COCONUT Deliverable D1
Algorithms for Solving Nonlinear Constrained and Optimization Problems: The State of The Art

June 8, 2001

The Coconut Project

Christian Bliek *ILOG*
Peter Spellucci *TU Darmstadt*
Luís N. Vicente *University of Coimbra*
Arnold Neumaier *University of Vienna*
Laurent Granvilliers, Eric Monfroy and Frederic Benhamou *University of Nantes*
Etienne Huens and Pascal Van Hentenryck *Catholic University of Louvain*
Djamila Sam-Haroud and Boi Faltings *Swiss Federal Institute of Technology*

Contents

Chapter 1

Introduction

Christian Bliek
ILOG
Sophia Antipolis
France

The goal of this document is to summarize the state of the art of algorithms for solving nonlinear constrained and optimization problems. These problems have received attention in different research areas. As a result different approaches exist to solve them. Each of the chapters below attempts to summarize the techniques developed in one particular area.

In Chapter 2 a summary of *Nonlinear Local Optimization* techniques is given. It covers the state of the art in the area of traditional numerical analysis. With respect to the other techniques described in this document, this area is the oldest and most established one. The algorithms are able to handle large scale nonlinear optimization problems. However, as the title indicates, these algorithms perform *local* optimization. That is, the solutions they produce are locally optimal, but not necessarily globally optimal. In certain problem classes local optimality implies global optimality. This is for example the case for convex problems. For these problems nonlinear local optimization algorithms can provide good approximations to global optima. Note however that these solutions remain approximations. Indeed local optimization algorithms do not provide conservative bounds on rounding errors.

Many techniques for solving nonlinear problems use derivative information. For a long time derivative information was obtained by numerical approximation. For example finite differences were used to approximate gradients. About 20 years ago a different technique was developed to compute derivatives. This technique called *Automatic Differentiation* is able to compute derivatives that are exact up to machine precision, provided the computer codes that define the functions are available. Automatic differentiation has since found its way in the area of numerical analysis. It also is extensively used by interval arithmetic and constraint satisfaction techniques. In this case the derivatives are not evaluated at a given point but over a given box instead. By taking proper care of the rounding error, this process can produce conservative enclosures of the derivatives over the box. In this project we focus on problems that can be stated under the form of arithmetic expressions. In Chapter 3 we review the state of the art in automatic differentiation for this type of problems.

Over the last decade a number of researchers have become interested in the area of *Nonlinear Global Optimization*. Chapter 4 provides an overview of the techniques that have been developed so far for solving these problems, and put a number of methods, including heuristic ones, in perspective. The chapter summarises a number of results on optimality conditions and certificates which most global optimization methods exploit in one form or another. It discusses various branching methods that can be used and also describes techniques related to *Linear Programming* and *Mixed Integer Linear Programming*.

The area of *Constraint Programming* solves optimization problems by exploiting the constraints of the problem to eliminate infeasible or suboptimal instantiations. This area has long been focused mainly on discrete problems. The interest in continuous domain problems is relatively recent. In contrast to interval arithmetic or global optimization, this area has mainly concentrated on algorithms for propagating nonlinear constraints. These *Constraint Propagation* techniques are reviewed in Chapter 5.

The various techniques described in this document are frequently used in isolation. However, one cannot say that one technique outperforms and is more general than all others. It turns out that the techniques are often complementary in performance and applicability. For example approximations by linear interval systems usually perform well close to a solution whereas propagation techniques are often most effective far from the solution. Also some techniques may not be applicable on some problems. For example linear programming techniques cannot be used to solve nonlinear problems. However, it can be used for the linear part of a nonlinear problem. The state of the art in *Solver Cooperation* is presented in Chapter 6.

In a number of engineering applications, it is not possible or desired to find an optimal solution. Instead one wishes to explore the sets of solutions of a problem and proceed to incrementally refine the problem formulation. This is often the case in the area of engineering design for example with non-routine design problems. The constraint propagation techniques described in Chapter 6 are well adapted to solve this type of problems. However, the enclosure of the sets of solutions they provide may not be very accurate. To address this problem a number of *Complete Enumeration* techniques have developed. They are reviewed in Chapter 7.

Chapter 2

Nonlinear Local Optimization

P. Spellucci
Department of Mathematics
TU Darmstadt
Germany

Overview by:
Luís N. Vicente
Department of Mathematics
Faculty of Sciences and Technology
University of Coimbra

2.1 Overview of the Field

Numerical methods for nonlinear optimization (nonlinear programming) problems are currently complex algorithms that incorporate several components to deal with the different aspects or features of the problems.

Most nonlinear optimization problems are defined by the minimization of an objective function on a feasible region defined algebraically by equality and/or inequality constraints. The geometry, structure, degree of nonlinearity, conditioning, and availability and computability of functions and its derivatives can vary dramatically from problem to problem. It is assumed that the data in nonlinear optimization is deterministic (otherwise one would enter in stochastic optimization).

Most numerical methods for nonlinear optimization are iterative, and each iteration is based on some form of step direction along which some displacement is performed. The step direction calculation is a fundamental tool and influences most of the features of the method, including its computational cost. Newton-based methods, quasi-Newton or secant methods, gradient and reduced-gradient methods, nonlinear conjugate gradient methods, are some of the most popular or well-known schemes to compute step directions. Basic SLP (sequential linear programming) and SQP (sequential quadratic programming) can also be seen as frameworks for computing step directions. The step direction is usually derived from a form of the first-order necessary conditions for local optimality.

The words *local* and *global* are used in nonlinear programming with a double meaning. When applied to minimizers they correspond to relative (local) or absolute (global) minimizers; a local or relative minimizer is a minimizer only in some neighborhood of the feasible region. The first and second-order necessary conditions provide good numerical characterizations for local optimality; they define goals or criteria that a numerical method for nonlinear optimization aim to achieve. The computation of global or absolute minimizers requires a different treatment and leads to the global optimization field.

One can also define *local* convergence and *global* convergence for algorithms (see Chapter 4). An algorithm is said to be globally convergent if it is able to approach a candidate for local optimality from any arbitrary starting point. Local convergence properties of a numerical method for nonlinear optimization speak about the rate or speed of convergence once the iterates enter a neighborhood of such points. An algorithm should have a good global to local behavior, i.e., should converge globally, entering eventually in a neighborhood of local convergence, and, once there, should exhibit a fast rate of convergence (superlinear or quadratic).

There are two main globalization strategies: line searches (that perform some evaluation along the step direction) and trust regions (that circumscribe the calculation of the step to a region of a priori defined size). Both require a merit function. For unconstrained optimization the objective function is itself the merit function. For constrained problems, a merit function is usually a linear combination of the objective function with a penalization of the feasibility violation. Recently, for constrained nonlinear optimization, a new global scheme has been proposed and investigated that uses the notion of a filter, based on multi-criteria optimization, to balance objective function and feasibility violation, as two criteria, avoiding therefore the use of merit functions.

To deal with the topology of the feasible region, additional techniques are commonly required. When inequalities are present, one often sees active-set strategies, interior-point techniques, or projection methods. Active-set strategies exploit the binding or activeness of inequality constraints (an inequality is said to be binding or active at a given feasible point when it acts like an equality at that point). The simplex method in linear programming is an active-set method that searches vertices (extreme points). Interior-point methods in linear programming follow the so-called central path, in the relative interior of the feasible set. In nonlinear programming, the central path may not be defined everywhere; interior-point methods typically follow the quasi-central path and relate with barrier methods. Projections onto the feasible region are efficient only in specific cases, like box constraints.

Solving the original problem by solving a sequence of parametrized problems is another approach. Hopefully, each parametrized problem is considerably easier to solve than the original, and can be solved by the numerical methods mentioned above. Penalty methods, barrier methods, and augmented Lagrangian methods are good examples of such techniques.

Last but not the least, there is a range of factors that can influence dramatically the numerical performance of algorithms for nonlinear optimization. The scaling of the problem is certainly one of the most important factors (for instance, many nonlinear optimization problems are posed originally in function spaces and its discretization carries along the original scaling). The efficiency of the linear algebra implemented within the algorithm, especially when it is sparse, is also determinant; inexactness can play an important role when iterative methods replace factorizations. Derivatives of functions, when not available, can be approximated using finite differences. When functions are evaluated using code for which the source is available, one can apply automatic differentiation to generate code for the derivatives (see Chapter 3). Quasi-Newton or secant methods can also be seen as alternatives when second-order derivatives are not available.

The field is enriched by the many problem classes that appear from applications: unconstrained optimization, quadratic programming, linearly constrained optimization, nonlinear least-squares (or data-fitting) problems, equality constrained optimization, box-constrained optimization, convex programming, etc. There are also application classes that may require proper treatment: optimal control, optimal design, parameter identification, inverse problems, optimization in finance, nonlinear network and transportation problems, etc. Other, perhaps less classical, techniques are becoming very popular in the engineering community, like derivative-free methods, response surface, surrogate modeling, space mapping, etc.

2.2 Introduction

Nonlinear Optimization came into live about fifty years ago with the seminal paper of Kuhn and Tucker [Kuhn and Tucker, 1951], very shortly only after the birth of LP praxis through Dantzig's pioneering work on the simplex algorithm. Since then it has grown to a mathematical discipline in its own right, with deep interconnections with nonlinear analysis and numerical analysis. The appearance of more and more powerful computer equipment and more and more involved numerical solution techniques made the solution of nonlinear models a routine job, which forty years ago were beyond any possible consideration. This is true at least for the solution of problems involving some hundred variables and some thousand constraints, solution meant in a local and weak sense, that means in the sense of identifying points which satisfy certain necessary optimality conditions. The research in this area came to some maturity about ten years ago and a large amount of software is now available even in the public domain [Mittelmann and Spellucci, 2000]. During the past ten years many working groups throughout the world aimed at solution techniques also capable of coping with true large-scale nonlinear optimization problems. Several overview papers about this subject appeared in recent years, e.g. [Conn *et al.*, 1994] and [Gould and Toint, 1999]. In this overview article we give a short introduction into *NLP* theory first and then review some of the most promising solution techniques. Whereas convex problems can be dealt with also in very high dimension successfully already, the treatment of nonconvex cases offers resistance to a satisfactory solution approach, since obviously methods which worked well for medium large problems cannot be transfered to very high dimensions.

2.3 Notation

Superscripts denote elements of sequences. Exponentiation occurs on arithmetic expressions only, i.e. we write $(\sigma)^2$ instead of σ^2. $\|\cdot\|$ denotes the euclidean norm. Superscript T denotes transposition. We avoid the clumsy notation $(x^T, y^T)^T$ for composite vectors and simply write (x, y). No confusion should result from this. We make frequent use of the notation $y_{\mathcal{A}} = \{(y_i) : i \in \mathcal{A}\}$ for a subvector of a vector y with indices in $\mathcal{A}$. Similarly for a matrix B $B_{\mathcal{A}}$ denotes the matrix made up from columns of B with indices in $\mathcal{A}$. Given a vector z $y = z^-$ denotes the vector with components $y_i = 0$ if $z_i \geq 0$ and $y_i = z_i$ otherwise. e denotes the vector $(1, \ldots, 1)^T$. $\nabla f(x)$ denotes the gradient of a function f, a column or a system of columns (in the case of a vectorfunction). It is always related to the Jacobian by

$$\nabla f(x) \;=\; (\mathcal{J}_f(x))^T \;.$$

2.4 Some theory

The general nonlinear optimization problem is

$$\textit{NLP:} \qquad \text{Minimize } f(x) \text{ with respect to } x \in \mathcal{S} \;,$$

where the so called feasible set $\mathcal{S}$ is defined through

$$\mathcal{S} \;=\; \{x \in \mathbb{R}^n : h(x) \;=\; 0,\; g(x) \;\geq\; 0\} \;.$$

We assume here that h and g are defined everywhere on $\mathbb{R}^n$ with values in some $\mathbb{R}^E$ and $\mathbb{R}^I$ and are of class C^2 at least, although some results are valid for the class C^1 too. The nonsmooth case requires quite special treatment, which is beyond the scope of this paper. For f we assume also differentiability of class C^2 on an open superset of $\mathcal{S}$. It is important to note that the same set $\mathcal{S}$ can be represented quite differently by functions h and g. It is the representation by a given triple (f, g, h) which defines a nonlinear programming problem in the sequel.

From a practical point of view, other formulations of *NLP* are more suitable, e.g. one standard form is

$$\mathcal{S} = \{x \in \mathbb{R}^n : \begin{array}{ccccc} x_u & \leq & x & \leq & x_o \\ b_u & \leq & Ax & \leq & b_o \\ c_u & \leq & c(x) & \leq & c_o \} , \end{array}$$

where $x_u,\ x_o,\ b_u,\ b_o,\ c_u,\ c_o$ are given constants, possibly $\in \{-\infty,\ \infty\}$ or possibly of equal size (defining an equation), and A a given matrix. This formulation reflects the different numerical treatment which may be applied to different parts of the constraints. In order to obtain necessary or sufficient optimality conditions, we need further assumptions on h and g, not on $\mathcal{S}$. These assumptions are known as constraint qualifications. Much research took place for obtaining minimal assumptions of this kind, and indeed there is known such a, in some sense minimal one, the *Guignard* constraint qualification [Guignard, 1969], [Gould and Tolle, 1971]. However, finally from additional considerations it turned out that the following one, which much more lends itself to a geometric interpretation, is "the" qualification one should have, first given by Mangasarian and Fromowitz in [Mangasarian and Fromovitz, 1967]:

$$\begin{array}{rl} \textit{MFCQ:} \quad \nabla h(x) & \text{is of full rank} \\ \text{and} & \\ \nabla h(x)^T z = 0,\ \nabla g_{\mathcal{A}}(x)^T z > 0 & \text{is solvable} \end{array}$$

where

$$\mathcal{A} = \mathcal{A}(x) = \{i \in I : g_i(x) \leq 0\}$$

is the so called "active set". (The full rank condition means that the gradients of the equality constraints are linearly independent. From this it follows via the implicit function theorem that the solution set of $\{z : h(x) = h(z)\}$ is locally (around x) a smooth nonlinear manifold in $\mathbb{R}^n$.) Geometrically *MFCQ* means that at x there exists a direction z which is tangential to the nonlinear manifold defined by $\{z : h(z) = h(x)\}$ whereas all active inequality constraints can be strongly improved towards feasibility along z. From the rank condition on ∇h it follows that it is also possible to find some $\tilde{z}$ which improves feasibility with respect to h simultaneously if $h(x) \neq 0$, [Spellucci, 1993a].

Observe that we have defined *MFCQ* here also for infeasible points, whereas the usual definition in the literature only takes feasible points into account. A much stronger condition is the so called regularity assumption

$$\textit{LICQ:} \quad (\nabla h(x), \nabla g_{\mathcal{A}}(x)) \text{ is of full rank .}$$

LICQ implies *MFCQ*. In order to see that consider the underdetermined linear system

$$\begin{pmatrix} \nabla h(x)^T \\ \nabla g_{\mathcal{A}}(x)^T \end{pmatrix} z = \begin{pmatrix} O \\ e \end{pmatrix} .$$

which is always solvable if the matrix has full row rank, as assumed. There are many problems of practical relevance, where the former is violated. *MFCQ* has some far reaching consequences:

Theorem 2.4.1 *Let $x^* \in \mathcal{S}$ be a local minimizer of problem NLP and MFCQ be satisfied there. Then there exists a* <u>*bounded set*</u> *$\mathcal{L}$ of multipliers $\lambda^* \in \mathbb{R}^I_+$ and for every such λ^* there is a unique multiplier $\mu^* \in \mathbb{R}^E$ such that there holds*

$$\begin{array}{rrcl} KTC: & \nabla f(x^*) - \nabla g(x^*)\lambda^* - \nabla h(x^*)\mu^* & = & 0\ , \\ & g_i(x^*)\lambda_i^* & = & 0 \\ & \text{with } \lambda^* \in \mathcal{L} \text{ and } \mu^* = \mu^*(\lambda^*) & & . \end{array}$$

Reverse, if KTC holds with a bounded set of multipliers (λ^*, μ^*), then MFCQ is satisfied. Finally, if f is convex, h affinely linear and g_i concave for all $i \in I$ (a convex optimization problem), then KTC is also sufficient for global optimality.

The proof of this can be found in [Gauvin, 1977] respectively in standard textbooks of nonlinear optimization, e.g. [Nocedal and Wright, 1999]. *MFCQ* has even more implications for a solution of *NLP*, e.g. it is the weakest known condition which guarantess exact penalizability of *NLP* based on properties of h and g alone, see e.g. [Burke, 1991] and also [Guddat *et al.*, 1997]. By exact penalization is meant a transformation of *NLP* into an unconstrained optimization problem, whose (local and global) solutions coincide with those of *NLP*. Such a penalization can be obtained via penalty functions of the form

$$f(x) + \gamma(||h(x)|| + ||g(x)^-||)$$

where γ is a suitably large constant and the norm is an absolute one, i.e. $||y|| = ||\ |y|\ ||\ \forall\ y$. Here and in the following we use the following convention: given a vector x by x^- respectively x^+ we understand a vector with components $\min\{0, x_i\}$ resp. $\max\{0, x_i\}$.

The Mangasarian-Fromowitz condition plays also a fundamental role in questions of stability of *NLP*. E.g.

Theorem 2.4.2 *Consider the perturbed problem NLP(g_0, h_0) where $\mathcal{S}$ is replaced by*

$$\mathcal{S}(g_0, h_0) = \{x \in \mathbb{R}^n : h(x) = h_0,\ g(x) \geq g_0\}\ .$$

For $x \in \mathcal{S}(0,0)$ there exist some $\tilde{x} \in \mathcal{S}(h_0, g_0)$ with $||x - \tilde{x}|| \leq C \times (||h_0|| + ||g_0^+||)$ for some constant $C = C(x)$ and (h_0, g_0) in some neighborhood of $(0,0)$ if and only if MFCQ holds at x.

For the proof, see [Robinson, 1976]. A similar result applies to perturbations of the problem functions h and g in a nonparametric (topological) sense [Guddat *et al.*, 1986]. Hence one may consider a problem instance of *NLP* without *MFCQ* as incorrectly stated. This is surely true as long as feasible points are considered only. But observe that in practice one also needs that condition for infeasible points. Unfortunately it is far from being true that *MFCQ* is naturally satisfied there. Consider the simple and extremely well behaved example with

$$n = 2,\ f(x) = -x_1 - x_2,\ h(x) = x_1^2 + x_2^2 - 2 \quad \text{and} \quad g(x) = x\ .$$

The feasible set is the quarter circle of radius $\sqrt{2}$ in the positive quadrant and even *LICQ* is satisfied here in a small neighborhood of the feasible set. However *MFCQ* is violated for $x_1 < 0,\ x_2 < 0$ and, surprisingly enough, some highly renowned codes fail on this example if provided with an initial guess in this region, if not g is given special treatment. (Most codes do not accept initial guesses which violate the given bound constraints. However, it is possible to construct similar examples with a more complicated, nonlinear g, hence this trouble remains.)

There exist also sufficient local optimality conditions for nonconvex problems. These involve the Hessian matix of the so called Lagrangian of the Problem *NLP*, namely

$$L(x, \lambda, \mu) = f(x) - h(x)^T \mu - g(x)^T \lambda\ ,$$

e.g.

Theorem 2.4.3 *Let be given a point $x^* \in \mathcal{S}$ which satisfies MFCQ and KTC with some pair (λ^*, μ^*). If in addition*

$$z^T \nabla^2_{xx} L(x^*, \lambda^*, \mu^*) z > 0 \quad \forall\ z \neq 0$$

with

$$\begin{aligned} \nabla h(x^*)^T z &= 0, \\ \nabla g_i(x^*)^T z &\geq 0 \text{ if } \lambda_i^* = 0 \\ \nabla g_i(x^*)^T z &= 0 \text{ if } \lambda_i^* > 0 \quad \forall\ i \in I \text{ and some } \lambda^* \in \mathcal{L} \end{aligned}$$

then x^* is a strong local minimizer of NLP.

For a proof, again see [Nocedal and Wright, 1999]. Assumptions as given in the previous theorems are also typical for the convergence analysis of optimization algorithms for solving *NLP*.

2.5 Unconstrained Minimization

The original problem

$$f(x) \stackrel{!}{=} \min_{x \in \mathbb{R}}$$

is typically reduced to the identification of a single stationary point, that means one solution of the (in general nonlinear) equation

$$\nabla f(x) \stackrel{!}{=} 0 \ .$$

There are known also methods which aim in identifying points which in addition satisfy the necessary second order condition "$\nabla^2 f(x^*)$ positive semidefinite". The classical approach here is the damped Newton method, either in form of employing a line search along the Newton direction or in form of a trust-region approach. Newtons method uses second derivatives and before the advent of automatic differentiation techniques this has been considered as a severe disadvantage. Much effort therefore has gone into the development of methods which avoid usage of second derivatives, which lead to the class of quasi-Newton methods. Astonishingly enough the use of e.g. the well analyzed methods from the Broyden class, especially the *BFGS*-update formula, often results in algorithms of higher efficiency than the usage of exact second derivatives, maybe because these methods in some sense remember the global behaviour of f whereas a true Newton step employs pure local information only. It can safely be stated that unconstrained problems of small and medium large dimension, say up to some hundred variables, can nowadays routinely and safely be solved using available quasi-Newton based codes, even if the Hessian matrix is rather illconditioned. Unfortunately, no method of practical relevance evolved in this class which were capable of dealing with sparsity in the Hessian (updates from the Broyden class always give full matrices even for a diagonal Hessian). However much progress has been made towards the solution of large-scale problems by exploiting special structure in the Hessian or by using methods adapted from large-scale linear solvers, see e.g. work by Griewank and Toint [Griewank and Toint, 1982], Nash [Nash, 1984], Buckley and Lenir [Buckley and Lenir, 1983], Byrd, Nocedal and Schnabel [Byrd *et al.*, 1994] and Liu and Nocedal [Liu and Nocedal, 1989]. Griewank and Toint consider problems which show a special additive structure, the class of so called partial separable functions, that are functions f with

$$f(x) \ = \ \sum_{j=1}^{m} f_j(x_j) \text{ with } x_j \text{ a lowdimensional subvector of } x \ .$$

The central idea is to devise updating formulae for the Hessians of the f_j by dense matrices A_j separately and combining these to a sparse approximation of the Hessian of f. There is public domain software which implements this. The method works well in the convex case and if dimensionality is small enough to allow direct solution techniques for the linear system with the matrix $\sum_j A_j$. All other methods mentioned can be characterized as modifications of Newton's method (so called inexact Newton methods) or of the method of conjugate gradients, known for the linear case since 1951. Using these methods problems with some 10000 variables can be solved without much trouble, at least if ill-conditioning of the Hessian is not too hard.

In high dimensional unconstrained minimization the main effort comes from the determination of the so called direction of descent d^k with $(\nabla f(x^k))^T d^k < 0$.

With this in mind the majority of methods which have been proposed up to now follow the linesearch model, i.e.

$$x^{k+1} \;=\; x^k + \sigma_k d^k \ ,$$

where σ_k is chosen such that $\{f(x^k)\}$ decreases strongly monotonically satisfying the principle of sufficient decrease

$$f(x^k) - f(x^{k+1}) \to 0 \Rightarrow \nabla f(x^k) \to 0 \ ,$$

with d^k satisfying the condition

$$\frac{|(\nabla f(x^k))^T d^k|}{||d^k||} \;\geq\; \phi(||\nabla f(x^k)||).$$

Here ϕ is a strongly monotonically increasing function with $\phi(0) = 0$, not necessarily explicitly known. The stepsize σ_k can be computed by a variety of algorithms, the simplest one being backtracking:

$$\begin{aligned} \sigma_k \;\;=\;\; & \text{argmax}_j\{\beta^j \sigma_{0,k} : j \;\in\; \mathbb{N}_0 \text{ and} \\ & \qquad f(x^k) - f(x^k + \beta^j \sigma_{0,k} d^k) \;\geq\; \delta \beta^j \sigma_{0,k}(-\nabla f(x^k))^T d^k\} \end{aligned}$$

where $0 \;<\; \beta \;<\; 1$ and $0 \;<\; \delta \;<\; \frac{1}{2}$ are two constants and $\sigma_{0,k}$ an initial guess, maybe simply 1 or some value obtained from a preliminary interpolation, e.g. using the values $f(x^k)$, $f(x^k + d^k)$ and $\nabla f(x^k)^T d^k$.
The so called trust-region methods also require a descent of this form, using formally $\sigma_k = 1$. Here a local model for f is used which has the form

$$\tilde{f}(d) \;=\; f(x_k) + (\nabla f(x^k))^T d + \frac{1}{2} d^T A_k d, \; ||d|| \;\leq\; \rho_k \ .$$

The (global) minimizer of this problem yields d^k and the step $x^{k+1} = x^k + d^k$ is accepted if

$$0 \;<\; \eta \;<\; \frac{f(x^k) - f(x^{k+1})}{\tilde{f}(0) - \tilde{f}(d^k)}$$

for some small constant $\eta \;>\; 0$, e.g. $\eta \;=\; 10^{-3}$. There is then an adaptive choice of ρ_k. If the test fails, then ρ_k is decreased and d^k must be computed anew, that means a further (large) linear system must be solved, which increases the algebraic complexity considerably. But the function information need not be computed anew and for very expensive evaluations the additional algebraic effort may be worthwhile. If the step is accepted, then ρ_{k+1} is chosen equal to or larger than ρ_k. Trust-region methods have advantages over linesearch methods because of weaker conditions on the models matrix A_k and stronger theoretical results (typically these methods converge to second order necessary points without using negative curvature information if $A_k = \nabla^2 f(x^k)$ for all k.) A trust-region Newton's method is realized in the code `TRON` of Lin and Moré [Lin and Moré, 1999].

In the (theoretically) ideal case the direction d^k is given by Newton's method. The linear system

$$\nabla^2 f(x^k) d^k \;=\; -\nabla f(x^k)$$

can be solved by direct methods for large n if the Hessian is very sparse only. Fortunately, it is not necessary to solve this system very precisely, at least far from the solution, without loosing the good convergence properties. This observation leads to the so called inexact Newton methods, where the Newton direction is approximated by some steps of the preconditioned conjugate gradient method or some solver based on the Lanzcos algorithm. This latter approach requires a routine which computes the product of the Hessian with a given arbitrary vector. In Nash's Code `TN/TNB`, which is available through *NETLIB* [Nash, 2001] this is replaced by taking a forward difference of the gradient $(\nabla f(x^k + \tau v) - \nabla f(x^k))/\tau$, which provokes additional problems. If the solution precision is controlled appropriately (one needs a final residual less than $\alpha_k ||\nabla f(x^k)||$, $\alpha_k \to 0$), then superlinear convergence speed is maintained. The computation of d^k requires several matrix-vector products respectively several evaluations of ∇f.

Alternatively one may use the so called quasi-Newton methods with limited-memory or the nonlinear preconditioned conjugate gradient method, applied directly to f. These methods generate directions of descent also for nonconvex f and need as essential information only one gradient of f every step. The nonlinear conjugate gradient method, originally proposed by Fletcher and Reeves and now known in a lot of variants is the simpler of these two approaches. Here d^k is computed recursively

$$\begin{aligned}
g^k &= \nabla f(x^k) \\
\hat{g}^k &= B g^k, (B \text{ is the preconditioner, positive definite }) \\
\gamma_k &= (g^k)^T \hat{g}^k \\
d^k &= \begin{cases} \hat{g}^k \text{ if } k = 0(\bmod N) \\ (1 - \frac{(g^k)^T d^{k-1}}{\gamma_{k-1}})\hat{g}^k + \frac{\gamma_k}{\gamma_{k-1}} d^{k-1} \text{ otherwise.} \end{cases}
\end{aligned}$$

N is the so called restartindex. In theory one has $N \geq n$, and, as shown by AlBaali [Al-Baali, 1985], N may be even infinite. With the exception of the leading factor of $\hat{g}^k$ these are the relations of the classical (linear) cg-algorithm. The modification forces $(g^k)^T d^k = \gamma_k > 0$, independently of the properties of f and σ_k. In practical computations it makes little sense to let N grow very much because the quality of the directions is deteriorated very strongly by roundoff and finally the determination of the stepsize σ comes into trouble.

Remark.: There is a point which often is neglected in theoretical discussions of this method. In the linear case, with a fixed positive definite Hessian, one knows for sure that the "optimal" stepsize produces descent and never checks that, whereas here monotonic descent of f must be verified, for fixing the stepsize. However, under the influence of roundoff, the test for descent may fail even if the direction d^k is a good one. □

If one chooses $N \geq n$ then N-step quadratic convergence of this method can be shown. However, for very large n this is practically irrelevant.

The usual quasi-Newton methods replace the Hessian G_k by a positive definite "approximation" A_k and compute d^k from

$$A_k d^k = -\nabla f(x^k)$$

resp. with $H_k = A_k^{-1}$

$$d^k = -H_k \nabla f(x^k) .$$

None of the quasi-Newton methods designed for the maintenance of a given sparsity pattern of the Hessian has shown practical success, whereas the successful ones all produce full matrices even for diagonal Hessian. That caused the research in limited memory methods. One of the best practical methods is based on the so called *BFGS*-formula which defines $\{A_k\}$ recursively using the differences

$$s^k = x^{k+1} - x^k$$

and

$$y^k = \nabla f(x^{k+1}) - \nabla f(x^k) .$$

It reads

$$A_{k+1} = A_k - \frac{A_k s^k (s^k)^T A_k}{(s^k)^T A_k s^k} + \frac{y^k (y^k)^T}{(y^k)^T s^k} ,$$

which for $(y^k)^T s^k > 0$ and positive definite A_k produces a positive definite A_{k+1}. (It is even possible to show that for uniformly convex f the eigenvalues of A_k and A_k^{-1} remain uniformly bounded under very weak conditions on the step $x^k \rightarrow x^{k+1}$, see e.g. [Byrd and Nocedal, 1989]. The condition $(y^k)^T s^k > 0$ is satisfied automatically for f uniformly convex and is enforced otherwise through the stepsize selection using the so called Powell-Wolfe conditions). For A_k resp. $H_k = A_k^{-1}$ there exists for the case $A_0 = \frac{1}{\gamma} I$ another closed formula [Byrd *et al.*, 1994]

$$H_{k+1} = \gamma I + C_k \begin{pmatrix} D_{11,k} & D_{12,k} \\ D_{21,k} & O \end{pmatrix} (C_k)^T$$

where

$$\begin{aligned} C_k &= \begin{pmatrix} S_k & \gamma Y_k \end{pmatrix} , \\ D_{21,k} &= -R_k^{-1} , \\ D_{12,k} &= (D_{21,k})^T , \\ D_{11,k} &= D_{12,k}(\Delta_k + \gamma (Y_k)^T Y_k) D_{21,k} \end{aligned}$$

with

$$
\begin{aligned}
(R_k)_{ij} &= \begin{cases} (s^i)^T y^j \text{ for } 0 \le i \le j \le k \\ 0 \text{ otherwise} \end{cases} \\
\Delta_k &= \operatorname{diag}_{0\le i\le k}((s^i)^T y^i) \ , \\
S_k &= (s^0, \dots, s^k) \ , \\
Y_k &= (y^0, \dots, y^k) \ .
\end{aligned}
$$

The idea of quasi-Newton methods with limited memory simply consists in defining in this formula Y_k, S_k, Δ_k and R_k using the last m instances

$$y^k, \dots, y^{k-m+1}, s^k, \dots, s^{k-m+1}$$

and a variable $\gamma = \gamma_k$ and fixing a formula for H_k this way. Of course this applies for $k > m$ only. Recommended values for m are in the range $\{7, \dots, 20\}$. This is implemented in the code `LBFGS(B)` of Byrd , Nocedal and coworkers [Byrd *et al.*, 1995]. With $m \ge n$ this method should, from theory, give the same results as the original method, if applied to uniformly convex quadratic f. Unfortunately this is far from being true. The numerical evaluation of H_k using this "direct" formula seems to be subject to numerical instability and further research in this direction is required. However, practioners like the method because of their simplicity. (no Hessian required). But from numerical experiments it became clear that for strongly nonquadratic f the incomplete Newton methods are to be preferred. Numerical experience is reported in the papers [Gilbert and Lemarechal, 1989], [Liu and Nocedal, 1989] , [Lucidi and Roma, 1997], [Zou *et al.*, 1993].

2.6 Bound constrained problems

For reasons of simple exposition we restrict the discussion to the problem

$$f(x) \stackrel{!}{=} \min_{x\in\mathbb{R}^n : x\ge 0} \ .$$

The case of general lower and upper bounds and of only partially constrained components of x can be derived with ease from this special one. The necessary first order optimality conditions now read

$$\nabla f(x) - \lambda = 0, \quad \lambda_i x_i = 0, \quad \lambda \ge 0 \ .$$

This can be rewritten as

$$\nabla f(x)_i = \begin{cases} 0 \text{ if } x_i > 0 \\ \ge 0 \text{ otherwise} \ . \end{cases}$$

The first methods for this type of problem were of the type of active set methods. Initially f is minimized using an unconstrained technique. However, by proper selection of the stepsize, the constraints remain satisfied. If a variable meets its lower bound, it is fixed there and minimization of f continues on a boundary linear manifold of the feasible set. This process continues until either a (resp. the) constrained minimizer is located or otherwise the components of the gradient corresponding to free variables became small whereas one of its components corresponding to a fixed variable stays relatively large and of negative sign, indicating a false constraining manifold. In the latter case the corresponding variable is freed and moved into the relative interior of the feasible set, with minimization of f continuing. Using some simple criteria for fixing what is meant by "small" and "relatively large" here convergence of this process to a constrained minimizer can be shown. This kind of method is simple to implement and robust, but necessarily very slow if many changes of the active set are required. Nash's code `TNB` is of this kind. All of the newer methods for this kind of problem however make use of a special intermediate step as soon as a variable meets its bound. This special step consists in (at least one) step of the gradient projection method. This is given by exact (for linear or quadratic f) or approximate minimization of f along the projection of the ray $x^k - \sigma\nabla f(x^k)$ onto the feasible set, which in the case of bound constraints is trivial , namely

$$\mathcal{P}_{\mathcal{S}}(x^k - \sigma\nabla f(x^k)) \;=\; \max\{0, x^k - \sigma\nabla f(x^k)\}\ .$$

with max for a vector understood componentwise. If the active set $\mathcal{A}$ does not change, then unconstrained minimization of f on a constraining manifold (using a method better than the mere gradient descent) is resumed. This leads to fast changes in the active set and rather efficient overall performance, e.g. Byrd et. al. [Byrd *et al.*, 1995] (code `LBFGSB`), Conn, Gould und Toint [Conn *et al.*, 1988a, Conn *et al.*, 1989] (code `SBMIN` gone into `LANCELOT`), [Conn *et al.*, 1988b], Moré and Lin [Lin and Moré, 1999] , code `TRON`, Moré and Toraldo [Moré and Toraldo, 1989], [Moré and Toraldo, 1991] (code `GPCG` by Felkel, see [Mittelmann and Spellucci, 2000]) and Felkels code `PL2`, again see [Mittelmann and Spellucci, 2000] and Felkels PhD dissertation [Felkel, 1999], see also Moré's `MINPACK 2` project [Averick and Moré, 1994].

Using this technique it is possible to solve bound constrained quadratic problems with almost the same effort as unconstrained ones. However, this author does not completely participate in the enthusiasm of many colleagues for gradient projection. It is true that the sole role of this method is the fast identification of the correct active set. Nevertheless, for a general nonlinear f it is necessary to check the descent of f along the projected ray if the function is nonconvex or if the computation of the two directional derivatives of f at the breakpoints of the ray cannot be computed cheaply. Because the optimal stepsize along the projection might be quite small for mildly illconditioned Hessian of f, this may lead to premature termination. The optimal stepsize along a ray $x - \sigma\nabla f(x)$ is of the form

$$\sigma^* \;\in [\frac{1}{\mathrm{eig}_{\max}(G)}, \frac{1}{\mathrm{eig}_{\min}(G)}](1 + \mathcal{O}(||\nabla f||))\ ,$$

where eig(G) are the eigenvalues of the Hessian G of f. The correction

$$\sigma^* \nabla f(x) \stackrel{def}{=} s$$

can be as small as

$$\frac{3\sqrt{3}}{2} \frac{1}{cond(G)} ||x - x_{\text{opt}}||(1 + \mathcal{O}(\frac{1}{cond(G)}))$$

whereas the difference $f(x) - f(x+s)$ may be of the order of

$$\frac{9}{4} \frac{1}{cond(G)} \text{eig}_{\min}(G) ||x - x_{\text{opt}}||^2$$

(which is much smaller than $f(x) - f(x_{\text{opt}})$.) If this value reaches the roundoff level in the evaluation of f the method terminates of course, at a relatively large value $||x - x_{\text{opt}}||$, even for $\nabla^2 f$ only mildly illconditioned. Numerical tests verified this behaviour [Pinz, 1996]. Therefore it might finally not be the best solution to move along constraining manifolds at all. There exist newer approaches which make use of a primal-dual approach, including movement in the dual variables $(\nabla f(x))_i$ for $x_i = 0$, e.g. [Di Pillo *et al.*, 2000]. Others aim in avoiding movement along constraining manifolds using moves that at least in a number of variables are always interior to the feasible set, e.g. [Branch *et al.*, 1999]. But thorough testing of all these solution approaches on an identical testbed are necessary to assess their relative merits.

2.7 General linearly constrained problems: active set methods

Linearly restricted problems are much easier to handle than those with general constraints. The feasible set is convex. There exist methods for the elimination of redundant equalities and inequalities from systems of linear equality and inequality constraints which work (at least theoretically). Numerically there might be trouble due to illconditioning. (Indeed there are lots of examples in the `NETLIB LP`-library where infeasibility is hard to discern.) With this reduction done *MFCQ* is satisfied globally for these systems. There exist finite algorithms to decide feasibility of such constraints (e.g. phase I of the simplex algorithm). No one of these properties translates to general constraints.

