union of hypersurfaces in our space of independent variables x. While there are counterexamples even for that (just consider the obvious case $F \equiv 0$ which results in the trivial constraint $0 = 0$, or the case $F \equiv 1$ which results in the always infeasible $1 = 0$), most practical implicit equality constraints form such hypersurfaces. Therefore, our heuristics are tuned for the common case. (This is in contrast to implicit inequality constraints which do not reduce the dimension of the search space except in degenerate cases.) In this case, the ideal goal would be to eliminate one dimension from the search space for each equality constraint. Given that this is not possible for the reasons discussed above, our algorithm aims instead at making the search space as narrow as possible in the excess dimension. This is done by introducing quadratic constraints in the global search which attempt to enclose the correct equality constraints from both directions. Since we cannot guarantee that the enclosures will always be rigorous (due to the lack of global information), we rectify our enclosures over time, such that all already computed points are always within the enclosures for the constraints (ignoring rounding errors as everywhere else in the algorithm). At the same time, we generate additional enclosures with every new point we retain, whether it results from a local or a global search (but not for the starting points).

The general approach to obtain such enclosures was suggested by Stefan Vigerske, who is using it successfully in his LaGO optimizer [35, 36]. However, we adapted his approach for our problem. At each new point, we generate two quadratic estimates for each implicit equality constraint, one from above and one from below. (Unlike LaGO, we do not sample new points for the sole purpose of improving the constraint estimates, but instead compute the optimal enclosures from the points we get during the main algorithm. The worst which can happen from lacking constraint information is to land at an infeasible point, which will naturally help improving our enclosures. Function evaluations being expensive, we cannot afford doing additional sampling at the expense of local search.) Consider the implicit equality constraint $F_{2;l}(x) = 0$. We start by computing enclosures

$$\underline{F}(x) = \sum_{i,j} \underline{a}_{ij} x_i x_j + \sum_i \underline{b}_i x_i + \underline{c}$$

and

$$\overline{F}(x) = \sum_{i,j} \overline{a}_{ij} x_i x_j + \sum_i \overline{b}_i x_i + \overline{c}$$

which are the best in the sense of the linear programs

$$\begin{array}{rl} \max & \sum_k \underline{F}(x_k) \\ \text{s.t.} & \forall k : \underline{F}(x_k) \le F_{2;l}(x_k) \\ & \underline{F}(x_{\text{new}}) = F_{2;l}(x_{\text{new}}) \end{array}$$

resp.

$$\begin{array}{rl} \min & \sum_k \overline{F}(x_k) \\ \text{s.t.} & \forall k : \overline{F}(x_k) \ge F_{2;l}(x_k) \\ & \overline{F}(x_{\text{new}}) = F_{2;l}(x_{\text{new}}) \end{array}$$

or in coordinates:

$$\begin{array}{rl} \max & \sum_{i,j} \left(\sum_k x_{ki} x_{kj}\right) \underline{a}_{ij} + \sum_i \left(\sum_k x_{ki}\right) \underline{b}_i + N\underline{c} \\ \text{s.t.} & \forall k : \sum_{i,j} x_{ki} x_{kj} \underline{a}_{ij} + \sum_i x_{ki} \underline{b}_i + \underline{c} \le F_{2;l}(x_k) \\ & \sum_{i,j} x_{\text{new};i} x_{\text{new};j} \underline{a}_{ij} + \sum_i x_{\text{new};i} \underline{b}_i + \underline{c} = F_{2;l}(x_{\text{new}}) \end{array}$$

resp.

$$\begin{array}{rl} \min & \sum_{i,j} \left(\sum_k x_{ki} x_{kj}\right) \overline{a}_{ij} + \sum_i \left(\sum_k x_{ki}\right) \overline{b}_i + N\overline{c} \\ \text{s.t.} & \forall k : \sum_{i,j} x_{ki} x_{kj} \overline{a}_{ij} + \sum_i x_{ki} \overline{b}_i + \overline{c} \ge F_{2;l}(x_k) \\ & \sum_{i,j} x_{\text{new};i} x_{\text{new};j} \overline{a}_{ij} + \sum_i x_{\text{new};i} \overline{b}_i + \overline{c} = F_{2;l}(x_{\text{new}}) \end{array}$$

where N is the number of points, i.e. $N = \sum_k 1$. For the first enclosure (using the data from the starting points), the last constraint, which requires an exact fit at the new point, is omitted. The linear programs are solved using the `lp_solve` library. Unlike LaGO, which proceeds to computing convex enclosures out of these, we work with the (in general nonconvex) quadratic enclosures directly.

We then apply the following relaxation: let's assume the enclosure $\underline{F}(x) \le F(x) \le \overline{F}(x)$ holds for all x. Then the pair of constraints $\overline{F}(x) \ge 0$ and $\underline{F}(x) \le 0$ forms a relaxation for the equality constraint $F(x) = 0$. We apply this to our constraint $F_{2;l}(x) = 0$, replacing it with the two constraints $\overline{F}(x) \ge -\tau$ and $\underline{F}(x) \le \tau$, where $\tau \ge 0$ is a small heuristic tolerance compensating for the fact that our enclosures may be off due to lack of information. We add these two constraints to our global search minimization problem, which is then solved in two steps (as doing it in one step did not achieve satisfying convergence with the two supported local optimizers): we first run the global search with only bound constraints, starting at the center of the box, then we use this point as the starting point for the fully constrained global search.

