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 13, 2010

Abstract

GloptLab is an easy-to-use testing and development platform for solving quadratic constraint satisfaction problems, written in Matlab.

The algorithms implemented in GloptLab are used to reduce the search space: scaling, constraint propagation, linear relaxations, strictly convex enclosures, conic methods, and branch and bound. All these methods are rigorous, hence it is guaranteed that no feasible point is lost. Finding and verifying feasible points complement the reduction methods. From the method repertoire custom made strategies can be built, with a user-friendly graphical interface.

GloptLab was tested on a large test set of constraint satisfaction problems, and the results show the importance of compose a clever strategy.

1 Introduction

GloptLab is a *testing and development platform*, implemented in Matlab, for rigorously solving *quadratic constraint satisfaction problems*, i.e., for finding multivariate points satisfying a given list of quadratic equations and inequalities, in a way that ensures that no possible solution is lost during the solution process. GloptLab supports rigorous input, converting decimal numbers that are not exactly representable into narrow interval coefficients accounting for the conversion errors. Coefficients may also be specified directly as narrow intervals.

GloptLab can also solve other algebraic constraint satisfaction problems, using the `dag2gloptlab` converter of the Coconut Environment, which introduces intermediate variables to transform algebraic problems to quadratic ones.

The constraints are represented in GloptLab in the form

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

Here $F : \mathbb{R}^n \to \mathbb{R}^m$ 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 points; the problem is called *infeasible* if there are no feasible points.

Rigorous methods for solving the constraint satisfaction problem (1) combine branch-and-bound techniques with methods to reduce the sets $\mathbf{x}$ and $\mathbf{F}$ to narrower sets $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{F}}$) with 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 process. All methods implemented in GloptLab are rigorous since they meet the above property. The methods in GloptLab return a certificate which can be used to verify the solution process and to automatically generate human-readable computer assisted proofs.

The reduction methods implemented in GloptLab were jointly developed with Arnold Neumaier, and are described in separate publications quoted in Section 3, where a short introduction to each of them is given. The input format is 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 [6]). We use different *linear relaxation* techniques to get finite bounds or decrease the size of the search space (see Domes & Neumaier [9]). *Strict convex enclosures* compute a nearly optimal interval hull of strictly convex constraints (see Domes & Neumaier [7]). *Conic methods* may lead to spectacular reductions of the search domain, but require a great deal of computation time (see Domes & Neumaier [10]). *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 [5]).

Some of the methods mentioned above make use of *external toolboxes*. Since the verification is often based on interval techniques, INTLAB (Rump [24]) is probably the most important toolbox, although some methods avoid using 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. [31]) or SDPT3 (Toh et al. [32]). The latter two can be also used to optimize 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 [16]. AMPL (Fourer et al. [11]), the Coconut Environment (Schichl [26]) and the SMPL parser (Markót [19]) are also used to convert to the internal representation of the problem.

There is a number of software packages for solving constraint satisfaction problems. The Numerica software by Hentenryck et al. [12] uses branch and prune methods and interval constraint programming to solve constraint satisfaction problems. The ICOS solver by Lebbah [17] 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 [14] uses explanation-based constraint programming, and propagates the constraints of the problem, learning from the failures of the solver. The prize-winning solver Baron by Sahinidis & Tawarmalani [25] can also solve constraint satisfaction problems. Initiated by the development of interval analysis on directed acyclic graphs by Schichl & Neumaier [28], the Coconut Environment [26, 27] has been developed as a global optimization software platform.

Typically, the solvers quoted require finite and not too large two-sided bound constraints to ensure the efficiency of the interval techniques. Formally, unbounded problems are often tightened (e.g., by Baron) by adding artificial bound constraints, with the resulting danger of excluding feasible points. Some of the best solvers (e.g., Baron) use unverified methods and return unverified results. The reason is that verifying the results or error control is often considered to be an unnecessary extra effort. However there is a number of cases where serious safety problems can arise from unverified results. This has motivated research in robotics (e.g., Merlet [20]) and more generally in safe computation techniques (Jansson [13], Keil [15], Lebbah et al. [18]). Uncertainties in the input data are even more often ignored. In general the modeling languages (e.g., AMPL [11] or GAMS by Brooke et al. [2]) do not support an exact treatment of rational or interval constraint

coefficients.

Section 5 contains the novel contribution as we discuss the integration of the above methods in the GloptLab environment, features ranging from the building of user defined solution strategies with the graphical user interface to the possibilities of extending the method repertoire. An example can be found in Section 6, while in Section 7 we present some test results of GloptLab analyzed by the Test Environment (see Neumaier et al. [23] and Domes et al. [4] for the current version). In the final section we summarize the most important features of GloptLab, give some perspectives and talk about future work.
More information is available at the GloptLab homepage:

`http://www.mat.univie.ac.at/~dferi/gloptlab.html`

The public version of GloptLab is available at:

`http://www.mat.univie.ac.at/~dferi/gloptlab/download.html`

2 Problem specification

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

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

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

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

or, with inequalities on vectors and matrices interpreted component-wise to the two-sided vector inequality $\underline{x} \leq x \leq \overline{x}$. 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 $x = (x_1, \ldots, x_n)^T$ 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 $p : \mathbb{IR}^n \to \mathbb{IR}$ satisfying