Minimization methods of the active set type (minimization on submanifolds) can easily be devised for general linear constraints, hence it is possible to maintain feasibility in linearly constrained optimization quite easily, once a feasible point is known. Some of the sequential quadratic programming (*SQP*)-methods discussed later in this paper use this fact. Contrary to this the gradient projection method cannot be implemented efficiently for this type of constraints, since the determination of the projection of an (infeasible) point onto a polyhedron involves the solution of a convex quadratic programming problem whose solution might be as costly as the solution of the original problem.

For the convex quadratic linearly constrained optimization problem, the so called convex *QP* problem, i.e.

$$f(x) \;=\; \frac{1}{2}(x)^T Ax - a^T x \quad \text{where } A \text{ is positive semidefinite} \quad .$$

with the constraints

$$g(x) \;=\; G^T x + g^0 \;\geq\; 0 \;, \quad h(x) \;=\; H^T x + h^0 \;=\; 0 \;,$$

including the case $A = O$, the *LP* problem, there exist specialized highly efficient methods, e.g. the simplex method , the dual simplex method and its generalizations to the quadratic case, e.g. methods given by Fletcher [Fletcher, 1971], Goldfarb [Goldfarb, 1972] and Goldfarb and Idnani [Goldfard and Idnani, 1983]. These methods are part of many optimization libraries and the core of most of the *SQP* methods for medium large problems, which will be dealt with later in this paper. Using sparse matrix methods these algorithms can also be extended to higher dimensional problems, for the quadratic case see e.g. [Gould, 1991]. For the *LP* problem there exists a bunch of implementations, commercial as well as in the public domain, capable of solving problems up to 10^6 variables, given sufficient sparsity in the constraints.

There is newer work [Forsgren and Murray, 1997] which deals with the possibility to combine classical active set technology with activation and inactivation of an arbitrary number of constraints per step, which makes these methods even more efficient, see also sections 3.4.6 and 3.4.7 in [Spellucci, 1993a]. Nevertheless the theoretical worst case complexity of such an approach is always exponential in dimension (and input size). Algorithmically, these methods can be quite involved.

Besides direct treatment of the active linear constraints via elimination techniques (using a choice of numerical linear algebra techniques) there exists also the possibility to transform the problem to a bound constrained one and using the powerful techniques described in the previous section for the latter. For simplicity of exposition let us assume that the constraints are already in the form used by the primal simplex method, namely

$$B^T x - b \;=\; 0 \;, x \;\geq\; 0 \;.$$

Indeed any linearly constrained problem can be transformed such that the constraints take this form. Let us also assume that B is of full rank (In reality this is often not the case, and the <u>numerical</u> treatment of rank deficiency is extremely delicate.) The first order necessary optimality conditions then read

$$\begin{aligned} \nabla f(x) - B\mu - \lambda &= 0 \;, \\ \lambda &\geq 0 \;, \\ \lambda_i x_i &= 0 \;, \; i = 1, \ldots, n \;. \end{aligned}$$

If f is convex then these conditions are also sufficient for optimality of a feasible point. If there exists a so called Slater point $\bar{x}$ with $B^T \bar{x} - b \;=\; 0 \;, \bar{x} \;>\; 0$, then the set of possible Lagrangian multipliers is bounded. Friedlander, Martínez and Santos [Friedlander *et al.*, 1994] have shown, that in the case f convex and of class C^2 the solution can be obtained via the following bound constrained minimization problem involving the primal and the dual variables x, μ and λ simultaneously:

$$M(x,\mu,\lambda) \;=\; \frac{1}{2}\big(||\nabla f(x) - B\mu - \lambda||^2 + ||B^T x - b||^2 + (x^T\lambda)^2\big) \;\stackrel{!}{=}\; \min_{x\geq 0,\ \lambda\geq 0} \;.$$

An obvious disadvantage of this approach is the increase in dimensionality. Kanzow [Kanzow, 1994], using the so called Fischer-function

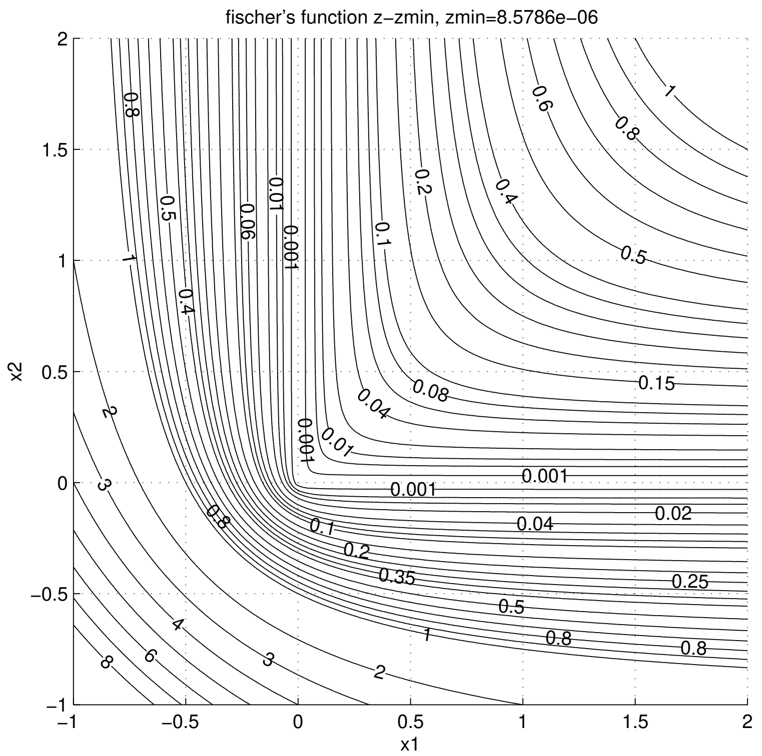
$$\phi(\alpha,\beta) \;=\; (\sqrt{\alpha^2+\beta^2} - \alpha - \beta)^2$$

succeeded in reformulating the problem as a completely unconstrained one:

$$K(x,\mu,\lambda) \;=\; \frac{1}{2}\big(||\nabla f(x) - B\mu - \lambda||^2 + ||B^T x - b||^2 + \sum_{i=1}^{n}\phi(x_i,\lambda_i)\big) \;\stackrel{!}{=}\; \min \;.$$

Fischer's function ϕ has the interesting property to be stationary exactly for $\alpha,\beta \geq 0, \alpha\beta = 0$ and penalizing simultaneously the primal and dual bound constraints as well as the so called complementarity condition $x_i\lambda_i = 0$. There are many other functions of this type which are heavily used in solution methods for the so called complementarity problem

$$F(x)^T y \;=\; 0, \quad F(x) \;\geq\; 0, \quad y \;\geq\; 0 \;.$$



Numerical tests with high dimensional convex QP problems have shown superiority of the Friedlander et. al. approach compared with Kanzow's. Both methods transform a convex quadratic problem to a nonconvex nonquadratic one, which may be considered a disadvantage. Indeed any convex QP problem can be rewritten as a convex bound constrained QP problem, as shown by the present author [Spellucci, 1993b]. With

$$L(x,\mu,\lambda) = f(x) - \mu^T(B^Tx - b) - \lambda^T x$$

and

$$\Phi(x,\mu,\lambda;\delta,\eta) = -L(x,\mu,\lambda) + \tfrac{\delta}{2}\,\|B^Tx - b\|^2 + \tfrac{\eta}{2}\,\|Ax - a - B\mu - \lambda\|^2$$

there holds the following: If A is positive semidefinite and in addition positive definite on the kernel of B^T, then the minimization of Φ with respect to $x,\ \mu,\ \lambda$ under the bound constraints

$$x \ \geq\ 0\ ,\ \lambda\ \geq\ 0$$

is equivalent to solving the original QP provided $\delta > 0$ and $\eta > \rho((Z^TAZ)^{-1})$, with Z an orthonormal basis of B^T's kernel. $\rho(.)$ denotes the spectral radius of a matrix. Here the trouble comes through the determination of η. Since the (nearly exact) determination of the theoretical lower bound for this parameter is out of discussion for a truly large-scale problem, much remains to be done here. All these approaches introduce an additional problem (also present for all known primal dual differentiable exact penalty functions for the general *NLP* problem): the condition number of the Jacobian of the corresponding *KTC*-system is in the order of the square of the condition number of the *KTC*-system of the original problem and there seems to be no simple way to alleviate this effect. It is an open question whether there exist exact primal-dual smooth penalty functions (a function for which a single unconstrained minimization gives the solution of *NLP* together with the multipliers) without that nasty effect. Numerical tests which such methods are given in [Felkel and Spellucci, 1996], [Spellucci, 1997]. In [Spellucci, 1997] the *QP* problems are constructed artificially with known solution and known condition number and minimized with the code `LBFGSB` of [Byrd *et al.*, 1995]. This solver was used with identical parameters and identical termination criterion. The computations were done on a HP9000/715 with double precision IEEE754 arithmetic (about 16 decimals). By proper selection of the termination criteria termination took place only if the relative changes in the objective function were below 10^{-13}. The tables which follow show the precision obtained. `DXR`, `DMR` and `DLR` denote the norm of the error divided by the norm of the true solution, measured in the euclidean norm, for the variables x, μ and λ respectively. The condition number of the Hessian and the reduced Hessian was chosen identically in $\{10, 100, 1000, 10000\}$, just the reciprocal of the value $\sigma_{\min}$ listed in the table. Such a condition number would not be considered overly bad. n varied from 1000 to 10000 in steps of 1000.

Method of Friedlander et. al.

$\sigma_{\min}$	$DXR \in$	$DMR \in$	$DLR \in$
10^{-1}	$[5 \cdot 10^{-6}, 4 \cdot 10^{-5}]$	$[6 \cdot 10^{-5}, 1 \cdot 10^{-3}]$	$[7 \cdot 10^{-7}, 3 \cdot 10^{-5}]$
10^{-2}	$[3 \cdot 10^{-4}, 3 \cdot 10^{-3}]$	$[5 \cdot 10^{-2}, 1 \cdot 10^{-1}]$	$[1 \cdot 10^{-5}, 3 \cdot 10^{-3}]$
10^{-3}	$[1 \cdot 10^{-3}, 1 \cdot 10^{-2}]$	$[8 \cdot 10^{-2}, 1 \cdot 10^{-1}]$	$[2 \cdot 10^{-5}, 3 \cdot 10^{-2}]$
10^{-4}	$[3 \cdot 10^{-3}, 3 \cdot 10^{-2}]$	$[9 \cdot 10^{-2}, 2 \cdot 10^{-1}]$	$[3 \cdot 10^{-5}, 6 \cdot 10^{-2}]$

Method of Kanzow

$\sigma_{\min}$	$DXR \in$	$DMR \in$	$DLR \in$
10^{-1}	$[2 \cdot 10^{-5}, 1 \cdot 10^{-2}]$	$[4 \cdot 10^{-4}, 1 \cdot 10^{-1}]$	$[9 \cdot 10^{-8}, 3 \cdot 10^{-2}]$
10^{-2}	$[2 \cdot 10^{-3}, 2 \cdot 10^{-2}]$	$[2 \cdot 10^{-1}, 3 \cdot 10^{-1}]$	$[6 \cdot 10^{-3}, 8 \cdot 10^{-2}]$
10^{-3}	$[7 \cdot 10^{-3}, 4 \cdot 10^{-2}]$	$[1 \cdot 10^{-1}, 2 \cdot 10^{-1}]$	$[8 \cdot 10^{-3}, 1 \cdot 10^{-1}]$
10^{-4}	$[1 \cdot 10^{-2}, 5 \cdot 10^{-2}]$	$[1 \cdot 10^{-1}, 3 \cdot 10^{-1}]$	$[6 \cdot 10^{-3}, 1 \cdot 10^{-1}]$

Method of Spellucci

$\sigma_{\min}$	$DXR \in$	$DMR \in$	$DLR \in$
10^{-1}	$[1 \cdot 10^{-6}, 2 \cdot 10^{-5}]$	$[2 \cdot 10^{-5}, 3 \cdot 10^{-4}]$	$[2 \cdot 10^{-11}, 3 \cdot 10^{-6}]$
10^{-2}	$[7 \cdot 10^{-5}, 3 \cdot 10^{-4}]$	$[5 \cdot 10^{-3}, 4 \cdot 10^{-2}]$	$[4 \cdot 10^{-12}, 1 \cdot 10^{-6}]$
10^{-3}	$[2 \cdot 10^{-4}, 1 \cdot 10^{-1}]$	$[5 \cdot 10^{-2}, 9 \cdot 10^{-2}]$	$[6 \cdot 10^{-12}, 2 \cdot 10^{-2}]$
10^{-4}	$[3 \cdot 10^{-3}, 3 \cdot 10^{-1}]$	$[4 \cdot 10^{-2}, 9 \cdot 10^{-2}]$	$[2 \cdot 10^{-11}, 4 \cdot 10^{-2}]$

Taking into account that such *QP* solvers may possibly be used as a block box solver within other algorithms it becomes clear from the above results, which indicate errors up to 30 percent for only moderately illconditioned problems, that the practical value of these approaches is quite questionable.

2.8 Linearly constrained problems: Interior-point methods

In the following we again restrict the discussion to the special problem

$$f(x) \stackrel{!}{=} \min$$

with the constraints

$$B^T x - b = 0 , \quad x \geq 0 .$$

Again we require full rank of B. At first we only consider a convex quadratic f,

$$f(x) = \frac{1}{2} x^T A x - b^T x , \quad \text{A positive semidefinite}$$

including the *LP* case. Since Karmarkar's seminal paper [Karmarkar, 1984] on the construction of methods of polynomial complexity for the *LP* problem there has been an enormous research effort on what is now known as "interior-point methods". Meanwhile the online archive [InteriorPoint, 2001] contains papers which appeared since 1994, and also a pointer to an older collection from Eberhard Kranich, making up some thousand contributions. Today the so called primal-dual interior-point methods are considered as the most powerful approach and we discuss here a typical candidate from this class. The development starts with considering the *KTC* conditions which in the convex case, given a Slater-point, are necessary and sufficient characterizations of the solution:

$$\begin{aligned} Ax - a - B\mu - \lambda &= 0 , \\ \lambda &\geq 0 , \\ x &\geq 0 , \\ \lambda_i x_i &= 0 , \; i = 1, \ldots , n \\ B^T x - b &= 0 . \end{aligned}$$

We consider this as a nonlinear system in x, μ and λ. Discarding the sign-constraints for x and λ we can write this as a nonlinear equation

$$F(x, \mu, \lambda) = \begin{pmatrix} Ax - a - B\mu - \lambda \\ -B^T x + b \\ X\lambda \end{pmatrix} = 0$$

where $X = diag(x_1, \ldots , x_n)$. We parametrize the problem changing the complementarity condition to

$$\begin{aligned} Ax - a - B\mu - \lambda &= 0 , \\ \lambda_i x_i &= \epsilon , \; i = 1, \ldots , n , \\ B^T x - b &= 0 . \end{aligned}$$

On the set of "interior" points

$$x > 0 \, , \lambda > 0$$

the Jacobian of this system is always regular. For $\epsilon > 0$ the unique solution describes a smooth curve with parameter ϵ, (known as the "central trajectory") which tends for $\epsilon \to 0$ to one of the solutions of the initial problem.
Under the additional condition of strict complementarity of one solution this limit point can be precisely characterized (it is the "analytic" center of the solution set).
Remark.: Observe that in the *LP*- and also in the not strictly convex *QP*-case the solution will not necessarily be unique. □
This solution curve is now computed pointwise and approximately only, using the damped Newton method for ϵ fixed, and varying ϵ slowly. During this computation the constraints $x > 0$, $\lambda > 0$ must be maintained. If this is done naively then one runs into serious trouble because of the singularities at the boundary and the true art in devising an efficient algorithm consists in a senseful change of ϵ and a senseful selection of the stepsize parameter for damping the Newton step. It turns out that ϵ_k should depend on the duality gap

$$(x^k)^T \lambda^k / n \; .$$

A typical algorithm is the one of Kojima, Mizuno and Yoshise [Kojima *et al.*, 1989]:

$$\begin{aligned}
\mathcal{J}_F(z^k)\Delta z_N^k &= -F(z^k) \text{ to be solved : Newton step} \\
\mathcal{J}_F(z^k)\Delta z_C^k &= \rho_k \hat{e} \text{ to be solved : centering step} \\
\rho_k &= \sigma_k((x^k)^T\lambda^k)/n \; , \\
\Delta z^k &= \Delta z_N^k + \Delta z_C^k \stackrel{def}{=} (\Delta x^k, \; \Delta\mu^k, \; \Delta\lambda^k)^T \\
\hat{\alpha}_k &= \min\{\min\{x_i^k/(-\Delta x_i^k) : \Delta x_i^k < 0\}, \\
&\qquad \min\{\lambda_i^k/(-\Delta\lambda_i^k) : \Delta\lambda_i^k < 0\}\} \; , \\
\alpha_k &= \min\{1, \tau_k \hat{\alpha}_k\} \; , \\
z^{k+1} &= z^k + \alpha_k \Delta z^k \; .
\end{aligned}$$

Here $z = (x, \mu, \lambda)$, $\tau_k \in]0,1[$, $\sigma_k \in [0,1[$ and $\hat{e} = (0, \ldots, 0, e)^T$ where $e = (1, \ldots, 1)^T \in \mathbb{R}^n$.
In the *LP* and convex *QP* case one can show total polynomial complexity of the algorithm for reaching the exact solution (in the case of exact rational input data) with a step number of $\mathcal{O}(\sqrt{n}L)$, where L is the so called input length. In the case of integer coefficients and an *LP* this is defined as

$$L = \sum_{i,j} \log_2(|b_{ij}| + 1) + \sum_j \log_2(|b_j| + 1) + \sum_i \log_2(|a_i| + 1) \; .$$

After a finite number of steps the correct constraining manifold can be identified exactly, which allows subsequently to abandon the path following and to move to the boundary in $\mathcal{O}(n)$ steps. Of course, this is a nice theoretical result (concerning the so called "small step" methods) but without any practical relevance. A rule of thumb states that "thirty steps suffice" (for the so called "large step" methods) to come sufficiently near to the desired solution. It remains to discuss how to solve the linear systems in this method in the true large-scale situation. The Jacobian of F reads

$$\mathcal{J}_F(z) \;=\; \begin{pmatrix} A & -B & -I \\ -B^T & O & O \\ \Lambda & O & X \end{pmatrix} \;,\; \Lambda \;=\; \mathrm{diag}(\lambda_i)\;.$$

A is the Hessian of f and hence positive semidefinite for convex f. If $A = O$, i.e. the *LP* case, then the solution of the linear systems with this matrix is relatively easy, since it can be reduced to a linear system with the (by assumption) positive definite and much smaller matrix $B^T\Lambda^{-1}XB$. In case of a general semidefinite A this is no longer possible and a first resort would be methods for large indefinite symmetric systems, e.g. the techniques developed by Duff, Reid and coworkers [Duff and Reid, 1983], [Duff and Reid, 1993]. Fortunately these matrices are not general indefinite, such that more efficient solution methods are available see e.g. [Gill *et al.*, 1996], [Vanderbei, 1995]. There exist other specialized techniques, e.g. iterative preconditioned techniques like those considered by Freund and Jarre [Freund and Jarre, 1994]. Primal-dual methods for the *LP* case are discussed in depth by St. Wright in [Wright, 1997].

Mittelmann [Mittelmann, 2000] performed a large number of comparative studies of several interior-point- and simplex-based methods. From his results it can be seen that in many instances interior-point methods can by far outperform the simplex method, but there are problems concerning the reliability of such codes, which however continue to diminish in time to date. Since the codes tested are contiuously updated and improved, the interested reader should directly consult Mittelmanns pages for obtaining the newest results.

It is possible to derive interior-point methods from the minimization of the so called logarithmic barrier function (a method known for a long time [Fiacco and McCormick, 1990b] and disregarded subsequently because of the inherent illconditioning. In the case of a problem satisfying the linear independence and second order sufficiency condition $E + |\mathcal{A}|$ of the eigenvalues of the Hessian tend to infinity whereas the remaining ones stay bounded. If path following is used and the exact Hessian is available, this has only little effect. However, if one uses a general method of descent then even slight illconditioning has a detrimental effect due to the loss of centrality. Restricted to the central path the conditioning of the problem is much better). The logaritmic barrier method penalizes a step towards the boundary by a term which grows to infinity there, multiplied by a parameter (the ϵ above) which is driven to zero, such that the minimizer is approached from the interior of the feasible set. Mostly one uses the so called logarithmic barrier function

$$f(x) - \epsilon\sum_{i=1}^{n}\ln(x_i) \;\stackrel{!}{=}\; \min_x, \quad B^Tx - b = 0$$

with $\epsilon \to 0$. The direct application of this idea requires a point $x > 0$, which satisfies the equality constraints. This defines a "feasible interior-point" approach, but there exist also "infeasible interior-point" methods, where the equality constraints are satisfied in the limit only.
Remark: If the QP problem is nonconvex, then there is no known algorithm of polynomial complexity to solve it. Even the decision whether a stationary point found is a local minimizer, is NP hard, [Sahni, 1974], [Vavasis, 1990]. The methods described above cannot be used for nonconvex cases. □
Remark: In addition to the results mentioned above it is at least of theoretical interest that it is also possible to obtain superlinear convergence using these methods by proper choice of the parameters, see e.g. [Potra *et al.*, 1993] □
If f is a general smooth convex function, then the same techniques can in principle be applied, simply replacing the matrix A there by $\nabla^2 f(x^k)$. The step size selection then has in addition to check the descent property of f as the sole additional complication. An algorithm around this ideas has been developed from `HOPDM`, see [Epelly *et al.*, 2000].

2.9 Nonlinearly constrained problems

Concerning nonlinearly constrained problems there is presently not yet such a large progress as with linearly constrained ones, although several successful solution approaches exist. The solution of small to medium-scale general nonlinear problems is nowadays routine. There exist commercial solvers with dense linear algebra, e.g. in the libraries NAG and IMSL and some more powerful ones which already use sparse matrix technology, also capable to deal with some thousand variables at least, see e.g. the optimization software guide by Moré and Wright [Moré and Wright, 1993] and the nonlinear programming FAQ [Fourer, 2001]. There are even codes in the public domain which perform quite well [Mittelmann, 2000], [Mittelmann and Spellucci, 2000].

From the older approaches like penalty and the classical augmented Lagrangian methods there survived only the generalized reduced gradient method, e.g. the code `GRG2` of Lasdon (see e.g. [Fourer, 2001]). This is an active set method and can be considered as an adaptation of the simplex method to nonlinear constraints. Typically the problem is formulated as

$$f(x) \stackrel{!}{=} \min \ , \ h(x) \ = \ 0, \ x_{\mathcal{J}} \ \geq \ 0 \ .$$

by introducing positive slacks for the original inequality constraints. Given a feasible point x a direction of descent d is computed which satisfies $(\nabla h(x))^T d = 0$ and $d_i = 0$ if $x_i = 0$ for $i \in \mathcal{J}$. For a prospective move

$$x \ \mapsto \ x + \sigma d$$

a restoration value $r(x, d, \sigma)$ is computed such that

$$h(x + \sigma d + r(x, d, \sigma)) \ \equiv \ 0 \ , x + \sigma d + r(x, d, \sigma) \ \geq \ 0 \ .$$

Of course $r(x,d,\sigma) = \mathcal{O}(\sigma^2)$. The computation of r requires the solution of a nonlinear system of equations for every instance of σ. σ itself is computed using e.g. backtracking for f along that arc. Freedom exists in defining d, it may be obtained by solving

$$\begin{pmatrix} A & B \\ B^T & O \end{pmatrix} \begin{pmatrix} d \\ -\begin{pmatrix} \mu \\ \lambda_{\mathcal{A}} \end{pmatrix} \end{pmatrix} = \begin{pmatrix} -\nabla f(x) \\ 0 \end{pmatrix}$$

with the sole restriction of A being positive definite on the kernel of B^T. Here A stands for the Hessian of the Lagrangian or a replacement thereof and B is $(\nabla h(x), (e^i)_{i\in\mathcal{A}})$. In addition one needs some inactivation scheme for eliminating indices from $\mathcal{A}$ based on the signs of the current multiplier estimates. This type of method is reliable but too expensive due to the maintenance of feasibility and the necessity of providing a feasible point first, which requires some phase I like method. Therefore other approaches are currently in the center of the interest. Especially *SQP* methods, which can be interpreted as modified Newton methods for the system *KTC* combined with some globalization scheme (mostly via a so called merit function) have shown enormous success for medium-scale problems.

However, for large-scale problems (with more than some thousand variables and even larger ones) the solution approaches have not yet matured. One might discern five different approaches to tackle such large-scale problems:

i Modification of well established methods simply by introducing sparse matrix techniques into the linear algebra part.

ii Transformation into bound constrained problems with subsequent use of the specialized solution techniques for the latter.

iii Methods of linearization *SLP*.

iv New variants of the *SQP* method.

v Direct adaptations of interior-point methods.

In the first group we find the further development of the code `MINOS` by Saunders [Saunders, 1994], work of Gill, Murray and Saunders on `SNOPT` [Gill *et al.*, 1997] (a special instance of a *SQP*-method which evolved from `NPSOL`) and for example the *SQP* methods of Nickel and Tolle [Nickel and Tolle, 1989], Betts and Frank [Betts and Frank, 1994], Bartholomew-Biggs and Hernandez [Bartholomew-Biggs and Hernandez, 1994], Ni [Ni, 1993] and Biegler and Nocedal [Biegler *et al.*, 1995].

`MINOS` is very efficient if there are only few and only mildly nonlinear constraints. Here a sequence of nonlinear but linearly constrained optimization problems is solved, following Robinson's method [Robinson, 1972]. Globalization is obtained by the classical exterior quadratic penalty function. The linear constraints are dealt with using the elimination technique of the simplex method (indeed, MINOS is the long standing reference for an implementation of the simplex method.) If nonlinearity is more pronounced and there are many nonlinear constraints, this method is inferior to other approaches.

SNOPT takes advantage of eventual sparsity of the gradients of the constraints using sparse matrix methods for a factorization, which is used in the steps of a primal QP solver within a SQP method. Using this factorization a reduced Hessian is defined which is needed to solve the KTC like systems. It is normally a full matrix, hence either one restricts oneself to problems with a rather modest number of free variables (a case for which SNOPT was originally designed, a discretized control problem) or one resorts to the use of limited-memory quasi-Newton matrices.

Similarly Bartholomew-Biggs's approach is based on exploiting sparsity of constraint gradients and limited-memory approximations of the Hessian of the Lagrangian (this time the full matrix). However, as other users of the limited-memory approach too, the authors report about poor performance of this technique in the case of nonconvex or illconditioned testcases. This corresponds to the numerical experience of the present author.

The papers [Bartholomew-Biggs and Hernandez, 1995, Bartholomew-Biggs and Hernandez, 1994] describe how sparse-matrix techniques can be used directly in the kernel of an otherwise unchanged SQP-code, the QP solver. Since one cannot assume that the Hessian of the Lagrangian is positive definite the arising systems are not quasi-definite as in interior-point methods for convex QP's. The key role here play techniques for solving sparse symmetric indefinite systems, as developed by Duff, Reid and coworkers and gone into the codes MA27 and MA47 of the Harwell library. This technique also plays a key role in many other approaches. There is a disadvantage: the user must specify the nonzero patterns of the gradients and of the Hessian of the Lagrangian. If one uses a modeling system, this will be provided automatically, but otherwise it may be a prohibitively painstaking job. These systems have the form

$$\begin{pmatrix} A & N \\ N^T & 0 \end{pmatrix} \begin{pmatrix} d \\ w \end{pmatrix} = -\begin{pmatrix} g \\ c \end{pmatrix} .$$

A being sparse requires exact second derivatives or difference approximations thereof, since quasi-Newton updates will be full. Finite differences might be very costly, if possible at all. For problems of very high dimension the Bunch-Parlett decomposition is of little use because of the increased fill in involved with it and also because of numerical problems occuring if pivoting is limited. Iterative solvers for such systems are known "in principle", e.g. the classic SYMMLQ [Paige and Saunders, 1975], see also [Bjoerck, 1997]. However, due to problems with detecting rank deficiency and indefiniteness of the reduced Hessian they are not reliable. But there is an intense research in this area, see e.g. [Freund and Jarre, 1994], [Gould *et al.*, 1998]. The last cited work applies the conjugate gradient method to the system with the reduced Hessian without setting this up explicitly.

There is an even more problematic point. Indefinite *QP* problems which occur quite often in the *SQP* method if applied to nonconvex problems using the exact Hessian are regularized by a spectral shift, using a norm of the matrix or a similar crude eigenvalue bound. This is done for example by Franke [Franke, 1995] and by Conn, Gould and Toint in [Conn *et al.*, 1996], with obviously bad outcome.

Ni's approach is a further development of the ideas of Nickel and Tolle. Here the inverse of the Hessian of the Lagrangian resp. of a positive definite replacement thereof is approximated by a limited-memory quasi-Newton update. The *QP* subproblem is solved via its dual. Problems with an incompatible primal *QP* are circumvented by regularizing the dual such that it always has a bounded solution.

Biegler, Nocedal und Schmid [Biegler *et al.*, 1995] consider the equality constrained case only. In principle they use the standard *SQP* approach, however decomposing the correction explicitly into a "vertical" and a "horizontal" part. (The "vertical" part here and at other places concerns always a correction of infeasibility. This correction is typically in the range space of the gradients of the (active) constraints). The essential contribution of this paper is the demonstration how to compute this decomposition efficiently in the large-scale case. Numerical results are presented in the followup paper [Biegler *et al.*, 2000].

2.9.1 Transformation into an only bound constrained problem

Transformation into an only bound constrained problem is the basic idea of the `LANCELOT` project of Conn, Gould and Toint [Conn *et al.*, 1991]. One introduces slack variables into the general inequality constraints and obtains a problem with only equality constraints as general constraints and simple bound constraints as the only inequality constraints. For reasons of simple presentation we consider the case that all variables are bounded from below by zero.

$$(NLP:\ )\left\{\begin{array}{ccc} f(x) & = & \min \\ x & \geq & 0 \\ c(x) & = & 0 \end{array}\right. .$$

Then the `LANCELOT` (_large _and _nonlinearly _constrained _extended _Lagrangian _optimization _technique) approach is based on the following

Theorem 2.9.1 *Let x^* be a local solution of NLP with multipliers μ^* and λ^* for the equality and inequality constraints. Assume $\nabla c(x^*)$ being of full rank and $\nabla^2_{xx}L(x^*,\mu^*,\lambda^*)$ positive definite on the kernel of $\nabla c(x^*)^T$. Then there exists some $\gamma_0 > 0$ such that for $\gamma \geq \gamma_0$*

$$x(\mu) \stackrel{def}{=} argmin_x\{f(x) - (\mu)^T c(x) + \frac{\gamma}{2}||c(x)||^2 \ : \ x \geq 0\}$$

is well defined on $\mathcal{U}(\mu^*)$ as a function of μ and

$$\Psi(\mu) \stackrel{def}{=} f(x(\mu)) - (\mu)^T c(x(\mu)) + \frac{\gamma}{2}||c(x(\mu))||^2$$

posesses in μ^* an unconstrained strict local maximizer.

□

For given μ $x(\mu)$ will be computed by a bound constrained minimization and $\Psi(\mu)$ is simply the minimal value in this process. The function $\Psi(\mu)$ itself is maximized with respect to μ. It is of great help that $\nabla\Psi(\mu) = -c(x(\mu))$ and $\nabla^2\Psi(\mu) = -\frac{1}{\gamma}I + \mathcal{O}(\frac{1}{\gamma^2})$. This has the consequence that for sufficiently large γ even the simplest gradient method will converge quickly. All these relations hold for a general c indeed locally only, whereas for an originally convex problem the validity of the approach is global. There exist theoretical bounds for γ_0, but these cannot be evaluated in practice. They involve the singular values of ∇c and the eigenvalues of the Hessian and the reduced Hessian of the original Lagrangian function. This facts and the necessity to choose γ appropriately (large enough, but not too large, since otherwise the minimization will become troublesome) make the approach problematic. But for a long time this had been the only access to really large-scale problems. Meanwhile alternatives evolved, see below. If the required strong regularity assumptions are not satisfied, this will cause additional trouble, although a rank deficient ∇c doesn't hurt too much. The inner/outer iteration is expensive although the outer maximization converges fast usually. It is possible to increase the efficiency by using the problems structure introduced by the slacks. This is discussed for example in the paper [Conn *et al.*, 1997]. The weak points of the extended Lagrangian approach are discussed in the book [Spellucci, 1993a], section 3.5.2 and 3.5.3, and also in [Guddat *et al.*, 1997].

2.9.2 Linearization methods

The pure linearization methods (*SLP*-methods) use the fact that there exists a large choice of highly sophisticated software for *LP* problems which is capable of solving problems with up to millions of variables and constraints (sparsity assumed). One computes a direction of descent d^k for a given exact penalty function at a given guess x^k as a solution of the *LP*-problem

$$\begin{array}{rcl}
\nabla f(x^k)^T d + \gamma w &=& \min\limits_{d,w} \, , \\
-g(x^k) - \nabla g(x^k)^T d &\le& w \, , \\
-h(x^k) - \nabla h(x^k)^T d &\le& w \, , \\
h(x^k) + \nabla h(x^k)^T d &\le& w \, , \\
w &\ge& 0 \, , \\
-\rho_k \ \le d_i^k \ \le \rho_k \ , \ i = 1, \ldots, n \, . &&
\end{array}$$

Here w is a slack variable, which serves the purpose to make the linear constraints of this subproblem always compatible, such that the subproblem is solvable, due to the artificial compactness conditions on d^k. One aims in choosing γ sufficiently large, such that $w = 0$ is achieved whenever this is possible. ρ_k is the so called "trust-region radius", which is computed adaptively, depending on the descent obtained for the penalty function (in this case the

l_∞ exact penaltyfunction.) This method cannot converge fast, it corresponds to the ordinary gradient descent in unconstrained minimization. But there exists a lot of enhancements, see e.g. [Fletcher and Sainz de la Maza, 1989], [Jonasson and Madsen, 1994].

2.9.3 Modifications of the *SQP* method

The articles [Gould and Toint, 1999] and [Murray, 1997] describe existing possibilities and the second one also gives some review of existing software. We restrict ourselves here to a short review of the most promising approaches for which there is some numerical experience.

Franke [Franke, 1995] uses a classical *SQP* method, essentially Schittkowski's method [Schittkowski, 1983]. The emerging *QP* subproblem

$$\frac{1}{2}x^T Ax - a \stackrel{!}{=} \min\ ,\ B^T x - b \ = \ 0,\ C^T x - c \ \geq \ 0$$

is then solved by an interior-point method, considering the parametrized *KTC* system

$$\begin{aligned} Ax - a - B\mu - C\lambda &= 0\ , \\ B^T x - b &= 0\ , \\ w = C^T x - c &> 0\ , \\ \lambda &> 0\ , \\ \Lambda w - \epsilon e &= 0 \end{aligned}$$

with parameter ϵ, $\epsilon \to 0$. This nonlinear system is solved approximately via Newton's method. The resulting linear systems are solved by a modification of the sparse Bunch-Parlett code from the `MESCHACH`-library of Steward and Leyk (to be found as a tar-file in `netlib/cephes`). Here A represents the Hessian of the Lagrangian. If this matrix turns out to be indefinite, then Franke regularizes it as done in [Bartholomew-Biggs and Hernandez, 1995], [Conn *et al.*, 1996] using estimates of the eigenvalues provided by Gerschgorin discs. As is to be expected the numerical results are disappointing, if this occurs. As a further severe restriction Franke's approach misses a systematic treatment of inconsistent *QP*-Problems, which occur rather often in nonconvex cases. The approach has gone into a code `HQP` (see [Mittelmann and Spellucci, 2000]) which solved a bunch of highdimensional problems from the `CUTE` collection successfully.

Plantenga [Plantenga, 1994] first transforms a general *NLP* into an equality constrained one with additional bound constraints, using slack variables for the inequality constraints. For reasons of simple presentation we again assume that all variables are subject to the positivity constraints:

$$\begin{aligned} f(x) &\stackrel{!}{=} \min\ , \\ c(x) &= 0\ , \\ x &\geq 0\ . \end{aligned}$$

The solution of this problem is then approximated by the minimization of a barrier function subject to the equality constraints:

$$\begin{aligned} f(x) - \epsilon \sum_{i=1}^{n} \ln(x_i) &\stackrel{!}{=} \min \ , \\ c(x) &= 0 \ , \\ x &> 0 \ , \end{aligned}$$

that means the barrier technique here is used at the outer stage as opposed to Franke's approach, which uses this at the inner stage. The resulting equality constrained problem in turn is solved using the method of Byrd und Omojokun. This latter is a specialized method for equality constrained problems, a variant of Burke's method [Burke, 1992]. This is a trust-region method.

In every step two QP problems with an additional trust-region constraints have to be solved.

$$\begin{aligned} ||\nabla c(x^k)^T v + c(x^k)||_2^2 &\stackrel{!}{=} \min \ , && (9.1) \\ ||v||_2^2 &\leq \zeta \Delta_k^2 \ , \\ \nabla f(x^k)^T d + \frac{1}{2} d^T G_k d &\stackrel{!}{=} \min \ , && (9.2) \\ \nabla c(x^k)^T d &= \nabla c(x^k)^T v^k \ , && (9.3) \\ ||d||_2^2 &\leq \Delta_k^2 \ . \end{aligned}$$

Here v^k is the solution of (9.1). This is an example of the so called "vertical" correction: locally it points vertically to the boundary of the feasible set. This vertical correction serves the purpose of diminishing the infeasibility. The parameter ζ is chosen arbitrarily, but fixed in $]0,1[$. In (9.2), (9.3) d is decomposed into

$$d = v^k + Z_k u$$

where Z_k is a basis of the kernel of $\nabla c(x^k)^T$. Then the condition (9.3) is satisfied automatically. In order to compute such a basis for the problem (9.2) one uses sparse matrix techniques, e.g. the code `MA28` of the Harwell library. The trust-region radius Δ_k is computed adaptively as usual, checking descent for the exact penalty function

$$f(x) - \epsilon \sum_{i=1}^{n} \ln(x_i) + \gamma ||c(x)||_2$$

and ϵ is diminished in the outer iteration. The first version of Plantenga showed performance comparable with `LANCELOT`. A detailed exposition is to be found in the paper [Lalee *et al.*, 1998]. This method has been further developed by Hribar [Hribar, 1996].

