

GLOPTLAB: A configurable framework for the rigorous global solution of quadratic constraint satisfaction problems

Ferenc Domes

Faculty of Mathematics, University of Vienna
Nordbergstrasse 15, A-1090 Vienna, Austria

May 7, 2008

Abstract. GLOPTLAB is a software environment for the rigorous solution of polynomial constraint satisfaction problems, written in MATLAB. All applied methods are rigorous, hence it is guaranteed that no feasible point is lost. Some emphasis is given to finding a bounded initial box containing all feasible points, in cases where other complete solvers rely on non-rigorous heuristics.

The inputs are converted to a quadratic format, which forms the basis of the algorithms implemented in GLOPTLAB, which are used to reduce the search space: scaling, constraint propagation, linear relaxations, strictly convex enclosures, conic methods, and branch and bound. From the method repertoire custom made strategies can be built, with a user friendly graphical interface.

1 Introduction

GLOPTLAB is a global optimization environment, for the rigorous solution of *quadratic constraint satisfaction problems*

$$F(x) \in \mathbf{F}, \quad x \in \mathbf{x}. \quad (1)$$

Here $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a vector valued quadratic function, and $\mathbf{F} \subseteq \mathbb{R}^m$, $\mathbf{x} \subseteq \mathbb{R}^n$ are sets defined by lower and upper bounds only. The $F_i(x) \in \mathbf{F}_i$ are *quadratic constraints* and the $x_i \in \mathbf{x}_i$ are *bound constraints*. An $x \in \mathbf{x}$ is called a *feasible point* or a *solution* if $F(x) \in \mathbf{F}$ is satisfied. The task is to find one or all feasible point; the problem is called *infeasible* if there are no feasible points.

Rigorous methods for solving the constraint satisfaction problem (1) which are used to reduce the sets \mathbf{x} and \mathbf{F} can be written as $\Gamma : (\mathbf{x}, \mathbf{F}) \rightarrow (\tilde{\mathbf{x}}, \tilde{\mathbf{F}})$ and have the property

$$\{x \in \mathbf{x} \mid F(x) \in \mathbf{F}\} \subseteq \{x \in \tilde{\mathbf{x}} \mid F(x) \in \tilde{\mathbf{F}}\}.$$

This guarantees that *no feasible points are lost* during the reduction of the sets \mathbf{x} and \mathbf{F} . All methods implemented in GLOPTLAB are rigorous, and thus they met the above property.

GLOPTLAB supports rigorous input, however since most of the methods are based on interval arithmetic these intervals should be narrow, consisting only of uncertainties due to rounding error in the representations of exact problems.

GLOPTLAB is implemented in MATLAB and is meant to be a *testing and development platform*.

GLOPTLAB contains a number of methods, jointly developed with Arnold Neumaier. A brief introduction to them is given in Section 3 while detailed description of these methods can be found in the references. The problems are converted into an internal polynomial format analyzed and preprocessed by the *problem simplification*. *Constraint propagation* is a fast and effective method which is also used as a part of other more complicated methods (see DOMES & NEUMAIER [4]). We use different *linear relaxation* techniques to get finite bounds or decrease the size of the search space (see DOMES & NEUMAIER [6]). *Strict convex enclosures* compute a nearly optimal interval hull of strictly convex constraints (see DOMES & NEUMAIER [5]). *Conic methods* may lead to spectacular reductions of the search domain, but require a great deal of computation time (see DOMES & NEUMAIER [8]). *Branch and bound* divides the search space into smaller subdomains and applies some of the above methods to reduce their size or even eliminate them when they do not contain feasible points. The boxes which remain after the branching can be merged to a single or fewer ones by computing their *interval hull* or finding and bounding the *clusters* of them. *Finding and verifying feasible points* are important if we search only for a single solution of the constraint satisfaction problem. Different *scaling algorithms* guarantee that the methods which are not scaling invariant do not run into difficulties due to bad scaling (see DOMES & NEUMAIER [7]).

Some of the above methods make use of *external toolboxes*. Since the verification is often based in interval techniques INTLAB (RUMP [23]) is probably the most important toolbox, and although some methods avoid to use it to speed up the computation, it is always needed to run GLOPTLAB. Unverified solutions of linear programs are obtained by using LPSOLVE (BERKELAAR et al. [1]), SEDUMI (STURM et al. [28]) or SDPT3 (TOH et al. [29]). The latter two can be also used to optimizing over symmetric cones which make them an essential part of the rigorous conic methods. In general, the nonrigorous parts are only used for generating approximations which are needed in subsequent rigorous computation steps. The algorithms for finding and verifying feasible points make use of local solvers like projected BFGS and conjugate gradient methods from KELLEY [15]. The conversion from the AMPL format to the internal problem representation of GLOPTLAB is done by AMPL (FOURER et al. [10]) in connection with the COCONUT environment SCHICHL [25], while the parsing and conversion from a simplified AMPL format is done by the SMPL parser (MARKÓT [18]).

In Section 4 we discuss the integration of the above methods in the GLOPTLAB environment, features like the building of user defined solution strategies or the graphical user interface and the possibilities of extending the method repertoire. In the last section some examples and test results are given.

There are a number of software packages for solving constraint satisfaction problems. The NUMERICA software by HENTENRYCK et al. [11] uses branch and prune methods and interval constraint programming to solve constraint satisfaction problems. The ICOS solver by LEBBAH [16] is a software package for the rigorous solution of nonlinear and continuous constraints, based on constraint programming and interval analysis techniques. The PALM system by JUSSIEN & BARICHARD [13] uses explanation-based constraint programming, and propagates the constraints of the problem, learning from failure and from the solver. The price winning solver BARON by SAHINIDIS & TAWARMALANI [24] can also solve solving

constraint satisfaction problems. Initiated by the development of interval analysis on directed acyclic graphs by SCHICHL & NEUMAIER [26], COCONUT Environment [9, 25] has been developed as a global optimization software platform.

Typically, the solvers quoted require finite and not too large two-sided bound constraints, in order that the interval techniques used are efficient. Formally, unbounded problems are often (e.g., by BARON) by adding artificial bound constraints, with the resulting danger of excluding feasible points. Many solvers use unverified methods and return unverified results. The reason is that verifying the results or error control is often considered as an unnecessary extra effort. However there are a number of cases where serious safety problems can arise from unverified results. This motivated future research in robotics (e.g., MERLET [19]) and more general in safe computation techniques (JANSSON [12], KEIL [14], LEBBAH et al. [17]). Uncertainties in the input data are even more often ignored. In general the modeling languages (e.g., AMPL by FOURER et al. [10] or GAMS by BROOKE et al. [2]) do not support an exact treatment of rational or interval constraint coefficients.