We never throw away enclosures, instead we accumulate more and more of them as the algorithm proceeds. We do, however, correct them if we find them to be wrong, i.e. if we find a new point x' with $F_{2;l}(x') < \underline{F}(x')$ or $F_{2;l}(x') > \overline{F}(x')$. This is simply done by adjusting the constant term $\underline{c}$ resp. $\overline{c}$ by the amount needed to make the new point fit, i.e. to obtain $F_{2;l}(x') = \underline{F}(x')$ resp. $F_{2;l}(x') = \overline{F}(x')$.

2.7 Implementation

Our implementation of the algorithm we have just described can be obtained at `http://www.tigen.org/kevin.kofler/bbowda/`. It is implemented in the ISO C99 language [21, 22] and licensed under the GNU General Public License, version 3 [15] or later, with special exceptions allowing to link with the third-party optimizers used: Peter Spellucci's DONLP2 [41, 42, 43], the Ipopt [51, 52]

optimizer from the COIN project and the `lp_solve` [5] implementation of the revised simplex method [7]. Ipopt needs a solver for sparse symmetric linear systems of equations. In our tests, we used MUMPS [1, 2, 3, 4] in sequential mode. The third-party libraries were slightly patched by us, see [25, Section 5.1] for details. A full description of our implementation, including documentation of the files describing the input model, which have to be filled in by the user, is provided in [25, Chapter 4].

3 Results

In this section, we discuss the performance of the algorithm, both in terms of speed and quality. We first describe the approach used to obtain the results, then present a summary of the results themselves. More detailed documentation is in [25, Chapter 5]. Finally, we discuss the performance of existing state-of-the-art algorithms on a representative example.

3.1 Testing Methods

We tested our implementation on a few test cases from the libraries 1 and 2 of the COCONUT Benchmark [39, 40]. These libraries contain common test problems converted to COCONUT's internal DAG representation, which can be converted to C code using a small modification (described in [25, Section 5.1]) of the APIs from the COCONUT Environment [38, 34]. Library 1 contains problems taken from GLOBALLib [16] and the Handbook of Test Problems in Local and Global Optimization [13]. Library 2 corresponds to Vanderbei's CUTE Test Collection [50].

It is important to note that these test cases involve cheap, analytical functions, not expensive black box functions as in the real-world problems our algorithm is targeted at. This implies that all time measurements essentially only measure the time spent within the algorithm. Therefore, the number of function evaluations is an important metric which must be taken into account when estimating the time spent on real-world problems.

One problem we encountered with this test set is that many variables in the test problems lack one or both bounds. (In some cases, the variables are truly unbounded, in others, bounds easily follow from the constraints.) Our algorithm, however, can only work with finite bounds. Moreover, it is important for the bounds for the x variables to be as close together as possible, as our algorithm performs better when more of the x within the bounds are also within the bounds for $y = F_1(x)$. This is mainly due to the fact that, by design, our global search only handles implicit equality constraints specially, not inequality constraints (i.e. the bounds on the variables given by explicit equality constraints). The reasons for this tradeoff are explained in sections 2.2 and 2.6. Therefore, artificial large bounds like $[-1000, 1000]$ are not usable for most problems, at least for x. (The bounds for y are less sensitive because they are only used in the local surrogate models where locality is ensured by the covariance ellipsoid, so those bounds being larger than necessary is not a big deal.) Thus, we set reasonable bounds for the variables by hand where they are missing. (Any added bounds will be presented together with the results below.) The number

of function values needed for convergence was determined by inspection of the intermediate results; this is appropriate for expensive function evaluations.

In addition, we ran some simple scalability tests: we tested a trivial quadratic objective function and the multidimensional Rosenbrock function (the precise expressions and bounds will be given in the next section), using both the explicit and the implicit formulation for the equality constraints, and checked up to what dimension convergence can be achieved, and at what speed.

We report results obtained both with DONLP2 and with Ipopt (using MUMPS as the linear solver) so the results with the different NLP optimizers can be compared, and also to reduce the influence of the performance of the third-party optimizers on our overall results.

The tests were run on a Pentium 4 Northwood 2.6 GHz running a fully-updated Fedora 7.

3.2 Results

In this section, we present the results obtained from the tests described above. In all our tests, we did not provide starting values by hand, relying instead on our automated starting point generation algorithm described in section 2.3.

In some cases, objective function values below the true minimum are returned. This can be explained by the fact that our algorithm can return points which are only feasible up to a given tolerance. It would be unrealistic to expect complete feasibility out of a black box algorithm, especially in the presence of implicit equality constraints. Where not otherwise specified, the feasibility tolerance, i.e. the maximum allowed constraint violation, was set to 0.001.