A completely different approach is taken by Boggs and coworkers, [Boggs *et al.*, 1996], [Boggs *et al.*, 1994], [Boggs *et al.*, 1999]. The algorithm can be used for problems with mixed constraints, but like the authors in their papers we restrict ourselves here to the inequality constrained case, changing also the formulation into our standard form with constraints of the form $g(x) \geq 0$. Therefore the original problem now reads

$$f(x^I) \stackrel{!}{=} \min \ , g(x^I) \ \geq \ 0 \ .$$

After introducing slacks $x^{II} \geq 0$ the problem becomes formally one with equality constraints, the difference lying in the fact that the slacks are kept strictly positive.

$$x \ = \ (x^I, x^{II}) \ , c(x) = g(x^I) - x^{II} \ = \ 0 \ .$$

The backbone of the method is the classical *SQP* method with Fletchers differentiable penalty function as a merit function. This is the function

$$F(x) \ = \ f(x) - \mu(x)^T c(x) + \frac{\gamma}{2} ||A(x)^{1/2} c(x)||^2$$

where

$$\mu(x) \ = \ (\nabla c(x)^T \nabla c(x))^{-1} \nabla c(x)^T \nabla f(x) \ .$$

Since the *QP* solver used is an interior-point method which maintains strict positivity of the slacks (by means of the stepsize selection) , these can be eliminated from the computation. That means that no increase of dimension occurs here. Infeasible initial values are treated by a "big M" method, the *QP* problems being of the type

$$\begin{aligned} c^T d + \tfrac{1}{2} d^T A d + M\theta &\stackrel{!}{=} \min \ , \\ B^T d + b + \theta e &\geq 0 \ , \\ \theta &\geq 0 \ . \end{aligned}$$

If the algorithm is to be used with an equality constrained problem, then this "big M" phase persists throughout (since a strictly feasible point for the original problem is never found). During the stepsize algorithm the estimate $\mu(x)$ for the multipliers and the weight matrix $A(x)$ for the quadratic penalty term are frozen in order to reduce the evaluation effort for the Fletcher-function. Nevertheless every function value requires the solution of a linear system with the matrix

$$(\nabla g^T(x^k) \nabla g(x^k) + Z_k),$$

where g is the vector of inequality constraints and Z_k the diagonal matrix built from the values of the slack variables $x^{k,II}$. This matrix is always positive definite, but may be illconditioned. This introduces some trouble if the system must be solved iteratively. Furthermore the parameter γ in Fletchers function is not so easy to obtain, which produces even more trouble for nonconvex problems.

The *QP* solver itself is an inexact one and works with small dimensional subproblems of dimension up to three, which in turn are solved exactly. The basis of these subspaces is spanned by solutions of systems of the form

$$(\nabla g(x^k) Z_k^{-2} \nabla g^T(x^k) + Q_k/\beta_k) p_{i,k} \;=\; t_{i,k}, \; i = 1,2,3$$

where Q_k is an approximation for the Hessian of the Lagrangian, and β_k is chosen sufficiently large. These are systems of exactly the same structure as they occur when applying the classical barrier method for inequality constrained problems in connection with Newtons method. Should Q_k turn out to be not positive definite then also directions of negative curvature for the Lagrangian are used. The paper [Boggs *et al.*, 1999] contains an overview over numerical results obtained so far.

In the approach of Felkel and Spellucci the starting point again is a classical *SQP* method, this time using the nondifferentiable exact penalty function

$$\Phi(x;\gamma) \;=\; f(x) + \gamma ||h(x), g(x)^-||_\infty$$

for a problem with mixed constraints, with γ chosen adaptively. This function has the advantage of requiring minimal global conditions on the problem. Its clear disadvantage is its scale dependency, which requires adaptive and dynamic scaling of the problem. For this task a satisfactory practical solution was given already in [Heinz and Spellucci, 1994].

A direction of descent for this function is obtained from the *QP* problem

$$\begin{aligned}
\frac{1}{2} d^T B_k d + \nabla f(x^k)^T d + \gamma w + \frac{1}{2}\alpha w^2 \;&=\; \min \;, \\
-g(x^k) - \nabla g(x^k)^T d \;&\leq\; we \;, \\
-h(x^k) - \nabla h(x^k)^T d \;&\leq\; we \;, \\
h(x^k) + \nabla h(x^k)^T d \;&\leq\; we \;, \\
w \;&\geq\; 0 \;,
\end{aligned}$$

where w is a single slack variable which makes the problem compatible. As usual, $e = (1, \ldots, 1)^T$. But of course one wishes to obtain $w = 0$ which in turn guarantees that d^k is also a direction of descent for the penalty term alone. This is obtained by increasing γ cautiously. The *QP* subproblem is not solved exactly as in [Heinz and Spellucci, 1994], but approximately only minimizing a shifted logarithmic barrier function for this problem (observe that there are only inequality constraints due to the transformation of an equality constraint into two inequalites with slack), maintaining the constraint $w \geq 0$. For a restriction $r_i(d)$ of the *QP* problem the barrier term is

$$b(r_i, s, \eta) \;=\; \begin{cases} \ln(r_i + s) & \text{if } r_i(x) \;>\; -s\eta \\ \alpha_0 + \alpha_1 r_i + \alpha_2 r_i^2 + \alpha_3 r_i^3 & \text{otherwise} \end{cases} \;,$$

where the parameters are chosen such that C^3 continuity is satisfied. Here $s > 0$ is the shift, which must be chosen sufficiently small but must not tend to zero and η is a parameter. This avoids the singular behaviour at the solution, [Polyak, 1992].

This minimization is accomplished by an inexact Newton method based on the Lanczos algorithm. This *QP* solver has first been tested as a stand alone solver for *QP* problems with great success, see [Felkel, 1996]. In the current version of the *NLP* solver there are to be observed several inefficiencies. These occur if the matrices B_k are not sufficiently positive definite or if there is a large number of strongly nonlinear equality constraints. In the last case there occur steeply grooved and narrow valleys (due to the choice of a possible large γ and a small slack w), which make the method slow even if combined with second order corrections. Also the descent property of d^k is sometimes lost if the solution precision in the *QP* solver is relaxed. Improvements are a theme of current research.

The filter*SQP* method of Fletcher and Leyffer [Fletcher and Leyffer, 1997, Fletcher and Leyffer, 1998c, Fletcher and Leyffer, 1998b, Fletcher *et al.*, 1999] avoids the usage of a merit function at all and thereby the complications involved in chosing the penalty parameters appropriately. Rather it considers *NLP* as a special instance of vector optimization

$$\begin{pmatrix} f(x^*) \\ \theta(x^*) \end{pmatrix} \overset{!}{\leq} \begin{pmatrix} f(x) \\ \theta(x) \end{pmatrix} ,$$

where

$$\theta(x) \overset{def}{=} \max\{||h(x)||_\infty, ||(g(x))^-||_\infty\} .$$

Two types of moves are considered. If the trust-region *QP* problem

$$\begin{aligned} m_k(d) \quad \overset{def}{=} \quad & f(x^k) + \nabla f(x^k)^T d + \tfrac{1}{2} d^T A_k d \overset{!}{=} \min_d \\ & g(x^k) + \nabla g(x^k)^T d \geq 0 , \\ & h(x^k) + \nabla h(x^k)^T d = 0 , \\ & ||d|| \leq \Delta_k , \end{aligned}$$

with the trust-region radius Δ_k is solvable, then its solution is taken as a prospective move and its acceptability is checked, otherwise a restoration move is computed, aiming in diminishing θ. This might be e.g. a solution of

$$\begin{aligned} w &\overset{!}{=} \min \\ g(x^k) + \nabla g(x^k)^T d + we &\geq 0 , \\ h(x^k) + \nabla h(x^k)^T d + we &\geq 0 , \\ -h(x^k) - \nabla h(x^k)^T d + we &\geq 0 , \\ ||d|| &\leq \Delta_k . \end{aligned}$$

The acceptance of a move and the trust-region radius are controlled by a so called "filter". This is a set of values $(f(x^s), \theta(x^s))$ none of which is dominated by the others in the sense of the half order of $\mathbb{R}^2$ and with $\theta(x^s) \neq 0$. If the QP problem is infeasible, the corresponding values are always added to the filter and a move is made by a restoration step, aiming in computing some pair $(x^{k+1},\ \Delta_{k+1})$ such that the corresponding new QP problem becomes feasible. Otherwise the values at x^k are added to the filter if

$$m_k(0) - m_k(d^k) \ \le\ \kappa \theta_k^2$$

for some constant $\kappa > 0$. The point $x^{k+1} = x^k + d^k$ is accepted if

$$\theta(x^{k+1}) \ \le\ (1-\gamma_\theta)\theta_j \quad \text{or} \quad f(x^{k+1}) \ \le\ f_j - \gamma_\theta \theta_j$$

for all values in the filter and if in addition

$$\frac{f(x^k) - f(x^{k+1})}{m_k(0) - m_k(d^k)} \ \ge\ \eta_1 \ >\ 0\ .$$

In this case Δ_{k+1} can be increased, in all other cases Δ_{k+1} is decreased. γ_θ is chosen arbitrarily but fixed in $]0,1[$. After adding a pair (θ_k, f_k) to the filter this is purged by deleting all pairs with $\theta_j \ \ge\ \gamma_\theta \theta_k$ <u>and</u> $f_j \ \ge\ f_k - \gamma_\theta \theta_k$. This method is quite flexible, in principle it uses a merit function whose weights may change every step (via scalarization of the vector optimization problem). The code `filter` can be used via `NEOS`.

2.9.4 Adaptation of interior-point methods

The modified SQP methods described above result in multistage algorithms of inner/outer iteration type. Since the QP problems inherit the combinatorial nature of the original problem, their solution itself may be quite costly, especially in the large-scale case. Therefore there was much effort in avoiding this by using some more direct approach.

Conn, Gould and Toint [Conn *et al.*, 1996] describe a new access for minimizing general nonlinear functions subject to linear equality and bound constraints. Nonlinear constraints could be incorporated via an augented Lagrangian. The problem

$$\begin{aligned} f(x) &\overset{!}{=} \min_x\ , \\ B^T x &= b\ , \\ x &\ge 0 \end{aligned}$$

is embedded into a higher dimensional one

$$\begin{aligned} f(x) + \frac{1}{2}\rho(\xi+1)^2 &\overset{!}{=} \min_{x,\xi}\ , \\ B^T x - b &= \xi(B^T x^0 - b)\ , \\ x &\ge 0\ , \\ \xi &\ge 0\ , \end{aligned}$$

where the penalty parameter ρ is fixed in principle but has to be determined adaptively. The Kuhn-Tucker conditions for this extended problem become parametrized and the parametric systems are solved by Newton's method, with the Bunch-Parlett-decomposition as linear solver:

$$\begin{array}{rcl} \nabla f(x) - B\mu - \lambda & = & 0 \, , \\ -(B^T x^0 - b)^T \mu + \rho(\xi + 1) & = & 0 \, , \\ B^T x - b - \xi r^0 & = & 0 \, , \\ X\Lambda e - \epsilon e & = & 0 \, , \\ X, \Lambda & > & 0 \, . \end{array}$$

The logarithmic barrier function

$$f(x) + \frac{1}{2}\rho(\xi + 1)^2 - \epsilon \sum_{i=1}^{n} \ln(x_i)$$

serves as a merit function for stepsize control. In the case $\nabla^2 f(x)$ not positive definite it is regularized by the additive term $||\nabla^2 f(x)||I$. Numerical results are reported for large QP problems. In the convex case the results are very good, compared e.g. with Goulds direct method , code `VE09`, but for nonconvex cases they are completely disappointing, as is to be expected from this crude regularization. Divising a good adaptive scheme for the penalty parameter ρ is difficult. It depends besides others on the reciprocal value of the smallest positive eigenvalue of the indefinite matrix

$$\begin{pmatrix} \nabla^2 f(x) & B \\ B^T & O \end{pmatrix} .$$

There one also tried to make use of the classical barrier and penalty functions more directly using some modifications for large dimensional problems in the hope to take advantage of the progress made for highdimensional unconstrained minimization, e.g. Nash and Sofer [Nash and Sofer, 1993] or Shanno and Breitfeld [Breitfeld and Shanno, 1994]. The results obtained this way are not overly encouraging. The main reason for this lies in the fact that one loses too much of the "centering" when the penalty parameter is changed. Along the central trajectory the behaviour of these functions is much more amenable than a bit outside, especially if one uses quasi Newton or conjugate gradient type methods.

Forsgren and Gill [Forsgren and Gill, 1998] use the minimization of a mixed penalty-barrier function which was already described by Fiacco and McCormick. One of the problems in connection with this function is the fact that the theoretical multiplier estimates provided by these functions, e.g.

$$\frac{\rho}{g_i(x(\rho))} \approx \lambda_i$$

are quite unreliable in practice. The authors try to alleviate this effect by adding penalty terms for these deviations in a primal-dual framework:

$$\begin{aligned}\Psi(x,\mu,\lambda;\epsilon,\gamma) &= f(x)+\frac{1}{2\epsilon}\sum_{i=1}^{p}(c_i(x)^2+\gamma(c_i(x)+\epsilon\mu_i)^2)\\ &\quad -\epsilon\sum_{i=1}^{n}(\ln(x_i)+\gamma(\ln(\frac{x_i\lambda_i}{\epsilon})+1-\frac{x_i\lambda_i}{\epsilon}))\ .\end{aligned}$$

This function is minimized simultaneously with respect to x, μ and λ. ϵ is the path parameter and we have $\epsilon \to 0$ but $\gamma > 0$ fixed. For ϵ fixed minimization is performed by the damped Newton method, making use also of directions of negative curvature, which are available through the Bunch-Parlett decomposition of the Hessian. A special pivoting rule for this decomposition is at the heart of their algorithm. Thereby convergence to second order stationary points can be guaranteed. Numerical results are described in [Laux, 1999]. The outcome of Laux's experiments is a severe scale dependency of the chosen penalty function. Furthermore a good choice of the parameters involved is problematic and last not least, a point satisfying the inequality constraints must be known as an initial guess.

In the work [Byrd *et al.*, 1996] a method is investigated which treats only the slack variables by an interior-point approach. A general *NLP* problem

$$f(x) \stackrel{!}{=} \min\ ,\ c_E(x)\ =\ 0\ ,\ c_I(x)\ \geq\ 0$$

is transformed into

$$f(x) \stackrel{!}{=} \min\ ,\ c_E(x)\ =\ 0\ ,\ c_I(x)-s\ =\ 0\ ,\ s\ \geq\ 0\ ,$$

and this in turn is embedded in a family of equality constrained problems

$$\begin{aligned}f(x)+\epsilon\sum_{i=1}^{m}\ln(s_i) &\stackrel{!}{=} \min\ ,\\ c_E(x) &= 0\ ,\\ c_I(x)-s &= 0\ .\end{aligned}$$

For ϵ fixed this is an equality constrained nonlinear optimization problem which is solved by a *SQP* method. A trust-region approach is taken. It turns out that in constructing the trust-region the slack variables need a special treatment, namely using a norm like

$$\left|\left|\begin{pmatrix} d_x \\ \max\{\beta,\Delta\}S^{-1}d_s\end{pmatrix}\right|\right|\ \leq \Delta$$

where d_x designates the change in x and d_s the change in s and, as usual, $S = \mathrm{diag}(s_1,\ldots,s_I)$. β is a chosen constant. A quite general convergence theory is developed. These investigations are continued in the paper by Byrd, Hribar and Nocedal [Byrd *et al.*, 1999]. There comes up an essentially new aspect, namely the formulation of an adequate quadratic model for the Lagrangian of the barrier-problem with respect to the slack variable s. It turns out that the obvious choice

$$\mu S^{-2}$$

for the quadratic part corresponding to s is not favourable, since it gives directions which tend to violate feasibility of s and hence enforce a reduction of the step size. Much better seems to be the choice

$$S^{-1}\Lambda \ ,$$

where Λ designates the diagonal matrix with the multiplier estimates for the equality constraints $c_i(x) + s = 0$. These estimates are computed separately using the least squares estimate from the *KTC* conditions for x and s fixed. The paper contains detailed discussion how to solve the subproblems efficiently and approximately only. Promising numerical results are presented for the implementation `NITRO`. `NITRO` can be tested via the `NEOS` submission tool.

Gay, Overton and M. Wright [Gay *et al.*, 1998] consider a general *NLP* problem with mixed equality and inequality constraints. Different from their competitors they treat the inequality constraints directly by barrier terms. Hence

$$f(x) \stackrel{!}{=} \min \ , \ h(x) \ = \ 0 \ , \ g(x) \ \geq \ 0$$

is solved via

$$f(x) - \epsilon \sum_i \ln(g_i(x)) \stackrel{!}{=} \min \ , \ h(x) \ = \ 0$$

with $\epsilon \to 0$. The parameter dependent *KTC* conditions

$$\begin{aligned} \nabla h(x)y + \nabla g(x)z - \nabla f(x) &= 0 \ , \\ h(x) &= 0 \ , \\ G(x)z - \epsilon e &= 0 \ , \ G = \operatorname{diag}(g_i(x)) \end{aligned}$$

are solved by Newtons method, after symmetrization and partial elimination:

$$\begin{pmatrix} -K & A \\ A^T & O \end{pmatrix} \begin{pmatrix} d_x \\ d_y \end{pmatrix} = \begin{pmatrix} \nabla f - Ay - \epsilon B^T G^{-1} e \\ -h \end{pmatrix} \text{ with } K \ = \ H + B^T G^{-1} Z B \ .$$

An indefinite K becomes regularized to definiteness, but such that its eigenvalues tending to infinity are not touched. Two merit functions are used, namely the norm of the right hand side of the Newton equation and periodically also some type of extended Lagrangian

$$f(x) - \epsilon \sum_i \ln g_i(x) - y^T h(x) + \frac{c}{2}||h(x)||^2 \ ,$$

(with c finite and in principle fixed). This device serves the purpose to prevent convergence of Newtons method to saddle points or maxima. Numerical results for low dimensional problems are reported. No complete convergence analysis seems to exist.

Vanberbei and Shanno [Vanderbei and Shanno, 1997] make use of the powerful *QP* solver `LOQO` of Vanderbei. It serves at the heart of a *SQP* code. Here too inequality constraints $g(x) \geq 0$ are transformed into equations using positive slacks which in turn are treated by the logarithmic barrier term. The method computes directions of descent for the inexact penalty barrier function

$$f(x) - \epsilon \sum_{i=1}^{m} \ln(s_i) + \frac{\beta}{2} ||g(x) - s||^2$$

from the reduced system

$$\begin{pmatrix} H & O & -A \\ O & -S^{-1}\Lambda & I \\ -A^T & I & O \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta s \\ \Delta \lambda \end{pmatrix} = \begin{pmatrix} -\nabla f + A\lambda \\ \epsilon S^{-1}e - \lambda \\ g(x) - s \end{pmatrix} ,$$

where $A = \nabla g(x)$ and $H = \nabla^2_{xx} L$. This descent property is obtained for β sufficiently large. Indefinite H is regularized by addition of a suitable multiple of the unit matrix. ϵ is chosen dependent on the duality gap:

$$\epsilon = \gamma \min\{(1-r)\frac{1-\xi}{\xi}, 2\}^3 \frac{s^T\lambda}{I}$$

with

$$0 < r < 1 , \quad \xi = \frac{\min_i s_i \lambda_i}{s^T\lambda/I} .$$

(Remember that I is the number of inequality constraints). In principle we must have $\beta \to \infty$, but since in practice one contents oneself with an approximate solution anyway the authors consider this fact as not so critical. The paper also describes a special treatment of bound- and interval constraints such that only general inequalities must be transformed by slacks. Equality constraints are dealt with directly by a quadratic penalty term. The paper also reports numerical experience. `LOQO` can also be used through `NEOS`.

Akrotirianakis and Rustem [Akrotirianakis and Rustem, 1997] describe an approach similar to the one given by Vanderbei und Shanno. Starting point is a *NLP* problem

$$f(x) \stackrel{!}{=} \min, \; g(x) = 0 , \; x \geq 0 .$$

As a merit function serves the mixed penalty-barrier function

$$\Phi(x;c;\epsilon) = f(x) - \epsilon \sum_{i=1}^{n} \ln(x_i) + \frac{c}{2} ||g(x)||^2$$

subject to the equality constraints

$$g(x) \;=\; 0 \;,$$

that means the equality constraints are treated in dublicate: they enter the merit function and are posed explicitly. The barrier parameter ϵ is controlled in an outer and the penalty parameter c in an inner iteration. For ϵ and c fixed the directions of descent come from a Newton step for a zero of the function

$$F(x,y,z;c,\epsilon) \;=\; \begin{pmatrix} \nabla f(x) - z + c(\nabla g(x))g(x) - \nabla g(x)y \\ g(x) \\ XZe - \epsilon e \end{pmatrix}$$

Here y represents the vector of Lagrange multipliers corresponding to the equality constraints and z is obtained as a new free variable from the relation

$$z \;=\; \epsilon X^{-1}e \;.$$

As usual $e = (1,\ldots,1)^T$. z corresponds to the Lagrange multipliers of the bounds. For sufficiently large c the descent property of this direction can be guaranteed and the adaptive computation of c is based on that fact. The Jacobian of F has as left upper block

$$H \;=\; \nabla^2 f(x) - y \times \nabla^2 g(x) + c\nabla g(x)(\nabla g(x))^T + cg(x) \times \nabla^2 g(x)$$

$H + X^{-1}Z$ must be invertible in order to have a well defined direction d. This is of course a restrictive condition, which also occurs in other work. Even stronger, uniform boundedness and uniform positive definiteness is assumed for this matrix in the convergence analysis. Different stepsizes are used for the primal and the dual variables. For a special version of the algorithm there is a proof of convergence. No numerical results are presented. The proof of the uniform boundedness of c needed for that seems to be incomplete.

2.9.5 Homotopy methods for the KTC-system

Whereas the work of the previous section results in some inner/outer iteration scheme with the possibility to choose subalgorithms independently it is also possible to use the primal-dual interior-point formulation known from convex *LP* and *QP* directly for a general *NLP*. This is shown in the interesting paper of El Bakry et. al. in [El-Bakry *et al.*, 1996]. These investigations have been continued by Durazzi [Durazzi, 2000]. The procedure is best interpreted as a transfer of the Kojima, Mizuno and Yoshise method from the *LP* to the the *NLP* case. As is known that one results in the *LP* case to the most efficient solvers presently known. Starting point is a general problem with mixed constraints

$h(x) = 0$ and $g(x) \geq 0$. The trajectory defined by the system

$$\begin{aligned}\nabla f(x) - \nabla h(x)\mu - \nabla g(x)\lambda &= 0\ ,\\ \lambda - z &= 0\ ,\\ h(x) &= 0\ ,\\ g(x) - s &= 0\ ,\\ ZSe - \epsilon e &= 0\ ,\end{aligned}$$

formally written as $(x, \mu, \lambda, z, s)(\epsilon)$ with $\lambda, z, s > 0$ is traced with $\epsilon \rightarrow 0$, the nonlinear system in turn being approximately solved by Newton's method. The Jacobian of this system can be symmetrized and once again one has to solve a large symmetric indefinite linear system every step. Some numerical evidence with small-scale problems is provided. There are some critical points. On point is the large number of artificial unknowns (a total of $n + 3m + p$, if m is the dimension of g and p that of h). The structure of the linear system is much less amenable than in the *LP* case. The convergence conditions are quite strong and seem to restrict the treatment to convex problems. Among other things one needs regularity of the matrix

$$\nabla^2_{xx} L(x, \mu, \lambda) + \nabla g(x) S^{-1} Z \nabla g(x)^T$$

for all values of x, μ, λ, z, s in a neighborhood of the trajectory.

A quite similar approach is followed by Epelly, Gondzio and Vial [Epelly *et al.*, 2000] who restrict themselves to the convex case from the very beginning.

2.10 Some numerical results

In the following we give an excerpt of the results obtained by the `COPS` project [Dolan and Moré, 2000]. The test comprises four solvers, `LANCELOT, LOQO, MINOS` and `SNOPT` in their newest version (2000). Default options were used for all codes. The problems are formulated in `AMPL`. Most of them are discretized control problems. There are also three large *QP* problems in the testsuite. The evaluation was done on a SUN ULTRA SPARC 2 under Solaris 7. We give here a part of the results only, for the lowest and the highest dimension tested. The table gives computing time in seconds and the optimal value of the objective function. "F" denotes a failure (either premature termination or time out). The numbers `x-y-z` give the number of variables, of the contraints and of the bound constraints.

problem/solver	LANCELOT	LOQO	MINOS	SNOPT
largest small polygon	12.83	3.49	2.11	1.14
50-49-50	7.9715e-01	7.64714e-01	7.64383e-01	7.79740e-01
largest small polygon	F	F	344.81	256.6
200-199-200			7.72803e-01	7.85040e-01
electron on sphere	3.98	0.84	6.22	9.65
75-25-0	2.43812e+02	2.43812e+02	2.43812e+02	2.43812e+02
electron on sphere	371.6	2437.78	F	1600.48
600-200-0	1.84389e+04	1.84389e+04		1.84390e+04
hanging chain	13.91	17.24	1.22	5.72
100-51-0	5.07230e+00	5.07226e+00	5.07226e+00	5.07226e+00
hanging chain	577.24	1028.35	73.9	F
800-401-0	5.06788e+00	5.06862e+00	5.06862e+00	
shape of cam	42.74	0.68	0.87	0.6
100-202-100	4.30178e+00(F)	4.28414e+00	4.28414e+00	4.28414e+00
shape of cam	1887.03	12.47	21.72	25.17
800-1602-800	4.85693e+00(F)	4.27427e+00	4.27426e+00	4.23739e+00
alpha-pinene	1426.01	28.85	1.98	3.74
505-500-5	1.96766e+01	1.98715e+01	1.98715e+01	1.98715e+01
alpha-pinene	F	16.87	F	235.44
4005-4000-5		1.98721e+01		1.98721e+01
marine population	623.75	2.07	6.58	85.37
615-592-15	1.97522e+07	1.97522e+07	1.97522e+07	1.97522e+07
marine population	F	38.4	F	1502.26
4815-4792-15		1.97465e+07		1.97465e+07
flow in channel	F	1.55	1.09	2.14
400-400-0		1.00000	1.00000	1.00000
flow in channel	F	22.54	32.65	98.5
3200-3200-0		1.00000	1.00000	1.00000
robot arm	F	1.03	2.82	10.22
460-300-306		9.14687	9.14687	9.14687
robot arm	F	F	161.18	2671.63
3610-2400-2406			9.14108	9.14101
particle steering	30.83	923.3	1.58	3.25
256-200-51	5.54672e-01	5.54668e-01	5.54668e-01	5.54668e-01
particle steering	2997.88	F	143.09	147.37
2006-1600-401	5.54552e-01		5.54572e-01	5.54573e-01

problem/solver	LANCELOT	LOQO	MINOS	SNOPT
goddar rocket	F	3.34	1.69	3.04
205-150-153		1.01281	1.01280	1.01282
goddar rocket	F	12.42	F	64.48
1605-1200-1203		1.01283	F	1.01283
hang glider	F	F	F	11.14
256-200-153				1.28239e+03
hang glider	F	F	F	1268.67
2006-1601-1203				1.24797e+03
cracking oil	918.28	1.37	5.04	5.25
503-500-3	5.23633e-03	5.23664e-03	5.23664e-03	5.23664e-03
cracking oil	F	49.62	161.99	179.71
4003-4000-3		5.23659e-03	5.23659e-03	5.23659e-03
methanol to hydrocarbon	196.62	2.13	5.05	12.92
605-600-5	9.02300e-03	9.02229e-03	9.02228e-03	9.02228e-03
methanol to hydrocarbon	F	45.2	263.67	512.16
4805-4800-5		9.02229e-03	9.02228e-03	9.02228e-03
catalyst mixing	7.71	0.66	2.39	3.99
303-200-101	-4.77480e-02	-4.80694e-02	-4.80605e-02	-4.80579e-02
catalyst mixing	424.61	8.25	17.88	181.24
2403-1600-801	-4.71856e-02	-4.80559e-02	-4.74787e-02	-4.80451e-02
elastic plastic torsion	3.01	2.99	108.31	125.58
1250-0-1250	-4.17510e-01	-4.17510e-01	-4.17510e-01	-4.17510e-01
elastic plastic torsion	17.19	15.86	F	F
5000-0-5000	-4.18239e-01	-4.18239e-01		
journal bearing	3.02	3.36	173.65	722.68
1250-0-1250	-1.54015e-01	-1.54015e-01	-1.54015e-01	-1.54015e-01
journal bearing	17.89	13.33	F	F
5000-0-5000	-1.55042e-01	-1.55042e-01		
obstacle	2.77	2.98	103.76	137.68
1250-0-1250	2.51948	2.51948	2.51948	2.51948
obstacle	16.33	F	F	F
5000-0-5000	2.50694e+00			

Besides these tests there are also many interesting benchmarks provided by H.D.Mittelmann [Mittelmann, 2000], showing the same effect: there is no clear cut winner and reliability of the codes is far beyond the level meanwhile reached for small- and medium-scale problems. Hence much remains to be done.

2.11 Conclusion

The great impact of interior-point methods on present optimization technology can clearly be seen from the discussion above. Summarizing it can be safely said that convex problems, be it *QP* or more general *NLP* problems, can be solved by a variety of methods with good success, even in the large-scale case. As T. Rockafellar states: "The great watershed in optimization isn't between linearity and nonlinearity but convexity and nonconvexity" [Rockafellar, 1993]. First of all we must be prepared to accept a local solution, possibly much weaker than the desired global one. Which of the possibly many local solutions is identified is subject to details of the algorithm's implementation and the initial guess and it is almost impossible to take influence on that. For a nonconvex problem often some kind of restoration steps must be used. This typically requires the linear independence of the gradients of the violated constraints or a least the selection of a subset of constraints which satisfy this condition. In the small-scale area the QR or the SVD decomposition of the matrix of these gradients helps to do that. This technique cannot be used for large-scale applications because the enormous fill in produced by orthogonal transformations. Other linear algebra techniques are much less reliable. The detection of infeasibility cannot be based on the unboundedness of the dual variables. The detection of nonconvexity via detection of indefiniteness of the Hessian of the Lagrangian is an unreliable process, especially for large scale applications. How to regularize a nonconvex problem, especially in connection with interior-point methods, is an open problem. The crude technique of addition of a multiple of the unit matrix is clearly inadequate. Hence much remains to be done in order to increase the reliability and efficiency of existing methods.

Chapter 3

Automatic Differentiation

Christian Bliek
ILOG
Sophia Antipolis
France

3.1 Introduction

The solution of constraint satisfaction and optimization problems usually requires the evaluation partial derivatives or Hessians. Traditional nonlinear programming algorithms (see Chapter 2) usually need to evaluate or approximate the gradient and/or Hessian at a given point in every iteration. Complete nonlinear solvers based on constraint satisfaction or interval arithmetic (see Chapter 4 and Chapter 5) also need gradient and/or Hessian information. In this case the desired information may not only be required at a given point but also over a given box.

Automatic differentiation is related to but different from symbolic differentiation. With symbolic differentiation explicit formulas for the derivatives are produced. To compute values for the derivatives, these explicit formulas are evaluated. With automatic differentiation, no explicit formula for the derivatives is ever produced. To compute values for the derivatives differentiation rules are used interleaved with a numerical evaluation process. The the advantage of automatic differentiation over symbolic differentiation followed by numerical evaluation is that information related to common subexpressions can be shared.

The basic assumption of automatic differentiation is that the functions to be differentiated can be expressed by a sequence of unary or binary arithmetic operators. More complicated operators can be included provided local partial derivative and Hessian information of the operators can be provided. In the context of this project we will focus on functions that can expressed under the form of arithmetic expressions.

3.2 Computation of Partial Derivatives

3.2.1 Forward Mode

As described in [Rall, 1981], automatic computation of partial derivatives is usually done recursively. In addition to the values of the expressions calculated at each node one also computes the gradient of the expression with respect to the input variables. For the multiplication one would for example compute:

$$\begin{aligned} z &= u \cdot v \\ \nabla z &= \nabla u \cdot v + u \cdot \nabla v \end{aligned}$$

Note that in general ∇z is a vector of length n, where n is the number of independent variables in the expression. As one can see this bottom up approach unfortunately has a complexity of $\mathcal{O}(Fn)$ where F is the complexity of one function evaluation.

3.2.2 Backward Mode

The forward computation of partial derivatives is not as efficient as it could be, indeed by performing operations on vectors intermediate results are not shared. This consideration led to the more efficient backward, top down, computation of partial derivatives [Speelpenning, 1980].

The forward mode computes the partial derivative $\frac{\partial w}{\partial x_i}$ of an intermediate result w with respect to the independent variables x_i. It then proceeds in a bottom up fashion to finally compute $\frac{\partial f}{\partial x_i}$. The backward mode computes the partial derivative $\frac{\partial f}{\partial w}$ where f is the function to be computed. It then proceeds in a top down fashion to finally compute $\frac{\partial f}{\partial x_i}$. This is illustrated in figure 3.1. The process of backward differentiation corresponds to a particular way of using

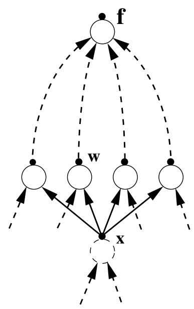


Figure 3.1: Differentiation in Reverse Mode

the chain rule

$$\frac{\partial f}{\partial x} = \sum_{w \in N_G^+(x)} \frac{\partial f}{\partial w} \frac{\partial w}{\partial x}$$

We use the notation $N_G^+(x)$ for the nodes in G that depend on x.

Note that the differentiation process requires that the values of the intermediate quantities be available. In the forward mode the evaluation and differentiation could be done simultaneously resulting in a single forward pass algorithm. However for the backward mode we need a first forward pass to perform the evaluation. This is followed by a pass in reverse mode to compute the derivatives. The process is illustrated in figure 3.2.

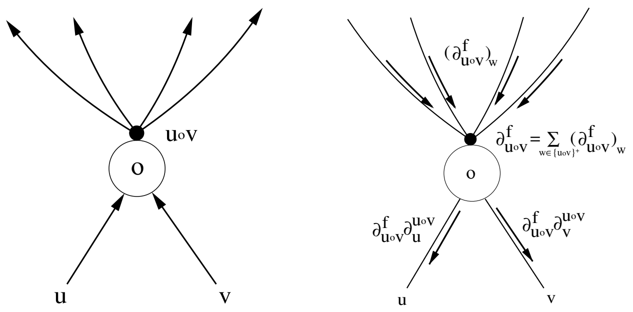


Figure 3.2: Evaluation pass and Backward Differentiation Pass

For the 4 basic arithmetic operators, the local derivatives are:

$$\begin{array}{rclrcl}
\frac{\partial u+v}{\partial u} & = & 1 & \frac{\partial u+v}{\partial v} & = & 1 \\
\frac{\partial u-v}{\partial u} & = & 1 & \frac{\partial u-v}{\partial v} & = & -1 \\
\frac{\partial u*v}{\partial u} & = & v & \frac{\partial u*v}{\partial v} & = & u \\
\frac{\partial u/v}{\partial u} & = & 1/v & \frac{\partial u/v}{\partial v} & = & -u/v^2
\end{array}$$

The backward differentiation algorithm is illustrated in figure 3.3.

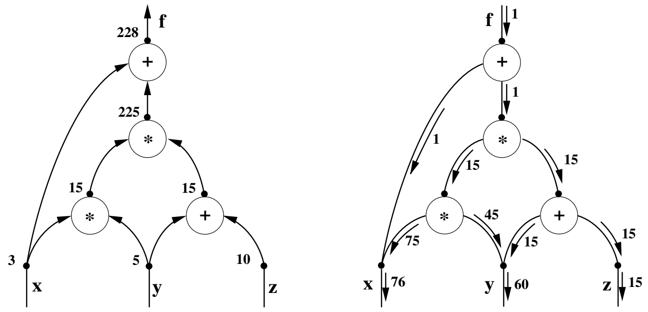


Figure 3.3: Example of Backward Differentiation Algorithm

The complexity of evaluating both the function value and the partial derivatives reduces to CF, where the constant factor C is 3 when all operators are at most binary [Baur and Strassen, 1983]. Iri [Iri, 1984] analyzes this algorithm from the perspective of graph theory, and shows how the value of C depends on the different measures of complexity. A comparison between the different approaches together with some practical applications of this technique can be found in [Griewank, 1989, Griewank, 1991].

3.2.3 Mixed Mode

Backward differentiation can provide a significant speedup for the computation of a single function with many independent variables. As more functions are computed by a single computational graph the speedup decreases, because each function requires its backward evaluation of derivatives. When many functions are provided by a common computational graph, but there is only one variable, forward evaluation is generally more efficient than backward evaluation. Indeed, in this case the situation is reversed.

It turns out that for differentiation of computational graphs that compute many functions from many independent variables that both the forward and the backward mode are outperformed by mixed mode strategies [Griewank and Reese, 1991, Bliek, 1992].

Derivative graphs allow for general computation model for partial derivatives. Where previously, nodes corresponded with arithmetic operators, they are now associated with intermediate variables. This mapping is illustrated in figure 3.4. In addition we label each arc (x, y) with the local partial derivative

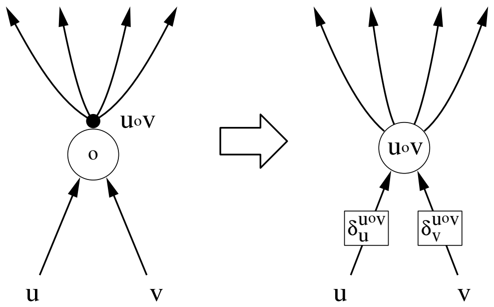


Figure 3.4: Mapping from Computational to Derivative Graphs

$\frac{\partial y}{\partial x}$ of y w.r.t. x Mathematically, derivative graphs express the relation:

$$dy = \sum_{x \in N_G^-(y)} \frac{\partial y}{\partial x} dx \tag{2.1}$$

$N_G^-(y)$ denotes the nodes in G on which y depends.