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

is called an interval enclosure of $p(x)$ in the box $\mathbf{x}$. There is a number of methods for defining $p(\mathbf{x})$, for example interval evaluation or centered forms (for details, see, e.g., Neumaier [22]). If for all $y \in p(\mathbf{x})$ an $x \in \mathbf{x}$ exists such that $p(x) = y$, then $p(\mathbf{x})$ is the range. If this only holds for $y = \inf p(\mathbf{x})$ and $y = \sup p(\mathbf{x})$, then $p(\mathbf{x})$ 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 directed rounding here. Let p be an expression, $\nabla\{p\}$ denotes the result when first the rounding mode is set to downward rounding, then p is evaluated. Similarly, $\Delta\{p\}$ denotes the result 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 replacement of 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). In contrast, using an interval arithmetic package, the user does not have control over switching the rounding mode, and frequent switches can significantly increase the 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 constraints, and quadratic constraints. We represent simple bounds as box constraints $x \in \mathbf{x}$. 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 constraints (but not the constant term) are collected into an $n^2 + n =: n_q$ dimensional column vector $q(x)$. There we choose

$$q(x) = (x_1, \ldots, x_n,\ x_1^2, \ldots, x_1x_n,\ \ \ldots\ x_nx_1, \ldots, x_n^2)^T$$

The coefficients of the ith constraint in the resulting monomial basis are collected in the ith 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 $A_{i:}q(x) \in \mathbf{F}_i$ $(i = 1 \ldots m)$, where $\mathbf{F}_i$ is a closed interval, and $A_{j:}$ denotes the jth 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 $m \times n_q$ 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 $A \in \mathbb{R}^{m \times n_q}$ such that $\underline{A} \le A \le \overline{A}$, 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

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

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

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

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

$$q(x) = (1, x_1, \ldots, x_n,\ x_1^2, \ldots, x_1x_n,\ \ \ldots\ x_nx_1, \ldots, x_n^2)^T \in \mathbb{R}^{n_q+1}.$$

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 user-defined univariate functions (see below) occur. Since GloptLab is user extensible, this more general representation may be useful to some developers. Let $I, J \subseteq \{1, \ldots, n\}$ be index sets then

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

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. 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

$$\begin{array}{ll} \min & A_{i:}q(x) \\ \text{s.t.} & Aq(x) \in \mathbf{F} \ \text{ for some } A \in \mathbf{A}, \\ & x \in \mathbf{x},\ x_J := \phi(x_I). \end{array} \tag{6}$$

with

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

and $\phi : \mathbb{R}^{n_u} \to \mathbb{R}^{n_u}$ 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 (e.g., variables used substituting univariate functions) and the n_s slack variables x^s by writing $x = (x^o \quad x^i \quad x^s)^T$ with $n = n_o + n_i + n_s$. There are no slack variables in the internal inequality representation (6) but slack variables may occur in the *internal equality representation*

$$\begin{array}{ll} \min & A_{i:}q(x) \in \mathbf{F}_{\text{obj}} \\ \text{s.t.} & Aq(x) = 0 \ \text{ for some } A \in \mathbf{A}, \\ & x \in \mathbf{x},\ x_J := \phi(x_I). \end{array} \tag{7}$$

with

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

and $\phi : \mathbb{R}^{n_u} \to \mathbb{R}^{n_u}$. The objective function is evaluated as an ordinary constraint, resulting in the bounds $\mathbf{F}_{\text{obj}}$.

The conversion from the AMPL format to the internal problem representation of GloptLab is done by AMPL in connection with the Coconut Environment [27], while the parsing and conversion from a simplified AMPL format is done by the SMPL parser [19]. 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 1 (Subsection 5.1).

3 The implemented methods

There is a number of different rigorous methods developed for and integrated in GloptLab. In this chapter we give a brief description of the most important methods of 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 be 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 of *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

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

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 [33]) algorithm, Morgan's algorithm (Morgan [21], Chapter 5), and the methods LP and ScaleIT described in Domes & Neumaier [5]. The computed scaling vectors are then stored and later used by different methods.

The task Simplify simplifies the problem, and computes the scaling vectors. The task parameters are:

Parameters of the task Simplify		
Parameter	Type	Description
Objective	selection	what to do with the objective? (remove, etc.)
Scaling	selection	select the used scaling method.
Linear solver	selection	select a possible linear solver for the scaling.

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 expressions, we can write each constraint without loss of generality in the form

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

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, we 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 [6].

The task Propagate completes one step of constraint propagation. The task parameters are:

Parameters of the task PROPAGATE		
Parameter	Type	Description
METHOD	selection	select separable or linear method.
FULL MODE	decision	select only the forward mode or the method.

3.3 Linear relaxations

Linear constraints of the form

$$Ex \geq b,\ x \in \mathbf{x}. \tag{9}$$

may be obtained by relaxing the constraints of (3). Every feasible point of the constraint satisfaction problem (3) satisfies (9) iff for all $x \in \mathbf{x}$ and $A \in \mathbf{A}$ the inequalities

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

hold. In this case the linear system (9) is called a linear relaxation of (3) (proof can be found in DOMES & NEUMAIER [9]). The relaxation (9) is found by computing interval enclosures, 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 [9].

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, then cut them with the original ones. Then 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 program. We find the d most promising directions (usually $d = 3$) and minimize the upper and lower bound from these 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 on the selected variables.

The task LINEAR applies a linear method to the problem. The task parameters are:

Parameters of the task LINEAR		
Parameter	Type	Description
METHOD	selection	select the method bound, contract or solve.
LINEAR SOLVER	selection	select an external linear solver.
EQU SOLVER	selection	select a method for solving equalities.