3.2.1 COCONUT Benchmark Library 1

We ran our optimizer on two examples from library 1 of the COCONUT Benchmark: `circle` and `dispatch`.

circle The best known result for the `circle` (*Circle Enclosing Points*) problem, found by `MINOS`, is:
`4.5742477881 at [5.3880763381, 6.3990975587, 4.5742477881]`

As there were no usable bounds for the three x variables, we provided the bounds $[0, 10]$ for all three. With `DONLP2` as the local optimizer, our algorithm found the solution:
`4.626279 at x=[5.069064,6.109449,4.626279]`
after 150 function evaluations and 17.430 seconds. (Increasing the number of allowed function evaluations did not lead to a better solution.) With `Ipopt` as the local optimizer, our algorithm found the solution:
`4.574240 at x=[5.388075,6.399094,4.574240]`
after 50 function evaluations and 52.046 seconds.

dispatch The `dispatch` (*Economic Load Dispatch Including Transmission Losses*) problem originates from power generation. The best known solution, found by `MINOS`, is:
`3155.2879268581 at [50.0000000000, 75.4858804799, 93.2622541695, 8.7481346494]`

The model does not provide bounds for x_4 and the objective function nor an upper bound for the inequality constraint. Therefore we computed the bounds from those for the first three x variables and rounded to obtain the following bounds: $x_4 \in [-200, 320]$, the objective $y_1 \in [-1000, 7000]$ and the inequality constraint $y_2 \in [210, 730]$ ($y_2 \geq 210$ was the original inequality constraint). The objective function includes a constant term 653.1000000000000227374 which cannot be represented by our implementation, this term has to be added to the optimum function value provided by our optimizer. With `DONLP2` as the local optimizer, our algorithm found the solution:
`2502.184619 at x=[50.000000,76.060592,92.712255,8.773665]`
(i.e. an actual optimum of 3155.284619) after 250 function evaluations and 51.357 seconds. With `Ipopt` as the local optimizer, our algorithm found the solution:
`2502.173543 at x=[50.000000,75.527606,93.221193,8.749709]`
(i.e. an actual optimum of 3155.273543) after 251 function evaluations (250 plus one extrapolated point) and 1 minute 10.768 seconds.

3.2.2 COCONUT Benchmark Library 2

We ran our optimizer on three examples from library 2 of the COCONUT Benchmark: `aljazzaf`, `twobars` and `maratos`.

aljazzaf The best known solution for the Aljazzaf example problem, found by `OQNLP`, is:
`75.0049000369 at [0.0000000000, 0.9999999940, 0.9999990004]`

We used the bounds $[-1, 1]$ for the x variables and $[0, 500]$ for y. With `DONLP2` as the local optimizer, our algorithm found the solution:
`74.962735 at x=[0.000209,0.999946,0.999368]`
after 201 function evaluations (200 plus one extrapolated point) and 2 minutes 1.482 seconds. With `Ipopt` as the local optimizer, our algorithm failed to find a point satisfying the feasibility tolerance. After 250 function evaluations and 19 minutes 32.282 seconds, extrapolation produced the 251^{st} point `[-0.000108,0.999998,1.000075]` which still fails to satisfy the tolerance.

twobars The `twobars` (*Structural analysis of the simplest two bar scheme*) problem originates from mechanics. The best known solution, found by `DONLP2`, is:
`1.50865 at [1.41163, 0.377072]`

This model already provides usable bounds for the x variables, for y we kept the default $[-1000, 1000]$ from our converter. With `DONLP2` as the local optimizer, our algorithm found the solution:
`1.557142 at x=[1.497032,0.286213]`
after 150 function evaluations and 8.603 seconds. (Increasing the number of allowed function evaluations did not lead to a better solution.) With `Ipopt` as the local optimizer, our algorithm found the solution:
`1.508620 at x=[1.411632,0.377006]`
after 100 function evaluations and 55.188 seconds.

maratos The `maratos` problem is hard to solve correctly because the variables in the optimal solution have very different scales. The best known solution,

found by DONLP2, is:
`-0.999999 at [1, -1.72277e-06]`

For this problem, we used the bounds $[-2, 2]$ for all x and y variables. We also decreased our feasibility tolerance to 10^{-5}. (We tried decreasing it further to 10^{-6}, but our optimizer failed to find a feasible point with that low a tolerance.) In addition, we modified the final output in `main.c` to print the second x coordinate for the retained optimum in exponential (`%lg`) format. With DONLP2 as the local optimizer, we obtained:
`-1.000000 at x=[1.000001,-5.2591e-05]`
after 100 function evaluations and 20.791 seconds. With `Ipopt` as the local optimizer, we obtained:
`-1.000000 at x=[1.000001,-8.36745e-05]`
after 100 function evaluations and 2 minutes 13.876 seconds.

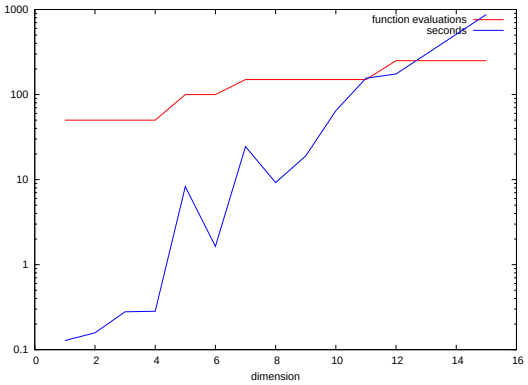
3.2.3 Scalability Tests

The results of the scalability tests will be presented in partly tabular, partly graphical form: d is the dimension of the problem, which we vary to test scalability. For every d, we chart the number of function evaluations, which we increased in steps of 50 until convergence was obtained, and the computation time in seconds, and we list the obtained result in a table. For conciseness reasons, we only list the objective function values, the corresponding x points can be found in [25, Section 5.2.3].

Quadratic Objective as Explicit Equality Constraint. We used the trivial quadratic objective $\sum_i \left(x_i - \left(\sqrt{2}-1\right)\right)^2$ and used the usual variable substitution to turn it into the explicit equality constraint $y = F_1(x) = \sum_i \left(x_i - \left(\sqrt{2}-1\right)\right)^2$ and the linear objective function y. We chose the bound constraints $[-1, 1]$ on x_i and $[-d, 2d]$, where d is the dimension of the vector x, on y. (The bounds on y are not a true constraint because $x_i \in [-1, 1]$ implies $y \in [0, 2d]$). We also tried the bounds $[-d, d]$ for y.