Condensation is the process of removing a node from a graph, and replacing its arcs with new arcs connecting its negative to its positive neighbors, unless they already are part of the graph. The label of a new arc is the product of the labels of the arcs of the eliminated path. For an existing arc its label is incremented by the same product. The condensation process is illustrated in

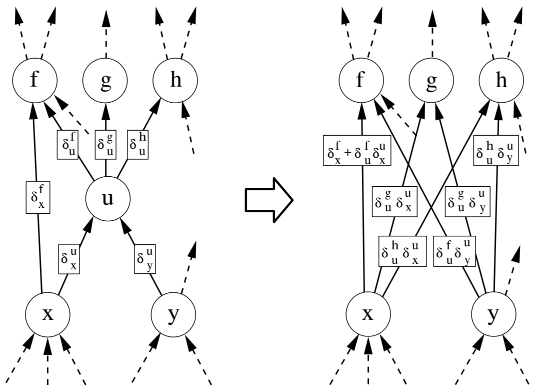


Figure 3.5: Node Condensation

figure 3.5. Algebraically, condensation of node u corresponds to the elimination of du from the set of equations (2.1) for $y \in N_G^+(u)$.

The graph obtained after condensation of a set of nodes is independent of the condensation order of these nodes [Rose and Tarjan, 1978]. When all nodes internal to the derivative graph are condensed, the labels of the remaining arcs contain the partial derivatives of the functions with respect to the independent variables. The computation of partial derivatives can therefore be viewed as the condensation of the internal nodes in the derivative graph.

One estimate of the cost to condense a node is the count of the number of multiplications required to perform the condensation. It turns out that the total cost to condense the nodes internal to the derivative graph depends on the order in which the nodes are condensed. The determination of the optimal condensation order is conjectured to be NP-complete. As a result a number of heuristics have been proposed to determine good condensation orders. One can for example use the Markowitz rule where one first condenses the nodes requiring the least number of multiplications [Griewank and Reese, 1991, Bliek, 1992]. A more robust approach is to use lookahead strategies in which one also takes into account an estimation of the cost remaining after condensation of a node [Bliek, 1992]. In cases where the computational cost of the derivatives is very important, it may be worthwhile to resort to techniques that attempt to find near-optimal or optimal condensation strategies. We refer the reader to [Bliek, 1992] and [Bischof and Haghighat, 1996] for a description of such techniques.

3.2.4 Improvements

The backward and the mixed modes require that the function values be available when the partial derivatives are computed. This requires the execution of a forward function evaluation pass before the partial derivatives are computed and the storage of the corresponding intermediate results. The drawback of this approach is the memory required to store these results is proportional to time required to compute the function values. In certain cases this memory consumption may be excessive.

One approach to address this problem is to trade time for space. Instead of storing the computed values for all intermediate quantities, one only stores the values at given time instances called "snapshots". To obtain the function values during the reverse pass, instead of looking them up, one recomputes them from the latest snapshot. By applying this approach recursively and taking snapshots halfway through the computation segments, one can reduce the memory consumption from $\mathcal{O}(F)$ to $\mathcal{O}(\log F)$ while increasing the running time from $\mathcal{O}(F)$ to $\mathcal{O}(F \log F)$ [Griewank, 1992].

Unlike the backward and mixed modes, the forward mode does not have excessive memory requirements. However, as indicated above, its time complexity $\mathcal{O}(Fn)$ is not as good as compared to the backward mode. It turns out that this complexity can be improved to $\mathcal{O}(F \log n)$ when the graph is a Kantorovic Tree [Ulbrich and Ulbrich, 1996]. Kantorovic trees are graphs where all intermediate results are used exactly once with the exception of the input variables which may be used more than once. Note that arithmetic expressions in which intermediate quantities cannot be referenced are Kantorovic trees. In practice it is often the case that expressions are of this type. To achieve this complexity the operations performed in the forward mode need to be carefully designed to exploit sparsity. We refer the reader to [Ulbrich and Ulbrich, 1996] for more details.

3.3 Interval Derivatives and Interval Slopes

The techniques presented in section 3.2 can be applied to compute derivatives at given points. In this case the values for the variables and for the seed values for the derivatives are floating point numbers. As a result, the computed values will be exact up to machine precision. However, one can also use these techniques to compute interval derivatives. In this case the values for the variables are intervals and the seed values for the derivatives are point intervals. The computed values will now be intervals that enclose the range of the derivatives over the given box.

The automatic differentiation techniques presented in this chapter are based on the assumption that the chain rule holds. One is therefore not limited to the computation of partial derivatives. In [Bliek, 1992, Bliek, 1997] it is shown that the chain rule also holds for Interval Slopes. As a result one can use automatic differentiation algorithms to compute them. In this case the evaluation part of algorithms needs to include interval evaluation as well as evaluation of the centers.

3.4 Computation of Hessians

The computation of the Hessian in forward mode is similar than the computation of the gradient. In addition to the gradient:

$$\nabla z = \frac{\partial z}{\partial x}\nabla x + \frac{\partial z}{\partial y}\nabla y$$

one now also computes the Hessian of the current intermediate result z with respect to the independent variables:

$$\nabla^2 z = \begin{pmatrix} \nabla x^T & \nabla y^T \end{pmatrix} \begin{pmatrix} \frac{\partial^2 z}{\partial^2 x} & \frac{\partial^2 z}{\partial x \partial y} \\ \frac{\partial^2 z}{\partial y \partial x} & \frac{\partial^2 z}{\partial^2 y} \end{pmatrix} \begin{pmatrix} \nabla x \\ \nabla y \end{pmatrix}$$

In this case we not only perform vector operations at each node, but also matrix operations. The cost of the operations per node therefore increases from $\mathcal{O}(n)$ to $\mathcal{O}(n^2)$. The overall cost of computing the Hessian in forward mode therefore is $\mathcal{O}(Fn^2)$.

The improvements obtained by using backward differentiation can be carried over for the computation of Hessians [Iri, 1984]. To compute partial derivatives of a function, one normally executes the reverse pass over the original computational graph of the function. However, one also could explicitly build the computational graph that corresponds with this computation. The total size of the resulting graph is approximately 3 times the one of the original graph. Given this extended graph that computes the derivatives $\frac{\partial f}{\partial x_i}$, one can now use the reverse mode n times to calculate the entries $\frac{\partial^2 f}{\partial x_i \partial x_j}$ of the Hessian matrix. The complexity of this procedure is thus $\mathcal{O}(Fn)$.

We indicated above that for Kantorovic graphs the complexity of the forward mode can be reduced to $\mathcal{O}(F \log n)$. The same technique can be used to improve the computation of the Hessian on Kantorovic graphs. In this case the complexity becomes $\mathcal{O}(Fn \log n)$ [Ulbrich and Ulbrich, 1996].

3.5 Sparse Matrix Techniques

A number of sparse matrix techniques have been devised to optimize the computation of finite differences. As indicated in [Coleman and Verma, 1998] these techniques can also be used to improve the efficiency of automatic differentiation techniques. Let us describe the idea for the computation of $\nabla^2 f$. When the matrix $\nabla^2 f$ is sparse and the sparsity pattern is known, then for an appropriate choice of q from the vector $\nabla^2 f q$ one can determine more than one column of $\nabla^2 f$. Suppose that we have the computational graph to determine J as described above. In this case we can compute $\nabla^2 f q$ in $\mathcal{O}(F)$ time using the forward mode by setting the *seed* vector $\frac{\partial x_j}{\partial x_j} = q_j$. To choose the appropriate vectors q graph coloring techniques can be used. If k vectors are sufficient to determine $\nabla^2 f$, then the computational complexity is Fk instead of Fn. Instead of relying on column computations, one can use row computations. In that case we would compute $p^T \nabla^2 f$ using the backward mode by setting the seed vector $\partial \frac{\partial f}{\partial x_i} / \partial \frac{\partial f}{\partial x_i} = p_i$. We refer the reader to [Coleman and Verma, 1998] for a more detailed treatment.

The method above is recommended to exploit sparsity when only the forward or the reverse mode are available. We believe that the performance of mixed mode algorithms should be comparable to above technique.

Note that some nonlinear programming techniques do not need to compute the full Jacobian matrix explicitly. Inexact Newton techniques for example only need to evaluate Jq. As explained above, this evaluation can be performed in $\mathcal{O}(F)$ time by seeding the input vector with q.

3.6 Conclusion

The advent of efficient automatic differentiation algorithms has had an important impact on nonlinear local optimization algorithms (see Chapter 2). Indeed, the computation of the derivatives is no longer a costly operation. As a result algorithms tend to employ exact derivative information more frequently. Automatic differentiation has also had a profound impact on Interval Arithmetic and Constraint Satisfaction Algorithms (see Chapter 4 and Chapter 5). Almost all state of the art algorithms in these areas resort to automatic differentiation.

Most efforts in the area of automatic differentiation have focused on computational efficiency. This concern is justified for nonlinear local optimization algorithms. However, when automatic differentiation is used for computing interval derivatives and interval slopes, computational efficiency is not the only concern. In this case the overestimation made by a particular algorithm is also very important. In our opinion this aspect deserves more attention.

Finally, we believe that the computation of Hessians may benefit from mixed mode automatic differentiation algorithms. Indeed, to compute the Hessian one could start from the computational graph extended to include the derivative computation. This graph is a multiple-input, multiple-output graph on which mixed mode strategies are expected to work well. We are not aware of any works that study the performance of mixed mode strategies for computing the Hessian.

Chapter 4

Constrained Global Optimization

Arnold Neumaier *Institut für Mathematik*
Universität Wien
Strudlhofgasse 4, A-1090 Wien
Austria

This chapter covers the state of the art of general purpose constrained global optimization, with emphasis on systematic techniques that are provably convergent.Since constraint propagation techniques are more thoroughly covered in Chapter 6, they are treated here only superficially.

4.1 Introduction

Global optimization is the task of finding the absolutely best set of admissible conditions to achieve an objective under given constraints, assuming that both are formulated in mathematical terms. It is much more difficult than convex programming or finding local minimizers of nonlinear programs, since the gap between the necessary (Karush-Kuhn-Tucker) conditions for optimality and known sufficient conditions for global optimality is tremendous.

Many famous hard optimization problems, such as the traveling salesman problem or the protein folding problem, are global optimization problems. The truth of the famous unresolved conjecture $P \neq NP$ [Garey and Johnson, 1979] would imply [Murty and Kabadi, 1987, Pardalos and Schnitger, 1988] that there are no general algorithms that solve a given global optimization problem in time polynomial in the problem description length. However, some large-scale global optimization problems have been solved by current methods, and a number of software packages are available that reliably solve most global optimization problems in small (and sometimes larger) dimensions. The Global Optimization WWW-site at

`http://solon.cma.univie.ac.at/~neum/glopt.html`

contains many links to online information about the subject.

In this course, the audience will be introduced to theory and techniques that form the backbone of these packages. Much of what is presented here is part of work done in the context of the COCONUT project [coconut, 2001] sponsored by the European Union, with the goal of integrating various existing systematic approaches into a uniform whole. Among the topics (and packages) discussed are:

- Branching methods using local information only (DIRECT, MCS, LGO)
- Branch and bound methods (BARON, αBB)
- Interval methods (GLOBSOL)
- Constraint satisfaction techniques (NUMERICA)
- Outer approximation and underestimation techniques
- Separable and semiseparable programs
- Certificates of infeasibility
- Sufficient conditions for global minimizers

Deliberately excluded are methods that are specific to special problem classes such as distance geometry or protein folding [Neumaier, 1997], methods specific to combinatorial optimization [Nemhauser and Wolsey, 1988, Nemhauser and Wolsey, 1989]. Moreover, the discussion of heuristic methods (that tend to find good local minimizers, but – especially when there are lots of them – not necessarily the global one) is limited to a short overview, and to techniques that remain useful for systematic methods.

No attempt has been made to be objective in selection and evaluation of the material; instead I have tried to give personal value judgments whenever I found it appropriate. At the present state of the art, where so many methods compete and very little comparative information is available, this seems justified. Thus I discuss the methods that I find most interesting, most useful, and most promising. (Usually, even for the topics I discuss, much more is in the references quoted.) And I hope that my selection bias will be justified by the future.

As one can see from the list of codes available for systematic global optimization given in Section 4.7, none of the available codes makes use of all available state-of-the-art techniques; indeed, many research groups on global optimization work with little knowledge of or care for what is going on in related areas. It is hoped that this course helps to change this lack of communication across the borders of the various traditions in global optimization.

Reviews from other perspectives are given in [Gray *et al.*, 1997, Pinter, 1996a, Törn, 2000, Törn *et al.*, 1999].

Acknowledgment.

I want to thank the Mathematics Department of the University of Colorado at Denver, and in particular Weldon Lodwick for the opportunity to give a course with the same title in April/May 2001.

4.2 Typical applications

To show the relevance of global optimization for both pure and applied mathematics, we sketch here a number of typical applications. Of course, this is only the tip of an iceberg...

(i) Many problems in graph theory are global optimization problems. For example the **maximum clique problem** asks for the maximal number of mutually adjacent vertices in a given graph. By a well-known theorem of [Motzkin and Strauss, 1965], an equivalent formulation is the indefinite quadratic program

$$\begin{array}{ll} \max & x^TAx \\ \text{s.t.} & e^Tx = 1, \quad x \geq 0, \end{array}$$

where A is the adjacency matrix of the graph and e is the all-one vector. Since the maximum clique problem is NP-hard, the same holds for all classes of global optimization problems that contain indefinite quadratic programming.

(ii) **Packing problems**. The problem is to place a number of k-dimensional ($k \leq 4$) objects of known shape within a number of larger regions of k-space of known shape such a way that there is no overlap and a measure of waste is minimized. The simplest packing problem is the **knapsack problem** where a maximal number of objects of given weights is to be placed into a container with given maximum weight capacity. Many packing problems arise in industry; but there are also a number of famous packing problems in geometry, of which the 300 year old **Kepler problem** of finding the densest packing of equal spheres in Euclidean 3-space was only solved recently by [Hales, 1998] (reducing the problem to several thousand linear programs and some interval calculations to ensure rigorous handling of rounding errors).

(iii) **Scheduling problems**. The problem is to match tasks (or people) and slots (time intervals, machines, rooms, airplanes, etc.) such that every task is handled in exactly one slot and additional constraints are satisfied. If there are several feasible matchings, one which minimizes some cost or dissatisfaction measure is wanted. Simple scheduling problems such as the **linear assignment problem** can be formulated as linear programs and are solved very efficiently, but already the related **quadratic assignment problem** is one of the hardest global optimization problems, where already most instances with about 30 variables are at the present limit of tractability.

(iv) **Nonlinear least squares problems**. In many applications, one needs to fit data to functional expressions. This leads to optimization problems with an objective function of a form such as

$$f(\theta) = \sum_l \|y_l - F(x_l, \theta)\|^2,$$

where x_l, y_l are given data vectors and θ is a parameter vector. Under certain assumptions, the most likely value of θ is the global minimizer; it generally must have a small objective function value at noise level if the model is to be deemed adequate. If the F_l are nonlinear in θ, a nonconvex optimization problem results that frequently has spurious local minima far above the noise level. A particularly obnoxious case is obtained for data fitting problems in **training neural networks**.

(v) **Protein folding**. The protein folding problem [Neumaier, 1997] consists in finding the equilibrium configuration of the N atoms in a protein molecule with given amino acid sequence, assuming the forces between the atoms are known. These forces are given by the gradient of the $3N$-dimensional potential energy function $V(x_1, \ldots, x_N)$, where x_i denotes the coordinate vector of the ith atom, and the equilibrium configuration is given by the global minimizer of V. Because short-range repulsive forces act like packing constraints, there are numerous local minima.

(vi) **Chemical equilibrium problems** [Floudas, 2000, McDonald and Floudas, 1995]. The task here is to find the number and composition of the phases of a mixture of chemical substances allowed to relax to equilibrium. Local optimization of the associated Gibbs free energy is notorious for giving wrong (nonglobal) solutions, and the need to solve such problems was one of the main driving forces for the development of constrained global optimization packages in the chemical engineering community, which till today is among the leaders in the field.

Many more applications can be found in the books by [Pinter, 1996b] and [Floudas and Pardalos, 1990].

4.3 Basic ideas

In this course we discuss systematic methods for finding the global minimizer(s) of an objective function subject to constraints. Such problems are typically much more difficult than local optimization problems, since it is often hard to decide whether a local minimizer found is global, and since one needs nonlocal space covering techniques to avoid being trapped in a region with only nonglobal local minimizers.

Basic to all systematic global optimization algorithms is the **branching principle** (Section 4.10). This technique consists in splitting (branching) the original problem recursively into subproblems which are sooner or later easy to solve. In pure branching methods, the more prospective branches are split more frequently, while in **branch and bound methods** one compute for each subproblem bounds on the objective function in the hope of being able to eliminate many subproblems at an early stage.

A very useful technique of **constraint propagation**, discussed in Section 4.13, allows to reduce the feasible region in many cases by exploiting properties of **separable constraints** of the form

$$\sum_{k \in K} q_k(x_k) \in \mathbf{b}$$

with simple, often linear or quadratic functions q_k of a single variable only. This technique may save a lot of branching steps and thus speeds up the branch and bound procedure. This is a reason why special care should be taken in presenting (or transforming) the problem in a form which has as much separability as possible, and we introduce the notion of a **semiseparable program** adapted to this feature. Section 4.18 addresses ways to transform general problems into semiseparable form by introducing appropriate extra variables. Semiseparable programs are also amenable to approximation by a **mixed integer linear program** (MILP), the only class of global optimization problems that has a long reputation of being successfully solvable even for large problem instances. We shall not discuss techniques for solving MILPs (see, e.g., [Bixby *et al.*, 2000, Nemhauser and Wolsey, 1988, Nemhauser and Wolsey, 1989, Wolsey, 1998]) but show how to approximate general global optimization problems by MILPs in Sections 4.17 and 4.18.

In order to be able to quickly eliminate subproblems it is important that one quickly locates good feasible points. This is usually done by local optimization (often in a somewhat rudimentary form); see Section 4.15. However, especially for problems with many local extrema, it is important to use some heuristics which (hopefully) prevents trapping of a local method in a high-lying local minimum. Suitable such tunneling techniques are discussed in Section 4.14.

Another basic principle, discussed in Section 4.20, is that of **outer approximation** of the feasible domain and **underestimation** of the objective function, in order to obtain **relaxed problems** which are convex and hence solvable by local methods. Indeed, this is the traditional way to obtain the bounds on the subproblem.

A useful tool for the automatic construction of tight bound constraints, outer approximations and underestimating functions in nonlinear problems is **interval arithmetic**; see Section 4.12. Though little known in the optimization community, interval arithmetic is an elegant way of computing with bound constraints, intervals, and simple higher dimensional geometric shapes like boxes and parallelepipeds. Its most prominent feature is that it allows strict estimates of the approximation error in linear and quadratic approximations of nonlinear functions over a box, thereby providing non-local information even in large boxes. In this course, we only give a very short introduction to this subject (just sufficient for writing programs); a much more extensive treatment can be found in [Neumaier, 1990]. Interval arithmetic can also be used to rigorously certify the validity of calculations with finite precision arithmetic, and some such applications to optimization are briefly treated in Section 4.22. The state of the art in 1997 of certified global optimization with interval methods is in [Kearfott, 1996b].

Further considerations relevant for a good global optimization programs are addressed in Section 4.21. In particular we consider the use of **cutting planes** and more general cutting surfaces.

Basic references. A basic reference on most aspects of global optimization is the *Handbook of Global Optimization* by [Horst and Pardalos, 1995]. It contains chapters written by the experts in the respective subfields, on global optimality conditions, complexity issues, concave minimization, dc methods, indefinite quadratic programming, complementarity problems, minimax problems, multiplicative programming, Lipschitz optimization, fractional programming, network problems, continuation methods, interval methods, and stochastic methods (including simulated annealing).

Some more recent books present the state of the art in deterministic global optimization from different perspectives: The interval point of view is in KEARFOTT's 1996 book *Rigorous Global Search* [Kearfott, 1996b]. The constraint propagation point of view is in the book *Numerica* by [Van Hentenryck *et al.*, 1997c]; See also the tutorial by [Lustig and Puget, 2000]. The convex analysis point of view is in the book *Deterministic Global Optimization: Theory, Algorithms and Applications* by [Floudas, 1999].

A comprehensive background on local optimization (needed as part of most good global optimization algorithms) can be found in the book *Numerical Optimization* by [Nocedal and Wright, 1999]. For interior point methods, this should be complemented by [Wright, 1997].

Other books on global optimization include [Forgó, 1988, Hansen, 1992, Horst *et al.*, 1995, Horst and Tuy, 1990, Ratschek and Rokne, 1984, Ratschek and Rokne, 1988, Van Laarhoven and Aarts, 1987, Zhigljavsky, 1991], and proceedings of conferences on global optimization include [Bomze *et al.*, 1996, Dixon and Szegö, 1975, Grossman, 1996, Floudas and Pardalos, 1992, Floudas and Pardalos, 2000].

4.4 Problem formulation

For the purpose of this course, a global optimization problem is specified in the form

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in \mathbf{x}, \quad F(x) \in \mathbf{F}, \quad x_I \text{ integral.} \end{array} \tag{4.1}$$

Here

$$\mathbf{x} = [\underline{x}, \bar{x}] = \{x \in \mathbb{R}^n \mid \underline{x} \le x \le \bar{x}\},$$

where $\underline{x} \in (\mathbb{R} \cup \{-\infty\})^n$ (with $\bar{x} \in (\mathbb{R} \cup \{\infty\})^n$, $\underline{x} \le \bar{x}$) is a bounded or unbounded **box** in $\mathbb{R}^n$, and x_I denotes the subvector $(x_{i_1}, \dots, x_{i_l})^T$ of x when $I = (i_1, \dots, i_l)$ is a list of indices. Inequalities between vectors are interpreted componentwise. $f : \mathbf{x} \to \mathbb{R}$ is a continuous objective function, $F : \mathbf{x} \to \mathbb{R}^m$ a vector of m continuous constraint functions $F_1(x), \dots, F_m(x)$, and $\mathbf{F}$ is a box in $\mathbb{R}^m$ defining the constraints on $F(x)$.

$$C = \{x \in \mathbf{x} \mid x_I \text{ integral}, F(x) \in \mathbf{F}\} \tag{4.2}$$

is the **feasible domain.** Points $x \in C$ are called **feasible**, and a **solution** of (4.1) is a feasible point $\hat{x} \in C$ such that

$$f(\hat{x}) = \min_{x \in C} f(x). \tag{4.3}$$

A **local minimizer** only satisfies $f(\hat{x}) \le f(x)$ for all $x \in C$ in some neighborhood of $\hat{x}$, and the solutions are precisely the **global minimizers**, i.e., the local minimizers with smallest objective function value. A **local (global) solver** is an algorithm or programming package designed for finding a local (global) minimizer. (We avoid the ambiguous term *optimizer* which may denote either a minimizer or a solver.)

The difficulties in global optimization stem mainly from the fact that there are generally many local minimizers but only one of them is the global minimizer (or just a few), and that the feasible region may be disconnected. (Consider, e.g., the set of positions in the Rocky Mountains below a certain altitude.) Already a linear objective function has one minimizer in each connected component, and local descent methods fail if they start in the wrong component.

Even the **constraint satisfaction problem**, i.e., the problem of deciding whether the feasible set is nonempty (and finding a feasible point) is frequently highly nontrivial, and may be essentially as difficult as the optimization problem itself (cf. Section 4.14). The usual device of minimizing a suitable measure of infeasibility does not work when the constraints are sufficiently nonlinear since this measure has itself local minima in which descent methods usually get stuck.

Usually it is possible to reformulate a global optimization problem such that f and F are **smooth**, i.e., twice continuously differentiable. Note that (4.1) is sufficiently flexible to take care of

- free variables x_i: take $\underline{x}_i = -\infty$, $\bar{x}_i = \infty$;
- nonnegative variables x_i: take $\underline{x}_i = 0$, $\bar{x}_i = \infty$;
- binary variables x_i: take $\underline{x}_i = 0$, $\bar{x}_i = 1$, $i \in I$;
- equality constraints $F_i(x) = 0$: take $\underline{F}_i = \bar{F}_i = 0$;
- inequality constraints $F_i(x) \leq 0$: take $\underline{F}_i = -\infty$, $\bar{F}_i = 0$.

If I is not empty then, if f and F are linear, (4.1) is called a **mixed integer linear program (MILP)**; and if f and F are convex, and $\underline{F}_i = -\infty$ for all nonlinear F_i, (4.1) is called a **mixed integer nonlinear program (MINLP)**. Strictly speaking, this term should apply for all problems (4.1); however, the techniques for MINLP use the convexity in an essential way, so that it is appropriate to reserve the term for the convex case. Nonconvex mixed integer global optimization problems have received little attention, but see, e.g., [Floudas, 1995, Sahinidis, 1996, Tawarmalani and Sahinidis, 2001]

The only class of global optimization problems that can be reliably solved for many large problem instances (say, $\approx 10^5$ variables and $|I| \approx 10^3$) is the class of MILPs. This is due to the fact that after fixing the integer variables one is left with a linear program, which can be solved efficiently. Instead of trying all integer combinations separately, branching techniques (branch and bound, branch and cut) combined with preprocessing the resulting linear programs drastically cut down the number of cases to be looked at. MINLP shares with MILP the feature that fixing all integer variables leads to a tractable problem, in this case a convex nonlinear program, for which every local minimizer is a solution; however, the dimensions are here more limited since nonlinear programming codes are significantly slower than their linear counterparts.

Most of constrained global optimization is nowadays best viewed as an adaptation of mixed integer programming technology to nonlinear problems. Historically, however, many of the techniques were devised independently by groups working in integer programming, combinatorial optimization, unconstrained optimization, interval analysis, and constraint logic programming.

Other important classes of global optimization problems:

- **simply constrained**: if $\dim F = 0$,
- **continuous**: if $I = \emptyset$,
- **bound constrained**: if simply constrained and continuous,
- **separable**: if $f(x) = \sum_{k=1}^{n} f_k(x_k)$ and $F(x) = \sum_{k=1}^{n} K_k(x_k)$,
- **factorable**: if f and F are obtained from constants and the x_k by applying a finite sequence of arithmetic operations and unary elementary functions,
- **reverse convex**: if f, F are concave, and $\underline{F}_i = -\infty$ for all nonlinear F_i.
- **DC**: if f, F are differences of convex functions.

4.5 First order optimality conditions

The material in this section is closely related to Chapter 2.4.

Traditional nonlinear programming provides the following necessary **(Karush-John) optimality conditions** for local minimizers. We assume that f, F are continuously differentiable, and denote by $f'(x)$ and $F'(x)$ the derivatives at x. Note that $f'(x)$ is a row vector and $F'(x)$ a matrix, the **Jacobian**.

4.5.1 Theorem. *([Karush, 1939, John, 1948])*

For every local minimizer $\hat{x}$ of (4.1) there are a number $\kappa \geq 0$ and a vector y, not both zero, such that the row vector

$$g^T = \kappa f'(\hat{x}) + y^T F'(\hat{x}) \tag{5.4}$$

satisfies

$$g_i \begin{cases} \geq 0 & \text{if } \underline{x}_i = \hat{x}_i < \bar{x}_i, \quad i \notin I, \\ \leq 0 & \text{if } \underline{x}_i < \hat{x}_i = \bar{x}_i, \quad i \notin I, \\ = 0 & \text{if } \underline{x}_i < \hat{x}_i < \bar{x}_i, \quad i \notin I, \end{cases} \tag{5.5}$$

$$y_i \begin{cases} \geq 0 & \text{if } \underline{F}_i < F_i(\hat{x}) = \bar{F}_i, \\ \leq 0 & \text{if } \underline{F}_i = F_i(\hat{x}) < \bar{F}_i, \\ = 0 & \text{if } \underline{F}_i < F_i(\hat{x}) < \bar{F}_i. \end{cases} \tag{5.6}$$

(Note that there is no restriction on g_i if $i \in I$ or $\underline{x}_i = \bar{x}_i$, and no restriction on y_i if $\underline{F}_i = F_i(\hat{x}) = \bar{F}_i$.)

We say that $\hat{x}$ satisfies a **constraint qualification** (CQ) if (5.4)–(5.6) hold for some $\kappa > 0$. In this case, one can scale g, κ, y to enforce $\kappa = 1$ and obtains the more frequently used **Kuhn-Tucker conditions** ([Kuhn, 1991, Kuhn and Tucker, 1951]). A sufficient condition for the constraint qualification is that the rows of $F'(\hat{x})$ are linearly independent; various weaker conditions guaranteeing CQ are known.

If $\kappa = 1$ then y is called an optimal **Lagrange multiplier** corresponding to $\hat{x}$ (it need not be unique). In this case, g is the gradient of the associated **Lagrangian** ([Lagrange, 1797])

$$L(x, y) = f(x) + y^T F(x)$$

at $x = \hat{x}$.

Note that minimizers with huge Lagrange multipliers are best considered as points nearly violating the constraint qualification, so that (5.4) holds with $y = O(1)$ and tiny κ.

If there are only nonnegativity constraints and equality constraints,

$$C = \{x \geq 0 \mid F(x) = b\},$$

corresponding to $\underline{x}_i = 0$, $\bar{x}_i = \infty$, $\underline{F}_i = \bar{F}_i = b_i$ then the conditions (5.6) are vacuous, and (5.5) reduces to the traditional **complementarity condition**

$$\min(g_i, x_i) = 0 \quad \text{for all } i.$$

4.5.2 Example. We consider the problem

$$\begin{array}{ll} \min & f(x) = -x_1 - 2x_2 \\ \text{s.t.} & F(x) = (x_1 - 1)^2 + (x_2 - 1)^2 = 1, \quad x_1, x_2 \in [-1, 1]. \end{array} \tag{5.7}$$

The feasible region is a quarter circle, and the contour lines of the objective function are linear, decreasing in the direction indicated in Figure 4.5. This implies that there is a local maximizer at P, a local minimizer at Q and a global minimizer at R. The solution is therefore $\hat{x} = \binom{0}{1}$. Since there are only two variables, we could analyze the problem graphically, but we could as well have proceeded symbolically as follows.

Assuming for simplicity the validity of the CQ, we find for the gradient of the Lagrangian

$$g = \begin{pmatrix} -1 \\ -2 \end{pmatrix} + y \begin{pmatrix} 2x_1 - 2 \\ 2x_2 - 2 \end{pmatrix}.$$

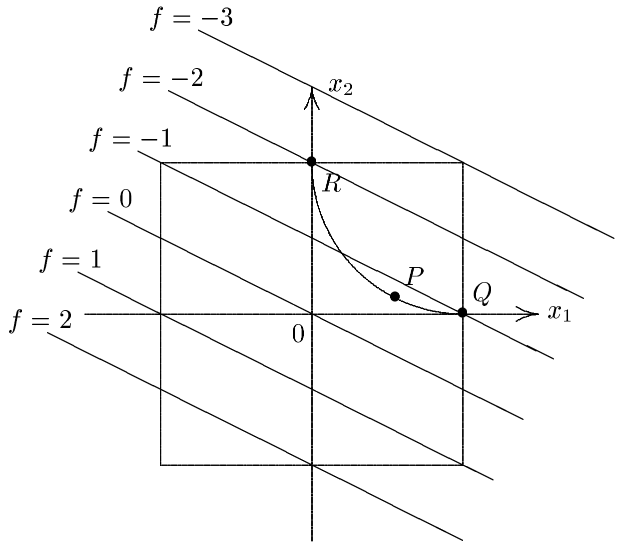


Figure 4.1: Feasible region and contour lines for problem (5.7).

The Kuhn-Tucker conditions require that $g_i \geq 0$ if $\hat{x}_i = -1$, $g_i \leq 0$ if $\hat{x}_i = 1$, and $g_i = 0$ otherwise. This leaves three cases for each component, and a total of $3 \cdot 3 = 9$ cases. If we assume $|\hat{x}_1|, |\hat{x}_2| < 1$ we must have $g = 0$, hence $\hat{x}_1 = 1 + 1/2y$, $\hat{x}_2 = 1 + 1/y$. Since $\hat{x}$ must be feasible, $y < 0$, and since $F(\hat{x}) = 1$, $y = -\frac{1}{2}\sqrt{5}$, $\hat{x} = (1 - 1/\sqrt{5}, 1 - 2/\sqrt{5})^T$, which is the local maximizer P. If we assume $\hat{x}_1 = -1$ or $\hat{x}_2 = -1$, or $\hat{x}_1 = \hat{x}_2 = 1$, we find a contradiction with $F(\hat{x}) = 1$. (These are 6 cases!) If we assume $|\hat{x}_2| < 1 = \hat{x}_1$ we find Q, and for $|\hat{x}_1| < 1 = \hat{x}_2$ we find R. Thus we have three feasible points satisfying the Kuhn-Tucker conditions, and a comparison of their function values shows that R is the global minimizer.

In general, we have three cases for each two-sided inequality and two for each one-sided inequality; since the number of independent choices must be multiplied, the total number of cases grows exponentially with the number of inequalities in the problem formulation. Thus this symbolical approach is limited to problems with few inequality constraints. Even then it only works if the resulting nonlinear equations are symbolically solvable and have few solutions only. Thus in general we need to resort to numerical methods.

We draw several conclusions from the example. First, there is a combinatorial aspect to the continuous global optimization problem, so that it resembles a mixed integer problem. Second, several cases can often be excluded by a single argument, which is the basis for the branch and bound approach to global optimization. Third, the Karush-John or Kuhn-Tucker conditions do not distinguish between maxima and minima (and other "stationary" points); all these need to be enumerated in a naive approach. Since there may be an exponential number of Kuhn-Tucker points, additional techniques are needed to reduce the search space.

In Section 4.16, we shall look at Lagrange multiplier techniques involving second order conditions that will address this last point.

4.6 Classification of techniques

Numerical methods for global optimization can be classified into three classes by distinguishing the available guarantees: heuristic methods, approximation methods, and systematic methods.

This course is mainly concerned with systematic methods; however since heuristics and approximation methods are frequently successful and, for many difficult problems, the only feasible choice, we give (later) an overview over the main posssibilities.

In this section, we only describe the qualitative features that distinguish the three classes of methods.

Heuristic methods. This class contains all methods that cannot be proved to find the global minimizer with a predictable amount of work, though they may work frequently. It contains several deterministic and most stochastic methods. For the latter, it is sometimes possible to prove convergence with probability arbitrarily close to 1 (if running arbitrarily long), but this does not yet guarantee convergence. (Moreover, the assumptions underlying the convergence proofs are frequently not verifiable for particular examples.) The simplest heuristic is **multiple random start**, consisting of picking random starting points and performing local optimizations from these points, in the hope that one of them is in the basin of attraction of the global minimizer. Most heuristics can be regarded as techniques devised to speed up this basic method, by picking the points more carefully and by doing the local optimizations only selectively.

Most of the research on good heuristics has been concentrated on global optimization methods for simply constrained problems only.

Approximation methods. This class contains all methods that transform the original problem by means of suitable approximations into a simpler global optimization problem that is more tractable. Solving the approximate problem yields an approximate solution for the original problem, and local optimization from this approximate solution gives the global minimizer of the original problem if the approximation was good enough, and usually a good local minimizer otherwise. Of great practical importance and widely used is the approximation by a MILP, since highly efficient software is available for solving the latter [Bixby *et al.*, 2000].

Systematic methods. This class contains all methods that (in exact arithmetic) are guaranteed to find the global minimizer with a predictable amount of work. Here predictable only means relative to known problem characteristics such as Lipschitz constants or other global information (needed for the convergence proof, but usually not for the algorithm itself). The bound on the amount of work is usually very pessimistic - exponential in the problem characteristics. It is only a weak guarantee that does not ensure that the algorithm is efficient in any sense, but it guarantees the absence of systematic deficiencies that prevent finding (ultimately) a global minimizer.

The simplest systematic method for bound constrained problems is **grid search**, where all points on finer and finer grids are tested, and the best point on each grid is used as a starting point for a local optimization. Since the number of points on a grid grows exponentially with the dimension, grid search is efficient only in one and two dimension. More efficient systematic methods generally combine branching techniques with one or several techniques from local optimization, convex analysis, interval analysis and constraint logic.

Generally, systematic methods (including approximation methods that reduce the problem to one treated by systematic methods) are more reliable than heuristic methods since, to the extent they work (which depends on the difficulty of the problem), they have built in guarantees.

Systematic methods with finite termination require more or less detailed access to global information about the problem. If only black box function (and sometimes gradient) evaluation routines are available, systematic methods will find the global minimizer with certainty after a finite time, but will never know when this is the case. Thus for systematic black box algorithms, stopping must be based on heuristic recipes.

Good heuristics (and probabilistic choices) also play a role in systematic methods, mainly to provide cheaply a good local minimizer that benefits the systematic search.

4.7 Software for systematic global optimization

Here we list some of the better systematic global optimization codes available on the WWW, with short comments on scope and method.

Some Branching Codes Using Function Values Only

The codes listed use black box function evaluation routines, and have heuristic stopping rules.

(i) **DIRECT, Divide Rectangles** (in Fortran, by [Gablonsky and Kelley, 2001])
`http://www4.ncsu.edu/~jmgablon/`

gblSolve, a MATLAB 5 implementation of DIRECT
`http://www.ima.mdh.se/tom/global/global.htm`

Implementations of a simple and efficient global optimization method by [Jones *et al.*, 1993] for bound constrained problems.

DIRECT is based on branching and a Pareto principle for box selection.

(ii) **MCS, Multilevel Coordinate Search** (by [Huyer and Neumaier, 1999])
`http://solon.cma.univie.ac.at/~neum/software/mcs/`

A Matlab program for bound constrained global optimization using function values only.