A public version of GLOPTLAB is expected to be available end of July 2008.

2 Problem specification

We represent simple bounds as box constraint $x \in \mathbf{x}$. A box (or interval vector) is a Cartesian product

$$\mathbf{x} = [\underline{x}, \bar{x}] := (\mathbf{x}_1, \dots, \mathbf{x}_n)^T$$

of (bounded or unbounded) closed, real intervals $\mathbf{x}_i := [\underline{x}_i, \bar{x}_i]$. Thus the condition $x \in \mathbf{x}$ is equivalent to the collection of simple bounds

$$\underline{x}_i \leq x_i \leq \bar{x}_i \quad (i = 1, \dots, n),$$

or, with inequalities on vectors and matrices interpreted componentwise, to the two-sided vector inequality $\underline{x} \leq x \leq \bar{x}$. Apart from two-sided constraints, this includes with $\mathbf{x}_i = [a, a]$ variables x_i fixed at a particular value $x_i = a$, with $\mathbf{x}_i = [a, \infty]$ lower bounds $x_i \geq a$, with $\mathbf{x}_i = [-\infty, a]$ upper bounds $x_i \leq a$, and with $\mathbf{x}_i = [-\infty, \infty]$ free variables.

We also consider a quadratic expression $p(x)$ in $x = (x_1, \dots, x_n)^T$ such that the evaluation at any $x \in \mathbf{x}$ is a real number. If

$$p(x) \in \mathbf{p}(x) \text{ holds for all } x \in \mathbf{x}$$

then any mapping $p : \mathbb{R}^n \rightarrow \mathbb{R}$ satisfying

$$p(x) \in \mathbf{p}(\mathbf{x}), \quad \text{for all } x \in \mathbf{x}. \quad (2)$$

is called an interval enclosure of $p(x)$ in the box \mathbf{x} . There are a number of methods for defining $\mathbf{p}(\mathbf{x})$, for example interval evaluation or centered forms (for details, see, e.g., NEUMAIER [21]). If for all $y \in \mathbf{p}(\mathbf{x})$ an $x \in \mathbf{x}$ exists such that $p(x) = y$, then $\mathbf{p}(\mathbf{x})$ is the range. If this only holds for $y = \inf \mathbf{p}(\mathbf{x})$ and $y = \sup \mathbf{p}(\mathbf{x})$ then $\mathbf{p}(\mathbf{x})$ is the interval hull $\square\{p(x) \mid x \in \mathbf{x}\}$. To get rigorous results when using floating point arithmetic, one needs an implementation of interval arithmetic with outward rounding.

Another – and somewhat trickier – alternative is to compute the upper and the lower bound of the range separately, without the use of interval arithmetic, by using monotonicity properties of the operations. To get rigorous results when using floating point arithmetic, one needs here directed rounding. For an expression p , we denote by $\nabla\{p\}$ the result obtained when first the rounding mode is set to downward rounding, then p is evaluated, and by $\Delta\{p\}$ the result obtained when first the rounding mode is set to upward rounding, then p is evaluated. We assume that negating an expression is done without error; thus, e.g., $\Delta\{-(x-y)\} = -\Delta\{x-y\}$. Careful arrangement allows in many cases to replace downward rounded expressions by equivalent upward rounded expressions. For example, $\nabla\{x-y\} = \Delta\{-(y-x)\}$. If this is possible, one can achieve correct results using only upward rounding (thus saving rounding mode switches), while in interval arithmetic, the rounding mode is often switched at the cost of computation time. However, not all expressions can be bounded from below or above using directed rounding only; and detailed considerations are needed in each particular case.

The constraint satisfaction problems in GLOPTLAB consist of simple bounds, linear constraint, and quadratic constraints. We represent simple bounds as box constraint $x \in \mathbf{x}$. The linear and quadratic constraints are represented in a sparse matrix notation. The linear, quadratic, and bilinear monomials occurring in at least one of the constraint (but not the constant term) are collected into an n_q -dimensional column vector $q(x)$. There we choose

$$q(x) = (x_1, \dots, x_n, x_1^2, \dots, x_1 x_n, \dots, x_n x_1, \dots, x_n^2)^T$$

The coefficients of the i th constraint in the resulting monomial basis are collected in the i th row of a (generally sparse) matrix A , and any constant term (if present) is moved to the right hand side. Thus the linear and quadratic constraints take the form $A_i q(x) \in \mathbf{F}_i$ ($i = 1 \dots m$), where \mathbf{F}_i is a closed interval, and A_j denotes the j th row of A .

As in the case of simple bounds, this includes equality constraints and one-sided constraints by choosing for the corresponding \mathbf{F}_i degenerate or unbounded intervals. In compact vector notation, the constraints take the form $Aq(x) \in \mathbf{F}$.

While traditionally the coefficients in a constraint are taken to be exactly known, we allow them to vary in (narrow) intervals, to be able to rigorously account for uncertainties due to measurements of limited accuracy, conversion errors from an original representation to our normal form, and rounding errors when creating new constraints by relaxation techniques. Thus the coefficient matrix A is allowed to vary arbitrarily within some interval matrix \mathbf{A} . The $m \times n$ interval matrix \mathbf{A} with closed and bounded interval components $\mathbf{A}_{ik} = [\underline{A}_{ik}, \overline{A}_{ik}]$, is interpreted as the set of all $A \in \mathbb{R}^{m \times n}$ such that $\underline{A} \leq A \leq \overline{A}$, where \underline{A} and \overline{A} are the matrices containing the lower and upper bounds of the components of \mathbf{A} .