3.4 Strictly convex enclosures

A quadratic inequality constraint with a positive definite 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 we compute the *directed Cholesky factorization*; an approximate factorization $A \approx R^T R$ with nonsingular upper triangular R such that the error matrix $A - R^T R$ of the factorization is tiny and guaranteed 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 [7] we find such a representation which makes the error as small as possible 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 ellipsoid defined by a Euclidean norm constraint

$$||Rx||_2^2 + 2a^T x \le \alpha. \tag{10}$$

There is also need for 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 $\Delta \ge 0$ then (10) implies

$$\|R(x - \tilde{x})\|_2 \le \delta := \gamma + \sqrt{\Delta}, \quad |x - \tilde{x}|_2 \le \frac{\delta}{\beta} d. \tag{11}$$

If $\Delta < 0$ then (10) has no solution $x \in \mathbb{R}$. For suitably chosen $\tilde{x}$ the bounds in (11) are optimal (for details and proof see Section 6,7, of Domes & Neumaier [7]).

By the second inequality of (11) 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.

The task Ehull finds the ellipsoid hull of strict convex constraints. The task parameters are:

Parameters of the task Ehull		
Parameter	Type	Description
Scaling	decision	apply the scaling factors found by the task Simplify

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 [29]:

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*

$$0 \leq \begin{pmatrix} 1 \\ x \end{pmatrix}^T G \begin{pmatrix} 1 \\ x \end{pmatrix} - \begin{pmatrix} 1 \\ \overline{x} - x \\ x - \underline{x} \end{pmatrix}^T Z \begin{pmatrix} 1 \\ \overline{x} - x \\ x - \underline{x} \end{pmatrix} - z^T Aq(x). \tag{12}$$

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 (12) are greater or equal to zero. □

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

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

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

$$\begin{array}{ll} \min & c^T y \\ \text{s.t.} & y_i \geq 0, \\ & \|r(y)\| \leq y_k, \\ & \frac{1}{2}\|s(y)\|^2 \leq y_j y_k, \\ & G \text{ symmetric and positive semidefinite,} \end{array} \tag{13}$$

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 (13) 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

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

using interval arithmetic. Since $\hat{p}(x)$ is a rigorous enclosure of the feasible domain and a quadratic expression with narrow 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 method is limited, and the number of iterative steps should be rather low. For details on the conic methods used in GloptLab (see Domes & Neumaier [7]).

The task Conic applies a conic method to the problem. The task parameters are:

Parameters of the task Conic		
Parameter	Type	Description
Method	selection	bound, ellipsoid, feasibility or fixellipsoid.
Conic solver	selection	select an external conic solver.
Reduce in x ...	numeric	the reduction directions for the bound method.
Z setting	selection	select strategy for setting the matrix Z.
Weight	decision	decide weighting the slack variables or not.
Verify	decision	verify the results or compute approximately.
Intround	decision	transform all coefficients into rational numbers.

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 $(k = 1, \ldots, s)$ such that $\mathbf{x} = \mathbf{x}^1 \cup \ldots \cup \mathbf{x}^s$ 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. The rigorous methods are applied to the first one, 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 width 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. This is the *depth-first* split method. Since the maximal memory needed by the depth-first split is low this is the branching method which is currently implemented in GloptLab. Choosing the ith direction in which a box is split is a critical issues; in GloptLab either the one where $\mathbf{x}_i$ has maximal width or the one where the constraints have maximal range $\sum_k \mathrm{wid}(A_{k:}q(\mathbf{x}_i))$ can be chosen. Different variable 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 results in an unnecessary large amount 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.

The task Split divides the current box into sub-boxes and applies a solution strategy to each of them. The task parameters are:

Parameters of the task Split		
Parameter	Type	Description
Method	selection	currently only the depthfirst is available.
Dir chooser	selection	this criteria sets the split variable index.
Split in x	numeric	select the components of x which can be split.
Absolute small	numeric	a small box has a small param. less than this.
Margin	numeric	margin between absolute and relative small.
Relative small	numeric	a small box has rel. small par. less than this.
Small box crit	selection	the criteria classifying the small parameters.
Max depth	numeric	the maximum allowed depth of the split.

The task MERGE BOXES merges the boxes found by the task SPLIT. The task parameters are:

Parameters of the task MERGE BOXES		
Parameter	Type	Description
METHOD	selection	select the interval hull or the cluster method.
TOLERANCE	numeric	tolerance between neighboring clusters.
DROP BOXES	decision	drop or leave the boxes found by the split.

4 Finding and verifying feasible points

An important step toward the rigorous solving of optimization problems is to find and verify feasible points of a constraint satisfaction problem.

To find a feasible point of the constraint satisfaction problem (3) we construct a smooth *feasibility distance function* $d(x) : \mathbb{R}^n \to \mathbb{R}$, which we minimize in the box $\mathbf{x}$ by using a local solver. The current selection of local solvers integrated in GLOPTLAB consists of BFGS, GRADIENT PROJECTION (both of them are MATLAB versions by KELLEY [16]), LBFGS-B (ZHU et al. [34]) and FMINCON (contained in the optimization toolbox for MATLAB). Note that most of these solvers require the gradient of the function $d(x)$, but since the feasibility distance function d is smooth this can be computed explicitly.

If an approximately feasible point x_f has been found, we try to find a box $\mathbf{b} \subseteq \mathbf{x}$ around x_f such that the existence of a feasible point inside of $\mathbf{b}$ is guaranteed by a mathematical existence theorem. This is called the *verification of feasible points*. For details on finding and verifying feasible points used in GLOPTLAB (see DOMES & NEUMAIER [8]).

In Section 7 we make use of finding feasible points to test our solver on a large test set of constraint satisfaction problems. The task FIND FEAS POINT can find and can verify a feasible point inside the current box. The task parameters are:

Parameter	Type	Description
LOCAL SOLVER	selection	select: bfgs, gradproj, lbfgs-b or fmincon.
MAX ITERATIONS	numeric	maximum number of solver iterations.
VERIFICATION	decision	try to verify the found point or not.
DELTA	numeric	δ constant used in the verification process.
SOLVER	selection	linear solver used for verification.