MCS is based on branching and sequential quadratic programming.

(iii) **LGO, Lipschitz Global Optimization** (commercial, by [Pinter, 1996b])
`http://is.dal.ca/~jdpinter/lgoide.htm`

An integrated development environment for global optimization problems with Lipschitz continuous objective and constraints.

LGO is based on branching and estimation of Lipschitz constants; constraints other than simple bounds are handled by L_1 penalty terms; interior convex constraints by projection penalties.

Some Branch and Bound Codes

The codes listed use global information (generally from required symbolic problem input). They have finite termination with guarantee that the global minimizer is found; in difficult cases storage or time limits may be exceeded, however, leading to appropriate error messages.

Not listed are the many MILP codes available (see the Global Optimization Web Page mentioned on p.1.)

(i) **BARON, Branch-And-Reduce Optimization Navigator** (in Fortran, by [Ryoo and Sahinidis, 1996, Sahinidis, 1996])
`http://archimedes.scs.uiuc.edu/baron/baron.html`

A general purpose solver for optimization problems with nonlinear constraints and/or integer variables. Fast specialized solvers for many linearly constrained problems. Easy to use GAMS/AMPL-like interface to the general purpose and specialized solvers.

BARON is based on branching and box reduction using convex relaxation and Lagrange multiplier techniques.

(ii) α**BB** (by [Adjiman *et al.*, 1998a, Adjiman *et al.*, 1996, Adjiman *et al.*, 1998b, Androulakis *et al.*, 1995])
`http://titan.princeton.edu/soft.html#abb`

Branch and bound code for nonlinear programs. The site has currently the description only; no code.

αBB is based on branching and bound by convex underestimation, using interval analysis to write nonlinearities in DC (difference of convexc function) form.

(iii) **GlobSol** (in Fortran 90, by [Kearfott, 1996b])
`http://www.mscs.mu.edu/~globsol/`

Branch and bound code for global optimization with general factorable constraints, with rigorously guaranteed results (even roundoff is accounted for correctly).

GlobSol is based on branching and box reduction using interval analysis to verify that a global minimizer cannot be lost.

(iv) **MINLP** (by [Fletcher and Leyffer, 1994, Fletcher and Leyffer, 1998a])
`http://www.maths.dundee.ac.uk/~sleyffer/MINLP.html`

Branch and bound code for mixed integer nonlinear programming; finding the global optimum is guaranteed only if all non-integer constraints are convex.

The site has currently the description only; no code. However, problems with AMPL input can be solved online via **NEOS** at
`http://www-neos.mcs.anl.gov/neos/solvers/IP:MINLP-AMPL/`

MINLP uses standard mixed integer programming techniques and convex underestimation.

(v) **GAMS/DICOPT** (commercial, by Grossmann)
`http://www.gams.com/solvers/dicopt/main.htm`

Solver for mixed Integer Nonlinear Programming (MINLP) problems.

Apparently based on convexity assumptions [Duran and Grossmann, 1986]; no details available about methods used.

(vi) **Numerica** (by [Van Hentenryck *et al.*, 1997c])

Branch and bound code for constrained optimization (with mathematically rigorous results).

This code (no longer available) was based on branching and box reduction using interval analysis and constraint satisfaction techniques. The box reduction and interval analysis algorithms of Numerica are now available in **ILOG Solver** at `http://www.ilog.com/products/solver/`

4.8 Heuristics for simple constraints

As mentioned before, heuristic methods have been mainly studied for simply constrained problems. Many different heuristics are known for simply constrained problems, and we sort them into three categories: **local descent techniques**, including among others multiple random start, clustering [Boender *et al.*, 1982] and Bayesian stochastic techniques [Mockus, 1989, Mockus, 1994], tunneling [Levy and Montalvo, 1985], and smoothing methods [Stillinger, 1985, Guddat *et al.*, 1990]; **non-monotonic search techniques**, including among others tabu search [Glover, 1989, Glover, 1990, Glover and Laguna, 1997], simulated annealing [Kirkpatrick *et al.*, 1983, Ingber, 1993, Van Laarhoven and Aarts, 1987], and deterministic variants such as threshold accepting [Dueck and Scheuer, 1990]; **ensemble methods**, including genetic algorithms [Holland, 1973, Forrest, 1993, Michalewicz, 1996] and variants such as ant colony minimization [Dorigo *et al.*, 1991].

No attempt is made to be representative or complete on referencing or describing the large literature on heuristic techniques; we only mention the 1975 book by [Dixon and Szegö, 1975], which marks the start of a tradition of comparing different heuristic global optimization methods, an excellent exposition of stochastic global optimization methods for bound constrained problems on the WWW by [Törn, 2000], and another WWW-survey of (mainly heuristic) methods by [Gray *et al.*, 1997]. For general heuristics in combinatorial optimization (where the underlying ideas are also called **metaheuristics**), see, e.g., [Aarts and Lenstra, 1997, Yagiura and Ibaraki, 2001].

Instead of describing technical details of the various methods (these vary from author to author and even from paper to paper), we give an informal view of the ideas, strengths and weaknesses of one method from every category, each based on analogies to natural processes where more or less global optima are reached. While these techniques are motivated by nature it is important to remember that processes in nature need not be the most efficient ones; at best it can be assumed to be efficient *given the conditions under which they have to operate* (namely an uncertain and changing environment that is potentially hazardous to those operating in it). Indeed, much of our present technology has vastly surpassed natural efficiency by unnatural means, and it would be surprising if it were different in global optimization. Even assuming that nature solves truly *global* optimization problems (a disputable assumption), simple lower estimates for the number of elementary steps – roughly corresponding to function evaluations – available to natural processes to converge are (in chemistry and in biology) in the range of 10^{15} or even more. This many function evaluations are unacceptable for present day computers, and will be so in the near future.

With a limited number of function evaluations, the quality of heuristic methods depends a lot on details of the implementation; comparisons on relative efficiency are virtually missing. Indeed, the techniques must generally be tuned to special classes of applications in order to be fast and competitive, which makes general purpose comparisons difficult and inconclusive.

Smoothing (= **homotopy** = **continuation**) **methods** are based on the intuition that, in nature, macroscopic features are usually an average effect of microscopic details; averaging smoothes out the details in such a way as to reveal the global picture. A huge valley seen from far away has a well-defined and simple shape; only by looking more closely, the many local minima are visible, more and more at smaller and smaller scales. The hope is that by smoothing a rugged objective function surface, most or all local minima disappear, and the remaining major features of the surface only show a single minimizer. By adding more and more details, the approximations made by the smoothing are undone, and finally one ends up at the global minimizer of the original surface.

In mathematical terms, one has to define a homotopy by introducing an additional parameter t into the problem in such a way that $t = 0$ gives the original problem, while $t = 1$ gives either a related convex problem or a related problem with a unique and known global minimizer. (There are various ways of doing this; homotopies whose parameter has a natural interpretation in the context of the original problem usually perform better.) Then a sequence of local problems is solved for $t = t_1, t_2 \ldots, t_N$, where the t_i form a decreasing sequence starting at 1 and ending at 0. Each time, the solution of the previous problem is taken as the starting point for the current problem. The quality of the final local minimizer depends on the homotopy, and frequently is the global or at least a good local minimizer.

There is no theoretical work on conditions that would ensure convergence to the global minimum. In particular, it is quite possible for such a method to miss the global minimum. However, for properly chosen homotopies, smoothing methods at least gives good local minima with a small number of function evaluations. (For more theory on homotopy methods, see e.g. [Guddat *et al.*, 1990, Section 6.3].)

Simulated annealing takes its intuition from the fact that the heating (annealing) and slow cooling of a metal brings it into a more uniformly crystalline state, that is believed to be the state where the free energy of bulk matter takes its global minimum. (Incidentally, even for the simplest potential energy functions, it is still an unsolved problem whether this is indeed true with mathematical rigor. Apart from that, even very pure crystals still have defects; i.e., the global minimum is not quite achieved in nature.) The role of temperature is to allow the configurations to reach higher energy states with a probability given by Boltzmann's exponential law, so that they can overcome energy barriers that would otherwise force them into local minima. This is quite unlike line search methods and trust region methods on which good local optimization programs are based.

In its original form, the simulated annealing method is provably convergent in a probabilistic sense but exceedingly slow; various ad hoc enhancements make it much faster. In particular, except for simple problems, success depends very much on the implementation used.

Genetic algorithms make use of analogies to biological evolution by allowing mutations and crossing over between candidates for good local optima in the hope to derive even better ones. At each stage, a whole population of configurations are stored. Mutations have a similar effect as random steps in simulated annealing, and the equivalent of lowering of the temperature is a rule for more stringent selection of surviving or mating individuals.

The ability to leave regions of attraction to local minimizers is, however, drastically enhanced by crossing over. This is an advantage if, with high probability, the crossing rules produce offspring of similar or even better fitness (objective function value); if not, it is a severe disadvantage. Therefore the efficiency of a genetic algorithm (compared with simulated annealing type methods) depends in a crucial way on the proper selection of crossing rules. The effect of interchanging coordinates is beneficial mainly when these coordinates have a nearly independent influence on the fitness, whereas if their influence is highly correlated (such as for functions with deep and narrow valleys not parallel to the coordinate axes), genetic algorithms have much more difficulties. Thus, unlike simulated annealing, successful tuning of genetic algorithms requires a considerable amount of insight into the nature of the problem at hand.

Both simulated annealing methods and genetic algorithms are, in their simpler forms, easy to understand and easy to implement, features that invite potential users of optimization methods to experiment with their own versions. The methods often work, if only slowly, and may be useful tools for applications where function values are not very expensive and the primary interest is to find (near-)solutions *now*, even when the reliability is uncertain and only subglobal optima are reached.

To make simulated annealing methods and genetic algorithms efficient, clever enhancements are essential. However, theoretical work on explaining the effectiveness of useful enhancements is completely lacking. I also haven't seen careful comparisons of the various options available and their comparative evaluation on standard collections of test problems.

In general, heuristic methods tend to fail systematically to find the global optimum on the more difficult problems in higher dimensions, but they frequently give relatively good points with a reasonable amount of effort. Beyond a certain number of function evaluations (that depends on the problem), progress slows down drastically if the global optimum has not yet been located already. This is unlikely to change in the future, although new heuristics and variations of old ones are discovered almost every year.

Fore general purpose global optimization, the most promising heuristics appear to be clustering methods (see the recent comparison by [Janka, 2000]), being fairly robust and fast. For heuristics adapted to problem structure, I'd favor smoothing methods (if a natural homotopy is available) and tabu search like strategies (being closer to systematic methods and having a kind of memory).

Heuristics for general constraints. For general constraints, heuristics is much less developed. Only the smoothing techniques extend without difficulties to general constraints. To use the other techniques, it is customary to rewrite problems with general constraints in an equivalent or approximately equivalent form with simple constraints only; see Section 4.9.

4.9 Bound constrained approximation

The second class category of global optimization methods consists of approximation methods. In this section we treat the approximation of general constrained problems by bound constrained problems using penalty and barrier functions. (The approximation of nonlinear problems by mixed integer linear programs will follow later.)

Penalty and barrier formulations. Traditionally (see [Fiacco and McCormick, 1990a]), constraints that cannot be handled explicitly are accounted for in the objective function, using simple l_1 or l_2 penalty terms for constraint violations, or logarithmic barrier terms penalizing the approach to the boundary. In both cases, the reformulation changes the solution, so that this is an instance of an approximation method, and the result should be used as a starting point for a subsequent local optimization of the original problem. There are also so-called exact penalty functions whose optimization gives the exact solution (see, e.g., [Nocedal and Wright, 1999]); however, this only holds if the penalty parameter is large enough, and what is large enough cannot be assessed without having global information.

The use of more general transformations gives rise to more precisely quantifiable approximation results. In particular, if it is known in advance that all constraints apart from the simple constraints are soft constraints only (so that some violation is tolerated), one may pick a transformation that incorporates prescribed tolerances into the reformulated simply constrained problem, using the following variation of a similar, but less flexible result of [Dallwig *et al.*, 1997].

4.9.1 Theorem. (Soft optimality theorem)

For some $\Delta > 0$, *let*

$$q(x) = \frac{f(x) - f_0}{\Delta + |f(x) - f_0|},$$

$$\delta_i(x) = \begin{cases} F_i(x) - \underline{F}_i & \text{if } F_i(x) \leq \underline{F}_i, \\ F_i(x) - \bar{F}_i & \text{if } F_i(x) \geq \bar{F}_i, \\ 0 & \text{otherwise.} \end{cases}$$

$$r(x) = \frac{2\sum \delta_i^2(x)/\sigma_i^2}{1 + \sum \delta_i^2(x)/\sigma_i^2}.$$

Then the global minimizer $\hat{x}$ *of the merit function*

$$f_{\text{merit}}(x) = 1 - (1 + q(x) + r(x))^{-1}$$

in $\mathbf{x}$ *either satisfies*

$$f(\hat{x}) \leq \min\{f(x) \mid F(x) \in \mathbf{F}, x \in \mathbf{x}\}, \tag{9.8}$$

$$F_i(\hat{x}) \in [\underline{F}_i - \sigma_i, \bar{F}_i + \sigma_i] \quad \text{for all } i, \tag{9.9}$$

or one of the following two conditions holds:

$$\emptyset = \{x \in [x] \mid F(x) \in \mathbf{F}\}, \tag{9.10}$$

$$f_0 < \min\{f(x) \mid F(x) \in \mathbf{F}, x \in \mathbf{x}\}. \tag{9.11}$$

Proof. If there is a feasible point with $f(x) \leq f_0$ then $q(x) \leq 0$, $r(x) = 0$ at this point. Since f_{merit} is monotone increasing in $q + r$, we conclude from $f_{\text{merit}}(\hat{x}) \leq f(\hat{x})$ that

$$q(\hat{x}) \leq q(\hat{x}) + r(\hat{x}) \leq q(x) + r(x) \leq q(x),$$

$$-1 + r(\hat{x}) \leq q(\hat{x}) + r(\hat{x}) \leq q(x) + r(x) \leq 0,$$

hence $f(\hat{x}) \leq f(x)$, giving (9.8), and $r(\hat{x}) \leq 1$,

$$q_i^2(\hat{x})/\sigma_i^2 \leq \sum q_i^2(\hat{x})/\sigma_i^2 \leq 1,$$

giving (9.9). □

Note that if a feasible point is already known we may choose f_0 as the function value of the best feasible point known, thus eliminating the possibility (9.10).

Of course there are other choices for $q(x)$, $r(x)$, $f_{\mathrm{merit}}(x)$ with the same properties. The choices given are simple and lead to a continuously differentiable merit function with Lipschitz-continuous gradient if f and F have these properties.

Moreover, the merit function is bounded above by 1 even when f and/or some F_i are unbounded. In particular, the formulation is able to handle so-called **hidden constraints**, where the conditions for infeasibility are not known explicitly but are discovered only when attempting to evaluate one of the functions involved. In such a case, if the function value routine returns an error code, the merit function value can be simply set to 1.

Projection penalties. A little known result by [Pinter, 1996b] may be used to get in certain cases (in particular, for linear and convex quadratic constraints) an exact reformulation as a nonsmooth but Lipschitz continuous simply constrained problem. The idea is to project infeasible point to the feasible domain.

To accomodate linear constraints (or convex quadratic ones), Pinter assumes that x_0 is a known interior point; such a point can be found by solving a linear program or a convex second order cone program. (If no interior point exists since the feasible set is in a lower-dimensional subspace, each feasible point has the form $x = x_0 + Cz$ with $z \in \mathbf{z}$ with x_0 in the relative interior of the feasible domain, and $\mathbf{z}$ a box with $0 \in \mathrm{int}\ \mathbf{z}$. Both x_0 and C can be found by techniques from convex analysis for finding an independent set of points spanning the affine subspace spanned by the feasible set. Reposing the optimization problem in terms of z reduces the dimension and yields a problem in which 0 is an interior point.)

For arbitrary $\gamma > 0$ we now define the modified objective function

$$\bar{f}(x) := f(\bar{x}) + \gamma\|\bar{x} - x\|^2, \tag{9.12}$$

where

$$\bar{x} = \lambda x_0 + (1 - \lambda)x \tag{9.13}$$

and $\lambda = \lambda_x \geq 0$ is smallest such that $\bar{x}$ satisfies the linear constraint. This is well-defined, since $\lambda = 1$ always works by choice of x_0. Each constraint contributes a lower bound $\in [0, 1]$ for λ, and the largest of these bounds is the desired value. In particular, a linear constraint $a^T x \le \alpha$ contributes a nonzero lower bound

$$\lambda \ge (a^T x - \alpha)/(a^T x - a^T x_0)$$

if both numerator and denominator of the right hand side are positive. A convex quadratic constraint similarly yields a quadratic inequality that can easily be solved for λ. (Convexity can be weakened to star-shapedness with respect to x_0.)

The modified objective function (9.12) is Lipschitz continuous; but nonsmooth at all points where the ray (9.13) hits a lower-dimensional face of the feasible domain. Note that to evaluate (9.12), function values are needed only at points satisfying the linear (or convex quadratic) constraints.

4.10 Pure branching methods

We begin our analysis of systematic methods for global optimization by looking at the options for methods that can access no global information about a problem. The information is made available via black box routines that provide **local information** only, i.e., function values and possibly gradients or Hessians at single points. A necessary and sufficient conditions for systematic methods based on local information only is given by the following important **density theorem** due to [Törn and Žilinskas, 1989]. It formalizes the simple observation that after finitely many local evaluations there are still many 'holes', i.e., balls not containing an already evaluated point, and there are many functions [Neumaier, 2001] that have the known function values, gradients and Hessians at the evaluation points but an arbitrarily low function value at the center of such a ball.

4.10.1 Theorem. *Any method based on local information only that converges for every continuous f to a global minimizer of f in a feasible domain C must produce a sequence of points $x^1, x^2, \ldots$ that is dense in C.*

Conversely, for any such method,

$$\liminf_{l \to \infty} f(x^l) = \min\{f(x) \mid x \in C\}.$$

A global optimization method based on local information only is called **convergent** if it satisfies the hypothesis of the density theorem. (Actual implementations of a convergent global optimization method usually are not truly convergent since they must have built in termination criteria that are necessarily heuristic.)

Convergence is a **minimal** requirement and does not make an algorithm good! For example, exhaustive grid search is convergent but far too slow in dimensions

> 2. (Compare with local optimization with line searches along the steepest descent direction, which is globally convergent but frequently very slow.) In a sense, the density theorem says that any convergent method must be ultimately exhaustive, though it may delay the detailed exploration of unpromising regions. Since, in practice, only a limited number of points can be explored, the behavior of a pure branching method is governed by its ability to find a good ordering of the points to be evaluated for which premature termination has no severe effect.

Three good systematic general purpose global optimization algorithms based on local information only are currently available: DIRECT [Jones *et al.*, 1993], MCS [Huyer and Neumaier, 1999] and LGO [Pinter, 1996b]. All work for bound constrained problems only and need the approximation techniques of Section 4.9 for more general problems. (Some of these are built in into LGO, but must be coded by the user for DIRECT and MCS.) All three algorithms enforce convergence by employing a branching scheme. They differ in how and when to split, and what is done within each box.

A **branching scheme** generates a sequence of rooted trees of boxes whose leaves cover the feasible set. At least one point in each box is evaluated. The first tree just has the original box as root and only leaf. Each other tree is obtained from the previous one by splitting one or several leaves. If the diameter of all boxes at all leaves converge to zero, convergence of the algorithm is straightforward.

The convergence to zero of the diameters is ensured by appropriate **splitting rules** that define when and how a box is split. For example, convergence is guaranteed when in each of a sequence of rounds, one

- always splits the oldest box along the oldest side, and finitely many other boxes, or
- always splits the longest box along the longest side and finitely many other boxes (where length = sum of length of sides),

provided that each split of the oldest (or longest) box produces boxes whose volume is at most a fixed fraction < 1 of the unsplit box. The possibility of 'and finitely many other boxes' (but not many if the code is to be robust!) can be used with considerable flexibility without destroying the convergence property.

Apart from the convergence requirement, the key to efficiency is a proper balance of global and local search. This is achieved in DIRECT by splitting in each round all boxes for which the pair (volume,midpoint function value) are not dominated by other such pair. Here (v, f) is dominated by (v', f') if both $v' < v$ and $f' > f$. In particular, the box of largest volume and the box with the best function value are never dominated and hence always split. MCS uses instead domination of pairs (l, f), where l is a suitably assigned level, and in addition employs local optimization steps (using sequential bound constrained quadratic programs) from appropriate candidate points. LGO uses lower bounds

$$L \geq \max_{k,l} \|f(x_k) - f(x_l)\|/\|x_k - x_l\|$$

on Lipschitz constants L obtained from the previous function evaluations to decide on the promising boxes to split first. (Upper bounds on L, and hence bounds on function values, cannot be obtained from local information only.)

4.10.2 Remark. The combination of a suitable branching strategy with the heuristic methods discussed earlier would make the latter systematic, and appears to be a fruitful research direction.

To improve on the density theorem we must find ways to throw away irrelevant parts of the feasible domain that are guaranteed not to contain a global minimizer. To be able to do this reliably, some kind of global information is necessary. This is utilized by box reduction techniques, discussed in Section 4.11 using a simple example, and afterwards in more depth.

4.11 Box reduction – an example

Box reduction techniques are based on a more or less sophisticated interplay of several components: logical constraint propagation, interval analysis, convex relations and duality arguments involving Lagrange multipliers. Before giving a more formal treatment, we illustrate simple arguments of each of these components by reconsidering Example 4.5.2.

Suppose that a local solver produced already the local minimizer $\hat{x} = \binom{0}{1}$ for the problem (5.7) discussed in Example 4.5.2, perhaps as the best local minimizer found by minimizing from a few random starting points. We use box reduction to check whether there is possibly a better feasible point. In fact we know already that this is not the case, but we obtained this knowledge in a way that works only for very simple problems. Thus we want to do it again, using only techniques of wide applicability.

The idea of box reduction is to use various arguments that allow to shrink the box without losing any feasible point that is at least as good as the best point found already. Since $\hat{x}$ is feasible with objective function value -2, any such point satisfies

$$f(x) = -x_1 - 2x_2 \leq -2, \tag{11.14}$$

$$F(x) = (x_1 - 1)^2 + (x_2 - 1)^2 = 1, \tag{11.15}$$

$$x_1 \in [-1,1], \quad x_2 \in [-1,1]. \tag{11.16}$$

Constraint propagation (see Section 4.13) consists in deducing better bounds for a variable by using the other bounds and one of the constraints. In particular, (11.14) implies $x_2 \geq 1 - x_1/2 \geq 0.5$ since $x_1 \leq 1$, and $x_1 \geq 2 - 2x_2 \geq 0$ since $x_2 \leq 1$, reducing the bounds to

$$x_1 \in [0,1], \quad x_2 \in [0.5,1].$$

Similarly, (11.15) implies $(x_1-1)^2 = 1-(x_2-1)^2 \geq 1-0.25 = 0.75$, hence $x_1 \leq 1-\sqrt{0.75} < 0.14$, giving the improved bound

$$x_1 \in [0, 0.14].$$

This bound could be used to improve again x_2 using (11.14); and by alternating use of (11.14) and (11.15) one would obtain a sequence of boxes shrinking towards $\hat{x}$. However this is a special feature of this simple example. In most cases, this simple substitution process gives no or only very little improvements after the first few good reductions. (Look at a problem with the constraints $x_1+x_2 = 0$, $x_1 - x_2 = 0$, $x_1, x_2 \in [-1,1]$ to see why.) However, it is a very cheap and easily formalizable process that gives important initial range reductions in many otherwise difficult problems.

Interval analysis (see Section 4.12) can be applied in a number of different ways. Here we use it to produce linear relaxations of the nonlinear constraint. The Jacobian of $F(x)$ at $x \in \mathbf{x} = ([0,0.14],[0.5,1])^T$ is

$$F'(x) = (2x_1-2, 2x_2-2) \in ([-2,-1.72],[-1,0]) = F'(\mathbf{x}).$$

The mean value theorem implies that, for any $\tilde{x} \in \mathbf{x}$,

$$F(x) \in F(\tilde{x}) + F'(\mathbf{x})(x-\tilde{x}) \quad \text{if } x \in \mathbf{x}.$$

Using $\tilde{x} = \hat{x}$ we find

$$1 \in 1 + [-2,-1.72]x_1 + [-1,0](x_2-1) = [1-2x_1, 2-1.72x_1-x_2];$$

the interval evaluation needs no case distinction since x_1 and $x_2 - 1$ happen to have constant sign. The lower bound gives no new information, but the upper bound leads to the new constraint

$$1.72x_1 + x_2 \leq 1.$$

By its derivation, this constraint is weaker than (11.15).

But since it is linear, the constraint is quite useful for **relaxation techniques** (see Section 4.20). It allows us to create a convex relaxation of the problem. Indeed, we may look at the relaxed linear program

$$\begin{array}{ll} \min & -x_1 - 2x_2 \\ \text{s.t.} & 1.72x_1 + x_2 \leq 1, \quad 0 \leq x_1 \leq 0.14, \quad 0.5 \leq x_2 \leq 1. \end{array} \tag{11.17}$$

By construction, every feasible point better than the best point is feasible for (11.17), hence the minimum of (11.17) will be a **lower bound** on the best possible objective function value of the original problem. Solving (11.17) gives the solution $\hat{x} = \binom{0}{1}$ with function value -2. Since this lower bound equals the best function value found so far for the original problem, the original problem has global minimum -2. This is a happy accident due to special circumstances: Our problem had already a linear objective function, and the global minimizer was at a corner of the feasible set. (But as we shall see, we can adapt the technique to work much more generally if the box is narrow enough.)

It might still be the case that there is a second, undiscovered global minimizer. This can be checked with **multiplier techniques**. We use the Lagrange multiplier $\hat{\lambda} = 2$ associated with linear constraint of (11.17) at the solution. The associated linear combination $-x_1 - 2x_2 + 2(1.72x_1 + x_2 - 1)$ is bounded by the best known function value -2 of the original problem, giving $2.44x_1 - 2 \leq -2$, hence $x_1 \leq 0$. Thus we must have $x_1 = 0$, and constraint propagation using (11.14) implies $x_2 = 1$. Thus the box has been reduced to $\hat{x}$, showing that it is the only global minimizer.

What generalizes? The problem discussed was deliberately kept simple so that the complete solution process could be demonstrated explicitly. In general, constraint propagation only gives limited reduction. Similarly, relaxed linear or convex programs usually only gives a lower bound on the least possible objective function value, but the linear combination derived from the Lagrange multiples frequently containts useful information that can be exploited by constraint propagation to get a further significant box reduction.

If the reduction process stalls or becomes slow, the box is split into two or more smaller boxes. On the smaller boxes, the same techniques may prove effective, and one alternates box reduction and box splitting until all boxes sizes are below some termination threshold. Usually, only very few boxes remain if good enough reduction techniques are used (pathological exceptions include min $x - x$ s.t. $x \in [0, 1]$). If no box remains, the problem is guaranteed to have no feasible point.

The total number of boxed processed is a measure of the difficulty of a problem for the particular algorithm used. Simple problems (like the illustrative example) only need a single box; in the worst case, an exponential number of boxes may be needed. In the latter case, time and storage limitations may force a premature termination; in this case the best point found is not verified to be a global minimizer.

4.12 Interval arithmetic

Interval analysis, the study of theory and algorithms for computing with intervals, is a large subject; see [Moore, 1979] (introductory) and [Neumaier, 1990] (advanced). Its importance for global optimization stems from several, interrelated facts:

- Interval analysis gives easily computable (though sometimes only very crude) bounds on expressions.
- Interval analysis allows one to control nonlinearities in a simple way (via centered forms).
- Interval analysis extends classical analysis in its ability to provide *semilocal* existence and optimality conditions, valid within a *pre-specified* local region around some point, while classical analysis generally only asserts the existence of such neighborhoods without enabling one to find them in a simple way.

We give here a short introduction to the basics, and mention the main techniques useful for global optimization.

If $\mathbf{a}$ and $\mathbf{b}$ are two intervals we define for any binary operation $\circ$

$$\mathbf{a} \circ \mathbf{b} := \square\{\tilde{a} \circ \tilde{b} \mid \tilde{a} \in \mathbf{a}, \tilde{b} \in \mathbf{b}\}, \tag{12.18}$$

provided the right hand side is defined. Here

$$\square S = [\inf S, \sup S]$$

denotes the **interval hull** of a set of real numbers, i.e., the tightest interval containing S. A monotonicity argument gives for addition and subtraction

$$\mathbf{a} + \mathbf{b} = [\underline{a} + \underline{b}, \overline{a} + \overline{b}], \tag{12.19}$$

$$\mathbf{a} - \mathbf{b} = [\underline{a} - \overline{b}, \overline{a} - \underline{b}], \tag{12.20}$$

and for multiplication and division

$$\mathbf{a} * \mathbf{b} \quad = \square\{\underline{a}\underline{b}, \underline{a}\overline{b}, \overline{a}\underline{b}, \overline{a}\overline{b}\}, \tag{12.21}$$

$$\mathbf{a}/\mathbf{b} \quad = \square\{\underline{a}/\underline{b}, \underline{a}/\overline{b}, \overline{a}/\underline{b}, \overline{a}/\overline{b}\} \quad \text{if } 0 \notin \mathbf{b}; \tag{12.22}$$

in most cases only two of these products or quotients need to be computed. We also define elementary functions $\varphi \in \{\mathrm{sqr}, \mathrm{sqrt}, \exp, \log, \sin, \cos, \dots\}$ of an interval $\mathbf{a}$ (and similarly $-\mathbf{a}, \mathbf{a}_+$, etc.) by

$$\varphi(\mathbf{a}) := \square\{\varphi(\tilde{a}) \mid \tilde{a} \in \mathbf{a}\} \tag{12.23}$$

whenever the right hand side is defined. Again $\varphi(\mathbf{a})$ can be computed from the value of φ a the endpoints of $\mathbf{a}$ and the interior extremal values, depending on the monotonicity properties of φ.

For details and further properties of interval operations see [Moore, 1979] (introductory) and [Neumaier, 1990] (advanced); we only remark here that some rules familiar from real arithmetics fail, and in particular the interval evaluation of different expressions equivalent in real arithmetic may give different results. E.g., (with $-\mathbf{a} := 0 - \mathbf{a} = [-\overline{a}, -\underline{a}]$)

$$\mathbf{a} + (-\mathbf{a}) = \mathbf{a} - \mathbf{a} \neq 0 \quad \text{except when } \underline{a} = \overline{a}.$$

Therefore, we also use the converse **inner operations**

$$\mathbf{a} \overset{\circ}{+} \mathbf{b} := [\underline{a} + \overline{b}, \overline{a} + \underline{b}], \tag{12.24}$$

$$\mathbf{a} \overset{\circ}{-} \mathbf{b} := [\underline{a} + \underline{b}, \overline{a} - \overline{b}]. \tag{12.25}$$

Here, expressions of the form $\pm\infty \mp \infty$ in (12.24) or (12.25) must be interpreted as $-\infty$ for the lower bounds and as $+\infty$ for the upper bounds. Note that the result of an inner operation is not necessarily an interval since it may happen that the lower bound is larger than the upper bound; giving an empty "interval".

All these operations are very simple to program. Note that many implementations of interval arithmetic are rather slow since they take care to guarantee correct (and often optimal) outward rounding, needed when interval arithmetic is used for mathematically rigorous certification (see Section 4.22). For global optimization without certification, unrounded fast interval operations usually suffice.

Important: When using unrounded interval arithmetic, proper safeguards must be taken at places (such as (12.24), (12.25) and intersections) where intervals might become (spuriously) empty due to accumulation of roundoff errors. In place of an empty result, a thin interval formed from the arithmetic mean of the two intersecting bounds should be returned in a safe implementation.

As already mentioned, an interval evaluation $f(\mathbf{x})$ of some expression f often overestimates the desired range $\text{Range}(f, \mathbf{x}) = \{f(x) \mid x \in \mathbf{x}\}$ of a function. However, under very mild conditions [Neumaier, 1990, Section 1.4], the evaluation over small boxes satisfies

$$f(\mathbf{x}) \subseteq \text{Range}(f, \mathbf{x}) + O(\varepsilon) \quad \text{if } \bar{x} - \underline{x} = O(\varepsilon);$$

we refer to this as the **linear approximation property** of simple interval evaluation.

Better enclosures, especially for small ε, can be obtained by **centered forms**; the simplest of these (but not the most efficient one, see [Neumaier, 1990, Chapter 2] for better methods based on slopes) is the **mean value form**: Due to the mean value theorem, we have

$$f(x) \in f(\xi) + f'(\mathbf{x})(x-\xi) \quad \text{if } x, \xi \in \mathbf{x}. \tag{12.26}$$

In particular, $\mathrm{Range}(f, \mathbf{x})$ is contained in $f(\xi) + f'(\mathbf{x})(\mathbf{x}-\xi)$, and it can be shown that, under mild conditions,

$$f(\xi) + f'(\mathbf{x})(\mathbf{x}-\xi) \subseteq \mathrm{Range}(f, \mathbf{x}) + O(\varepsilon^2) \quad \text{if } \bar{x} - \underline{x} = O(\varepsilon);$$

we say, that the mean value form (as other centered forms) has the **quadratic approximation order**.

Apart from interval evaluation and centered forms, we need **interval Newton methods** for verifying solutions of nonlinear systems of equations. The prototype (but again not the most efficient method; see [Neumaier, 1990, Chapter 5] for better methods based on slopes and Gauss-Seidel iteration) is KRAWCZYK's [Krawczyk, 1969] method. To check for solutions of $F(x) = 0$ with $x \in \mathbf{x}$, Krawczyk multiplies the vector version of (12.26) by a matrix C and adds x to find

$$x \in K(\mathbf{x}, \xi) := \xi + CF(\xi) + (CF'(\mathbf{x}) - I)(\mathbf{x} - \xi).$$

For $\xi \in \mathbf{x}$, the resulting **Krawczyk operator** $K(\mathbf{x}, \xi)$ has the following properties, typical of interval Newton methods:

(i) Any zero $x \in \mathbf{x}$ of F lies in $\mathbf{x} \cap K(\mathbf{x}, \xi)$.

(ii) If $\mathbf{x} \cap K(\mathbf{x}, \xi) = \emptyset$ then $\mathbf{x}$ contains no zero of F.

(iii) If $K(\mathbf{x}, \xi) \in \mathrm{int}\mathbf{x}$ then $\mathbf{x}$ contains a unique zero of F.

(i) and (ii) follow directly from the above derivation, while (iii) is a consequence of Banach's fixed point theorem.

The most important part is (iii), since, applied to the Karush-John conditions, it allows the elimination of large regions around a local minimizer; cf. section 4.16. However, (i) and (ii) are also useful as ways to reducing a box or eliminating it, if it contains no zero. This is implemented in GLOBSOL [Kearfott, 1996b] and Numerica [Van Hentenryck *et al.*, 1997c].

Convexity check. Interval analysis can be used to check the convexity of a function $f : \mathbf{x} \to \mathbb{R}$ in some box $\mathbf{x}$. Let $\mathbf{G}$ is a matrix of intervals (usually simply called an *interval matrix*), calculated as an enclosure of $f''(x)$ for $x \in \mathbf{x}$, then, with $r = \max\{\overline{x}_k - \underline{x}_k \mid k = 1, \ldots, n\}$, the linear approximation property implies that $|\bar{G} - \underline{G}| = O(r)$. Such a statement implies that $|\mathbf{G} - \tilde{G}| = O(r)$ for all individual matrices $\tilde{G} \in \mathbf{G}$, with absolute values taken component-wise. In particular, if $\hat{G}$ is positive definite then, provided the underlying box is not too wide, all matrices in $\mathbf{G}$ are definite, too; and if this is the case, f is convex in $\mathbf{x}$. The following constructive criterion for simultaneously checking the definiteness of all members of an interval matrix was given in [Neumaier, 1996].

4.12.1 Theorem. *(Sufficient conditions for convexity)*

Let $f : \mathbf{x} \to \mathbb{R}$ be twice continuously differentiable on the compact box $\mathbf{x}$, and suppose that $\mathbf{G}$ is a symmetric interval matrix such that

$$f''(x) \in \mathbf{G} \quad \textit{for all} \;\; x \in \mathbf{x}. \tag{12.27}$$

(i) If some symmetric matrix $G_0 \in \mathbf{G}$ is positive definite and all symmetric matrices in $\mathbf{G}$ are nonsingular then they are all positive definite, and f is uniformly convex in $\mathbf{x}$.

(ii) In particular, this holds if the midpoint matrix

$$\check{G} = (\sup \mathbf{G} + \inf \mathbf{G})/2$$

is positive definite with inverse C, and the preconditioned radius matrix

$$\Delta = |C| \operatorname{rad} \mathbf{G},$$

where

$$\operatorname{rad} \mathbf{G} = (\sup \mathbf{G} - \inf \mathbf{G})/2,$$

satisfies the condition

$$\|\Delta\| < 1 \tag{12.28}$$

(in an arbitrary norm).

Proof. (i) Since the eigenvalues are continuous functions of the matrix entries and the product of the eigenvalues (the determinant) cannot vanish, no eigenvalue changes sign. Hence the eigenvalues of all matrices in $\mathbf{G}$ are positive, since this is the case for the positive definite member. Thus all matrices in $\mathbf{G}$ are positive definite. By well-known results, uniform convexity of f now follows from (12.27).