We therefore pose the quadratic constraint satisfaction problem in the form

$$Aq(x) \in \mathbf{F}, \quad x \in \mathbf{x}, \quad A \in \mathbf{A}. \quad (3)$$

If we introduce additional n_s slack variables x^s then quadratic constraint satisfaction problem in the equality form is given by

$$Aq(x) = 0, \quad x \in \mathbf{x}, \quad A \in \mathbf{A}. \quad (4)$$

where n_o is the number of the original variables x^o , $x = (x^o \ x^s)^T$, $n = n_o + n_s$ and

$$q(x) = (1, x_1, \dots, x_n, x_1^2, \dots, x_1 x_n, \dots, x_n x_1, \dots, x_n^2)^T \in \mathbb{R}^{n_q+1}.$$

Although currently not used, the GLOPTLAB format supports a more general representation which allows the user to define non-quadratic optimization problems. Quadratic constraint satisfaction problems are the special case where the objective function is constant and no univariate functions occur. Since GLOPTLAB is user extensible, this more general representation may already be useful to some users. The definition

$$x_j := \phi_k(x_i) \text{ with } j \in J, \ k \in \{1, \dots, n_u\}, \ i \in I \quad (5)$$

assigns the univariate function ϕ_k depending on the variable x_i to the variable x_j . For example, if $(i, j, k) = (3, 4, 2)$ and $\phi_2(z) = \sin(z + \pi/3)$ then $x_4 := \phi_2(x_3) = \sin(x_3 + \pi/3)$ is an additional univariate non-quadratic constraint definition. The term $x_J := \phi(x_I)$ in (6) represents all n_u univariate function definitions, whereby the index sets $I, J \subseteq \{1, \dots, n\}$. For future development purposes we also define an *objective function*, of which we can search for a local or a global minimum, inside the feasible domain.

The non-linear optimization problem

$$\begin{aligned} \min \quad & A_i q(x) \\ \text{s.t.} \quad & Aq(x) \in \mathbf{F} \text{ for some } A \in \mathbf{A}, \\ & x \in \mathbf{x}, \ x_J := \phi(x_I). \end{aligned} \quad (6)$$

with

$$x \in \mathbb{R}^n, \ q(x) \in \mathbb{R}^{n_q}, \ A \in \mathbb{R}^{m \times n_q}, \ i \leq n, \ |I| = |J| = n_u$$

and univariate $\phi : \mathbb{R}^{n_u} \rightarrow \mathbb{R}^{n_u}$. is called the *internal inequality representation* of GLOPTLAB.

Since the above representation is often obtained from converting non-quadratic problems, by introducing additional intermediate variables, we differentiate between the n_o original variables x^o , n_i intermediate variables x^i and the n_s slack variables x^s by writing $x = (x^o \ x^i \ x^s)^T$ with $n = n_o + n_i + n_s$. There are no slack variables in the internal inequality representation (6) but may occur in the *internal equality representation*

$$\begin{aligned} \min \quad & A_i q(x), \ i \in \{1, \dots, n\} \\ \text{s.t.} \quad & Aq(x) = 0 \text{ for some } A \in \mathbf{A}, \\ & x \in \mathbf{x}, \ x_J := \phi(x_I). \end{aligned} \quad (7)$$

with

$$x \in \mathbb{R}^n, \ q(x) \in \mathbb{R}^{n_q+1}, \ A \in \mathbb{R}^{m \times (n_q+1)}, \ |I| = |J| = n_u,$$

and univariate $\phi : \mathbb{R}^{n_u} \rightarrow \mathbb{R}^{n_u}$

The conversion from the AMPL format to the internal problem representation of GLOPTLAB is done by AMPL (FOURER et al. [10]) in connection with the COCONUT environment [9], while the parsing and conversion from a simplified AMPL format is done by the SMPL parser (MARKÓT [18]). Converting the GLOPTLAB problem representation (.DEF, .GLB files) to AMPL or SMPL formats is also possible. More information about the conversion possibilities can be found in Figure 2 (Subsection 4.5).

3 The implemented methods

There are a number of different rigorous methods developed and integrated in GLOPTLAB. In this chapter we give a brief description of the most important methods in this constantly expanding repertoire.

3.1 Problem simplification and scaling

This is usually the first step after reading a problem. In the problem simplification phase several preprocessing steps are done: We first identify and *remove bound constraints* from the general constraints, and store their bounds in the box \mathbf{x} . *Unbounded constraints* – where the corresponding interval F_i in 6 is unbounded – *are removed*. Possibly *redundant constraints are identified* and can be optionally removed. The *problem can transformed into the equality representation* (4) by introducing additional slack variables. *Additional structural characteristics* like sparsity pattern are also derived.

The polynomial scaling problem consists in *finding a constraint scaling vector* $r \in \mathbb{R}_+^m$ and a *variable scaling vector* $c \in \mathbb{R}_+^n$ such that the scaled problem

$$x \in \mathbf{x}, \quad A^s q(x) \in \mathbf{F}^s \quad \text{with} \quad A_{ik}^s := r_i |A_{ik}| q(c)_k, \quad \mathbf{F}_i^s := r_i \mathbf{F}_i \quad (8)$$

is well-scaled in an appropriate sense. Which properties constitute a well-scaled problem is a somewhat ill-defined matter, because it highly depends on the applications and is not easily quantifiable. Intuitively, a scaling algorithm should somehow decrease large variations between appropriately weighted sums of logarithms of the coefficients of the matrix A ; the weights should reflect the expected size of the values of the monomials. In GLOPTLAB we can choose between the HOMPACk (WATSON & TERRY [30]) algorithm, Morgan's algorithm (MORGAN [20, Chapter 5]), and the methods LP and SCALEIT described in DOMES & NEUMAIER [7]. The computed scaling vectors are then stored and later used by different methods.

3.2 Constraint propagation

Filtering techniques which tighten a box are called *constraint propagation* if they are based on single constraints only. *Forward propagation* uses the bound constraints to improve the bounds on the general constraints; *backward propagation* uses the bounds on the general constraints to improve the bounds on the variables.

Since (3) only consists of quadratic expression, we can write each constraint without loss of generality in the form

$$\sum_k (a_k x_k^2 + b_k x_k) + \sum_{\substack{j,k \\ j > k}} b_{jk} x_j x_k \geq c, \quad x \in \mathbf{x}, \quad (9)$$

where the $a_k x_k^2$ are the quadratic, the $b_k x_k$ the linear and the $b_{jk} x_j x_k$ the bilinear terms.

We first separate the constraint by approximating or bounding the bilinear terms, then we apply the forward propagation step: we compute the enclosure \mathbf{p}_k of each univariate quadratic term $p_k(x_k) := a_k x_k^2 + b_k x_k$ where the uncertainties \mathbf{a}_k and \mathbf{b}_k of the constraint coefficients are also taken into account. Then we use the \mathbf{p}_k to verify that the constraint is feasible, to get a new bound on each $p_k(x_k)$ and to find a new lower bound for the constraint. If the constraint has been found feasible, can apply the backward propagation step and find the set of all x_k with $a_k x_k^2 + b_k x_k \in \mathbf{p}_k$. Finally, if we cut the bounds found with the original bound on the variables, we may obtain tighter bound constraints.