5 Integration of methods

The development of GLOPTLAB started in 2005 by Prof. Arnold Neumaier and myself. In the beginning, we experimented with constraint propagation techniques in order to reduce the search space of quadratic problems. Since GLOPTLAB was primary designed as a testing and development platform, we used the interpreted language MATLAB because of its ease of use and its graphical capabilities. In order to aid the development we developed a graphical interface providing a visual representation of the constraints and the current bound constraints during the reduction process. This was very useful for the debugging and testing of our programs. Since we intended later to extend the program to solve non-quadratic constraint satisfaction and optimization problems, we developed a general internal format in an early stage, and made only minor changes to it later.

As we added new methods like the ellipsoid hull and branch and bound to the method repertoire, we needed to decide which of the methods should be used in order to find a fast and reliable solution procedure, and how often and in which order they should be applied. Instead of making a fixed choice, we decided to create a task processor, and a strategy builder. To have an easy way to create, save, load and modify strategies, we added this features to the graphical user interface. We designed the GUI as a layer which is clearly separated from the solver engine, so that a batch solution of the problems is also possible.

Whenever we found a problem where our existing methods seemed to perform poorly we added new functionality, tasks, and parameters in order to improve the performance. For example, we added linear relaxations and conic programs. These lead to the development of a user extensible method repertoire. In order to obtain useful answers in case the complete search could not finish in the given time limit, we developed a method of finding feasible points. To be competitive with other rigorous solvers we created a method for verifying feasible points close to a near-feasible approximation.

Now GloptLab has a rich selection of rigorous methods that we can use to build strategies and then apply it to solve quadratic constraints satisfaction problems.

We summarize the most interesting features of GloptLab:

- There is a well structured input format representing global optimization problems (already presented at the beginning of Section 3).
- At present only quadratic constraints are solved. The solution of non-quadratic, algebraic problems is possible by using the AMPL to GloptLab converter from the Coconut Environment, which automatically transforms algebraic terms to quadratic ones by introducing intermediate variables.
- The whole environment is implemented in a completely modular way, allowing easy portability of individual methods to other solvers and languages (see Subsection 5.1).
- Easy to use for prototyping and for development of new techniques in the context of other methods (Subsection 5.2).
- The strategy builder allows to test different strategies for different problem classes (Subsection 5.2).
- Interactive solution of a particular problem: it is possible to stop the execution of the strategy, remove and add new tasks to it and then resume the solution process. This approach can greatly reduce the solution time (Subsection 5.2).
- Contributors can add their own method with only minimal knowledge of the other parts of the software (see Subsection 5.3).
- The graphical user interface (Subsection 5.4) supports both the easy building of solution strategies and the visualization of the solution process.
- Using the batch mode of GloptLab, it is possible to run solution strategies in the Test Environment [4], or on processors without graphical support (Subsection 5.5).

In the following subsections we discuss the above items, however because of the extent of the topics we omit some details. More information can be found in the documentation files which are part of the GloptLab environment.

5.1 GloptLab structure diagram

An overview of the structure of GloptLab is given in Figure 1. This diagram emphasizes how the software is structured, and gives some overview over currently implemented features. These consist of the dependency of the methods of the external solvers, building strategies from different tasks, conversion possibilities from other input formats, measuring performance, or saving statistical information about all solved problems. The small table in the corner of Figure 1 shows which parts of GloptLab are internal, external, or GUI components. All external packages integrated into GloptLab are free. The software packages used are listed in the following table, and are packaged with the current GloptLab version. They can be downloaded and installed separately but in this case the corresponding path variables have to be set manually by editing the Gloptlab.cfg file or by using the editor of the graphical user interface.

Solver	required?	Function
Intlab	necessary	interval arithmetic
Coconut Environment	optional	converter of non-quadratic problems
Sedumi	optional	solver for conic and linear programs
SDPT3	optional	solver for conic and linear programs
lpsolve	optional	solver for linear programs

When Intlab and at least one package from the above selection which is capable to solve both conic and linear programs is installed, all current features of GloptLab can be used. Since the external solvers are connected to the GloptLab solver engine through an interface, adding new linear or conic solvers is not difficult. The new solvers are automatically recognized by the strategy builder opening new options in the task selection process.

5.2 Solution strategies

As shown in Figure 1, to solve a problem or a list of problems we need a strategy. A *solution strategy* or simply a *strategy* is a list of *tasks* used to solve a problem. A task could be the implementation of one of the methods described in Chapter 3, but there are other tasks like loops, conditions and breaks to extend the functionality and ensure the versatility of a strategy. Strategies are built comfortably by using the graphical strategy builder of the user interface (also see 1 in Figure 2), automatically ensuring a correct strategy syntax. New methods are automatically recognized by the strategy builder (for details see Subsection 5.3). The strategies can be applied to the problems either directly using the GUI, or they can be saved for later use and for the execution of batch solution jobs.