(ii) $G_0 = \check{G}$ belongs to $\mathbf{G}$, and condition (12.28) implies strong regularity of the interval matrix $\mathbf{G}$ ([Neumaier, 1990], Section 4.1) and hence nonsingularity of all matrices in $\mathbf{G}$. Thus (i) applies. $\square$

In many cases, the Hessian of the augmented Lagrangian can be shown to have the form

$$f''(x) = \sum u_i A_i \quad \text{with } u_i \in \mathbf{u}_i,$$

for suitable constructively available real matrices A_i and intervals $\mathbf{u}_i$. In this case, the above result can be strengthened (with virtually the same proof) by replacing $\check{G}$ and Δ with

$$\check{G} = \sum \check{u}_i A_i$$

and

$$\Delta' = \sum \operatorname{rad} \mathbf{u}_i |CA_i|,$$

respectively, where

$$\check{u} = (\sup \mathbf{u} + \inf \mathbf{u})/2$$

and

$$\operatorname{rad} \mathbf{u} = (\sup \mathbf{u} - \inf \mathbf{u})/2.$$

Indeed, it is not difficult to see that for $G = \sum \mathbf{u}_i A_i$, we always have $0 \leq \Delta' \leq |\Delta|$, so that the refined test is easier to satisfy.

Other sufficient conditions for convexity, the basis of the αBB method [Adjiman *et al.*, 1996, Androulakis *et al.*, 1995] are in [Adjiman *et al.*, 1998b]

4.13 Constraint propagation

In many cases, constraints can be used to tighten the box constraints. The general technique is called **constraint propagation** and was pioneered in constraint logic [Cleary, 1987, Older and Vellino, 1993], but has also forerunners in presolve techniques in mathematical programming [Anderson and Anderson, 1995, Lodwick, 1989]; see [Babichev *et al.*, 1993, Benhamou *et al.*, 1994, Benhamou and Older, 1997, Chen and van Emden, 1995, Hager, 1993, Hyvönen and De Pascale, 1996, Kearfott, 1991, Van Hentenryck, 1989, Van Hentenryck *et al.*, 1997b, Van Hentenryck *et al.*, 1997c] for further developments. Chapter 6 contains an in depth treatment of constraint propagation from the point of view of constraint programming (which emphasizes solution strategies); here we concentrate on particular formulas for implementing single steps of the propagation.

We follow the setup by ([Dallwig *et al.*, 1997]), which handles linear constraints (and more generally block separable constraints) without the need to decompose the constraints into primitive pieces defined by single operations.

4.13.1 Proposition. *Let the q_k be real-valued functions defined on $\mathbf{x}_{J_k}$.*

(i) If

$$\overline{q}_k \geq \sup\{q_k(x_{J_k}) \mid x_{J_k} \in \mathbf{x}_{J_k}\}, \quad \overline{s} \geq \sum_k \overline{q}_k, \tag{13.29}$$

then

$$x \in \mathbf{x}, \quad \underline{a} \leq \sum_k q_k(x_{J_k}) \quad \Rightarrow \quad q_k(x_{J_k}) \geq \underline{a} - \overline{s} + \overline{q}_k \quad \textit{for all } k. \tag{13.30}$$

(ii) If

$$\underline{q}_k \leq \inf\{q_k(x_{J_k}) \mid x_{J_k} \in [x_{J_k}]\}, \quad \underline{s} \leq \sum_k \underline{q}_k, \tag{13.31}$$

then

$$x \in \mathbf{x}, \quad \sum_k q_k(x_{J_k}) \leq \overline{a} \quad \Rightarrow \quad q_k(x_{J_k}) \leq \overline{a} - \underline{s} + \underline{q}_k \quad \textit{for all } k. \tag{13.32}$$

Proof. The assumptions of (i) imply

$$q_k(x_{J_k}) \geq \underline{a} - \sum_{l \neq k} q_l(x_{J_l}) \geq \underline{a} - \sum_{l \neq k} \overline{q}_l \geq \underline{a} + \overline{q}_k - \overline{s},$$

hence the conclusion in (13.30) holds. (ii) is proved in the same way. □

The proposition is applied as follows to tighten box constraints. Suppose that $x \in \mathbf{x}$. For any constraint of the form

$$\underline{a} \leq \sum_k q_k(x_{J_k}) \tag{13.33}$$

we form the quantities (13.29). (This is straightforward if the q_k depend on a single variable x_k only, $J_k = \{k\}$; in the most important cases, q_k is linear or quadratic in x_k, and the supremum is very easy to calculate; in more complicated cases, upper resp. lower bounds can be calculated with interval arithmetic.) Then one checks the condition

$$\underline{a} \leq \overline{s} \ ; \tag{13.34}$$

if it is violated then (13.33) is clearly inconsistent with $x \in \mathbf{x}$ (and in the branch and bound application, the corresponding subproblem can be discarded). If (13.34) holds, one can exploit the conclusion in (13.30), provided that one can compute the set of $x_{J_k} \in \mathbf{x}_{J_k}$ (or a superset) such that

$$q_k(x_{J_k}) \geq \overline{q}_k + \underline{a} - \overline{s}. \tag{13.35}$$

If $\underline{a}$ is sufficiently close to $\overline{s}$ then x_{J_k} will be forced to be close to the global maximum of q_k over the interval $\mathbf{x}_{J_k}$, thus reducing the component $\mathbf{x}_{J_k}$ and hence the box $\mathbf{x}$. This procedure can be applied for each k in turn to get an optimally reduced box. One can similarly proceed for separable constraints of the form $\sum q_k(x_{J_k}) \le \overline{a}$.

Again, in the separable case ($J_k = \{k\}$), computing the set of x_k with (13.35) is easy, especially for linear or quadratic q_k. If q_k is nonmonotonic, it may happen that the resulting set is disconnected; then one has to make a choice between taking its convex hull – which is an interval –, or of considering splitting the box into subboxes corresponding to the connected components.

In case of two-sided constraints $\sum q_k(x_{J_k}) \in \mathbf{a}$, which subsumes for $\mathbf{a} = q_0$ the equality constraint $\sum q_k(x_{J_k}) = q_0$, one can combine (13.30) and (13.32) using interval arithmetic.

4.13.2 Proposition. *Suppose that*

$$\mathbf{q}_k \supseteq \square\{q_k(x_{J_k}) \mid x_{J_k} \in \mathbf{x}_{J_k}\}, \quad \mathbf{r} \supseteq \mathbf{a} - \sum_k \mathbf{q}_k. \tag{13.36}$$

(i) If $0 \notin \mathbf{r}$ *then the conditions*

$$x \in \mathbf{x}, \quad \sum_k q_k(x_{J_k}) \in \mathbf{a} \tag{13.37}$$

are inconsistent.

(ii) Any x *satisfying* (13.37) *also satisfies*

$$q_k(x_{J_k}) \in \mathbf{r} \overset{\circ}{+} \mathbf{q}_k \quad \textit{for all } k. \tag{13.38}$$

Proof. (13.37) implies $0 \in \mathbf{a} - \sum q_k(x_{J_k}) \subseteq \mathbf{a} - \sum \mathbf{q}_k$, hence $0 \in \mathbf{r}$. Now suppose that $0 \in \mathbf{r}$. In the notation of the previous proposition we have

$$q_k(x_{J_k}) \in [\underline{a} - \overline{s} + \overline{q}_k, \overline{a} - \underline{s} + \underline{q}_k] = [\underline{a} - \overline{s}, \overline{a} - \underline{s}] \overset{\circ}{+} \mathbf{q}_k,$$

and since $\mathbf{r} = \mathbf{a} - [\underline{s}, \overline{s}] = [\underline{a} - \underline{s}, \overline{a} - \overline{s}]$, this implies (13.38). □

Again, condition (13.38) can be used to tighten $\mathbf{x}_{J_k}$ whenever

$$\mathbf{q}_k \not\subseteq \mathbf{r} \overset{\circ}{+} \mathbf{q}_k. \tag{13.39}$$

We give details for the most important case of quadratic (and including linear) functions, dropping indices for a moment.

4.13.3 Proposition. *Let* $\mathbf{c}$ *be an interval,* $a, b \in \mathbb{R}$*, and put*

$$\mathbf{d} := (b^2 + 4a\mathbf{c})_+, \quad \mathbf{w} := \sqrt{\mathbf{d}} \ (\textit{if } \mathbf{d} \neq \emptyset)$$

Then

$$\{x \in \mathbb{R} \mid ax^2 + bx \in \mathbf{c}\} = \begin{cases} \emptyset & \textit{if } \mathbf{d} = \emptyset, \\ \emptyset & \textit{if } a = b = 0 \notin \mathbf{c}, \\ \mathbb{R} & \textit{if } a = b = 0 \in \mathbf{c}, \\ \dfrac{\mathbf{c}}{b} & \textit{if } a = 0, \\ \dfrac{-b - \mathbf{w}}{2a} \cup \dfrac{-b + \mathbf{w}}{2a} & \textit{otherwise.} \end{cases}$$

Proof. $ax^2 + bx = \tilde{c} \in \mathbf{c}$ is equivalent to $x = \tilde{c}/b$ when $a = 0$, and to $x = (-b \pm \sqrt{b^2 + 4a\tilde{c}})/2a$ otherwise; in the latter case, the expression under the square root must be nonnegative and hence lies in $\mathbf{d}$. Since the varying $\tilde{c}$ occurs in these formulas only once, the range over $\tilde{c} \in \mathbf{c}$ is given by $\mathbf{c}/b$ if $a = 0$ and by $(-b \pm \sqrt{d})/2a$ otherwise (use monotonicity!). □

In the application to tightening boxes, one must of course intersect these formulae with the original interval. If the empty set results, the subproblem corresponding to the box $\mathbf{x}$ can be eliminated. (But remember to be cautious when using unrounded interval arithmetic!) If a disjoint union of two intervals results one either splits the box into two boxes corresponding to the two intervals or one leaves $\mathbf{x}_k$ unaltered; the first alternative is advisable only when the gap in the interval is quite large.

Note that the differences in Proposition 4.13.1 and the numerators in Proposition 4.13.3 may suffer from severe cancellation of leading digits, which requires additional attention in an actual implementation.

Consistency concepts. In constraint logic programming (see the book [Van Hentenryck *et al.*, 1997c] and the references at the beginning of this section), there are a number of consistency concepts that describe the strength of various tightening techniques. Essentially, a box is **consistent** with respect to a set of tightening procedures if their application does not reduce the box. A simple recursive argument invoking the finiteness of machine-representable boxes shows that every box can be reduced to a consistent box with finitely many applications of the tightening procedures in an arbitrary order. (Depending on the rules used, the resulting reduced box – called a **fixed point** of the tightening procedures – may or may not depend on the order of applying the rules.)

From a practical point of view, it is not advisable to apply the available rules until the fixed point is reached. The reason is that frequently the first few reductions are substantial, and later ones only reduce the box by tiny fractions. The convergence speed may be arbitrarily slow, as for example for the pair of constraints $x_1+x_2=0, x_1-qx_2=0$ with $q\in]0,1[$, where the unique fixed point (with respect to the simple tightening described above) reduces the volume in each step by a factor of q, which for q close to one is very inefficient compared to, say, a linear programming relaxation (which gives the result immediately).

Thus one has to be selective in practice, using suitable strategic rules for when to use which tightening strategy. The choice is usually done by various ad hoc recipes that balance the likely gain and the amount of work needed.

Semiseparable constraints. With some more work, the above techniques can be utilized also for semiseparable constraints. We need the following result.

4.13.4 Lemma. *If A is a rectangular matrix such that A^TA is nonsingular then*

$$|u_k| \le \sqrt{((A^TA)^{-1})_{kk}}\|Au\| \quad \textit{for all } u.$$

Proof. Let $A=QR$ be an orthogonal factorization of A with $Q^TQ=I$ and R square nonsingular. Then $A^TA=R^TR$ and $\|Au\|=\|Ru\|$. Since

$$|u_k| = |(R^{-T}e^{(k)})^TRu| \le \|R^{-T}e^{(k)}\|_2\|Ru\|_2,$$

$$\|R^{-T}e^{(k)}\|_2^2 = (e^{(k)})^TR^{-1}R^{-T}e^{(k)} = ((R^TR)^{-1})_{kk},$$

the assertion follows. □

Now suppose that we have a semiseparable inequality of the form

$$\sum q_k(x_k) + (x-x^0)^TH(x-x^0) \le \bar{a}, \tag{13.40}$$

with possibly nonsymmetric H. Using a modified Cholesky factorization $H+H^T=R^TR-D$ with a (nonnegative) diagonal matrix D, we can rewrite (13.40) as

$$0 \le \frac{1}{2}\|R(x-x^0)\|_2^2 \le \bar{a} - \sum q_k(x_k) + \frac{1}{2}(x-x^0)^TD(x-x^0). \tag{13.41}$$

The right hand side of (13.41) is a separable quadratic form, hence can be written as $\bar{a}-\sum\tilde{q}_k(x_k)$ with $\tilde{q}_k(x_k)=q_k(x_k)-\frac{1}{2}D_{kk}(x_k-x_k^0)^2$. Therefore, Proposition 4.13.1(ii) applies. Moreover, one gets the extra inequality $\|R(x-x^0)\|_2^2 \le 2(\overline{a}-\underline{s})$, which, together with the lemma gives the further inequalities

$$|x_k - x_k^0| \le \sqrt{2(\overline{s}-\underline{a})((R^TR)^{-1})_{kk}}, \tag{13.42}$$

which may help to tighten $\mathbf{x}_k$.

Block separable constraints. For only block-separable constraints ($|J_k| > 1$), the q_k are multivariate and one needs to resort to suboptimal interval techniques.

How to exploit the enclosures from Proposition 4.13.1 and 4.13.2 to tighten the box depends on the special form of E. In many cases, one can in turn solve directly for each variable involved, substituting an enclosing interval for all other variables.

If this is not possible directly one can use the mean value form (or another centered form) to rewrite a constraint $F_i(x) \in \mathbf{F}_i$ as

$$F_i(\xi) + F'(\mathbf{x})(x - \xi) \cap \mathbf{F}_i \neq \emptyset;$$

this is now a separable expression with interval coefficients that can be processed as above to tighten the box. This way of proceeding is called **conditioning** in [Van Hentenryck *et al.*, 1997c], and used in the Numerica [Van Hentenryck *et al.*, 1997c] package. Similarly, by using a Taylor expansion to second order with an interval Hessian, one gets a semiseparable expression with interval coefficients that can be processed in principle as above. (However, the interval coefficients cause here additional complications.)

4.14 Certificates of infeasibility

One may consider solving a global optimization problem as a **sequential nonlinear programming method** (SNLP), where local optimization (NLP) steps that improve a feasible point to local optimality alternate with tunneling steps that produce better (nearly) feasible points by some **tunneling procedure**. For systematic methods based on branching, the 'tunneling' is done by finding nearly feasible points during inspection of the subproblems.

The success of the tunneling step can be speeded up ([Dallwig *et al.*, 1997]) by solving on selected subboxes nonlinear least squares problems that minimize the sum of squares of the constraint violations, and (if a best feasible point with function value f^{best} is already available) the violation of $f(x) \leq f^{\text{best}} - \Delta$, where $\Delta \geq 0$ is some measure of minimal gain in function value. Alternatively, one may use the soft optimality theorem (Theorem 4.9.1) in place of least squares. (See also [Guddat *et al.*, 1990] for tunneling by continuation.) Thus, in a sense, the global optimization of (4.1) consists in solving a sequence of harder and harder feasibility problems

$$\begin{array}{ll} \text{find} & x \\ \text{s.t.} & x \in \mathbf{x}, \quad F(x) \in \mathbf{F}, \quad x_I \text{ integral}, \\ & f(x) \leq f^{\text{best}} - \Delta. \end{array} \tag{14.43}$$

Typical global optimization methods spend perhaps 5% of their time on finding a global minimizer, and the remaining 95% on the verification that there is no significantly better feasible point, i.e., showing that the feasibility problem (14.43) has no solution. Also, hard problems need a significant amount of time to find the first feasible point. Thus the initial and final (dominant) stages of a global optimization solution process are essentially identical with that for a feasibility problem.

In particular, general feasibility problems, also called **constraint satisfaction problems**, can be as hard as general global optimization problems, and the techniques needed for solving constraint satisfaction problems are essentially the same as those for solving global optimization problems.

If the functions defining the problem are continuously differentiable, bound constrained and linearly constrained problems can usually be solved, unless they have no feasible point. In the latter case, a **certificate of infeasibility** can be given that allows an easy check that this is the case. For linear constraints, the following result applies; in the formulation we use basic interval arithmetic (see, e.g., [Neumaier, 1990]).

4.14.1 Theorem. *The set of points satisfying*

$$x \in \mathbf{x}, \quad Ax \in \mathbf{b} \tag{14.44}$$

is empty if and only if there is a multiplier vector y such that

$$(y^T A)\mathbf{x} \cap y^T \mathbf{b} = \emptyset. \tag{14.45}$$

Proof. If x satisfies (14.44) then the left hand side of (14.45) contains $y^T Ax$ and hence is nonempty. Thus (14.45) implies that (14.44) cannot be satisfied. The converse is a simple consequence of the Lemma of Farkas and the fact [Neumaier, 1990, Section 3.1] that $a^T\mathbf{x} = \{a^T x \mid x \in \mathbf{x}\}$. □

Thus a certificate of infeasibility consists in a multiplier vector y satisfying (14.45), and is, e.g., a byproduct of phase 1 of a simplex algorithm. Similar certificates are available for convex constraints. If there are nonconvex nonlinear constraints, there are simple certificates for infeasibility of

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

such as a multiplier vector y with

$$y^T F(\mathbf{x}) \cap y^T \mathbf{F} = \emptyset, \tag{14.46}$$

where $F(\mathbf{x})$ is an interval evaluation of F at the box $\mathbf{x}$, or

$$y^T F(\xi) + (y^T F'(\mathbf{x}))(\mathbf{x} - \xi) \cap y^T \mathbf{F} = \emptyset, \tag{14.47}$$

where $F'(\mathbf{x})$ is an interval evaluation of the Jacobian F' at the box $\mathbf{x}$. Similarly, if already a feasible point with objective function value f^{best} is known then a multiplier vector y with

$$\min_{x \in \mathbf{x}} (f(x) + y^T F(x)) > f^{\text{best}} + \sup y^T \mathbf{F} \tag{14.48}$$

is a certificate of infeasibility for a global minimizer, and the left hand side can be bounded from below by interval evaluation or a centered form, giving a verifiable sufficient condition.

But unlike in the linear case, there is no guarantee that such a certificate exists (or can be found easily if it exists). Moreover, local solvers may fail because they are not able to find a feasible point, even if one exists. Indeed, finding a feasible point is in the latter case already a global problem that cannot be handled by local methods.

Good certificates of infeasibility of the form (14.47) are, however, available for small boxes not too close to the feasible domain. This follows from the linear approximation property of interval arithmetic. A suitable multiplier vector y can be obtained in this case from the linearized problem, or ([Schichl and Neumaier, 2001]) from a more detailed analysis of the output of a tunneling optimization.

Thus, in combination with branching, we can certify the nonexistence of solution in a covering of almost all of the initial box.

4.15 The role of local optimization

Local optimization routines are an important part of most global solvers. As we have seen, they are used for two different purposes:

(i) to find feasible points if the feasible domain has a complicated definition, and to find better local minimizers when (after successful tunneling) a feasible point better than the previously best local minimizer has been found;

(ii) to solve auxiliary optimization problems such as relaxations of the original problem (for generating improved bounds) or bound constrained approximations (for tunneling).

We discussed tunneling already in Section 4.14. The other kind of auxiliary local optimization problems to be solved are simpler in structure since they 're-lax' the problem in some way. A **relaxation** is a modification of the original problem whose solution is tractable and gives some information about the possible location of the global minimizer. In the past, mainly linear and convex relaxation have been used, since for these, local optimization provides global solutions, which usually implies useful global information about the original problem. More recently, nonconvex quadratic relaxations were found useful, too. We shall discuss various ways of obtaining and using relaxations in Section 4.20.

Considerations of superlinear convergence of local optimization algorithms imply that one generally uses **sequential quadratic programming** (SQP) techniques, that solve a sequence of related quadratic programs whose solution converges (under certain conditions, cf. below) to a local minimizer of the original problem; if the starting point is feasible (which, initially, need not be the case), the function value of the local minimizer is at or below that of the starting point.

To give the reader a rough idea of times and difficulties, here are some (completely unreliable but catchy) rules of thumbs. If the time needed to solve a linear program of a certain size is LP then solving a problem of comparable size may take perhaps

$$QP = 5 * LP$$

for a convex quadratic program,

$$QP' = 10 * LP$$

for a local minimizer of a nonconvex quadratic program,

$$SQP = 30 * QP$$

for a convex nonlinear program,

$$SQP' \geq 200 * QP$$

for a local minimizer of a nonconvex nonlinear program,

$$GLP_f \geq 100 * SQP$$

for *finding* a global minimizer of a nonconvex nonlinear program, and

$$GLP_v \geq 1000 * SQP$$

for *verifying* that it is a global minimizer.

We now comment on the needed properties of local optimization software. Usually, is is more important that the local solver is fast than that it is very robust (i.e., guaranteed to succeed), since lack of robustness in some of the local optimizations is made up for by the structure of the global solution process. To help control the amount of work done in the local part, it should be possible to force a premature return with a less than optimal point when some limit (of time or number of function values) is exceeded. Nevertheless, the local solver should be good to ensure that solving a problem with a unique minimizer (which is automatically global) by the global solver does not take much longer than a good local solver would need.

Modern nonlinear programming codes are usually "globally convergent" in some sense. The global convergence proofs (to a **local** minimizer only!) usually make more or less stringent assumption that imply the absence of difficulties in finding feasible points. Formally, we may say that a local optimization algorithm is **globally convergent** if there is a continuous function $d_{\text{feas}} : \mathbb{R}^n \to \mathbb{R}$ (defining a 'distance to feasibility') such that

$$d_{\text{feas}}(x) \geq 0, \quad \text{with equality iff } x \text{ is feasible}$$

and the algorithm produces for arbitrary continuous problems and arbitrary starting points a sequence of $x^l \in \mathbb{R}^n$ satisfying one of the following conditions:
(i) x^l converges to the set of points satisfying the Karush-John conditions (and, possibly, second order necessary conditions);
(ii) $d_{\text{feas}}(x^l) \to 0$ and $f(x^l) \to -\infty$;
(iii) x_l converges to the set of points where the objective or some constraint function is not continuously differentiable;
(iv) $d_{\text{feas}}(x^l) \to 0$, $\|x^l\| \to \infty$;
(v) $d_{\text{feas}}(x^l)$ converges to a nonzero local minimum of d_{feas}.

Conditions (i) and (ii) characterize the achievement of the optimization goal, while conditions (iii)–(v) characterize various modes of unavoidable failure. Failures of type (iii) or (iv) are usually attributed to bad modeling or bad choice of the optimization methods. Some methods such as bundle methods can cope with lack of differentiability hence do not lead to case (iii).

A failure of type (v) is unavoidable if there is no feasible point. However, failures of type (v) may happen for problems with nonconvex constraints even though feasible points exist. One could say that from a local point of view, an optimization problem is **easy** (for an algorithm) if (v) cannot occur whenever a feasible point exists. A local algorithms may be considered as good if among its easy problems are all problems with convex constraints only, and all problems satisfying certain strong versions [Burke, 1991] of the Mangasarian-Fromovitz [Mangasarian and Fromovitz, 1967] constraint qualification (See also Chapter 2.4). Ideally, a good local algorithm would provide in these cases a certificate of infeasibility whenever it detects case (v).

4.16 The cluster problem and second-order information

When programming a simple branch and bound algorithm for global optimization, one quickly notices that it is fairly easy to eliminate boxes far away from the global minimizer, while, especially in higher dimensions, there remains a large cluster of tiny boxes in a neighborhood of the global minimizer that is difficult to eliminate. The occurence of this situation is called the **cluster problem**. Often, algorithms try to avoid the cluster problem by providing only a Δ-optimal solution; i.e., the program stops when it has shown that there is no feasibel point with an objective function value of $f^{\text{best}} - \Delta$, where f^{best} is the function value of the best feasible point found so far. However, when Δ is small (as one wants to have it) then the cluster problem is still present, although to a less pronounced degree.

[Kearfott and Du, 1994] studies the cluster problem for unconstrained global optimization, and discovered that the source of the problem was the limited accuracy with which the function values were bounded. In particular, they showed that the cluster problem disappears if, for x in a box of diameter $O(\varepsilon)$, one can bound the overestimation of $f(x^{\text{best}}) - f(x)$ by $O(\varepsilon^3)$. Here we give a simplified version of his result.

Let $\hat{x} \in \mathbb{R}^n$ be a global minimizer of $f(x)$, and let $\hat{G}$ be the Hessian at $\hat{x}$. Near the global minimizer, we have

$$f(x) = \frac{1}{2}(x - \hat{x})^T \hat{G}(x - \hat{x}) + O(\|x - \hat{x}\|^3).$$

Suppose we can bound the objective function value over a box of diameter ε with an accuracy of $\Delta = K\varepsilon^{s+1}$, $s \leq 2$. Then no box of diameter ε containing a point x with $\frac{1}{2}(x - \hat{x})^T \hat{G}(x - \hat{x}) + O(\|x - \hat{x}\|^3) \leq \Delta$ can be eliminated. For sufficiently small Δ, this describes a nearly ellipsoidal region with volume proportional to $(2\Delta)^{n/2}/\det G$, and any covering by boxes of diameter ε contains at least $const \cdot (2\Delta)^{n/2}/(\varepsilon^n \det G)$ boxes. The number of uneliminated boxes is therefore proportional to at least

$$\begin{array}{ll} \varepsilon^{-n/2} & \text{if } s+1=1, \\ const^{n/2}/\det G & \text{if } s+1=2, \\ (const \cdot \varepsilon)^{n/2}/\det G & \text{if } s+1=3. \end{array}$$

We see that for $s+1=0$, the number grows immensely as ε gets small. For $s+1=1$, the number of boxes needed, while (for small ε) essentially independent of ε may still grow exponentially with the dimension, and is especially large for problems where the Hessian at the solution is ill-conditioned. However, the number is guaranteed to be small (for small ε) when $s+1=2$.

For constrained global optimization, a similar argument applies in a reduced manifold, with the result that in the formulas, n must be replaced by $n-a$, where a is the maximal number of constraints active at the solution, with linearly independent constraint gradients.

Clearly, to bound the overestimation over a box of diameter $O(\varepsilon)$ by $O(\varepsilon^3)$ requires that one knows the Hessian up to $O(\varepsilon)$, and to be able to bound the deviation from a quadratic model. Thus it is necessary to have access to second-order information. Unfortunately, in more than one dimension, no method is known that bounds function values over an arbitrary narrow box of diameter $O(\varepsilon)$ by $O(\varepsilon^3)$. (In a single dimension, this is possible; see [Cornelius and Lohner, 1984] and [Neumaier, 1990, Section 2.4].)

Fortunately, however, it turns out that using interval Hessian matrices (which, for 3-times differentiable functions have the requires $O(\varepsilon)$ accuracy, see [Neumaier, 1990, Section 1.4]), there are several ways to avoid the cluster problem, at least when the global minimizer is nondegenerate, i.e., satisfies the second-order sufficient conditions for a local minimizer.

Explicit global Hessian information can be used, as in GLOBSOL [Kearfott, 1996b] and NUMERICA [Van Hentenryck *et al.*, 1997c], by interval Newton methods (see Section 4.12 applied to the Karush-John conditions discussed in Section 4.5. These may verify the existence of a unique solution of the Karush-John conditions in some box around the best point found, and hence allow to shrink that box to a single point. (Since the Karush-John conditions are homogeneous in the multipliers, the condition that κ and y are not both zero may be represented as the constraint $\kappa^2+y^Dy=1$, where D is an arbitrary diagonal matrix with positive entries.)

Alternatively, one may use global Hessian information to verify the second-order sufficient conditions for a global minimizer given in [Neumaier, 1996]. They apply to smooth nonlinear programs of the form

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in \mathbf{x}, \quad F(x) = 0. \end{array} \tag{16.49}$$

Thus it is necessary to introduce slack variables to rewrite general inequality constraints as equality constraints.

4.16.1 Theorem. *Let $\hat{x}$ be a Kuhn-Tucker point for the nonlinear program* (16.49), *with associated multiplier z, and let*

$$y := f'(\hat{x})^T - F'(\hat{x})^T z, \tag{16.50}$$

$$D = \operatorname{Diag}\left(\sqrt{\frac{2|y_1|}{\overline{x}_1 - \underline{x}_1}}, \dots, \sqrt{\frac{2|y_n|}{\overline{x}_n - \underline{x}_n}}\right). \tag{16.51}$$

If, for some continuously differentiable function $\varphi : \mathbb{R}^m \to \mathbb{R}$ with

$$\varphi(0) = 0, \quad \varphi'(0) = z^T, \tag{16.52}$$

the **generalized augmented Lagrangian**

$$\hat{L}(x) := f(x) - \varphi(F(x)) + \frac{1}{2}\|D(x-\hat{x})\|_2^2 \tag{16.53}$$

is convex in $[u,v]$, then $\hat{x}$ is a global solution of (16.49). *Moreover, if $\hat{L}(x)$ is strictly convex in $[u,v]$, this solution is unique.*

A choice for φ that works in some neighborhood of a strong (=nondegenerate) global minimizer is given in [Neumaier, 1996], together with further implementation hints. The convexity can be checked by means of interval arithmetic; see Section 4.12. If these conditions hold in some box, one can again shrink this box to a single point.

One can use any of these techniques to construct boxes $\mathbf{y}$ that are guaranteed to contain no global minimizer except if already detected, resulting in exclusion constraints. An **exclusion constraint** is a constraint of the form

$$x \notin \mathbf{y}.$$

It can be used to reduce an arbitrary box $\mathbf{x}$ by intersecting it with $\mathbf{y}$ and taking the interval hull, which may result in a smaller box. If there was no reduction but the intersection is strictly contained in $\mathbf{x}$, one may also want to resort to multisection, cf. (19.93). Interesting exclusion boxes are those that are constructed around local minimizers, since this helps fighting the cluster problem.

Whenever we have a tentative approximate global minimizer $\tilde{x}$, we try to find simultaneously a large box $\mathbf{x}$ and a tiny box $\mathbf{z}$ such that any global minimizer $\hat{x} \in \mathbf{x}$ satisfies $\hat{x} \in \mathbf{z}$. This allows to use $\mathbf{x}$ as an exclusion region while $\mathbf{z}$ is stored in an output list as a box containing a putative minimizer. (After terminating the branching process, these boxes need to be checked again for possible elimination.)

Since we expect that $\tilde{x}$ has a function value optimal within $O(\epsilon)$, but knowing that this only enforces that $\tilde{x}$ has an accuracy of $O(\sqrt{\epsilon})$ (possibly less in case of singular Hessians), we start with a box

$$\mathbf{x} = [\tilde{x} - \sqrt{\epsilon}u, \tilde{x} + \sqrt{\epsilon}u]$$

for some suitable vector u, and apply the available reduction techniques until no significant improvement results. Call the resulting box $\mathbf{z}$. If second-order techniques are used to do the box reduction, then z is usually a tiny box or empty.

If $\mathbf{z}$ is empty, $\tilde{x}$ was not a good approximation but we know that $\mathbf{x}$ contains no solution. If $\mathbf{z}$ is nonempty, it is likely that $\mathbf{z}$ contains a solution. Indeed this is always the case if *only* interval Newton-like reduction techniques are used and $\mathbf{z} \subseteq \operatorname{int} \mathbf{x}$. (This requires some qualifying conditions that ensure that one can verify boundary-freeness conditions such as those in [Neumaier, 1990, Chapter 5.3-4].) Thus one may store $\mathbf{z}$ in the list of output boxes together with a flag whether existence (and possibly uniqueness) was verified.

If $\mathbf{z}$ is still a box of significant size, we must have been close to a degeneracy; splitting would probably not improve this and lead to an exponential number of boxes; thus it is preferable to put this box also in the list of output boxes to indicate that a low resolution candidate for a solution has been found. (This way of handling degeneracies is due to [Kearfott, 1996a].)

No matter what case we have been in, we always know that $\mathbf{x}$ does not contain a solution not yet in the output list. Therefore, we may add the exclusion constraint $x \notin \mathbf{x}$ to the problem description. However, one can often make $\mathbf{x}$ even bigger. So we try recursively

$$\mathbf{x}^0 = \mathbf{x},\ \mathbf{z}^0 = \mathbf{z} \quad \text{but } \mathbf{z}^0 = \mathrm{mid}(x) \text{ if } \mathbf{z} = \emptyset,$$

$$\mathbf{x}^{l+1} = 2\mathbf{x}^l \overset{\circ}{-} \mathbf{z}^l,\ \mathbf{z}^{l+1} = \mathrm{reduce}(\mathbf{x}^{l+1});$$

using the available ways of reducing $\mathbf{x}^l$ until $\mathbf{z}^{l+1} \subseteq \mathbf{x}^l$ or $\mathbf{z}^{l+1} = \mathbf{x}^{l+1}$. Then we have the generally stronger new exclusion constraint $x \notin \mathbf{x}^l$. (This way of generating exclusion constraints, using interval Newton methods, is due to [Van Iwaarden, 1986], who calls the technique **backboxing**, and is part of GLOBSOL [Kearfott, 1996b].)

4.17 Semilinear constraints and MILP

Let us call a constraint **semilinear** if, for arguments x in a *bounded* box $\mathbf{x}$, it is equivalent to a finite list of linear constraints and integer constraints; usually the latter involve additional auxiliary variables. The objective function $f(x)$ called **semilinear** if the inequality $f(x) \le x_0$, where x_0 is an additional variable, is semilinear. A **semilinear program** is an optimization problem with a semilinear objective function and a bounded feasible domain defined by semilinear constraints only. Since we can rewrite an arbitrary global optimization problem

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in C \end{array}$$

in the form

$$\begin{array}{ll} \min & x_0 \\ \text{s.t.} & x \in C,\ f(x) \le x_0, \end{array}$$

it is clear from the definition that any semilinear program can be rewritten as a mixed integer linear program by the introduction of additional variables.

The remarkable fact that every factorable optimization problem can be arbitrarily closely approximated by a sequence of semilinear programs (see Section 4.18) implies that one can use MILP software to obtain arbitrarily good approximate solutions of factorable optimization problems. To make this observation computationally useful we need to handle two tasks:

(i) Find interesting classes of semilinear constraints and constructive procedures for translating such constraints into linear and integer constraints; see Section 4.18.

(ii) Show how to approximate factorable constraints by semilinear constraints.

In this section we look at task (i). All linear constraints and integer constraints are trivially semilinear. A **binary constraint**

$$z \in \{0,1\}$$

is semilinear since it can be written in the equivalent form

$$z \in [0,1], \quad z \text{ integral.}$$

A list x_K of variables constrained by

$$\sum_{k \in K} x_k = 1, \quad x_k \in \{0,1\} \ (k \in K), \tag{17.54}$$

where K is some index set, is called a **special ordered set** (SOS). Clearly, the constraint

$$x_K \text{ is a SOS} \tag{17.55}$$

is also semilinear. Because (17.54) can hold only if all but one of the x_k $(k \in K)$ vanish, (17.55) is equivalent to requiring that

$$x_K = e^{(k)} \quad \text{for some } k, \tag{17.56}$$

where $e^{(k)}$ is the unit vector with a one in position k and zeros elsewhere. (A special ordered set of size two is just a pair of complementary $(0,1)$-variables, and one of its variables is redundant.) Since special ordered set are ubiquitous in MILP formulations, any MILP solver has special facilities to make efficient use of special ordered sets.

Many techniques for translating semilinear constraints are consequences of the following basic result.

4.17.1 Theorem. *Let $F_k : C_0 \to \mathbb{R}^{m_k}$ $(k = 1,\dots,d)$ be scalar- or vector-valued functions such that*

$$F_k(x) \geq \underline{F}_k \quad \textit{for all } x \in C_0 \tag{17.57}$$

with finite $\underline{F}_k \in \mathbb{R}^{m_k}$. Then there is a point $x \in C_0$ such that

$$F_1(x) \geq 0 \ \vee \ \dots \ \vee \ F_d(x) \geq 0 \tag{17.58}$$

if and only if there are $z \in \mathbb{R}^d$ and $x \in C_0$ such that

$$z \quad \textit{is a SOS}, \tag{17.59}$$

$$F_k(x) \geq \underline{F}_k(1 - z_k) \quad \textit{for all } k = 1,\dots,d. \tag{17.60}$$

(The symbol $\vee$ denotes the logical operation **or**. The operation **and** is simply given by the comma, and we follow the convention that the comma is binding stronger than $\vee$.)

Proof. If (17.58) holds then $F_k(x) \geq 0$ for some k, and $z = e^{(k)}$ satisfies (17.59) and (17.60). Conversely, if (17.59) holds then $z = e^{(k)}$ for some k, and (17.60) implies $F_k(x) \geq 0$; the other constraints in (17.60) are automatically satisfied because of (17.57). □

Note that (17.57) can always be satisfied if C_0 is bounded and the F_k are continuous.

A constraint of the form (17.58) is called a **disjunctive constraint**. The theorem implies that **linear** disjunctive constraints, where all $F_k(x)$ are affine functions of x, are semilinear, since then (17.60) consists of linear constraints.

More generally, linear disjunctive constraints of the form

$$A_1 x \in \mathbf{b}_1 \quad \vee \quad \dots \quad \vee \quad A_d x \in \mathbf{b}_d \tag{17.61}$$

are semilinear since we can rewrite each $A_k x \in \mathbf{b}_k$ in the form

$$\begin{pmatrix} A_k x - \underline{b}_k \\ \bar{b}_k - A_k x \end{pmatrix} \geq 0.$$

Note that we can rewrite (17.61) in the equivalent form

$$A_k x \in \mathbf{b}_k \quad \text{for some } k \in \{1 : d\}. \tag{17.62}$$

Since many practically relevant constraints can be cast in the form (17.61), this makes the theorem a very useful tool for recognizing semilinear constraints and translatng them into a MILP formulation. (There is also an extended literature on disjunctive programming not based on transformations to MILP; for pointers see [Jeroslov, 1977, Sherali and Shetty, 1980].)

For example, **semicontinuous (semiinteger) variables** are variables x_k constrained by

$$x_k = 0 \quad \vee \quad x_k \in \mathbf{a} \tag{17.63}$$

and

$$x_k = 0 \quad \vee \quad x_k \in \mathbf{a}; \quad x_k \text{ integral}, \tag{17.64}$$

respectively, which are semilinear constraints.