The method is cheap, rigorous, and does not require interval arithmetic since only directed rounding is used. It is often used in other methods for verifying approximate solutions. In general, if used as a stand alone technique more than one step of constraint propagation is done successively, until no further significant reduction takes place.

A more detailed description of our constraint propagation can be found in DOMES & NEUMAIER [4].

3.3 Linear relaxations

Linear constraints of the form

$$Ex \geq b, \quad x \in \mathbf{x}. \quad (10)$$

may be obtained by relaxing the constraints of (3). Every feasible point of the constraint satisfaction problem (3) satisfies (10) iff for all $x \in \mathbf{x}$ and $A \in \mathbf{A}$ the inequalities

$$Aq(x) + \underline{b} - \underline{F} \leq Ex \leq Aq(x) + \bar{b} - \bar{F}$$

hold. In this case linear system (10) is called a linear relaxation of (3) (proof can be found in DOMES & NEUMAIER [6]). The relaxation (10) is found by computing interval enclosures (2), by using constraint propagation from Subsection 3.2 and by finding linear under and overestimators: the function $u(x)$ is called a linear underestimator of $p(x)$ in the box \mathbf{x} , if for all $x \in \mathbf{x}$, $u(x) \leq p(x)$ holds. Similarly, the function $v(x)$ is called a linear overestimator of $p(x)$ in the box \mathbf{x} , if for all $x \in \mathbf{x}$, $p(x) \leq v(x)$ holds.

After linearizing the constraints we apply different methods to improve the bound constraints $x \in \mathbf{x}$. These methods are explained in detail in DOMES & NEUMAIER [6].

If some bounds in \mathbf{x} are infinite and the feasible domain is bounded, the *linear bounding* method is used to get finite bound constraints. This requires the approximate solution of a *single* linear program and a single constraint propagation step to generate new finite and rigorous bounds. The only purpose of this method is to bound the feasible domain, and leads to no further improvements if applied more than one time.

In *linear contraction* we first compute new bounds on the constraints, cut them with the original ones. Afterward a modified Gauss-Jordan elimination is used to precondition the system, then either a direct interval evaluation or a single constraint propagation step is used to get new bounds on some or all of the variables.

Among the methods based on linear relaxations, the *LP contraction* is the method which requires the most computational time since in each step we solve more than one linear

programs. We find the d most promising directions (usually $d = 3$) and minimize the upper and lower bound from this directions. This requires the approximate solution of $2d$ linear programs, of which the dual solutions are used to generate new constraints. Propagating the new constraints may improve the bound of the selected variables.

3.4 Strictly convex enclosures

A quadratic inequality constraint with a strictly convex Hessian matrix defines an ellipsoid whose interval hull is easy to compute analytically. However, to cope efficiently with rounding errors is nontrivial.

For a real, symmetric matrix A a nonsingular triangular matrix R we compute a *directed Cholesky factor* such that the error matrix $A - R^T R$ of the factorization is tiny and guaranteed to be positive semidefinite. Clearly, this implies that A is positive definite; conversely (in the absence of overflow), any sufficiently positive definite symmetric matrix has such a factorization with R representable in floating point arithmetic. In DOMES & NEUMAIER [5] we find such a representation which makes the error small as feasible and works even for nearly singular matrices.

We use the directed Cholesky factorization to transform a strictly convex quadratic constraint of the constraint satisfaction problem (3) into an an ellipsoid defined by a Euclidean norm constraint

$$\|Rx\|_2^2 + 2a^T x \leq \alpha. \quad (11)$$

There is also need of scaling when factoring ill-conditioned matrices before applying the factorization. Therefore the scaling computed in the simplification is used before the directed Cholesky factorization is applied.

We derive the optimal box enclosure of this ellipsoid; we find constants β , γ , Δ and a vector $d > 0$ such that if $\Delta \geq 0$ then (11) implies

$$\|R(x - \tilde{x})\|_2 \leq \delta := \gamma + \sqrt{\Delta}, \quad |x - \tilde{x}|_2 \leq \frac{\delta}{\beta} d. \quad (12)$$

If $\Delta < 0$ then (11) has no solution $x \in \mathbb{R}$. For suitable chosen \tilde{x} the bounds in (12) are optimal (for details and proof see DOMES & NEUMAIER [5, Section 6,7]).

By the second inequality of (12) we get rigorous bounds

$$\mathbf{u} := \left[(\delta/\underline{\beta})d - \tilde{x}, (\delta/\overline{\beta})d + \tilde{x} \right]$$

on the variables x . If we do this for each strictly convex quadratic constraint of the constraint satisfaction problem (3) and cut the resulting bounds with the original ones we may get tighter bound constraints.

By this method we get rigorous bounds on all n variables, obtainable with $O(n^3)$ operations. This should be used only once per problem since successive application gives no further improvement of the bounds.

3.5 Conic methods

Conic methods approximate the general constraints by hyperplanes, balls or hyperellipsoids, using semidefinite or conic programming in order to find sharp bounds on the feasible set of a quadratic constraint satisfaction problem. The conic methods use the internal equality form (4) and are based on the following proposition; improved by the techniques of SCHICHL & NEUMAIER [27]:

3.1 Proposition. *If G is positive semidefinite and $Z \leq 0$, than for any $x \in \mathbf{x}$ with $Eq(x) = 0$, we have*

$$0 \leq \begin{pmatrix} 1 \\ x \end{pmatrix}^T G \begin{pmatrix} 1 \\ x \end{pmatrix} - \begin{pmatrix} 1 \\ \bar{x} - x \\ x - \underline{x} \end{pmatrix}^T Z \begin{pmatrix} 1 \\ \bar{x} - x \\ x - \underline{x} \end{pmatrix} - z^T Aq(x). \quad (13)$$

Proof. Since by (4) the equality $Aq(x) = 0$ holds for all $x \in \mathbf{x}$ and by the definition of positive definiteness all terms on the right hand side of (13) are greater or equal to zero. \square