A simple solution strategy, where all tasks have automatically generated default parameters, looks like:

5.1 Strategy. (simple sample)

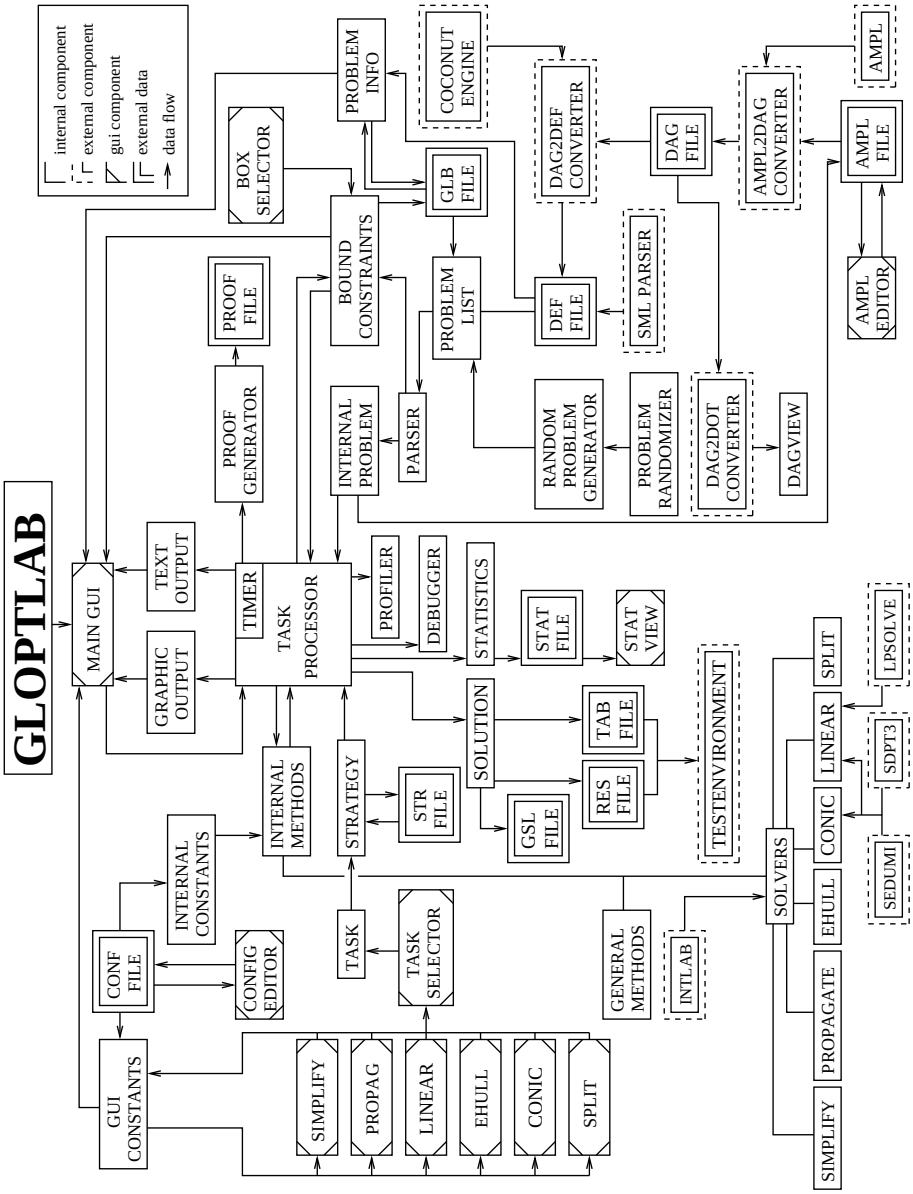

Figure 1: GloptLab structure

```
1: Read Problem
2: Simplify
3: Feasibility
4: Begin Condition
5:    Break
6: End Condition
7: Begin While
8:    Propagate
9:    Feasibility
10:    Begin Condition
11:       Break
12:    End Condition
13: End While
14: Begin Split
15:    Propagate
16:    Feasibility
17:    Begin Condition
18:       Break
19:    End Condition
20: End Split
21: Merge
22: Begin Postprocess
23:    Merge
24:    Feasibility
25: End Postprocess
26: Pause
27: Finish
```

while a more sophisticated one is:

5.2 Strategy. (complex sample)

```
1: Read Problem
2: Simplify
3: Ehull
4: Linear
5: Feasibility
6: Begin Condition
7:    Break
8: End Condition
9: Conic
10: Begin While
11:    Propagate
12:    Linear
13:    Feasibility
14:    Begin Condition
15:       Break
16:    End Condition
17: End While
18: Begin Split
19:    Propagate
20:    Linear
21:    Feasibility
22:    Begin Condition
23:       Break
24:    End Condition
25: End Split
26: Merge
27: Begin Split
28:    Propagate
29:    Linear
30:    Feasibility
31:    Begin Condition
32:       Break
33:    End Condition
34: End Split
35: Begin Postprocess
36:    Merge
37:    Feasibility
38: End Postprocess
39: Pause
40: Finish
```

Each method may have several input parameters (which are omitted in the above strategies), all of which have default values. For example, the while loop starts with BEGIN WHILE and ends with END WHILE and has, as parameters, the minimal gain percentage MINGAIN, the maximum number of iteration MAXITER and the width of a small box SMALL. The special parameters PRT and DEB can be set for every method and determine the level of the text and the debugging output. For more details on the various parameters see the tables at the end of the sections describing the different methods.
The input parameters depend on the implementation of the corresponding task, the definition of which must include their documentations. Therefore older strategies may become invalid if a task description has been changed. If strategies are built and updated using the GLOPTLAB graphical user interface, this is automatically recognized and the invalid lines are flagged for correction.

5.3 User defined methods

The different methods are integrated into GLOPTLAB in a uniform way such that the repertoire of methods can be extended easily. New functions can be written for each task, including those presented in Section 3 (constraint propagation, linear methods, conic methods, branch and bound), without knowledge of the GLOPTLAB code. For example if someone creates a new linear method it is automatically recognized by GLOPTLAB as such if it is placed into the `Gloptlab/Source/UserDefined/` directory as an `m`-file called `gllinear_*.m` and can be selected in the strategy generator as one of the options for linear methods. Samples for user defined methods in each class can be found in the `Gloptlab/Source/UserDefined/` directory.