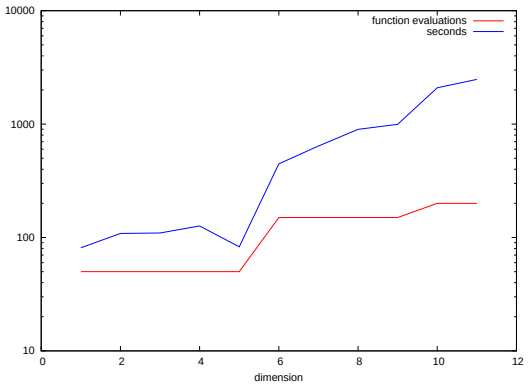
With the bounds $[-d, 2d]$ and DONLP2, we obtained the following results:

d	result
1	0.000000
2	0.000000
3	0.000000
4	0.000000
5	0.000000
6	0.000000
7	0.000006
8	0.000004
9	0.000024
10	0.002135
11	0.005927
12	0.005325
15	0.010697



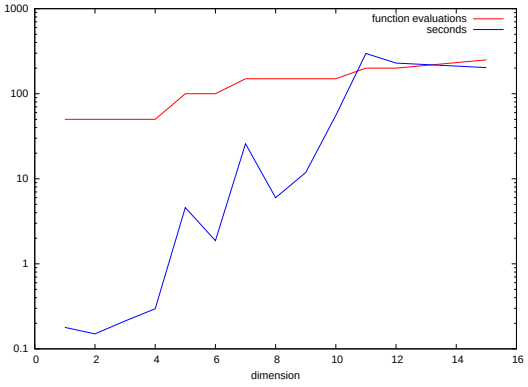
With the bounds $[-d, 2d]$ and `Ipopt`, we obtained the following results:

d	result
1	0.000000
2	0.000000
3	0.000000
4	0.000000
5	0.000000
6	0.001864
7	0.000082
8	0.090652
9	0.005280
10	0.008180
11	0.027408



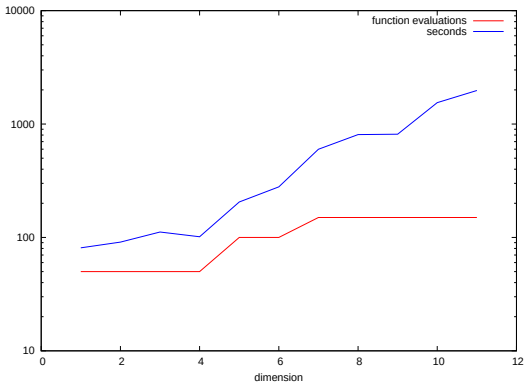
With the bounds $[-d, d]$ and `DONLP2`, we obtained the following results:

d	result
1	0.000000
2	0.000000
3	0.000000
4	0.000000
5	0.000000
6	0.000000
7	0.000003
8	0.000005
9	0.000033
10	0.000014
11	0.000120
12	0.001256
15	0.000030



With the bounds $[-d, d]$ and `Ipopt`, we obtained the following results:

d	result
1	0.000000
2	0.000000
3	0.000000
4	0.000000
5	0.000101
6	0.039120
7	0.008043
8	0.002461
9	0.000392
10	0.004097
11	0.034805



For $d = 6$, increasing the number of function evaluations from 100 to 150 did not improve the result either.

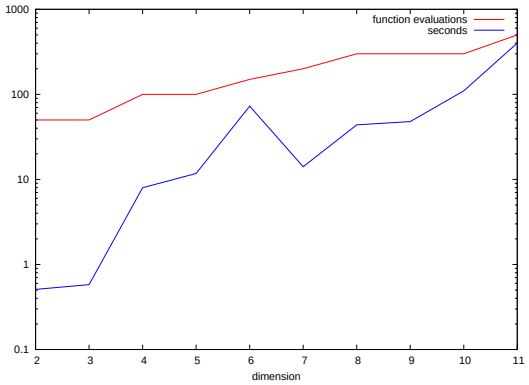
Quadratic Objective as Implicit Equality Constraint. In order to test equality constraints in implicit $f(x) = 0$ form, we rewrote the above problem by formulating the objective as an implicit rather than an explicit equality

constraint:

$$\sum_{i=1}^{d-1} \left(x_i - \left(\sqrt{2}-1\right)\right)^2 - x_d = 0.$$

To get these examples to converge in a timely manner for dimensions 7 and higher, a few tweaks were needed. First of all, we had to disable treatment of implicit equality constraints in the global search (see section 2.6) for dimension 7 and higher, because the LPs needed to approximate the implicit equality constraint for the global search took too long to compute: without this option, solving $d = 7$ with `DONLP2` appeared to converge, but took over 3 hours! `lp_solve` also ran into numerical problems in higher dimensions, which means most of that time was wasted without finding actual enclosures for our constraint. And secondly, we had to increase the feasibility tolerance `tol` from the default .001 in higher dimensions because the tolerance could not be reached, possibly partly due to the global search not being able to take the constraint into account. With `DONLP2`, we obtained the following results:

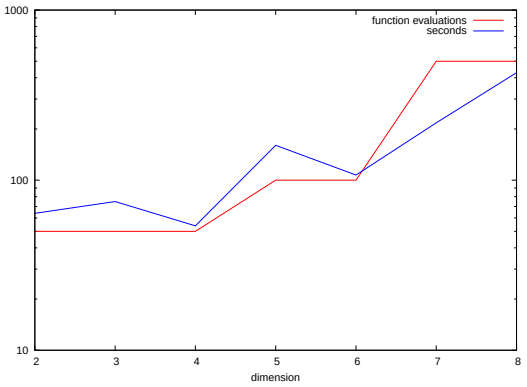
d	tol	result
2	.001	-0.000608
3	.001	-0.000804
4	.001	0.000008
5	.001	-0.000984
6	.001	0.007643
7	.01	0.000680
8	.01	-0.001772
9	.1	-0.034728
10	.1	-0.039629
11	.5	-0.046004



The reduction in computation time between dimensions 6 and 7 is due to the fact that we disabled the LPs for the enclosure of the implicit equality constraint in the global search for dimension 7 and higher, otherwise the computation time would have increased significantly instead.

With `Ipopt`, we obtained the following results:

d	tol	result
2	.001	-0.000629
3	.001	-0.000781
4	.001	-0.000599
5	.001	0.000139
6	.001	-0.000747
7	.1	-0.098857
8	.5	-0.050436



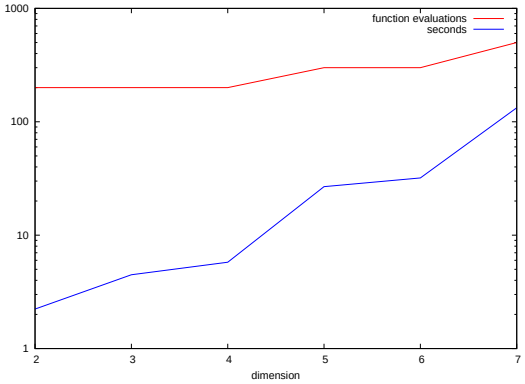
Rosenbrock Function as Explicit Equality Constraint. For this test, we used the multidimensional Rosenbrock function, formulated as an explicit equality constraint

$$y = \sum_{i=1}^{d-1} \left((1 - x_i)^2 + 100 \left(x_{i+1} - x_i^2 \right)^2 \right),$$

with the bound constraints $[-1, 2]$ on x_i and $[-1000, 2700]$ on y.

With `DONLP2`, we obtained the following results:

d	result
2	0.001435
3	0.011359
4	0.008234
5	0.046058
6	0.020165
7	0.053847



With `Ipopt`, we could not obtain convergence even for $d = 2$:

d	function evaluations	result	time (seconds)
2	200	0.135674	367.647

Higher values for the number of function evaluations also failed to improve the solution.

Rosenbrock Function as Implicit Equality Constraint. The last, and hardest, scalability test, was the multidimensional Rosenbrock function written as an implicit equality constraint

$$\sum_{i=1}^{d-2} \left((1 - x_i)^2 + 100 \left(x_{i+1} - x_i^2 \right)^2 \right) - x_d = 0$$

with the bound constraints $[-1, 2]$ on $x_1, \ldots, x_{d-1}$ and $[-1, 1]$ on x_d. In this example, our primary measurement target was not so much speed, but whether convergence can be obtained at all.

With `DONLP2`, we obtained the following results:

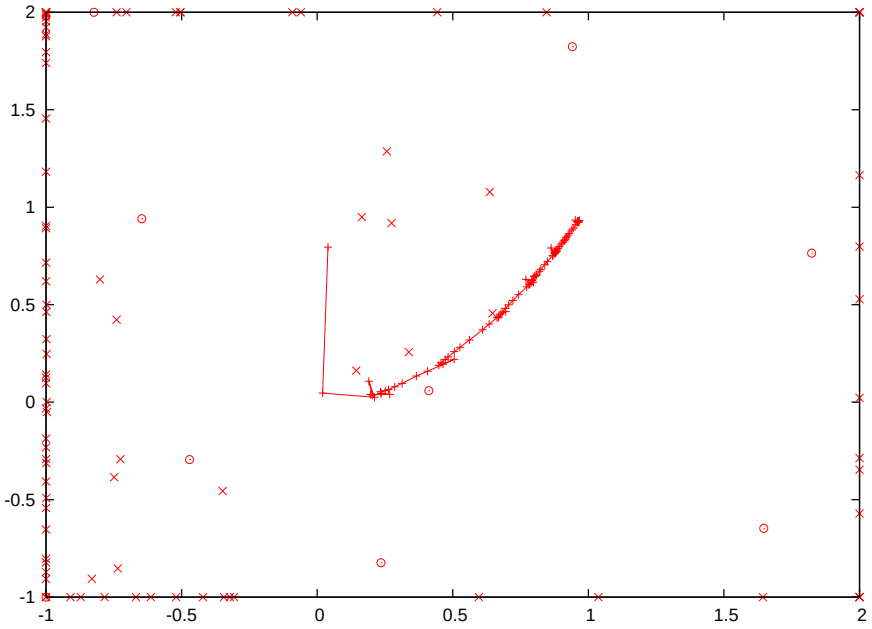
d	function evaluations	result	time (seconds)
3	100	0.018488	4.896
3	200	0.000040	18.634
4	does not converge		

With `Ipopt`, we could not obtain convergence even for $d = 3$.