Now if G is positive semidefinite and $Z \leq 0$ the equation

$$\begin{pmatrix} 1 \\ x \end{pmatrix}^T G \begin{pmatrix} 1 \\ x \end{pmatrix} \leq \begin{pmatrix} 1 \\ \bar{x} - x \\ x - \underline{x} \end{pmatrix}^T Z \begin{pmatrix} 1 \\ \bar{x} - x \\ x - \underline{x} \end{pmatrix} + z^T Eq(x) + p(x)$$

implies that $0 \leq p(x)$. To find the positive semidefinite matrix G , the matrix Z , the vector z and free parameters in $p(x)$ we solve the conic program

$$\begin{aligned} \min \quad & c^T y \\ \text{s.t.} \quad & y_i \geq 0 \\ & \|r(y)\| \leq y_k \\ & \frac{1}{2} \|s(y)\|^2 \leq y_j y_k \\ & G \text{ symmetric and positive semidefinite.} \end{aligned} \quad (14)$$

with suitably chosen objective, non-negativity constraints $y_i \geq 0$, norm constraints $\|r(y)\| \leq y_k$, rotated conic constraints $\frac{1}{2} \|s(y)\|^2 \leq y_j y_k$ and the semidefiniteness constraint for the matrix G . Choosing one of the quadratic expressions

- $p(x) = \pm x_i + \zeta$ and minimizing ζ ,
- $p(x) = -\sum_{i=1}^n x_i^2 + \zeta$ and minimizing ζ ,
- $p(x) = -1$ and minimizing 0,
- $p(x) = -\|\omega \circ x\|^2 + 2\xi^T(\omega \circ x) + \delta$ with $\|\xi\| \leq \zeta$ and minimizing $\zeta + \delta$,

results in interesting enclosures of the feasible domain. Since the conic program (14) is solved by an approximate solver we get the approximate solutions \hat{G} , \hat{Z} and \hat{z} , and we need to verify the results by computing

$$\hat{p}(x) := \begin{pmatrix} 1 \\ x \end{pmatrix}^T \hat{G} \begin{pmatrix} 1 \\ x \end{pmatrix} - \begin{pmatrix} 1 \\ \bar{x} - x \\ x - \underline{x} \end{pmatrix}^T \hat{Z} \begin{pmatrix} 1 \\ \bar{x} - x \\ x - \underline{x} \end{pmatrix} - \hat{z}^T E q(x),$$

using interval arithmetic. Since $\hat{p}(x)$ is a rigorous enclosure of the the feasible domain and a quadratic expression with tiny interval coefficients, we can use constraint propagation on it and may obtain tighter bound constraints.

Since the solution of the conic programs is rather costly, the maximal dimension of problems solved by this methods is limited, and the number of iterative steps should be rather low. For details on the conic methods used in GLOPTLAB see DOMES & NEUMAIER [5].

3.6 Branch and bound

Using branch and bound on the constraint satisfaction problem (3) means that we partition the bound constraints \mathbf{x} into s smaller subboxes, x^k ($k = 1, \dots, s$) such that $\mathbf{x} = \mathbf{x}^1 \cup \dots \cup \mathbf{x}^s$ and use rigorous methods $\Gamma_i(\mathbf{x}^k, \mathbf{F})$ on each \mathbf{x}^k separately. The methods applied to a subbox may reduce its width and even eliminate it if it contains no feasible points. There are different branching strategies; but in general they can be classified by the amount of memory they need. Recursive splitting selects a variable and splits the original box in this variable into two new boxes. To the first one the rigorous methods are applied while the second one is stored on a stack. If the first box is reduced but not eliminated by the methods, it is split again, whereby the second part is again stored on the stack. This is done until the actual box is empty, a minimal with of the current box is reached, or the maximal number of elements on allowed the stack is exceeded. Then the last box is popped from the stack, reduced and split by using the same procedure. Since the maximal memory needed by this splitting strategy is low it is the branching method which is currently implemented in GLOPTLAB. The variable x_i in which the a box is split is either the one where \mathbf{x}_i has maximal width or the one where the constraints have maximal range $\sum_k \text{wid}(A_{k:q}(\mathbf{x}_i))$. Different variables selection methods and splitting strategies may be included in the future.

Recursive splitting results in a finite cover of the feasible domain by nonempty subboxes of a given maximum size. We can either return all boxes found or create the *interval hull* of them. Connected components of the union of the subboxes define *clusters*, which can be separately bounded by their interval hull. Since returning all boxes found often result in an unnecessary large number of output and computing a single interval hull for distinct connected components is a crude approximation, therefore in most cases computing interval hull of clusters is the method of selection.

4 Integration of methods

The methods presented in the previous section are integrated as part of the GLOPTLAB environment.

4.1 Solution strategies

A *solution strategy* is a list of methods used to solve a problem. There are different additional features like loops and pauses to ensure that the user can generate many different strategies. Here we only give a sample solution strategy:

```
01: Read Problem - maxvars=15, cdeb, cpvt, box, prt
02: Simplify - scal=LP, obj=use refpoint, prt
03: Linear (bound) - eqsolver=fast, lsolver=SDPT3, prt
04: Linear (solve) - eqsolver=standard, lsolver=SDPT3, prt
05: Ehull - scal, prt
06: Feasibility (find point) - mindist=1e-005, delta=1e-005, ...
07: Begin While - mingain=0.2, maxiter=21, prt, small=0.5
08:     Begin While - mingain=0.2, maxiter=3, prt, small=0.5
09:         Conic (ellipsoid) - Zset=auto, csolver=SDPT3, ...
10:         Propagate (separable) - full, prt
11:         Conic (bound) - Zset=auto, csolver=SDPT3, ...
12:         Propagate (separable) - full, prt
13:         Begin While - mingain=0.2, maxiter=5, prt, small=0.5
14:             Propagate (separable) - full, prt
15:             Linear (contract) - eqsolver=standard, ...
16:         End While
17:         Feasibility (find point) - mindist=1e-005, ...
18:     End While
19:     Feasibility (find point) - mindist=1e-005, delta=1e-005, ...
20:     Begin Split (maxvariation) - mindist=0.0001, ...
21:         Feasibility (find point) - mindist=1e-005, ...
22:         Begin While - mingain=0.2, maxiter=5, prt, small=0.5
23:             Linear (contract) - eqsolver=standard, ...
24:             Propagate (separable) - full, prt
25:         End While
26:     End Split
27:     Merge (cluster) - drop, prt
28: End While
29: Pause
30: End
```

As one can see, each method can have several input constants. These constants can vary in different ranges, and their value has to be set when building a strategy. For example for the while loop which starts with BEGIN WHILE and ends with END WHILE the minimal gain

percentage MINGAIN, the maximum number of iteration MAXITER and the width of a small box SMALL has to be decided. The special parameters PRT and DEB can be set for every method and they determine the level of the text output generated by them.