Each method accesses the problem in either the inequality representation (6) or the equality representation (7) (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 may consist of new bound constraints, found feasible points, linear relaxations, new general constraints, etc. The writer of the methods must ensure for all rigorous tasks that the results are indeed rigorous.

5.4 Graphical user interface

The *graphical user interface* of GloptLab consists of areas for entering problems, for defining strategies, for displaying the solver progress and for configuring GloptLab (see Figure 2).

Building a strategy in the graphical user interface is done by inserting, editing or removing tasks using the strategy builder (marked 1 in Figure 2). In the graphical user interface not only the strategies can be edited but a single problem or a *problem list* (marked 2 in Figure 2) 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 2), while the graphical output for problems of every dimension is found in the graphical output window (marked 4 in Figure 2). Important information for the currently selected problem (name, number of variables and constraints etc.) can be viewed in the right lower part (marked 5 in Figure 2). Creating new problems or converting existing ones into the GloptLab format is also possible with the conversion tools and by using the internal GloptLab editor. They can be accessed from the panel marked with 6 in Figure 2. In the central long panel (marked 7 in Figure 2) the parameters for the graphical output, the statistical 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 configuration files, and the parameters contained in them can be edited by the user.

5.5 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 (`.gls`) and `.res` files as well. The latter is needed for the TEST ENVIRONMENT [4], which allows one to compare the results and the performance of GloptLab with other solvers.

5.6 Notes

The current version of GloptLab can be obtained at the official GloptLab homepage:
`http://www.mat.univie.ac.at/~dferi/gloptlab.html`.

6 Examples

The comparison of the performance to non-rigorous solvers which are implemented in a compiled language like C++ is difficult. We also emphasize that the strength of GloptLab lies in the easy use, extendability, the interactive solution and finding all solutions of a problem and not in outperforming non rigorous solvers by solving a whole test set of problems using a single default strategy. However we use the following two dimensional example to demonstrate one of the advantages of GloptLab: the quadratic constraint satisfaction problem

$$\begin{aligned} -3x_1^2 + x_2x_1 + x_2^2 &= -2 \\ x_1^2 + 3x_1x_2 - 3x_2^2 &= 10 \end{aligned} \tag{14}$$

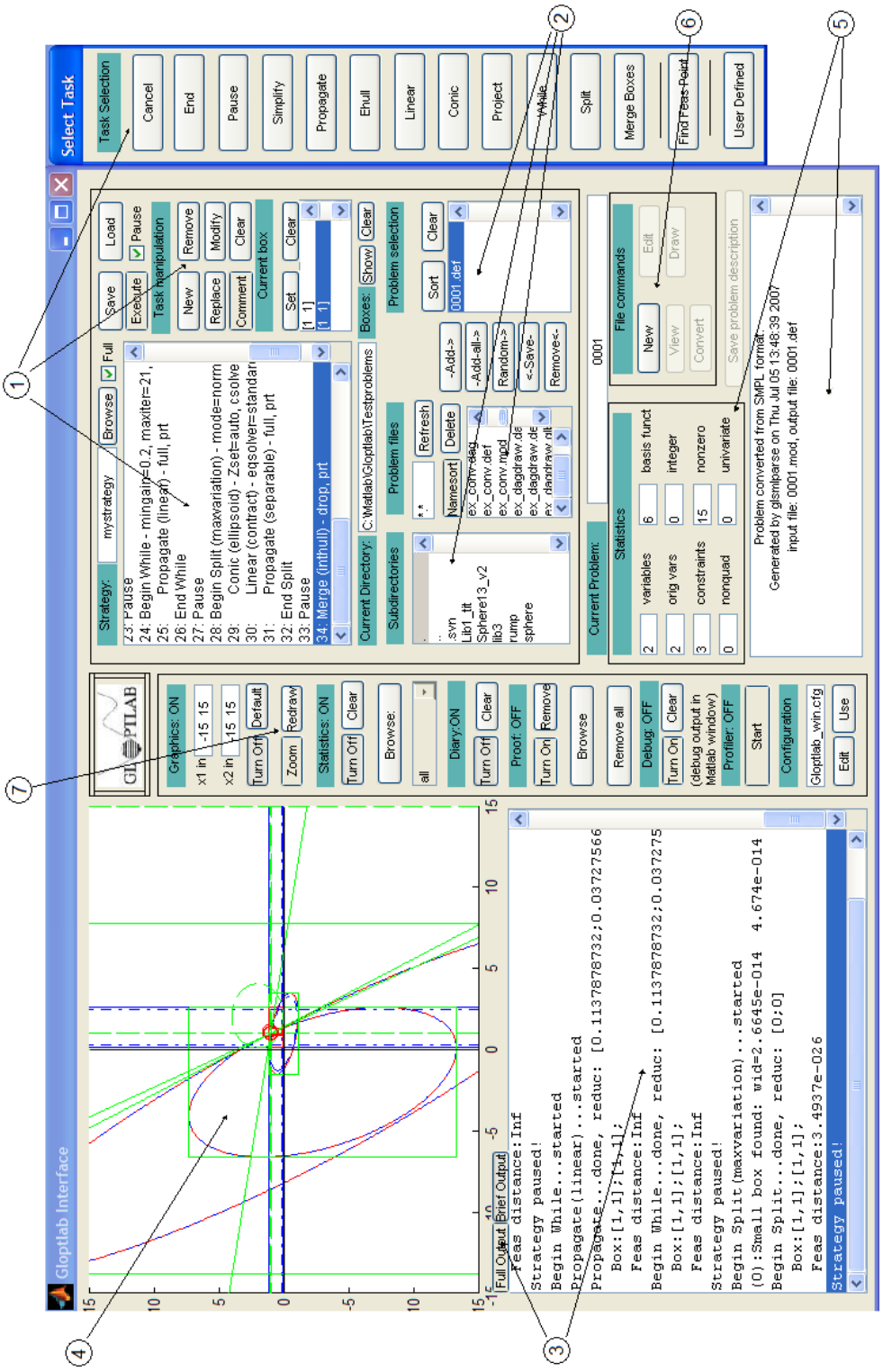

Figure 2: Gloptlab GUI; for explanations screenshot the text.

has no solution. The graph of (14) generated by the graphical user interface of GloptLab can be found in Figure 3.