3.2.4 Graphical Representation

The following figure shows the points evaluated for the original 2-dimensional Rosenbrock function expressed as an explicit equality constraint using `DONLP2` as the local optimizer. The circles represent the

starting points, the X-es the points found by the global search and the pluses connected by a line the points found by the local search.



3.3 Comparison

Finally, we evaluated existing publicly available solvers for a comparison. As our testcase, we considered the (2-dimensional) Rosenbrock function as an implicit equality constraint as it is representative of the hardest problem type our algorithm can solve.

As explained in the introduction, we found that many of the state-of-the-art gradient-free solvers (NOMAD [30], CONDOR [48, 49], the OpenOpt [29] solvers) were entirely unable to solve this type of problem, leaving us with just a small choice of candidates for a comparison. We tried three solvers on our testcase, two of which follow evolutionary approaches, the third one a Generating Set Search (GSS) strategy. As representatives of evolutionary algorithms, we picked two codes from the top 5 at the CEC-06 Special Session on Constrained Real-Parameter Optimization [44]: DMS-PSO [31] by J.J. Liang and P.N. Suganthan, which has achieved rank 2, and the DDE code [32] by Efrén Mezura-Montes, a derivative of which has achieved rank 4. (The codes for ranks 1 and 3 appear not to be publicly available.)

- The DDE code [32] by Efrén Mezura-Montes is an implementation of diversity differential evolution. We found it able to solve the Rosenbrock function as an implicit equality constraint, even the multidimensional test case with 8 variables (7-dimensional Rosenbrock function), but at the expense of a huge number of function evaluations. By default, the algorithm always does exactly 225090 function evaluations. While this can be tuned by reducing the number of generations or the number of individuals per generation, we found that even on the 3-dimensional case, i.e. the standard 2-dimensional Rosenbrock function, the algorithm was no longer

able to achieve convergence with any settings which significantly reduce the number of function evaluations. In contrast, our algorithm converged to a near-optimal point with only 100 function evaluations and achieved a precision of $4 \cdot 10^{-5}$ in only 200 function evaluations, i.e. a factor of over 1000 fewer.

- DMS-PSO (Dynamic Multi-Swarm Particle Swarm Optimizer) [31] by J.J. Liang and P.N. Suganthan is another evolutionary approach. By default, the algorithm always does exactly 500000 function evaluations. We found that the number of function evaluations required to achieve convergence varied significantly from test run to test run due to strong nondeterministic behavior, but was always above 100000. For reliable convergence, we found a number of function evaluations on the order of the default 500000 to be required, i.e. a factor of 2500 more than with our algorithm. From these two results, we conclude that evolutionary approaches are powerful, but not of practical use for functions which are really expensive to evaluate.
- HOPSPACK [27] is a framework for derivative-free optimization which ships with an implementation of the Generating Set Search (GSS) algorithm. It theoretically supports implicit black-box equality constraints. Unfortunately, it gets stuck at the non-optimal point (.8112, .6756) for the Rosenbrock function as an implicit equality constraint (the value found for the third variable is .03468 which is within the feasibility tolerance of .001 with a constraint violation of $9.760 \cdot 10^{-4}$, the exact value of the Rosenbrock function at that point is .03567, the optimum is 0 at $(1, 1)$), giving up and claiming convergence after 1708 function evaluations.

We conclude that our algorithm is a big improvement over the state of the art for the case of implicit black-box equality constraints that are expensive to evaluate.

4 Conclusion

We tested our algorithm on a few low-dimensional example programs from the COCONUT benchmark and performed some scalability tests. We obtained satisfying results on the problems we tested: on the five low-dimensional problems from the COCONUT benchmark we tested, convergence was obtained in all cases, though we were unable to solve the Aljazzaf and Maratos examples to full precision. The algorithm scales up to a dimension of around 15 for the simple quadratic objective. It is also able to optimize the multidimensional Rosenbrock function up to dimension 7. The problems with implicit equality constraints did not scale as well, however we were able to optimize the 2-dimensional Rosenbrock function as a 3-dimensional implicit equation to a very high precision. We observed a strong dependence of the results on the local optimizer used: for some test cases, the best result was obtained with `DONLP2`, for others with `Ipopt`; the result with the other optimizer was often significantly worse.

All in all, we found the results to be competitive, at least for low enough dimensions. Our algorithm found the optimum up to the desired tolerance (with at least one of the two local optimizers we tried for the surrogate models) in all the testcases we tried up to a dimension of 7 to 15 (depending on the problem) for explicit $y = f(x)$ formulations and 3 to 11 for implicit $f(x) = 0$ ones.

While we were successful at proving the concept and obtained some very promising results, there are several potential improvements which can be the subject of future research.

As can be seen in the graphical representation, our global search produces many points on the border of the box being searched and few points in the interior. Fine-tuning the global search to produce more interior points is likely to improve global convergence.

Another possible improvement may be obtained for the stopping criterion: currently, our algorithm always evaluates exactly the maximum number of points, it is unable to verify whether convergence has already occurred or not.

Implicit equality constraints in higher dimensions are another place where there is definitely room for improvement. Most, if not all, algorithms currently on the market fail to handle black box implicit equality constraints, so our algorithm is pioneering this domain. In the lowest dimensions, our algorithm handles such constraints very well, but in medium to high dimensions (starting at around 5 to 7), we ran into both numerical and speed-related difficulties with our linear programs.