4.2 Graphical user interface

Although the strategies can be modified manually, it is better to use the *graphical user interface* of GLOPTLAB to edit them (see Figure 1). Building a strategy in the graphical user interface is done by inserting, editing or removing tasks (marked 1 in Figure 1). There is also syntax check included, which prevents the creation of incorrect strategies. In the graphical user interface not only the strategies can be edited but a single problem or a *problem list* (marked 2 in Figure 1) can be solved by executing a strategy using the EXECUTE button. The text output of the solution procedure can be found in the text output window (marked 3 in Figure 1), while the graphical output for problems in two variables in the graphical output window (marked 4 in Figure 1). Important informations of the currently selected problem (name, number of variables and constraints etc.) can be viewed in the right lower part (marked 5 in Figure 1). Creating new problems or converting existing ones in the GLOPTLAB format is also possible with the conversion tools and the internal GLOPTLAB editor. They can be accessed from the panel marked with 6 in Figure 1. In the central long panel (marked 7 in Figure 1) the parameters for the graphical output, the statistic database, the automatically generated proofs, the profiler and the general configuration can be accessed and modified. The GLOPTLAB configuration consists of several global parameters like the path of the external solvers or the width of a box which is assumed as tiny, and all the default values of the parameters used in the different task. There can be different configurations files, and the parameters contained can be edited by the user.

4.3 Batch solution

Although GLOPTLAB can be completely controlled by using the graphical user interface, the latter is only an additional layer built on the GLOPTLAB core and not essential for using the software. Alternatively, it is possible to solve one or more problems with a selected strategy by using the UNIX GLOPTSOLVE or the MATLAB GLOPTSOLVE.M scripts.

GLOPTLAB can generate autosave files (.SAV), solution files in the GLOPTLAB format (.GSL) and .RES files as well. The latter is needed for the TESTENVIRONMET NEUMAIER et al. [22] which allows one to compare the results and the performance of GLOPTLAB with other solvers.

4.4 User defined methods

The different methods are integrated in GLOPTLAB in an uniform way such that the method repertoire can be easily extended. New functions can be easily written for each category presented in this section (constraint propagation, linear methods, conic methods, branch and bound) without an in depth knowledge of GLOPTLAB itself. If someone creates a

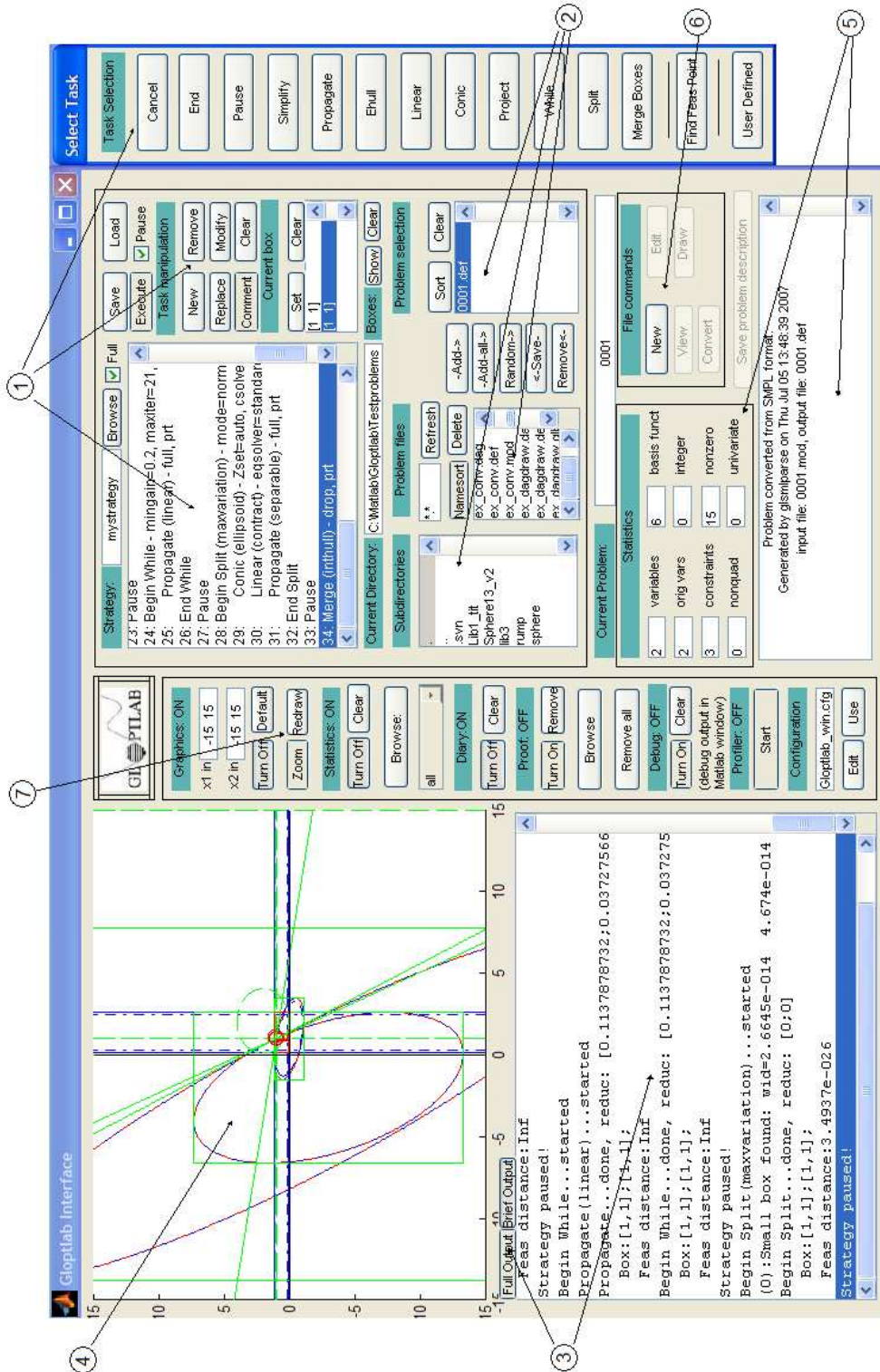


Figure 1: Gloptlab GUI; for explanations see text.

new method and places it into the GLOPTLAB/ SOURCE/USERDEFINED/ directory as an M-file called GLLINEAR.TEST.M it is automatically recognized by GLOPTLAB as a linear method and can be used similarly as the predefined ones. Samples for user defined methods can be found in the GLOPTLAB/SOURCE/USERDEFINED/ directory. Each method gets

the problem in either the internal inequality (6) or equality (7) representation (which are easily converted into each other) together with a number of additional control parameters (maximal depth, linear solver name, etc.) specific for each category. The results returned by the methods can consist of new bound constraints, found feasible points, linear relaxations, new general constraints, etc. The writer of the methods must ensure that all results are rigorous.