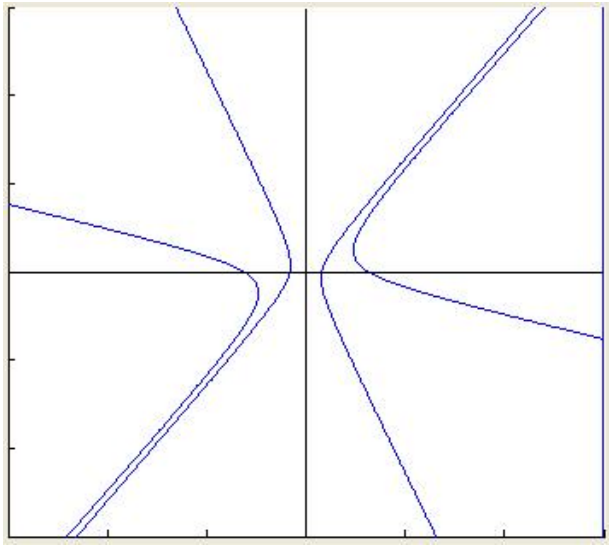

Figure 3: Two dimensional example consisting of two equality constraints.

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 the problem by adding 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 5.2, and verified infeasiblity 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.

7 Some test results

In this section we present some promising test results of GloptLab. The LaTeX tables containing the results are automatically generated by the Test Environment [4], which we used

for checking the solutions for correctness. We tested GloptLab on the library Lib3 of the Coconut Environment Testset (see Shcherbina et al. [30]), containing 308 constraint satisfaction problems. We solved the problems by using the sample strategies 5.1 and 5.2. These strategies are configured not to accept problems containing non-algebraic functions or more than 100 variables. The maximal time allowed for the solution of a single problem was 120 seconds.

Gloptlab on `Lib3` using the strategy (5.1)								
stat	all	wr	easy location			hard location		
			+G	-G	I	+G	-G	I
all	308	0	121	124	0	14	29	20
G	125	0	111	0	0	14	0	0
X	76	0	0	57	0	0	8	11
TU	95	0	8	59	0	0	21	7
U	12	0	2	8	0	0	0	2

Gloptlab summary statistics						
lib	all	accept	+G	G!	G?	I?
Lib3	308	232	135	125	0	0

Gloptlab on `Lib3` using the strategy (5.2)								
stat	all	wr	easy location			hard location		
			+G	-G	I	+G	-G	I
all	308	0	130	115	0	19	24	20
G	139	0	120	0	0	19	0	0
X	76	0	0	57	0	0	8	11
TU	85	0	10	52	0	0	16	7
U	8	0	0	6	0	0	0	2

Gloptlab summary statistics						
lib	all	accept	+G	G!	G?	I?
Lib3	308	232	149	139	0	0

Table legend: `stat` - solution status; `all` - the number of problems given to the solver; `accept` - problems accepted by the solver; `wr` - number of wrong claims (the sum of `G?` and `I?`); `easy location` - problems which have been classified as easy; `hard location` - problems which have been classified as hard. Status codes: `G` - the result claimed to be a global optimizer; `+G` - a global solution was found; `-G` - no global solution was found; `G!` - correctly claimed global solution; `G?` - wrongly claimed global solution, `I` - infeasible problem; `I?` - wrongly claimed infeasibility, `L` - local solution found; `TL` - timeout reached and a local solution was found; `U` - unresolved (no solution found or error message); `X` - model not accepted by the solver.

The tables show that from the 232 accepted problems we have found 135 correct solution (125 of them was claimed as correct) by using the first strategy and 149 correct solution (139 of them was claimed as correct) by using the second one. Within the same allowed solution time we solved 14 more problems with the second strategy as with the first one. This is approximately 10 percent of the accepted problems, and one third of them was a problem which is classified as a hard one. Indeed; 35 percent more of the hard problems was solved with using the second strategy. This significant difference was caused by the more sophisticated methods and the clever structure of the second strategy. The results show the importance of building a good strategy, as well as the process of testing different methods as the part of a strategy.

8 Conclusion and perspectives

Apart from the actual methods implemented in GloptLab, the major innovation is the ease with which it is possible to write strategies, to extend the method repertoire, and to test selected

methods on selected test sets as part of a strategy. In GloptLab users can build, test and optimize their own strategies, they can store and easily share them with other people. Moreover, users can implement and test their own methods without the need of extensive knowledge of the GloptLab implementation itself. The graphical representation of the solution process greatly simplifies the identification of the weak points of a method or a strategy.

Future work on GloptLab in our research group in Vienna includes porting the most useful methods and strategies to the Coconut Environment to increase the execution speed. Since the graphical layer is separated from the main solution engine, exporting parts of the implementation to other programs and the conversion to other programming languages should be easy. We also plan to add further methods to the method repertoire, and to search for optimal solution strategies. One of our goals is to develop an automatic strategy selection, which adapts the strategy to the problem solved. Numerous other features like generating human readable proofs and automatically building statistical databases, already available in a rudimentary form, will be fully developed in the future.