The extrapolation technique used to obtain feasible points with good objective function values in the presence of implicit equality constraints could also be a target for improvement: when it works, it usually produces very good points, however sometimes the extrapolation matrix is too ill-conditioned and sometimes bad input points are missed by the outlier detection and force bad extrapolated points.

It may also be worthwhile to explore the possibility of handling inequality constraints (i.e. the bounds on the variables given by explicit equality constraints) more like the implicit equality constraints are handled, both in terms of restricting the global search to the feasible area and by applying the final extrapolation step.

Given the high sensitivity of the results to the local optimizer used for the surrogate and density models, it would be worthwhile to try other local optimization methods for the surrogate models. It might even be worth a try to run a global optimization method on the global density models.

Finally, the current implementation is a prototype and does not always use the most efficient algorithms for its computations. To get it to scale to higher dimensions, optimizing the implementation for speed may be worthwhile.

References

[1] P. Amestoy, I. Duff, A. Guermouche, J. Koster, J.-Y. L'Excellent, S. Pralet et al.: MUMPS. Software package downloadable from `http://graal.ens-lyon.fr/MUMPS/`

[2] P.R. Amestoy, I.S. Duff and J.-Y. L'Excellent: Multifrontal parallel distributed symmetric and unsymmetric solvers. Comput. Methods in Appl. Mech. Eng., 184, pp. 501–520, 2000.

[3] P.R. Amestoy, I.S. Duff, J. Koster and J.-Y. L'Excellent: A fully asynchronous multifrontal solver using distributed dynamic scheduling. SIAM Journal of Matrix Analysis and Applications, Vol 23, No 1, pp. 15–41, 2001.

[4] P.R. Amestoy, A. Guermouche, J.-Y. L'Excellent and S. Pralet: Hybrid scheduling for the parallel solution of linear systems. Parallel Computing Vol 32 (2), pp. 136–156, 2006.

[5] M. Berkelaar, P. Notebaert, K. Eikland et al.: lp_solve. Software package downloadable from `http://groups.yahoo.com/group/lp_solve/`

[6] R. Carter, J. Gablonsky, A. Patrick, C. Kelley, and O. Eslinger: Algorithms for noisy problems in gas transmission pipeline optimization. Optimization and Engineering, 2 (2001), pp. 139–157.

[7] G.B. Dantzig: Computational Algorithm of the Revised Simplex Method. Rand Corporation, 1953.

[8] A. Dempster, N. Laird and D. Rubin: Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society, Series B, 39(1):1–38, 1977.

[9] D.E. Finkel and C.T. Kelley: Convergence analysis of the DIRECT algorithm. Optimization On-line Digest, August 2004.

[10] R. Fletcher and S. Leyffer: Nonlinear programming without a penalty function. Mathematical Programming, 91:239–270, 2002.

[11] R. Fletcher, S. Leyffer and Ph.L. Toint: A Brief History of Filter Methods. SIAG/Optimization Views-and-News, 18(1):2–12, 2007.

[12] A. Flexer, E. Pampalk and G. Widmer: Novelty detection based on spectral similarity of songs. Proceedings of 6$^{\text{th}}$ International Conference on Music Information Retrieval, pp. 260–263, September 2005.

[13] C.A. Floudas, P.M. Pardalos, C.S. Adjiman, W.R. Esposito, Z.H. Gümüs, S.T. Harding, J.L. Klepeis, C.A. Meyer and C.A. Schweiger: Handbook of Test Problems in Local and Global Optimization. Kluwer, Dordrecht 1999.

[14] S. Forrest: Genetic algorithms: principles of natural selection applied to computation. Science 261 (1993), pp. 872–878.

[15] Free Software Foundation: GNU General Public License, Version 3. Free Software Foundation, 2007. `http://www.gnu.org/licenses/gpl-3.0.html`

[16] GamsWorld group et al.: GLOBAL Library. GAMS World. `http://www.gamsworld.org/global/globallib.htm`

[17] S.Y. Gogolenko and V. Svjatnyj: Architecture aware parallelization of solvers for PDE systems on geometrical graphs. Computer Science - Research and Development, 23(3-4):225-230, 2009.

[18] J. Holland: Genetic algorithms and the optimal allocation of trials. SIAM J. Computing 2 (1973), pp. 88–105.

[19] L. Ingber: Simulated annealing: Practice versus theory. Math. Comput. Modelling 18 (1993), pp. 29–57.

[20] D.R. Jones, M. Schonlau and W.J. Welch: Efficient Global Optimization of Expensive Black-Box Functions. Journal of Global Optimization, 13:455–492, 1998.

[21] ISO C Working Group JTC1/SC22/WG14: ISO/IEC 9899:1999 – Programming languages – C. International Organization for Standardization, 1999.

[22] ISO C Working Group JTC1/SC22/WG14: WG14/N1124 – ISO/IEC 9899:TC2. Committee draft, 2005. `http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf`

[23] D.R. Jones, C.D. Perttunen and B.E. Stuckman: Lipschitzian optimization without the Lipschitz constant. J. Optimization Th. Appl. 79 (1993), pp. 157–181.

[24] S. Kirkpatrick, C.D. Geddat, Jr., and M.P. Vecchi: Optimization by simulated annealing. Science 220 (1983), pp. 671–680.

[25] K. Kofler: Black Box Optimization with Data Analysis. Diploma thesis. Universität Wien, 2007.