4.5 Structure diagram

An overview of the structure of GLOPTLAB is given in Figure 2.

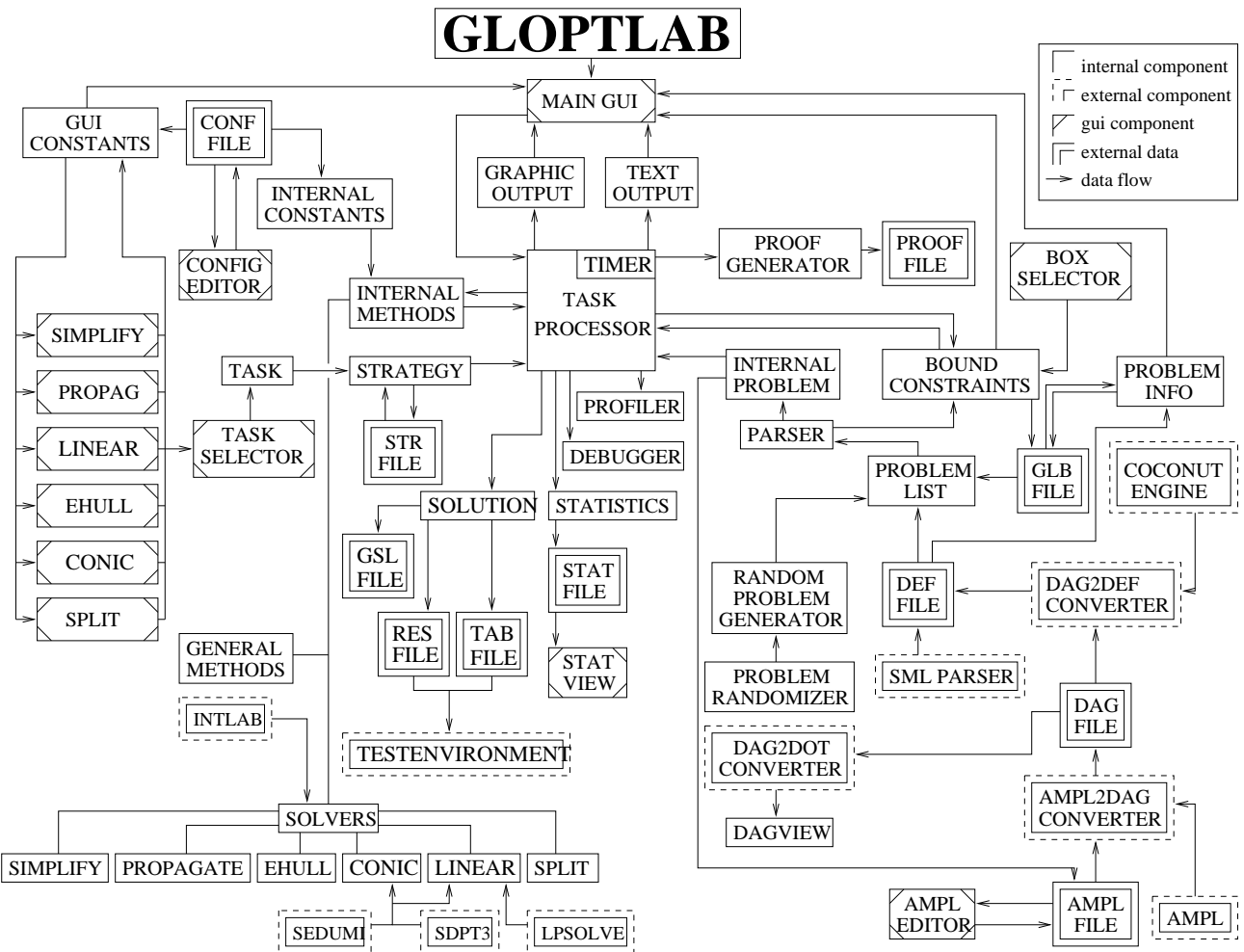


Figure 2: GlopLab structure

5 Example

The following two dimensional example demonstrates the advantage when using GLOPTLAB. The quadratic constraint satisfaction problem

$$\begin{aligned} -3x_1^2 + x_2x_1 + x_2^2 &= -2 \\ x_1^2 + 3x_1x_2 - 3x_2^2 &= 10 \end{aligned} \tag{15}$$

has no solution. The graph of (15) generated by the graphical user interface of GLOPTLAB can be found in Figure 3.

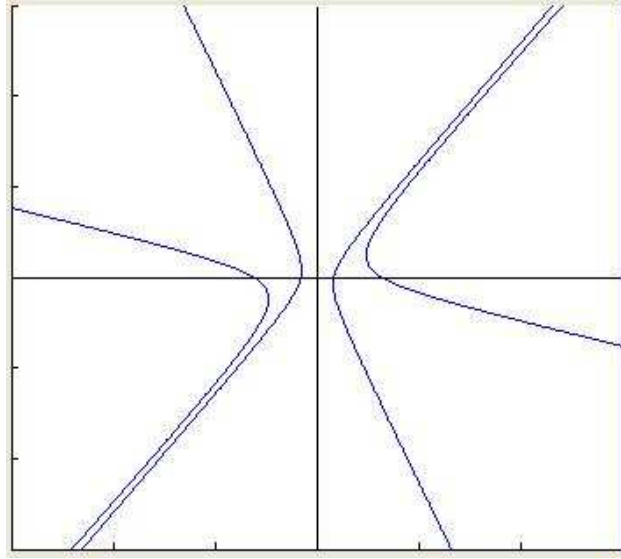


Figure 3: Two dimensional example consisting two equality constraints.

We tested some state of the art solvers by using the NEOS SERVER (see. CZYZYK et al. [3]) and obtained following results:

- The global solver BARON, found the problem infeasible after completing 41 iteration steps in approximately 0.3 seconds. However the message

User did not provide appropriate variable bounds.
We may not be able to guarantee globality.

is hidden in the log file returned by the solver. Thus, we tried to set artificial bounds, and when we used $-10^4 \leq x_1, x_2 \leq 10^4$ this message disappeared, showing that BARON cannot cope with unbounded bound constraints.

- The local solver KNITRO returned after 35 major iterations and 178 function evaluations the message:

EXIT: Convergence to an infeasible point.
Problem appears to be locally infeasible.
If problem is believed to be feasible, try multistart to search for feasible points.