We also intend to extend GloptLab to rigorously solve non-quadratic optimization problems. As discussed in the problem specification, the internal problem representation of GloptLab allows non-quadratic, univariate functions. A converter from a general optimization problem in AMPL format to the internal format is already implemented.

External contributors are welcome to join the project by implementing and testing their own user-defined methods. User-defined methods submitted to us will be permanently added to the method repertoire of future versions of GloptLab if they are promising enough.

Acknowledgements

I would like to thank Arnold Neumaier, for his help and support and Martin Fuchs for his creative ideas and corrections. Numerous suggestions by the referees, which markedly improved the presentation of the paper, are gratefully acknowledged. This paper was made possible through the research grant FSP 506/003 of the University of Vienna.

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, M. Fuchs, and H. Schichl. The Optimization Test Environment. *Optimization Methods and Software*, 2010. submitted. URL `http://www.mat.univie.ac.at/~dferi/testenv.html`.

[5] F. Domes and A. Neumaier. A scaling algorithm for polynomial constraint satisfaction problems. *Journal of Global Optimization*, 43:327–345, 2008. URL `http://www.mat.univie.ac.at/~dferi/publ/Scaling.pdf`.

[6] F. Domes and A. Neumaier. Constraint propagation on quadratic constraints. *Constraints*, 2009. ISSN 1383–7133. URL `http://www.mat.univie.ac.at/~dferi/publ/Propag.pdf`.

[7] F. Domes and A. Neumaier. Rigorous enclosures of ellipsoids and directed Cholesky factorizations. submitted, 2009. URL http://www.mat.univie.ac.at/~dferi/publ/Cholesky.pdf.

[8] F. Domes and A. Neumaier. Finding and verifying feasible points of polynomial constraint satisfaction problems. in preparation, 2010. URL http://www.mat.univie.ac.at/~dferi/publ/.

[9] F. Domes and A. Neumaier. Rigorous filtering using linear relaxations. in preparation, 2010. URL http://www.mat.univie.ac.at/~dferi/publ/.

[10] F. Domes and A. Neumaier. Using conic programs to solve quadratic constraint satisfaction problems. in preparation, 2010. URL http://www.mat.univie.ac.at/~dferi/publ/.

[11] R. Fourer, D. M. Gay, and B. W. Kernighan. AMPL – a modeling language for mathematical programming, 2002. Software. URL http://www.ampl.com/.

[12] P. Van Hentenryck, L. Michel, and Y. Deville. *Numerica. A modeling language for global optimization*. MIT Press, 1997.

[13] 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.

[14] 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.

[15] C. Keil. Lurupa - rigorous error bounds in linear programming. In *Algebraic and Numerical Algorithms and Computer-assisted Proofs*, 2005.

[16] C. T. Kelley. Iterative methods for optimization – Matlab codes, 1999. Software. URL http://www4.ncsu.edu/~ctk/matlab_darts.html.

[17] Y. Lebbah. iCOs – Interval COnstraints Solver, 2003. URL http://ylebbah.googlepages.com/icos.

[18] Y. Lebbah, C. Michel, M. Rueher, D. Daney, and J-P. 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.

[19] M. C. 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.

[20] J-P. Merlet. Solving the forward kinematics of a Gough-type parallel manipulator with interval analysis. *International Journal of Robotics Research*, 23(3):221–236, 2004. URL citeseer.ist.psu.edu/merlet04solving.html.

[21] A. Morgan. *Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems*. Prentice-Hall, 1987.

[22] A. Neumaier. *Interval methods for systems of equations*, vol. 37 of *Encyclopedia of Mathematics and its Applications*. Cambridge Univ. Press, Cambridge, 1990.

[23] A. Neumaier, O. Shcherbina, W. Huyer, and T. Vinkó. A comparison of complete global optimization solvers. *Math. Programming B*, 103:335–356, 2005.

[24] S. M. Rump. INTLAB – INTerval LABoratory, 1998 - 2008. URL http://www.ti3.tu-harburg.de/~rump/intlab/.

[25] 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.

[26] H. Schichl. Mathematical modeling and global optimization, habilitation thesis, 2003. URL http://www.mat.univie.ac.at/~herman/papers/habil.pdf.

[27] H. Schichl, M. C. Markót, A. Neumaier, Xuan-Ha Vu, and C. Keil. The COCONUT Environment, 2000-2010. Software. URL http://www.mat.univie.ac.at/coconut-environment.

[28] H. Schichl and A. Neumaier. Interval Analysis on Directed Acyclic Graphs for Global Optimization. *Journal of Global Optimization*, 33(4):541–562, 2005.

[29] H. Schichl and A. Neumaier. Transposition theorems and qualification-free optimality conditions. *Siam Journal Optimization*, 17:1035–1055, 2006. URL http://www.mat.univie.ac.at/~neum/ms/trans.pdf.

[30] O. Shcherbina, A. Neumaier, D. Sam-Haroud, Xuan-Ha Vu, and Tuan-Viet Nguyen. Benchmarking global optimization and constraint satisfaction codes. In Ch. Bliek, Ch. Jermann, and A. Neumaier, editors, *Global Optimization and Constraint Satisfaction*, pp. 211–222. Springer, 2003. URL http://www.mat.univie.ac.at/~neum/ms/bench.pdf.

[31] J. F. Sturm, O. Romanko, and I. Pólik. SeDuMi, 1997 - 2008. URL http://sedumi.mcmaster.ca/.

[32] 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.

[33] 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.

[34] C. Zhu, R. H. Byrd, and J. Nocedal. L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization, 1997. URL http://www.ece.northwestern.edu/~nocedal/lbfgsb.html.