A **special ordered set (SOS) of type 2** is a vector $\lambda \in \mathbb{R}^d$ satisfying

$$\lambda \geq 0, \quad \sum_{k=1}^{d} \lambda_k = 1,$$

at most two λ_k with adjacent indices are nonzero.

Since the latter constraint can be formulated as

$$\lambda_k + \lambda_{k+1} = 1 \quad \text{for some } k,$$

it is disjunctive; hence the constraint

$$x_K \text{ is a SOS of type 2} \tag{17.65}$$

is semilinear. Many MILP codes have special facilities for the efficient handling of SOS of type 2.

An **exclusion constraint** of the form

$$x \notin \text{int}\mathbf{x}. \tag{17.66}$$

where x is a box, is semilinear since it is a disjunction of the constraints

$$x_k \leq \underline{x}_k \quad \vee \quad x_k \geq \overline{x}_k.$$

Conditional linear constraints of the form

$$Ax \in \mathbf{a} \quad \text{if } Bx < b \tag{17.67}$$

are semilinear since (17.67) is equivalent to

$$Ax \in \mathbf{a} \quad \vee \quad (Bx)_1 \geq b_1 \quad \vee \quad \ldots \quad \vee \quad (Bx)_d \geq b_d,$$

where d is the number of rows of B and b. (Conditional linear constraints with $=$ or $\leq$ in place of $<$ in (17.67) are apparently not semilinear in general since their disjunctive form contain strict inequalities, which – according to our definition – are not regarded as linear constraints. However, conditional linear constraints where the condition involves integer variables only are semilinear since the condition can be replaced by an equivalent strict inequality conditions.)

Certain **minimum and maximum constraints** are also semilinear. A constraint of the form

$$a^T x \leq \min_{i=1:d} (Ax - b)_i \tag{17.68}$$

is equivalent to the linear constraints

$$a^T x \leq (Ax - b)_i \quad \text{for } i = 1 : d.$$

The reverse constraints

$$a^T x \geq \min_{i=1:d} (Ax - b)_i \tag{17.69}$$

is equivalent to the linear disjunctive constraint

$$a^T x \geq (Ax - b)_1 \quad \vee \quad \ldots \quad \vee \quad a^T x \geq (Ax - b)_d.$$

Similarly, a constraint of the form

$$a^T x \geq \max_{i=1:d} (Ax - b)_i \tag{17.70}$$

is equivalent to the linear constraints

$$a^T x \geq (Ax - b)_i \quad \text{for } i = 1 : d,$$

and the reverse constraint

$$a^T x \leq \max_{i=1:d} (Ax - b)_i \tag{17.71}$$

is equivalent to the linear disjunctive constraint

$$a^T x \leq (Ax - b)_1 \quad \vee \quad \ldots \quad \vee \quad a^T x \leq (Ax - b)_d.$$

The constraints

$$a^T x = \min_{i=1:d} (Ax - b)_i, \tag{17.72}$$

$$a^T x = \max_{i=1:d} (Ax - b)_i \tag{17.73}$$

are also semilinear, since they are equivalent to (17.68), (17.69) and (17.70), (17.71), respectively. In particular, **linear complementarity constraints** [Billups and Murty, 2000]

$$\min(a^T x - \alpha, b^T x - \beta) = 0 \tag{17.74}$$

and constraints of the form

$$a^T x - \alpha \leq |b^T x - \beta| \tag{17.75}$$

$$a^T x - \alpha = |b^T x - \beta|, \tag{17.76}$$

$$a^T x - \alpha \geq |b^T x - \beta| \tag{17.77}$$

are semilinear; the latter since we can write the absolute value in the form $|b^T x - \beta| = \max(\beta - b^T x, b^T x - \beta)$. Their MILP reformulation needs a single binary variable only since the associated SOS has size two. (Linear complementarity constraints arise in bilevel programming, see, e.g., [Ben-Ayed, 1993, Luo *et al.*, 1996, Outrata *et al.*, 1998, Shimizu *et al.*, 1997, Vicente and Calamai, 1994], in which the inner optimization problem is a linear program.)

Certain other **piecewise linear constraints** are also semilinear. Of particular interest are those of the form

$$a^T x \le q(x_i), \tag{17.78}$$

where q is a continuous, piecewise linear function of a *single* variable with a finite number of derivative discontinuities. Let $\xi_0 < \xi_1 < \cdots < \xi_d$ be a list of **nodes** such that $x_i \in [\xi_0, \xi_d]$ and q is linear in each interval $[\xi_{k-1}, \xi_k]$. Then

$$q(\xi) = q_k + q'_k(\xi - \xi_k) \quad \text{for } \xi \in [\xi_{k-1}, \xi_k], \tag{17.79}$$

where

$$q_k = q(\xi_k), \quad q'_k = \frac{q(\xi_k) - q(\xi_{k-1})}{\xi_k - \xi_{k-1}}.$$

Therefore, (17.78) can be rewritten as a disjunction of the d constraints

$$\xi \in [\xi_{k-1}, \xi_k], \quad a^T x \le q_k + q'_k(\xi - \xi_k)$$

for $k = 1, \ldots, d$. Since these are linear constraints, (17.78) is equivalent to a linear disjunctive constraint. The constraints

$$a^T x \ge q(x_i), \tag{17.80}$$

$$a^T x = q(x_i), \tag{17.81}$$

are semilinear by the same argument, with $\ge$ or $=$ in place of $\le$.

Piecewise linear constraints may also be modelled by SOS of type 2; see [Beale, 1988, Section 10.3] and [Beale and Forrest, 1976, Beale and Tomlin, 1970, Dantzig *et al.*, 1958, Tomlin, 1970]. Indeed, if $q(x)$ is piecewise linear with nodes $\xi_{1:d}$ and corresponding function values $q_k = q(x_k)$ then we may write an arbitrary argument x as

$$x = \sum \xi_k \lambda_k, \quad \lambda \text{ is a SOS of type 2}, \tag{17.82}$$

where λ is a SOS of type 2, and find

$$q(x) = \sum q_k \lambda_k.$$

Therefore, if we add the semilinear constraints (17.82), we may replace each occurence of $q(x)$ by $\sum q_k \lambda_k$. This even works for unbounded variables, and for general separable constraints $\sum q_l(x_l) \in \mathbf{a}$ with piecewise linear q_l. Many modern MILP programs have special features that allow them to handle piecewise linear constraints using special ordered sets of type 2.

Many combinatorial constraints are semilinear. For example, **all-different constraints** of the form

$$\text{the components of } x_K \text{ are distinct integers} \tag{17.83}$$

are semilinear, since we can rewrite them as

$$x_k \text{ integral for } k \in K; \quad |x_j - x_k| \geq 1 \quad \text{for } j, k \in K.$$

A **cardinality constraint**

$$\text{there are exactly } s \in \mathbf{s} \text{ nonzero } x_k \ (k \in K)$$

is semilinear if we know that x_K is integral and nonnegative. Indeed, an equivalent condition is the existence of binary numbers z_k such that

$$\begin{aligned} z_k &= 1 \quad \text{if } x_k > 0, \\ z_k &= 0 \quad \text{if } x_k < 1, \\ \sum_{k \in K} & z_k \in \mathbf{s}, \end{aligned}$$

and these are semilinear constraints.

4.18 Approximation by semilinear programs

The preceding results are of importance for factorable global optimization problems since one can use them to approximate the latter by semilinear programs. The ideas go back to [Markowitz and Manne, 1957] and [Dantzig, 1960] for approximate separable nonconvex programming using piecewise linear constraints. (A Lagrangian method by [Falk and Soland, 1969] gives piecewise linear relaxations, but in general, these do not yield arbitrarily good approximations.) With a trick due to [Pardalos and Rosen, 1987] (for the special case of indefinite quadratic programs) that allows to transform multivariate quadratic expressions into separable form, everything extends easily to the semiseparable case; see (18.86) below. For indefinite quadratic programs, this is discussed in detail in [Horst *et al.*, 1995]. The general factorable case, but with weaker outer approximations for bilinear terms, that do not guarantee convergence of the outer approximation, is already in [McCormick, 1976].

With a suitable reformulation, arbitrary factorable optimization problems (and many nonfactorable ones) can be rewritten in such a way that the objective function is linear and all constraints are either semilinear, or of the form (17.78), (17.80), (17.81) with continuous functions φ of a single variable. To see this, we introduce an auxiliary variable for every intermediate result; then the objective function is just a variable, hence linear, and the constraints are simple constraints or equations involving a single operation only,

$$x_k = \varphi(x_i), \tag{18.84}$$

$$x_k = x_i \circ x_j \quad (\circ \in \{+,-,*,/,\hat{}\}). \tag{18.85}$$

Equations (18.84) have the form (17.81), and equations (18.85) are linear if $\circ \in \{+,-\}$. For $\circ = /$, we get equivalent constraints $x_i = x_k x_j$, and for $\circ = \hat{}$ (the power), we can rewrite the constraint as

$$y_k = x_j y_i, \quad y_k = \log x_k, \quad y_i = \log x_i.$$

(Powers with constant exponents are treated as a case of (18.84).) It remains to consider products. But $x_k = x_i x_j$ is equivalent to

$$\alpha x_i + \beta x_j = u, \quad \alpha x_i - \beta x_j = v,$$

$$w = v^2, \quad w + 4\alpha\beta x_k = u^2,$$

for arbitrary α, $\beta \pm 0$. The first two are linear constraints in x_i, x_j, u, v, and the others are of the form (17.81). This proves that the reformulation can always be done. However, it is clear that in most cases many fewer intermediate variables need to be introduced since affine expressions $a^T x + \alpha$ can be left intact, as can all expressions depending only on a single variable. Moreover, as we shall see in a moment, quadratic and bilinear expressions can be handled more efficiently.

Therefore, it is advisable to do in a first step only those substitutions needed to transform the problem such that the new objective function $f(x) =: F_0(x)$ and the components $F_i(x)$ $(i = 1 : m)$ of the new constraint function vector $F(x)$ are **semiseparable**, i.e., of the form

$$F_i(x) = \sum_{(j,k)\in K_i} \varphi_j(x_k) + x^T H_i x + c_i^T x + \gamma_i \quad (i = 0 : m) \tag{18.86}$$

with nonlinear univariate functions φ_{ik} and (generally extremely sparse) matrices H_i. Note that linear terms may be absorbed into the sum, and quadratic and bilinear terms into $x^T H_i x$.

In a second step, the quadratic terms are rewritten as a weighted sum of squares,

$$x^T H_i x = \frac{1}{2} \sum_{j\in J_i} d_j (r_j^T x)^2. \tag{18.87}$$

This is always possible, usually in many ways; e.g., by a symmetric indefinite factorization

$$H_i + H_i^T = R^T D R, \quad D \text{ diagonal},$$

which gives

$$2x^T H_i x = (Rx)^T D(Rx) = \sum D_{kk}(Rx)_k^2.$$

(For numerical stability one needs to take care of scaling issues, to ensure that no unavoidable cancellation of significant digits takes place.) Using (18.87) and substituting new variables for the $r_j^T x$, we see that we can achieve in this second step the **separable form**

$$F_i(x) = \sum_{(j,k)\in K_i} \varphi_j(x_k) + c_i^T x + \gamma_i \quad (i = 0:m) \tag{18.88}$$

with increased K_i. Constraints of the form $F_i(x) \le \bar{F}_i$ are now replaced by

$$\sum_{(j,k)\in K_i} y_j + c_i^T x + \gamma_i \le \bar{F}_i,$$

$$y_j \ge \varphi_j(x_k) \quad \text{for} \quad (j,k) \in K_i,$$

and similarly for the objective function. Constraints of the form $F_i(x) \ge \underline{F}_i$ are replaced by

$$\sum_{(j,k)\in K_i} y_j + c_i^T x + \gamma_i \ge \underline{F}_i,$$

$$y_j \le \varphi_j(x_k) \quad \text{for } (j,k) \in K_i,$$

and similarly for the objective function. And two-sided constraints $F_i(x) \in \mathbf{F}_i$ with finite $\mathbf{F}_i$ are replaced by

$$\sum_{(j,k)\in K_i} y_j + c_i^T x + \gamma_i \in \mathbf{F}_i,$$

$$y_i = \varphi_j(x) \quad \text{for} \quad (j,k) \in K_i.$$

Thus, in this third step, the required form has been achieved, and generally much more parsimoniously. (A few more variables could be saved by leaving in each nonlinear $F_i(x)$ one of the nonlinear terms unsubstituted.)

So far, no approximation was done; the reformulated problem is equivalent to the original one. In a final **approximation step**, constraints of the form (17.78), (17.80), (17.81) are replaced by piecewise linear constraints. If (as traditionally done [Beale and Forrest, 1976]) just an approximation is desired, one simply uses in place of φ the piecewise linear function obtained by interpolating some function values of φ.

However, with little more work only, **outer approximations** can be constructed if we have two piecewise linear approximations $\underline{\varphi}$, $\bar{\varphi}$ with the same nodes $\xi_0 < \cdots < \xi_d$, satisfying

$$\underline{\varphi}(\xi) \le \varphi(\xi) \le \bar{\varphi}(\xi) \quad \text{for} \quad \xi \in [\xi_0, \xi_d]. \tag{18.89}$$

To get $\xi_0 = \underline{x}_i$ and $\xi_d = \bar{x}_i$, one needs good bounds $\mathbf{x}_i$ on x_i, which can usually be calculated by constraint propagation (see Section 4.13).

$\underline{\varphi}$ and $\bar{\varphi}$ can be found by exploiting convexity properties of φ, which are well-known for elementary functions and can be determined with interval analysis for factorable univariate functions. Given (18.89), the constraint (17.78) *implies* (and not only approximates) the semilinear constraints

$$\xi \in [\xi_{k-1}, \xi - k], \quad a^T x \leq \bar{\varphi}_k + \bar{\varphi}'_k(\xi - \xi_k) \quad \text{fore some } k,$$

the constraint (17.80) implies the semilinear constraints

$$\xi \in [\xi_{k-1}, \xi_k], \quad \underline{\varphi}_k + \underline{\varphi}'_k(\xi - \xi_k) \leq a^T x \quad \text{fore some } k,$$

and the constraints (17.81) implies the semilinear constraints

$$\xi \in [\xi_{k-1}, \xi_k], \quad \underline{\varphi}_k + \underline{\varphi}'_k(\xi - \xi_k) \leq a^T x \leq \bar{\varphi}_k + \bar{\varphi}'_k(\xi - \xi_k) \quad \text{fore some } k.$$

Moreover, by making the gap between the bounds in (18.89) sufficiently small, the approximation by these semilinear constraints becomes better and better.

As one can see, the complexity of the resulting MILP formulation depends on the number of nonlinear operations (but in a problem-dependent fashion because of the quadratic bilinear terms), and grows linearly with the number of nodes used in the piecewise linear approximation. Hence it is an efficient technique only if the number of nonlinear operations is not too large, and the approximation not too close.

4.19 The branch and bound principle

The branch and bound principle is a general label (invented in [Land and Doig, 1960, Little *et al.*, 1963]) to denote methods to split a problem recursively into subproblems which are sooner or later eliminated by showing that the subproblem cannot lead to a point better than (or as least as good as) the best point found so far. The latter is often checked by computing lower bounds on the objective function, and the splitting produces new branches in the tree of all subproblems tried, according to so-called **branching rules**; hence the name "branch and bound". But in practice, the subproblems are best treated in a more flexible fashion, allowing also to eliminate subproblems only partially.

General references for branch and bound in global optimization include [Beale, 1988, Berner, 1996, Epperly and Pistikopoulos, 1997, Epperly and Swaney, 1996, Floudas, 1995, Grossman, 1990, Horst *et al.*, 1995, Jansson and Knüppel, 1992, Kearfott, 1996b, Pinter, 1996b, Van Hentenryck *et al.*, 1997c].

For a global optimization problem

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in \mathbf{x}^{\text{init}}, \quad F(x) \in \mathbf{F}, \quad x_I \text{ integral}, \end{array} \tag{19.90}$$

a natural way to define subproblems is to choose boxes $\mathbf{x} \subseteq \mathbf{x}^{\text{init}}$ of the initial box $\mathbf{x}^{\text{init}}$, and to consider the subproblems

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in \mathbf{x}, \quad F(x) \in \mathbf{F}, \quad x_I \text{ integral}, \end{array} \tag{19.91}$$

i.e., each subproblem is characterized by (and stored as) the box over which the problem is solved. The branching process then consists in splitting a box $\mathbf{x}$ into two or several smaller boxes whose union is $\mathbf{x}$. The most typical branching rule is to select a **bisection coordinate** j and to split the j-th component of the box at a **bisection point** ξ. Thus, the current box $\mathbf{x}$ is replaced by two subboxes $\mathbf{x}^{\text{low}}, \mathbf{x}^{\text{upp}}$ with

$$\begin{array}{l} \mathbf{x}_k^{\text{low}} = \mathbf{x}_k^{\text{upp}} = \mathbf{x}_k \quad \text{if } k \neq j, \\ \mathbf{x}_j^{\text{low}} = [\underline{x}_j, \xi], \quad \mathbf{x}_j^{\text{upp}} = [\xi, \overline{x}_j]. \end{array} \tag{19.92}$$

This branching rule is termed **bisection**. The bisection point ξ is often taken as the **midpoint** $\xi = (\overline{x}_j + \underline{x}_j)/2$ of the interval $\mathbf{x}_j$; but this fails when there are infinite bounds, and is inefficient when the interval ranges over several orders of magnitude. In this case, a more useful bisection point is a **safeguarded geometric mean**, defined by

$$\xi = \operatorname{sign} \underline{x}_j \sqrt{\underline{x}_j \overline{x}_j} \quad 0 < \underline{x}_j \overline{x}_j < \infty,$$

and otherwise

$$\begin{array}{ll} \xi = 0 & \text{if } \underline{x}_j < 0 < \overline{x}_j, \\ \xi = \min(\mu, q\overline{x}_j) & \text{if } \underline{x}_j = 0, \\ \xi = \max(-\mu, q\underline{x}_j) & \text{if } \overline{x}_j = 0, \\ \xi = q^{-1}\underline{x}_j & \text{if } \underline{x}_j > 0, \\ \xi = q^{-1}\overline{x}_j & \text{if } \overline{x}_j < 0, \end{array}$$

where $q \in]0,1[$ is a fixed constant (such as $q = 0.01$) and variables whose initial interval contains 0 are assumed to be most likely of magnitude μ.

The branching coordinate is more difficult to choose, but the speed of a branch and bound algorithm may be heavily affected by this choice. For a good algorithm, the choice should be scaling invariant, but the details depend on how the algorithm treats the individual subproblems.

Sometimes, a **trisection** branching rule is used which splits some component of a box into three intervals. Also, **multisection** branching rules may be employed; only one natural choice is described here. Suppose we know that a subbox $\mathbf{x}^0$ of $\mathbf{x}$ cannot contain a solution of (19.90). Then we can cover $\mathbf{x} \setminus \mathbf{x}^0$ by (at most) $2n$ subboxes, namely, for $j = 1, \dots, n$,

$$\begin{aligned}
\mathbf{x}_k^{2j-1} &= \mathbf{x}_k^{2j} = \mathbf{x}_k^0 \quad \text{if } k < j, \\
\mathbf{x}_j^{2j-1} &= [\underline{x}_j, \underline{x}_j^0], \quad \mathbf{x}_j^{2j} = [\overline{x}_j^0, \overline{x}_j], \\
\mathbf{x}_k^{2j-1} &= \mathbf{x}_k^{2j} = \mathbf{x}_k \quad \text{if } k > j.
\end{aligned} \tag{19.93}$$

However, this may yield long and thin slices, and is then rather inefficient.

For a comparison of some branching rules for bound constrained problems see [Csendes and Ratz, 1997, Ratz, 1996, Ratz and Csendes, 1995].

The **bounding rule** in its classical variant requires the solution of a **convex relaxation**, i.e., a convex (and often linear) optimization problem whose feasible set contains the feasible set of the subproblem (**outer approximation**) and whose objective function is at no feasible point larger than the original objective function (**underestimation**). If the convex problem is infeasible, the subproblem is infeasible, too, and can be discarded. If the convex problem is feasible, its solution provides a lower bound on $f(x)$, and when this lower bound is larger than the value of f^{best} for some feasible point x^{best} known (stored in a list of **best feasible points found so far**) we conclude that the subproblem does no longer contribute to the solution of the global optimization problem, and hence can be discarded.

Clearly, this procedure is equivalent to adding the constraint $f(x) \leq f^{\text{best}}$ to the definition of the subproblem, and checking infeasibility of the resulting reduced subproblem. This suggests a more general approach to defining subproblems by adding other **cuts**, i.e., derived inequalities that have to be satisfied at a global minimizer. If these inequalities are linear, the cuts define hyperplanes and are referred to as **cutting planes**. Branch and bound methods using cuts are frequendly labelled **branch and cut**.

Another important approach to handling subproblems uses constraint propagation and related techniques that define **tightening** (or **narrowing**, or **pruning**) **rules** which serve to reduce (as much as easily possible) the box defining a subproblem without changing its feasible set. If tightening results in an empty box, the subproblem is eliminated; if not, the subproblem may still have been reduced so much that many branching steps are saved. Fast and simple tightening rules use constraint propagation, discussed in Section 4.13; more expensive rules are discussed in Section 4.16. The balancing of work done in tightening versus work saved through less branching is a delicate matter, which at present more or less depends on ad hoc recipes.

Note that tightening techniques may be applied not only to the original constraints but to any constraints that must be satisfied at the global minimizer. This includes cutting planes (see Section 4.21) and the equations and inequalities derived from the Karush-John optimality conditions (see Section 4.5). In particular, software based on interval techniques (GLOBSOL [Kearfott, 1996b], Numerica [Van Hentenryck *et al.*, 1997c]) make essential use of the latter.

4.20 Relaxations

The final few sections just collect some odds and ends that should be treated in detail in a more complete survey, but have been left out in the present version for lack of time.

One of the well-developed sides of global optimization is the use of **linear and convex relaxations** to find a lower bound the value of the objective function, which allows one to discard boxes where this lower bound is larger than the function value $f^{\rm best}$ of the best feasible point found so far. Linear and convex relaxations are the main tool used in the packages αBB [Adjiman *et al.*, 1998a, Adjiman *et al.*, 1996, Adjiman *et al.*, 1998b, Androulakis *et al.*, 1995] and BARON [Sahinidis, 1996].

The current version of these notes do not cover this, since it is well-covered in [Floudas, 1995, Floudas, 1999]. Some of the topics to be covered in a future version include:

- recognizing convexity
- second order corrections enforcing convexity (as used in αBB; see [Adjiman *et al.*, 1998b])
- Lagrangian probing techniques implemented in BARON [Sahinidis, 1996]
- Optimal envelopes for bilinear terms [Al-Khayyal and Falk, 1983]

4.21 Cutting planes and cutting regions

A **cutting region** or simply a **cut** is an inequality not in the original problem formulation that must hold for any global minimizer; if the inequality is linear, it is called a **cutting plane** [Gomory, 1960]. A lot is known about cutting planes in mixed integer linear programming; see, e.g. [Nemhauser and Wolsey, 1988, Nemhauser and Wolsey, 1989, Wolsey, 1998]; we are here rather interested in techniques for the smooth case.

We have already met two kinds of derived constraints that cut off part of the feasible region: The constraint

$$f(x) \le f^{\rm best}$$

cuts of points worse than the best feasible point found so far (with function value f^{best}), and exclusion constraints, discussed in Section 4.16, cut off a region around local minimizers that do not contain any other, better minimizer.

Lagrangian aggregation techniques are discussed in [Schichl and Neumaier, 2001].

Surely there is much more to be explored here.

4.22 Certification of global minimizers

For the solution of precise mathematical problems (such as the Kepler problem [Hales, 1998]), and also for safety-critical optimization problems, it is necessary to have a complete mathematical guarantee that the global minimizer has been found. This requires special attention since numerical computations are affected by rounding errors. Fortunately, interval arithmetic, if performed with directed (outward) rounding, is able to give mathematical guarantees even in the presence of rounding errors.

For a description of the main techniques currently available in global optimization with mathematical guarantee, see, e.g., [Kearfott, 1996b, Kearfott, 1996a]. These techniques are part of the packages GLOBSOL [Kearfott, 1996b] and Numerica [Van Hentenryck *et al.*, 1997c].

One may also be interested in providing a minimal number of mathematically rigorous certificates that constitute a proof that some point in a narrow computed box is in fact a global minimizer. We have already seen certificates of infeasibility in Section 4.14. These certificates are mathematically valid only if the corresponding conditions have been evaluated in exact arithmetic; and additional safeguards are needed to ensure their validity in finite precision arithmetic. Virtually nothing has been done so far with regard to this problem.

4.23 Test problems and testing

An important part of the development of global optimization software is the careful testing of proposed methods. For useful test problem collections, see, e.g., [Floudas *et al.*, 1999, Huyer and Neumaier, 1999, Janka, 2000, Jansson and Knüppel, 1995, Walster *et al.*, 1985]. In particular, [Huyer and Neumaier, 1999] contains a test suite containing the traditional test set of low-dimensional problems by [Dixon and Szegö, 1975], together with test results for DIRECT, MCS, and many other heuristic optimization methods. [Janka, 2000] contains a comparison of stochastic global optimization routines on a large number of low-dimensional test problems from different sources.

[Jansson and Knüppel, 1995, Walster *et al.*, 1985] contain test results for some interval methods on a large number of low-dimensional test problems.

The recent *Handbook of Test Problems in Local and Global Optimization* [Floudas *et al.*, 1999] contains a large collection of test problems for local and global optimization problems of types including multi-quadratic programming, bilinear and biconvex, generalized geometric programming, general constrained nonlinear optimization, bilevel optimization, complementarity, semidefinite programming, mixed-integer nonlinear optimization, combinatorial optimization, and optimal control problems, – from application areas including pooling/blending operations, heat exchanger network synthesis, phase and chemical reaction equilibrium, robust stability analysis, batch plant design under uncertainty, chemical reactor network synthesis, parameter estimation and data reconcilliation, conformational problems in clusters of atoms and molecules, pump network synthesis, trim loss minimization, homogeneous azeotropic separation system, dynamic optimization problems in parameter estimation and reactor network synthesis, and optimal control problems.

The algebraic test problems of this collection are available in the GAMS modeling language and the differential-algebraic problems are supplied in the MINOPT modeling language. All test problems can be downloaded from the web site `http://titan.princeton.edu/TestProblems/`

Test problems for local optimization should also pass global optimization solvers; the traditional test set for low-dimensional unconstrained problems is that by [Moré *et al.*, 1981], with optional bounds from [Gay, 1984]. A number of these problems have in fact several local minimizers and are therefore global optimization problems.

Bob Vanderbei maintains in

`http://www.sor.princeton.edu/~rvdb/ampl/nlmodels/`

a large collection of AMPL files for constrained nonlinear optimization problems from practical applications; also included are the the CUTE collection and the more academic but useful low-dimensional problem collection of Hock & Schittkowski.

The Mathematical Programming Society has guidelines [Jackson *et al.*, 1990] for reporting results of computational experiments based on [Crowder *et al.*, 1979, Greenberg, 1990, Ratliff and Pierskalla, 1981]. See also [Barr *et al.*, 1995].

Chapter 5

Constraint Propagation

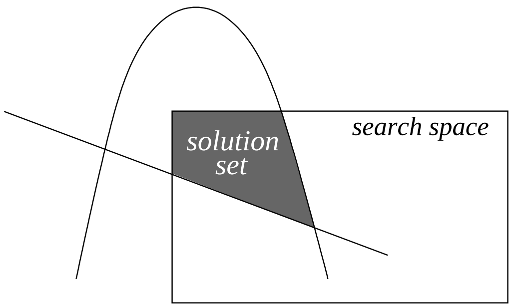
Laurent Granvilliers
Frédéric Benhamou
Institut de Recherche en Informatique de Nantes
Université de Nantes
B.P. 92208, F-44322 Nantes cedex 3, France

Etienne Huens
Departement d'ingénierie informatique
Université Catholique de Louvain
Place Sainte-Barbe, 2
B-1348 Louvain-la-Neuve, Belgique

5.1 Introduction

A **constraint satisfaction problem** (in short CSP) is given by a set of constraints expressing relations between the unknowns of a problem (variables), and a search space defined as the Cartesian product of variable domains (see Figure 5.1). The solution set of a CSP (feasible region) is the set of all elements from the search space that satisfy all the constraints (feasible points or solutions).

In this chapter we discuss a set of techniques for solving CSPs based on the **branch-and-prune approach**, that has been inspired by traditional branch-and-bound algorithms for optimization problems (see Chapter 4). Basically a branch-and-prune algorithm alternates pruning and branching steps of the search space to characterize the solution set of a CSP. A precise meaning of the



$$\begin{cases} y + 0.5x^2 - 2 \leqslant 0 \\ 2y + x + 2 \geqslant 0 \\ x, y \in [-1, 6] \times [-4, 1] \end{cases}$$

Figure 5.1: A constraint satisfaction problem.

output has to be given for specific instances of CSPs (finite domains, rationals in infinite precision, intervals ...), and specific algorithms (local search, complete methods ...). Branching operations (enumeration, bisection or splitting) guide the search for solutions in sub-domains. Pruning operations remove elements from the search space that do not satisfy the constraints. Reductions of the search space before branchings improve the practical complexity of the process though the general problem is NP-hard.
Note that, if the pruning method is **complete** (i.e., no solution is lost), and if no element remains after the computation, then the feasible region of the CSP is empty.
See Section 5.3.

Local consistency techniques are pruning methods that are polynomial in the length of problem description [Waltz, 1975a] [Montanari, 1974] [Mackworth, 1977] [Freuder, 1978]. Reductions (filterings) are computed by eliminating elements from the search space that do not verify some consistency property of constraints with respect to the current search space.
The most known property and the most used in practice for CSPs over finite domains is arc consistency [Mackworth, 1977]. Arc consistency can be roughly defined as follows: a CSP is arc consistent if for each constraint c and for each variable x, each value in the domain of x participates in a solution of c. A filtering for arc consistency just consists in removing all the values from the domain of x (inconsistent values) that do not participate in any solution of c.

Example 1 (consistency) *Consider a constraint $x_1 - x_2 = 1$, and $x_1 \in \{-1, 0, 1\}$, $x_2 \in \{-1, 0, 1\}$. The value -1 from the domain of x_1 is inconsistent, since there exists no value a_2 from the domain of x_2 such that $-1 - a_2 = 1$. A filtering based on arc consistency derives $x_1 \in \{0, 1\}$ and $x_2 \in \{-1, 0\}$.*

Reductions by consistency techniques are generally restricted to subsets (in general, singletons as for arc consistency) of the set of constraints (they are said to be local). **Constraint propagation algorithms** enforce reductions for

the whole CSP, until no domain can be contracted or a desired property (e.g., a threshold for the precision of domains) is obtained, navigating through the network of constraints connected by the variables. They can be modeled by narrowing [Benhamou, 1996] or chaotic iterations [Apt, 1999], i.e., sequences of application of contracting and monotonic functions on domains. The order in which constraints are visited, or more generally the order of application of reduction functions, does not influence the output domains (**confluence property**). However, this order determines the efficiency of the reduction process, and then several strategies have been defined.
See Section 5.4.

Example 2 (propagation) *Consider two constraints $c_1 : x_1 - x_2 = 1$ and $c_2 : x_1 > 0$, given $x_1 \in \{-1, 0, 1\}$, $x_2 \in \{-1, 0, 1\}$. A filtering based on arc consistency with respect to c_1 derives $x_1 \in \{0, 1\}$ and $x_2 \in \{-1, 0\}$. Then considering c_2 allows one to compute $x_1 \in \{1\}$ (the value 0 is inconsistent). Now, if c_1 is visited another time, the domain of x_2 is reduced to $\{0\}$. In other words the modification of the domain of x_1 using c_2 is propagated towards c_1 through the sharing of variable x_1.*

In the following we consider numeric CSPs (continuous constraint systems), i.e., nonlinear constraints (equations or inequalities) over the real numbers and closed interval domains. Since the search space (box) of a numeric CSP is in general infinite, traditional consistency techniques over finite domains have been adapted to deal with **approximations**, and more specifically, intervals. Pioneering works are presented in [Cleary, 1987, Davis, 1987, Hyvönen, 1992, Lhomme, 1993, Faltings, 1994, Benhamou and Older, 1997, Faltings and Gelle, 1997]. The essential idea is to consider a finite set that covers the computational domain, the **approximation domain**, for instance closed/opened floating-point intervals (intervals bounded by floating-point numbers). A second idea is to restrict consistency tests to the bounds of domains in order to reduce the computational complexity of propagation algorithms. This way, only bounds of domains are tightened.
The best known consistency notions for interval domains are **hull consistency** [Benhamou and Older, 1997] and **box consistency** [Benhamou *et al.*, 1994], whose main feature is to embed interval arithmetic [Moore, 1966] into constraint satisfaction methods. They are implemented in most constraint programming systems like CLP(BNR) [Benhamou and Older, 1997], ILOG Solver [Puget and Leconte, 1995], Numerica [Van Hentenryck *et al.*, 1997c] and Prolog IV [PrologIA, 1996]. A comparison of these techniques can be found in [Collavizza *et al.*, 1999].
See Sections 5.5, 5.6, 5.7.

Example 3 (narrowing) *Consider a linear constraint $\sum_{i \in I} a_i x_i = b$ and the variable domains $\boldsymbol{b}_i$. Given $k \in I$, a new lower bound for the domain of x_k can* *be computed as follows:*

$$\min(\boldsymbol{b}_k) := b - \sum_{j \in I - \{k\}} a_j \times \text{bound}(\boldsymbol{b}_j)$$

where $\text{bound}(\boldsymbol{b}_j) = \max(\boldsymbol{b}_j)$ *if* $a_j > 0$*, and* $\text{bound}(\boldsymbol{b}_j) = \min(\boldsymbol{b}_j)$ *otherwise. Note that here, the result is not guaranteed to be reliable since roundoff errors of numerical computations in a machine are not taken into account (this aspect will be discussed in Section 5.2).*
Note also that checking only the consistency of the bounds of domains sufficies to guarantee the consistency of sub-domains when the intersection of the constraints and the search space is convex (for instance, linear constraints and interval domains).

The **cooperation of pruning techniques** can greatly improve precision, see e.g., [Moore, 1966] [Neumaier, 1990]. The main idea is to combine redundant computations (reliable approximations of one quantity) from possibly heterogeneous solvers (see Figure 5.2). Nevertheless such a cooperation may also slow-down reduction processes, especially when the precision is not much improved.
For instance the cooperation of hull consistency and box consistency can be controlled so as to process subsets of data from the CSP [Benhamou *et al.*, 1999]. Another approach [Moore, 1966] [Neumaier, 1990], implemented in Numerica, consists in computing linear relaxations of CSPs by means of Taylor series expansions. Figure 5.2 illustrates the combination of a linear approximation and a centered approximation of a function, each one contracting the vertical component of the box.
See Section 5.8.

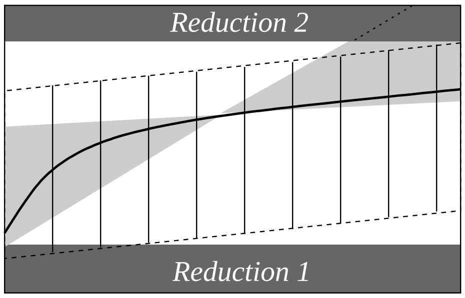


Figure 5.2: Cooperation of pruning techniques.

The **cooperation of symbolic-numeric algorithms** aims at transforming constraint expressions in a form that is more efficiently processed by numeric pruning techniques. For instance, a root-finding method for univariate polynomials can be combined with Gröbner bases that generate triangular systems. In our context two main problems have to be handled:

- the dependency problem of interval arithmetic: the precision of interval evaluation of real expressions depends on the occurrences of variables. There is a need for transforming constraint expressions;
- the locality of interval reasonings: constraints are processed one by one. Then the result is computed as an intersection of approximations (one per constraint), though the approximation of the solution set is in general much more precise. There is a need for combining constraints.

See Chapter 6.

The rest of this chapter is organized as follows. Interval arithmetic, which is at the basis of the pruning methods described here is introduced in Section 5.2. The general scheme of branch-and-prune computation is presented in Section 5.3. The basics of the processing of numerical constraints by constraint propagation with interval domains are introduced in Section 5.4. In the three sections 5.5, 5.6, and 5.7, specific local consistency techniques are presented, namely hull consistency, box consistency and bound consistency. Finally some ideas for cooperating pruning techniques are discussed in Section 5.8, and several symbolic-interval processes are presented in Chapter 6.

Notations. Let $\mathbb{R}$ denote the set of real numbers.

Consider a CSP $(C, \mathbf{b})$ over a set of variables $\{x_1, \ldots, x_n\}$. The search space of the CSP is denoted by $\mathbf{b} = \boldsymbol{b}_1 \times \cdots \times \boldsymbol{b}_n$, where $\boldsymbol{b}_i \subseteq \mathbb{R}$ is the domain of x_i for $i = 1, \ldots, n$. Assuming that no confusion is possible, the search space is also interpreted as the interval vector $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)^T$. Given a constraint c, let Var_c denote the set of variables occurring in c, and ρ_c the relation (subset of $\mathbb{R}^n$) defined by c.

Consider two constraints c and c'. The constraints are equivalent if $\rho_c = \rho_{c'}$. Constraint c is more general than c' if $\rho_{c'} \subseteq \rho_c$.

Given a scalar $\mathrm{a} = (a_1, \ldots, a_n)^T \in \mathbf{b}$ and a constraint $c(x_{i_1}, \ldots, x_{i_k})$ from C, $c(\mathrm{a})$ (or $c(a_1, \ldots, a_n)$) means $(a_{i_1}, \ldots, a_{i_k})^T \in \rho_c$. In other words the constraint is verified (satisfied) by a if the restriction of a to Var_c belongs to the underlying relation ρ_c.