[26] J. Liang, T.P. Runarsson, E. Mezura-Montes, M. Clerc, P.N. Suganthan, C.A. Coello Coello and K. Deb: Problem Definitions and Evaluation Criteria for the CEC 2006. Special Session on Constrained Real-Parameter Optimization, Technical Report, Nanyang Technological University, Singapore, 2006

[27] T.G. Kolda, T. Plantenga et al.: HOPSPACK. Software package downloadable from `https://software.sandia.gov/trac/hopspack/wiki/HopspackTeam`

[28] M. Krasnyk, S.Y. Gogolenko, M. Ginkel, K. Bondareva, O. Milokhov and I. Babenko: Diana. Software package downloadable from `http://www.mpi-magdeburg.mpg.de/projects/diana/people.html`

[29] D.L. Kroshko et al.: OpenOpt. Software package downloadable from `http://openopt.org/`

[30] S. Le Digabel et al.: NOMAD. Software package downloadable from `http://www.gerad.ca/nomad/Project/Home.html`

[31] J.J. Liang and P.N. Suganthan: Dynamic Multi-Swarm Particle Swarm Optimizer with a Novel Constraint-Handling Mechanism. Special Session on Constrained Real-Parameter Optimization, CEC-06, Vancouver, 2006.

[32] E. Mezura-Montes: Diversity Differential Evolution to Solve Constrained Optimization Problems. Software package downloadable from `http://www.lania.mx/~emezura/publications.php?option=2`

[33] Z. Michalewicz: Genetic Algorithm + Data Structures = Evolution Programs, 3rd ed. Springer, New York, 1996.

[34] A. Neumaier: Complete Search in Continuous Global Optimization and Constraint Satisfaction. Acta Numerica 2004 (A. Iserles, ed.). Cambridge University Press 2004, pp. 271–369.

[35] I. Nowak and S. Vigerske: LaGO. Software package downloadable from `https://projects.coin-or.org/LaGO`

[36] I. Nowak and S. Vigerske: LaGO - a (heuristic) Branch and Cut algorithm for nonconvex MINLPs. Preprint, Institut für Mathematik, Humboldt-Universität zu Berlin (ISSN 0863-0976), 12. `http://www.mathematik.hu-berlin.de/publ/pre/2006/P-06-24.ps`

[37] J. Sacks, W.J. Welch, T.J. Mitchell and H.P. Wynn, Design and analysis of computer experiments (with discussion), Statistical Science 4, pp. 409–435, 1989.

[38] H. Schichl et al.: COCONUT Environment. Software package downloadable from `http://www.mat.univie.ac.at/coconut-environment/`

[39] O. Shcherbina et al.: COCONUT Benchmark. Package downloadable from `http://www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html`

[40] O. Shcherbina, A. Neumaier, D. Sam-Haroud, Xuan-Ha Vu and Tuan-Viet Nguyen, Benchmarking global optimization and constraint satisfaction codes. Ch. Bliek, Ch. Jermann and A. Neumaier (eds.), Global Optimization and Constraint Satisfaction, pp. 211–222. Springer, Berlin 2003.

[41] P. Spellucci: DONLP2. Software package downloadable from `http://www.mathematik.tu-darmstadt.de:8080/ags/ag8/Mitglieder/spellucci_en.html`

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

[43] P. Spellucci: A new technique for inconsistent problems in the SQP method. Math. Meth. of Oper. Res. 47, (1998), 355–400. Physica Verlag, Heidelberg, Germany.

[44] P.N. Suganthan: Special Session on Constrained Real-Parameter Optimization, CEC-06, Vancouver, Canada, 17-21 July. Web page. `http://www3.ntu.edu.sg/home/EPNSugan/index_files/CEC-06/CEC06.htm`

[45] D.M.J. Tax, R.P.W. Duin: Outlier detection using classifier instability. Amin A. et al. (eds.), Advances in Pattern Recognition, Proc. Jont IAPR Int. Workshop SSPR'98 and SPR'98. Lecture Notes in Computer Science, Springer, pp. 593–601, 1998.

[46] A. Törn and A. Žilinskas: Global Optimization. Lecture Notes in Computer Science, Vol. 350. Springer-Verlag, Berlin, 1989.

[47] P.J.M. Van Laarhoven and E.H.L. Aarts: Simulated Annealing: Theory and Applications. Kluwer, Dordrecht, 1987.

[48] F. Vanden Berghen, H. Bersini: CONDOR. Software package downloadable from http://www.applied-mathematics.net/optimization/CONDORdownload.html

[49] F. Vanden Berghen, H. Bersini: CONDOR, a new parallel, constrained extension of Powell's UOBYQA algorithm: Experimental results and comparison with the DFO algorithm. Journal of Computational and Applied Mathematics, Elsevier, Volume 181, Issue 1, September 2005, pp. 157–175.

[50] B. Vanderbei, H.Y. Benson et al.: Cute Models. Package downloadable from http://www.sor.princeton.edu/~rvdb/ampl/nlmodels/cute/index.html

[51] A. Wächter et al.: Ipopt. Software package downloadable from https://projects.coin-or.org/Ipopt

[52] A. Wächter and L.T. Biegler: On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming. Mathematical Programming 106(1), pp. 25–57, 2006.

[53] P.R. Wolf and C.D. Ghilani: Adjustment Computations. John Wiley and Sons, Inc., New York, 1997.

[54] J.H. Wolfe: Pattern clustering by multivariate mixture analysis. Multivariate Behavioral Research, 5 (1970), pp. 329–350.