- The rigorous global solver ICOS modified by adding the the artificially set bounds of $-10^8 \leq x_1, x_2 \leq 10^8$. This happened without additional warning. It found the modified problem infeasible after 145 splits. The execution time was 4.38 seconds.
- We used GLOPTLAB with the solution strategy in Subsection 4.1, and verified infeasibility in 0.860 seconds. GLOPTLAB did not set any artificial bounds on the variables, and needed no branching since the conic ellipsoid enclosure verified that the problem is infeasible.

References

- [1] M. Berkelaar, J. Dirks, K. Eikland, and P. Notebaert. LpSolve, 1991 - 1999. URL <http://lpsolve.sourceforge.net/5.5/>.
- [2] A. Brooke, D. Kendrick, and A. Meeraus. GAMS: A user's guide, 1992. URL citeseer.ist.psu.edu/brooke92gams.html.
- [3] J. Czyzyk, M. Mesnier, and J. Moré. The NEOS Server. *IEEE Journal on Computational Science and Engineering*, 5:68–75, 1998. URL <http://www-neos.mcs.anl.gov/>.
- [4] F. Domes and A. Neumaier. Constraint propagation on quadratic constraints. submitted, 2008. URL <http://www.mat.univie.ac.at/~dferi/publ/Propag.pdf>.
- [5] F. Domes and A. Neumaier. Directed Cholesky factorizations and applications. submitted, 2008. URL <http://www.mat.univie.ac.at/~dferi/publ/Cholesky.pdf>.
- [6] F. Domes and A. Neumaier. Linear methods for quadratic constraint satisfaction problems. in preparation, 2008. URL <http://www.mat.univie.ac.at/~dferi/publ/Linear.pdf>.
- [7] F. Domes and A. Neumaier. A scaling algorithm for polynomial constraint satisfaction problems. to appear, 2008. URL <http://www.mat.univie.ac.at/~dferi/publ/Scaling.pdf>.
- [8] F. Domes and A. Neumaier. Using conic programs to solve quadratic constraint satisfaction problems. in preparation, 2008. URL <http://www.mat.univie.ac.at/~dferi/publ/Conic.pdf>.
- [9] H. Schichl et al. The COCONUT Environment, 2000–2008. Software. URL <http://www.mat.univie.ac.at/coconut-environment>.
- [10] R. Fourer, D. M. Gay, and B. W. Kernighan. AMPL – a modeling language for mathematical programming, 2002. Software. URL <http://www.ampl.com/>.
- [11] P. Van Hentenryck, L. Michel, and Y. Deville. *Numerica. A Modeling Language for Global Optimization*. MIT Press, 1997.
- [12] C. Jansson. VSDP: A MATLAB software package for verified semidefinite programming. In *Conference paper of NOLTA 2006*, p. 327330, 2006. URL <http://www.ti3.tu-harburg.de/paper/jansson/Nolta06.pdf>.

- [13] N. Jussien and V. Barichard. The PaLM system: explanation-based constraint programming. In *Proceedings of TRICS: Techniques for Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, pp. 118–133, September 2000. URL <http://www.emn.fr/jussien/publications/jussien-WCP00.pdf>.
- [14] Christian Keil. Lurupa - rigorous error bounds in linear programming. In *Algebraic and Numerical Algorithms and Computer-assisted Proofs*, 2005.
- [15] C. T. Kelley. Iterative methods for optimization – Matlab codes, 1999. Software. URL http://www4.ncsu.edu/~ctk/matlab_darts.html.
- [16] Y. Lebbah. iCOs – Interval COstraints Solver, 2003. URL <http://ylebbah.googlepages.com/icos>.
- [17] Yahia Lebbah, Claude Michel, Michel Rueher, David Daney, and Jean-Pierre Merlet. Efficient and safe global constraints for handling numerical constraint systems. *SIAM Journal on Numerical Analysis*, 42(5):2076–2097, 2005. URL <http://ylebbah.googlepages.com/research>.
- [18] M. Cs. Markót. SMPL - A Simplified Modeling Language for Mathematical Programming. Technical report, University of Vienna, 2008. URL <http://www.mat.univie.ac.at/~dferi/Gloptlab/index.html>.
- [19] J-P. Merlet. Solving the forward kinematics of a Gough-type parallel manipulator with interval analysis. *Int. J. of Robotics Research*, 23(3):221–236, 2004. URL citeseer.ist.psu.edu/merlet04solving.html.
- [20] A. Morgan. *Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems*. Prentice-Hall, 1987.
- [21] A. Neumaier. *Interval Methods for Systems of Equations*, vol. 37 of *Encyclopedia of Mathematics and its Applications*. Cambridge Univ. Press, Cambridge, 1990.
- [22] A. Neumaier, O. Shcherbina, W. Huyer, and T. Vinko. A comparison of complete global optimization solvers. *Math. Programming B*, 103:335–356, 2005.
- [23] S. M. Rump. INTLAB – INTerval LABoratory, 1998 - 2008. URL <http://www.ti3.tu-harburg.de/~rump/intlab/>.
- [24] N. V. Sahinidis and M. Tawarmalani. *BARON 7.2.5: Global Optimization of Mixed-Integer Nonlinear Programs*, User’s Manual, 2005. URL <http://www.gams.com/dd/docs/solvers/baron.pdf>.
- [25] H. Schichl. Mathematical modeling and global optimization, habilitation thesis, 2003. to appear. URL <http://www.mat.univie.ac.at/~herman/papers/habil.pdf>.
- [26] H. Schichl and A. Neumaier. Interval Analysis on Directed Acyclic Graphs for Global Optimization. *Journal of Global Optimization*, 33(4):541–562, 2005.
- [27] H. Schichl and A. Neumaier. Transposition theorems and qualification-free optimality conditions. *Siam J. Optimization*, 17:1035–1055, 2006. URL <http://www.mat.univie.ac.at/~neum/ms/trans.pdf>.

- [28] Jos F. Sturm, O. Romanko, and I. Pólik. SeDuMi, 1997 - 2008. URL <http://sedumi.mcmaster.ca/>.
- [29] K.C. Toh, M.J. Todd, and R.H. Tutuncu. SDPT3 – a Matlab software package for semidefinite programming, 1999. URL <http://www.math.nus.edu.sg/~mattohkc/sdpt3.html>.
- [30] L. T. Watson and L. Terry. HOMPACK: a suite of codes for globally convergent homotopy algorithms, 1985. URL <http://deepblue.lib.umich.edu/dspace/bitstream/2027.42/8204/5/ban6930.0001.001.pdf>.