Consider a n-ary constraint c, a search space $\mathbf{b}$, and a natural $i \in \{1, \ldots, n\}$. The i-th projection of c is defined as the set $\pi_i(c) = \{r_i \mid \exists (r_1, \ldots, r_n) \in \rho_c\}$. The i-th projection of c in $\mathbf{b}$ is defined as the set $\pi_i(c, \mathbf{b}) = \{r_i \mid \exists (r_1, \ldots, r_n) \in \rho_c \cap \mathbf{b}\}$.

5.2 Interval arithmetic

Interval arithmetic has been designed by R. E. Moore [Moore, 1966] to tackle roundoff errors of numerical computations and uncertainties of real parameters. It is implemented in constraint propagation to compute reliable bounds of real expressions.

We briefly present machine arithmetic (more details can be found in [Goldberg, 1991] [Muller, 1997]), interval arithmetic [Moore, 1966] [Alefeld and Herzberger, 1983] [Neumaier, 1990], and the notion of interval form used to compute bounds of real expressions.

5.2.1 Machine arithmetic

Let $\mathbb{R}^\infty$ be the real line compactified with the infinities $\{-\infty, +\infty\}$, and let $\mathbb{F} \subset \mathbb{R}$ be a finite subset of rational numbers corresponding to binary floating-point numbers in a given format [IEEE, 1985]. Let $\mathbb{F}^\infty$ denote $\mathbb{F} \cup \{-\infty, +\infty\}$.

For every $a \in \mathbb{R}$, let $a{\uparrow}$ be the smallest element in $\mathbb{F}^\infty$ greater than or equal to a, and $a{\downarrow}$ the greatest element in $\mathbb{F}^\infty$ smaller than or equal to a. Note that the rounding mode of floating-point computations can be set by program instructions. The IEEE standard defines four different modes (towards $-\infty$, towards $+\infty$, towards zero, nearest float), the rounding modes towards the infinities being intensively used for interval computations.

For every $a \in \mathbb{F}$, let a^+ be the smallest element in $\mathbb{F}^\infty$ greater than a, and a^- the greatest element in $\mathbb{F}^\infty$ smaller than a.

5.2.2 Interval arithmetic

Given $a \in \mathbb{F} \cup \{-\infty\}$ and $b \in \mathbb{F} \cup \{+\infty\}$, the set

$$\boldsymbol{x} = \{r \in \mathbb{R} \mid a \leqslant r \leqslant b\} = [a, b] = [\underline{\boldsymbol{x}}, \overline{\boldsymbol{x}}]$$

is an **interval**. An interval point $[a, a]$ is also denoted by a. Let $\mathbb{I}$ denote the set of intervals. Note that we consider only closed intervals but more general intervals are discussed in [Alefeld and Herzberger, 1983]. The *width* $\mathrm{w}(\boldsymbol{x})$ of an interval $\boldsymbol{x}$ is the rational number $(\overline{\boldsymbol{x}} - \underline{\boldsymbol{x}}){\uparrow}$. The width of an interval vector $\mathbf{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$ is defined as $\max_i \mathrm{w}(\boldsymbol{x}_i)$. An interval is said to be *canonical* if none of its bounds is an infinity and if it contains at most two elements of $\mathbb{F}$. An interval vector is canonical if each of its component is canonical.

The computation of enclosing intervals of sets of real numbers (interval hull) is an essential operation in interval arithmetic.

Definition 1 (interval hull) *Given a set $\rho \subseteq \mathbb{R}$, the* interval hull *of ρ is the tightest interval enclosing ρ, defined by*

$$\Box\rho = [\min(\rho){\downarrow}, \max(\rho){\uparrow}].$$

The interval hull of n-ary relations is interpreted componentwise, as follows:

$$\Box(\rho_1, \ldots, \rho_k) = (\Box\rho_1, \ldots, \Box\rho_k).$$

In the framework of interval arithmetic, a real quantity is represented by an enclosing interval; an interval operation is a set theoretic extension of the corresponding real operation. Interval operations are implemented by floating point computations on bounds of intervals, as follows:

$$\begin{array}{lcl}
[a,b] + [c,d] & = & [(a+c){\downarrow}, (b+d){\uparrow}] \\
[a,b] - [c,d] & = & [(a-d){\downarrow}, (b-c){\uparrow}] \\
[a,b] \times [c,d] & = & [\min(ac, ad, bc, bd){\downarrow}, \max(ac, ad, bc, bd){\uparrow}] \\
[a,b] \,/\, [c,d] & = & [\min(a/c, a/d, b/c, b/d){\downarrow}, \max(a/c, a/d, b/c, b/d){\uparrow}] \ \text{ if } 0 \notin [c,d] \\
 & = & [-\infty, +\infty] \ \text{ otherwise}
\end{array}$$

The elementary functions are extended so as to compute the hull of the real range, as follows:

$$\boldsymbol{\varphi}(\boldsymbol{x}) = \Box\{\varphi(a) \mid a \in \boldsymbol{x}\}$$

For instance, we have:

$$\mathbf{exp}([a,b]) = [\exp(a){\downarrow}, \exp(b){\uparrow}]$$

Nevertheless the computation of the exact interval hull cannot be always guaranteed for every elementary function, since the precision of floating-point computations can be more than one ulp (unit in the last place, i.e., distance between the two floating-point numbers that are closest to a real number). Implementation details of elementary functions can be found in [Muller, 1997].

The following properties hold:

- $+$ and $\times$ are commutative;
- $+$ and $\times$ are not associative due to rounding errors;
- the distributivity law of real arithmetic is no more valid, and only the *subdistributivity* applies: for all $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{I}$, we have

$$\boldsymbol{x} \times (\boldsymbol{y} + \boldsymbol{z}) \subseteq \boldsymbol{x} \times \boldsymbol{y} + \boldsymbol{x} \times \boldsymbol{z}.$$

- the interval operations are *monotonic*: for $\Diamond \in \{+, -, \times, /\}$ and all $\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{x}_2, \boldsymbol{y}_2 \in \mathbb{I}$ such that $\boldsymbol{x}_1 \subseteq \boldsymbol{y}_1$ and $\boldsymbol{x}_2 \subseteq \boldsymbol{y}_2$, we have $\boldsymbol{x}_1 \Diamond \boldsymbol{x}_2 \subseteq \boldsymbol{y}_1 \Diamond \boldsymbol{y}_2$.

5.2.3 Interval forms

Bounds of real expressions can be computed using *interval forms* (interval extensions or inclusion functions). More precisely, an interval form of a continuous function f over $\mathbb{R}$ is an interval function designed to compute supersets (enclosing intervals or outer approximations) of the range of f over a domain. Let $f^u(\mathrm{D})$ denote the range of a real function f over a domain D.

Definition 2 (interval form) *Let $f : \mathbb{R}^k \to \mathbb{R}$ be a continuous function over a domain* D. *A function $\boldsymbol{f} : \mathbb{I}^k \to \mathbb{I}$ is an* interval form *of f if for every $\mathbf{x} \in \mathbb{I}^k$ such that $\mathbf{x} \subseteq \mathrm{D}$, the following property holds:*

$$\boldsymbol{f}(\mathbf{x}) \supseteq f^u(\mathbf{x})$$

Moreover the classical interval forms defined below are inclusion monotonic, i.e., for every $\mathbf{x}, \mathbf{y}$ such that $\mathbf{x} \subseteq \mathbf{y}$, $\boldsymbol{f}(\mathbf{x}) \subseteq \boldsymbol{f}(\mathbf{y})$.

Consider a function $f : \mathbb{R}^k \to \mathbb{R}$, an interval form $\boldsymbol{f}$ of f, and a real number $\alpha > 0$. The *order of convergence of* $\boldsymbol{f}$ is α if there exists a real number $c \geqslant 0$ such that for every $\mathbf{x} \in \mathbb{I}^k$:

$$\mathrm{w}(\boldsymbol{f}(\mathbf{x})) - \mathrm{w}(\Box f^u(\mathbf{x})) \leqslant c \cdot \mathrm{w}(\mathbf{x})^\alpha$$

In other words this is an information on the precision of $\boldsymbol{f}$ with respect to the variable domains. The aforementioned formula suggests to preferably evaluate intervals forms of order 1 for large domains (linear convergence), and higher order interval forms for tight domains (width $w < 1$).

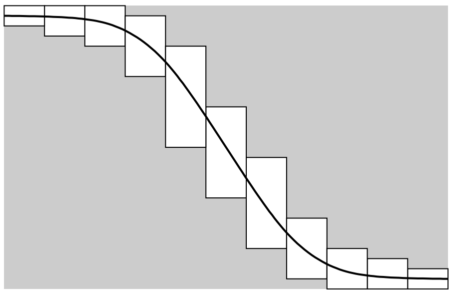


Figure 5.3: Interval evaluation of a function.

Figure 5.3 illustrates an interval evaluation of the range of a function f over a domain. The outer box corresponds to an evaluation over the whole domain (the range of f is included in the vertical component of the box). It can be refined if the domain is split, the evaluation being performed on each sub-domain. Such a division process is at the basis of most root-finding algorithms.

The definition is general enough to permit the design of various kinds of interval forms through symbolic transformations of constraint expressions, Taylor expansions, and other approximation techniques. A widely used notion is the **natural form** of a real function, that is a natural extension to interval arithmetic of a syntactic expression of the function. The natural form is obtained from the given expression by replacing each real number with its interval hull, each variable with an interval-valued variable, and each operation with the corresponding interval operation. The evaluation of the natural form of a function is often called *interval evaluation of the function.* The natural form has a linear convergence.

Example 4 (interval evaluation) *The natural form of* $f(x_1, x_2) = x_1 x_2 - 2$ *is the function* $\boldsymbol{f}(\boldsymbol{x}_1, \boldsymbol{x}_2) = \boldsymbol{x}_1 \times \boldsymbol{x}_2 - [2, 2]$ *(assuming that* $2 \in \mathbb{F}$*). An approximation of the range of* f *over* $[0, 1] \times [3, 5]$ *can be obtained as* $\boldsymbol{f}([0, 1], [3, 5]) = [-2, 3]$.

One would believe that the situation is ideal, since each real function can be reliably approximated by an interval function. Unfortunately, the computed intervals may be arbitrarily large compared to the real range. The main reason is the **dependency problem** of interval arithmetic: a variable is replaced with its domain during interval evaluations; as a consequence its occurrences are decorrelated.

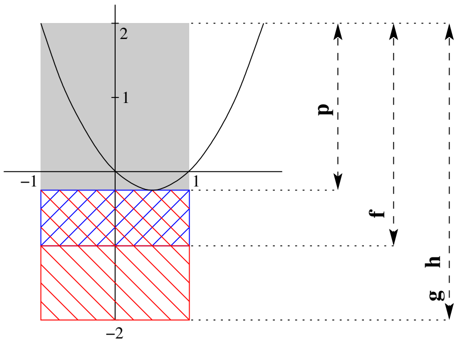
Example 5 (dependency problem (1)) *Let* $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{x} - \boldsymbol{x}$ *be an interval function. Given* $\boldsymbol{a} = [a, b]$, $\boldsymbol{f}(\boldsymbol{a})$ *is equal to* $[(a - b)\!\downarrow, (b - a)\!\uparrow]$. *For instance,* $\boldsymbol{f}([0, 1]) = [-1, 1]$ *though* $f^u([0, 1]) = [0, 0]$.

Furthermore different symbolic expressions may give different interval evaluations (see the following example).

Example 6 (dependency problem (2)) *Consider four expressions of a real function and their natural forms. Then,*

$$
\begin{array}{llll}
f(x) & = x^2 - x & \boldsymbol{f}([-1, 1]) & = [-1, 2] \\
g(x) & = x(x - 1) & \boldsymbol{g}([-1, 1]) & = [-2, 2] \\
h(x) & = xx - x & \boldsymbol{h}([-1, 1]) & = [-2, 2] \\
p(x) & = (x - 0.5)^2 - 0.25 & \boldsymbol{p}([-1, 1]) & = [-0.25, 2]
\end{array}
$$

Note that $\boldsymbol{p}(\boldsymbol{x}) = p^u(\boldsymbol{x})$ *since* p *contains only one occurrence of* x. *The following figure shows the computed evaluations for these four forms.*



The dependency problem is connected to the subdistributivity law. In fact there is clearly a strong motivation to factorize an expression in order to have precise interval computations. However there is an ideal case, when each variable of an expression occurs once. Moore's theorem guarantees the real range to be exactly enclosed by the interval evaluation (apart from rounding errors).

In order to handle the dependency problem and then to control nonlinearities, several different interval forms have been defined. They can be separated in two categories: syntactic-based forms and centered forms.

5.2.4 Syntactic-based interval forms

There are various kinds of interval forms derived by symbolic transformations. The essence of the method is to symbolically compute an equivalent real expression, and then to consider its natural form. The **Horner form** of an univariate polynomial

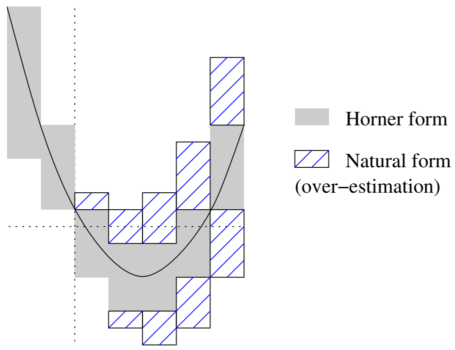
$$p(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_k x^k$$

is the polynomial

$$\boldsymbol{q}(\boldsymbol{x}) = \square\{\alpha_0\} + \boldsymbol{x}(\square\{\alpha_1\} + \cdots + \boldsymbol{x}(\square\{\alpha_{k-1}\} + \square\{\alpha_k\}\boldsymbol{x})).$$

The Horner form can be evaluated in $O(k)$ operations. The Horner form is often more precise than the natural form, which is a direct consequence of the subdistributivity property (a factorized expression gives more precise results).

Example 7 (Horner form) *Consider a function $f(x) = x^2 - 4x + 1$, its natural form and its horner form $\boldsymbol{h}(\boldsymbol{x}) = 1 + \boldsymbol{x}(-4 + \boldsymbol{x})$. The following figure illustrates their evaluations for $x \in [-2, 5]$.*



Nevertheless, this property is not always verified since the evaluation of an even power removes the negative part of the resulting interval.

Example 8 (evaluation of even/odd powers) *Let* $\boldsymbol{x} = [-1, 1]$. *Then,*

$$\boldsymbol{x}^4 \in [0, 1] \subset [-1, 1] \ni \boldsymbol{x} \times \boldsymbol{x}^3.$$

As shown in [Stahl, 1995], the **Bernstein form** [Rokne, 1979] [Farouki and Rajan, 1987] [Hong and Stahl, 1995] [Garloff and Graf, 1999] [Goualard, 2000] of an univariate polynomial is precise since a polynomial which is expressed in the Bernstein basis is numerically stable. However the evaluation process is difficult: this form depends on the variable domains and it must be computed after each modification of a domain.

The **nested form** [Stahl, 1995] of a quasi-polynomial (polynomial whose coefficients are real expressions) is a quasi-polynomial obtained from a sequence of factorizations of two products of variables. At each step two products are selected if the total degree (the sum of exponents) of their greatest common divisor is maximal. The nested form can be evaluated in $O(kn)$ operations if n is the number of variables, and k the number of terms in the initial expression. It has been generalized in [Ceberio and Granvilliers, 2000] in order to consider the selection as a strategy which can be adapted according to the situations.

Example 9 (nested form) *Consider a function*

$$f(x_1, x_2, x_3) = 2x_1x_2^2x_3 - x_2^2x_3^2 + \exp(x_1)x_1^2x_3$$

and its natural form $\boldsymbol{f}$. *The nested form of* f *is generated after two factorization steps, as follows:*

$$\begin{array}{rcl} f'(x_1, x_2, x_3) & = & \exp(x_1)x_1^2x_3 + (2x_1 - x_3)x_2^2x_3 \\ f''(x_1, x_2, x_3) & = & (\exp(x_1)x_1^2 + (2x_1 - x_3)x_2^2)x_3 \end{array}$$

Let $\boldsymbol{g}$ *denote the nested form of* f *corresponding to the natural form of* f''. *Given* $\mathrm{x} = ([1, 1], [0, 1], [0, 1])^T$, *we have* $\boldsymbol{f}(\mathrm{x}) = [-1, 4.72]$ *and* $\boldsymbol{g}(\mathrm{x}) = [0, 4.72]$.

The dual notion of the nested form is the **distributed form** obtained from an expression by developing multiplications with respect to additions. This form is weak in general but it can be better suited to specific methods. For instance a polynomial root-finding method that encloses a polynomial by two outer polynomials based on the distributed form has been introduced in [Hong and Stahl, 1994] and implemented in Newton [Van Hentenryck *et al.*, 1997a].

5.2.5 Centered forms

In the following, let $\mathbf{b}$ be the interval vector of variable domains, and $\mathbf{a}$ the interval hull of an element (generally the center) of $\mathbf{b}$.

Consider a function $f : \mathbb{R}^k \to \mathbb{R}$ and an interval form $\boldsymbol{f}$ of f. The generic centered form $\boldsymbol{c}$ of f, introduced by Moore [Moore, 1966], is defined as

$$\boldsymbol{c}(\mathbf{x}) = \boldsymbol{f}(\mathbf{a}) + \boldsymbol{g}(\mathbf{x} - \mathbf{a}) \cdot (\mathbf{x} - \mathbf{a})$$

where $\boldsymbol{g}$ is a *slope function* of f, i.e., a function that satisfies $\boldsymbol{f}(\mathbf{x}) - \boldsymbol{f}(\mathbf{a}) \in \boldsymbol{g}(\mathbf{x} - \mathbf{a})$. Centered forms have a quadratic convergence (see [Ratschek and Rokne, 1980] [Ratschek, 1980]).

A family of centered forms are obtained through an expansion of f in Taylor series around a point, where the error term is bounded by its evaluation over the whole domain [Moore, 1966] [Caprani and Madsen, 1980] [Berz and Hoffsttter, 1998]. The simplest Taylor form is the **mean value form**:

$$\boldsymbol{m}(\mathbf{x}) = \boldsymbol{f}(\mathbf{a}) + \nabla \boldsymbol{f}(\mathbf{b}) \cdot (\mathbf{x} - \mathbf{a})$$

Note that the mean value form depend on $\boldsymbol{f}$ and $\nabla \boldsymbol{f}$. There is a strong motivation to use tight interval forms, for instance nested forms may be preferred to natural forms.

The **second order Taylor form** is the following:

$$\boldsymbol{t}(\mathbf{x}) = \boldsymbol{f}(\mathbf{a}_0) + \nabla \boldsymbol{f}(\mathbf{a}) \cdot (\mathbf{x} - \mathbf{a}) + \frac{1}{2} \nabla^2 \boldsymbol{f}(\mathbf{b}) (\mathbf{x} - \mathbf{a})^2$$

Note that $\boldsymbol{t}$ is more precise than $\boldsymbol{m}$.

C. Bliek's PhD thesis [Bliek, 1992] is a good introduction to these notions. See [Bliek, 1997] and Chapter 3 for the evaluation of derivatives and slopes.

5.2.6 Combining interval forms

A nave approach to combining forms consists in intersecting the evaluations of different forms of a function. However this may result in useless computations, even if the precision is trivially improved. In fact this is a difficult problem since the accuracy of a form may also depend on domains (see the following example).

Example 10 (precision of interval forms) *Let $f(x) = x^2 + x$ and $g(x) = x(x+1)$ be two expressions of a real function and consider their natural forms $\boldsymbol{f}$ and $\boldsymbol{g}$.*

$$\begin{array}{lll}
\boldsymbol{f}([0,1]) = [0,2] & \boldsymbol{f}([-1,0]) = [-1,1] & \boldsymbol{f}([-1,1]) = [-1,2] \\
\boldsymbol{g}([0,1]) = [0,2] & \boldsymbol{g}([-1,0]) = [-1,0] & \boldsymbol{g}([-1,1]) = [-2,2]
\end{array}$$

We see that $\boldsymbol{f}$ and $\boldsymbol{g}$ are comparable over $[0,1]$; $\boldsymbol{g}$ is tighter than $\boldsymbol{f}$ over $[-1,0]$; and $\boldsymbol{f}$ is tighter than $\boldsymbol{g}$ over $[-1,1]$ (because of the even power).

The precision of an interval form with respect to the domains can be measured by the order of convergence. For instance, it is well-known that the Taylor form is more precise than the natural one for tight domains, and less precise for large domains (see the following example), since the Taylor form converges to the range of the function with a quadratic convergence on the width of domains and the convergence of the natural one is linear.

Example 11 (natural/mean value form) *Consider a function $f(x) = x^2 - x$. Let $\boldsymbol{f}$ be the natural form of f and $\boldsymbol{m}$ be the mean value form of f. Let $\boldsymbol{a} = [0,10]$ and $\boldsymbol{b} = [0.4, 0.6]$. We have:*

$$\begin{array}{llll}
f^u(\boldsymbol{a}) & = [-0.25, 90] & f^u(\boldsymbol{b}) & = [-0.25, -0.24] \\
\boldsymbol{f}(\boldsymbol{a}) & = [-10, 100] & \boldsymbol{f}(\boldsymbol{b}) & = [-0.44, -0.04] \\
\boldsymbol{m}(\boldsymbol{a}) & = [-75, 115] & \boldsymbol{m}(\boldsymbol{b}) & = [-0.27, -0.23]
\end{array}$$

Note that the range of f can be evaluated using the natural form of the equivalent expression $(x-0.5)^2 - 0.25$ where x occurs only once.

Furthermore Ratschek and Rokne [Ratschek and Rokne, 1995] suggest to use a centered form only if the width of the largest variable domain is smaller than $1/2n$, where n is the number of variables.

5.3 The branch-and-prune algorithm

An interval-based branch-and-prune algorithm takes as input a CSP and computes a set of boxes of a given width (precision). Each box is processed by two operations: domain pruning to contract it, and domain bisection to generate more precise sub-boxes. If the pruning method is complete, then each solution of the CSP is contained in at least one computed box. The generic branch-and-prune algorithm is presented in Table 5.3. The efficiency

```
BranchAndPrune( (C,b) : CSP ; ε : ℝ⁺ ): set of boxes
begin
  d := Prune(C,b)              % consistency algorithm
  if d = ∅ then
    return ∅                   % no solution is computed in b
  else if w(d) ⩽ ε then
    return d                   % d has the desired precision
  else
    Choose a dimension i ∈ {1,...,n} s. t. w(d_i) > ε
    Split d_i in d'_i ∪ d''_i          % bisection in two parts
    d' := (d_1,...,d_{i-1},d'_i,d_{i+1},...,d_n)
    d'' := (d_1,...,d_{i-1},d''_i,d_{i+1},...,d_n)
    return BranchAndPrune( (C,d') ; ε) ∪ BranchAndPrune( (C,d'') ; ε)
  fi
end
```

Table 5.1: The generic branch-and-prune algorithm.

of `BranchAndPrune` depends on the ability of `Prune` to contract the variable domains, the number of domains generated by a bisection, and the strategy for choosing the next variable domain to bisect. We refer the reader to [Kearfott, 1987] [Van Hentenryck *et al.*, 1997a].

Recall that the efficiency of `Prune` is connected with the precision of interval forms implemented in consistency techniques (see the previous section), and the way to process a conjunction of constraints, i.e., an intersection of relations.

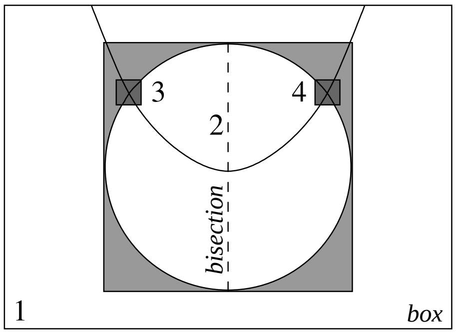
Several variable choice strategies have been implemented, e.g. in Numerica, the most efficient being:

- *largest*: at each branching step, the largest domain is chosen. Note that for CSPs over finite domains, the dual strategy is the most efficient (tightest domain);
- *round-robin*: a domain is bisected according to a fixed order of the variables.

Example 12 (branch-and-prune computation) *Consider the CSP*

$$(\{x_1^2 + x_2^2 = 2, x_2 = x_1^2\}, [-3,3] \times [-2,2]).$$

The following figure illustrates a possible computation in four steps: 1) pruning of the initial box; 2) bisection; 3) pruning of the leftmost sub-domain (approximation of the first solution); 4) pruning of the rightmost sub-domain (approximation of the second solution).



5.4 Interval constraints

Interval constraints are integrated in the constraint solving and optimization engines of many modern constraint programming languages, and have been used to solve industrial applications in areas like mechanical design, chemistry, aeronautics, medical diagnosis or image synthesis. One may cite the CLP systems `CHIP` [Lee and Van Emden, 1993] `CLIP` [Hickey, 2000], `CLP(BNR)` [Benhamou and Older, 1997], `DeclIC` [Goualard *et al.*, 1999], ECL$^{\mathrm{i}}$PS$^{\mathrm{e}}$ [Wallace *et al.*, 1997] and `PrologIV` [PrologIA, 1996], the C++ library `ILOG Solver` [Puget and Leconte, 1995], and the specific system `Numerica` [Van Hentenryck *et al.*, 1997c]. Furthermore few platforms, e.g., `Unicalc` [Semenov and Leshchenko, 1994] and the system of [Marti and Rueher, 1995], implement a symbolic-interval cooperative solver.

The term **interval constraint** is a generic one denoting a constraint (that is, a first order atomic formula such as an equation/inequation, or more generally a relation) in which variables are associated with intervals denoting their domains of possible values. In general, intervals are defined over real numbers though the concept is general enough to address other constraint domains (e.g., naturals, Booleans, lists, sets, etc.). When defined over reals, interval constraint sets are often called continuous constraint systems or numeric constraint satisfaction problems.

Constraint satisfaction over the real numbers (computation of a constraint relation or more generally a solution set of a constraint system) is generally intractable due to the limitations of machine arithmetic. Hence, J. G. Cleary [Cleary, 1987] has proposed to implement constraint satisfaction over the intervals, which is based on the notion of interval form of a constraint relation.

Definition 3 (interval form) *A relation* $\Gamma \subseteq \mathbb{I}^k$ *is an* interval form *of* $\rho \subseteq \mathbb{R}^k$ *if for all* $\mathrm{r} \in \mathbb{R}^k$, $\mathbf{x} \in \mathbb{I}^k$ *we have* $\mathrm{r} \in \mathbf{x} \wedge \mathrm{r} \in \rho \Rightarrow \mathbf{x} \in \Gamma$.

In other words, an interval form of a relation contains at least any box containing a solution point.

In practice, interval forms of constraint relations are obtained as follows: we consider interval forms of each term of constraints and the "possibly" interval interpretation of relation symbols. Given $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{I}$ and $\bowtie \in \{=, \leqslant, \geqslant\}$, we have

$$\boldsymbol{x} \bowtie \boldsymbol{y} \quad \Leftrightarrow \quad \exists\, a \in \boldsymbol{x}\ \exists\, b \in \boldsymbol{y}\ (a \bowtie b).$$

The computational expressions are the following:

$$\begin{array}{rcl} \boldsymbol{x} = \boldsymbol{y} & \Longleftrightarrow & \boldsymbol{x} \cap \boldsymbol{y} \neq \varnothing \\ \boldsymbol{x} \geqslant \boldsymbol{y} & \Longleftrightarrow & \overline{\boldsymbol{x}} \geqslant \underline{\boldsymbol{y}} \\ \boldsymbol{x} \leqslant \boldsymbol{y} & \Longleftrightarrow & \underline{\boldsymbol{x}} \leqslant \overline{\boldsymbol{y}} \end{array}$$

Now let $c : f(x_1, \ldots, x_n) \bowtie g(x_1, \ldots, x_n)$ be a constraint. Let $\boldsymbol{f}$ be an interval form of f and let $\boldsymbol{g}$ be an interval form of g. The main fact is that the set

$$\{\mathbf{x} \in \mathbb{I}^n \mid \boldsymbol{f}(\mathbf{x}) \bowtie \boldsymbol{g}(\mathbf{x})\}$$

defines an interval form of ρ_c which is computable. In our context, it allows one to check whether a domain belongs or not to a given interval form of a constraint relation. Moreover, if the domain does not belong to this interval form, the search space can be contracted since the use of interval arithmetic guarantees that a problem which is non satisfiable over the intervals is inconsistent over the real numbers. In the following we call the *natural form* of a constraint relation the interval form obtained from the natural forms of the constraint terms.

Example 13 (narrowing by evaluation) *Consider a constraint $x^2 - x = y + 1$ and its natural form. The constraint has no solution in the box $[0,1] \times [-1,1]$ since*

$$([0,1]^2 - [0,1]) \cap ([-1,1] + 1) = \varnothing.$$

Given a constraint system, the aim is to implement not only constraint satisfaction that permits either rejecting a domain or not, but also domain pruning (narrowing, reduction or contraction) to remove inconsistent values (real numbers that do not take part in any solution of the constraint system). The process of domain pruning can be modeled by the application of narrowing operators [Benhamou, 1996].

Definition 4 (narrowing operator) *Let c be a k-ary constraint. A* narrowing operator *for c is a mapping $\theta : \mathbb{I}^k \to \mathbb{I}^k$ such that for every $\mathbf{x}, \mathbf{y} \in \mathbb{I}^k$, we have the following properties:*

$$\begin{array}{rrcl} \textit{contraction:} & \theta(\mathbf{x}) & \subseteq & \mathbf{x} \\ \textit{completeness:} & \rho_c \cap \mathbf{x} & \subseteq & \theta(\mathbf{x}) \\ \textit{monotonicity:} & \mathbf{x} \subseteq \mathbf{y} & \Rightarrow & \theta(\mathbf{x}) \subseteq \theta(\mathbf{y}) \end{array}$$

```
Prune ($S = \{(\theta_1, c_1), \ldots, (\theta_m, c_m)\}$: set of narrowing operators ; $\mathbf{b}$: box) : box
% the set $S$ is generated from a given CSP $(C, \mathbf{b})$
begin
  Queue all pairs operator-constraint from $S$ in $Q$
  repeat
    Select an element $(\theta_i, c_i)$ from $Q$    % selection
    $\mathbf{d}$ := $\theta_i(\mathbf{b})$    % application (pruning)
    if $\mathbf{d} = \varnothing$ then
      return $\varnothing$    % inconsistency of the CSP
    else if $\mathbf{d} \neq \mathbf{b}$ then
      $Q$ := $Q \cup \{(\theta_j, c_j) \in S \mid \exists x_k \in Var_{c_j},\ \boldsymbol{d}_k \neq \boldsymbol{b}_k\}$    % propagation
      $\mathbf{b}$ := $\mathbf{d}$
      if $\theta_i$ is idempotent then $Q$ := $Q \setminus \{(\theta_i, c_i)\}$ fi    % $\theta_i(\mathbf{d}) = \mathbf{d}$
    else
      $Q$ := $Q \setminus \{(\theta_i, c_i)\}$    % $\theta_i(\mathbf{d}) = \mathbf{d}$
    fi
  until $Q$ is empty    % fixed-point:    for all $\theta$, $\theta(\mathbf{b}) = \mathbf{b}$
  return $\mathbf{b}$
end
```

Table 5.2: AC3-like Constraint Propagation Algorithm.

Note that the narrowing operators implemented in most constraint solvers share the property of idempotence, i.e., $\theta(\theta(\mathbf{x})) = \theta(\mathbf{x})$ for every $\mathbf{x} \in \mathbb{I}^k$.

The classical interval-based narrowing operators implement constraint satisfaction over the intervals. The existing algorithms only differ in the interval forms used for evaluation. More recently, the design of local consistency techniques has led to the implementation of more powerful narrowing operators which locally perform domain pruning with respect to a constraint and a variable. Such techniques (hull consistency, box consistency, bound consistency) are described in the following sections.

Example 14 (narrowing by bisection/evaluation) *Consider a constraint* $x^2 - x = 2$ *and its natural form. The domain* $[-1, 1]$ *cannot be eliminated by interval evaluation since* $[-1, 1]^2 - [-1, 1] \equiv [-1, 2]$*. However the sub-domain* $[0, 1]$ *is inconsistent since* $[0, 1]^2 - [0, 1] \equiv [-1, 1]$*.*

Given a numeric CSP, a **constraint propagation** (interval narrowing) algorithm takes as input a set of narrowing operators associated with constraints from the CSP, and generates a box approximating its solution set. Intuitively the narrowing operators are applied in sequence until a fixed-point is reached. Note that several operators can be associated with the same constraint in order to implement solver cooperations. The procedure of generation of narrowing operators from CSPs will be described for the specific techniques presented thereafter.

In the interval constraint framework, the classical constraint propagation algorithm, described in Table 5.4, is very similar to the arc consistency algorithm `AC3` [Mackworth, 1977]. The loop invariant of the algorithm is the following: every narrowing operator θ not in Q (the propagation queue) is such that $\theta(\mathbf{b}) = \mathbf{b}$. At every step of the `repeat` loop the strategy consists in applying a selected operator θ_i from Q, and adding in Q all the narrowing operators θ_j depending on a modified domain $\boldsymbol{d}_k$, i.e., the domain of a variable x_k occurring in its associated constraint c_j. The algorithm stops when a stable state is reached, i.e., no (strict) narrowing is possible with respect to any constraint. The result of the main step is to remove (some) incompatible values from the domains of the variables occurring in c_i. The main properties of `Prune` are the following:

- it terminates in finite time, since each step is contracting and the inclusion relation over an approximation domain (here, $\mathbb{I}$) is a well-founded relation;
- it is complete (the final box contains all solutions of the initial system included in $\mathbf{b}$), since each narrowing operator is complete;
- it is confluent (selection of constraints in the main loop is strategy independent), and it computes the greatest common fixed-point of the narrowing operators that is included in the initial box [Benhamou and Granvilliers, 1997].

These properties rely on domain theory and more precisely on properties of monotone operators over lattices [Henkin *et al.*, 1971, Apt, 1999].

5.5 Hull consistency

In this section, we describe the implementation of narrowing operators enforcing hull consistency. First we define the notion of arc consistency in order to draw the difference with respect to consistencies over continuous domains.

5.5.1 Global consistency and arc consistency

Different levels of consistency have been defined, the strongest one being global consistency.

Definition 5 (global consistency) *A CSP* $(\{c_1, \ldots, c_m\}, \mathbf{b})$ *is* globally consistent *if each domain,* $\boldsymbol{b}_i$ *for* $i = 1, \ldots, n$*, corresponds to the* i*-th projection of the solution set of the CSP, i.e.,*

$$\boldsymbol{b}_i = \{r_i \mid (r_1, \ldots, r_n) \in \mathbf{b} \cap \rho_{c_1} \cap \cdots \cap \rho_{c_m}\}$$

In other words, a CSP is globally consistent if no domain can be contracted while preserving the solution set. In practice, weaker properties are implemented, like k-consistency [Freuder, 1978], the most used for finite domain CSPs being arc consistency [Waltz, 1975a, Mackworth, 1977].

Definition 6 (arc consistency) *Consider a constraint $c(x_1, \ldots, x_n)$ and a search space* $\mathbf{b}$. *Let $i \in \{1, \ldots, n\}$ be a natural. The i-th projection of c is* arc consistent *w.r.t.* $\mathbf{b}$ *if*

$$\boldsymbol{b}_i = \pi_i(c, \mathbf{b}).$$

In other words, for all $r_i \in \boldsymbol{b}_i$, there exists $r_1 \in \boldsymbol{b}_1, \ldots, r_{i-1} \in \boldsymbol{b}_{i-1}, r_{i+1} \in \boldsymbol{b}_{i+1}, \ldots, r_n \in \boldsymbol{b}_n$ such that $(r_1, \ldots, r_n) \in \rho_c$; $(r_1, \ldots, r_{i-1}, r_{i+1}, r_n)$ is called a support of r_i.
A constraint is arc consistent if each of its projections is arc consistent. A CSP is arc consistent if all constraints are arc consistent.

A filtering for arc consistency consists in eliminating from each domain each value that has no support with respect to one of the constraints. Nevertheless, for continuous domains, enforced consistencies only approximate arc consistency since some solutions may be unrepresentable with floating-point numbers.

5.5.2 Hull consistency

Hull consistency [Benhamou and Older, 1997] is an elementary extension of arc consistency at the approximation level.

Definition 7 (hull consistency) *Consider a constraint $c(x_1, \ldots, x_n)$ and a search space* $\mathbf{b}$. *Let $i \in \{1, \ldots, n\}$ be a natural. The i-th projection of c is* hull consistent *w.r.t.* $\mathbf{b}$ *if*

$$\boldsymbol{b}_i = \Box\pi_i(c, \mathbf{b}).$$

A constraint is hull consistent if each of its projections is hull consistent. A CSP is hull consistent if all constraints are hull consistent.

Due to round-off errors introduced by the use of floating-point numbers, computing the interval hull of a real set is a difficult task in itself. Moreover, the precision of many arithmetic functions such as exp, cos ... is not guaranteed by the IEEE 754 standard [IEEE, 1985]; consequently, their actual precision is implementation dependent.

Algorithm `HC3revise` [Cleary, 1987] [Benhamou *et al.*, 1994][1] partly overcomes this problem by enforcing hull consistency over a decomposition of simple (primitive) constraints rather than considering the user constraint c. For example, the constraint $c : x_1 + x_2 \times x_3 = x_4$ might be decomposed into

[1] `HC3revise` is our own denomination that is justified by its very close relation to `AC3revise`.

$\{x_2 \times x_3 = z, x_1 + z = x_4\}$ with the addition of the new variable z (fresh variable). `HC3revise` is implemented using relational interval arithmetic [Cleary, 1987] (see an example below). This process can be seen as constraint inversion. The main advantage of such an approach is that computation of hull consistency can be implemented very efficiently for the set of simple constraints supported by the constraint programming system. The drawbacks are twofold:

P1) the structure of initial constraints is broken;

P2) introduction of new variables due to the decomposition process drastically hinders domain tightening for the variables the user is interested in. As pointed out in [Benhamou *et al.*, 1994], this is particularly true when the same variables appear more than once in the constraints since each occurrence of a variable x is considered as a new variable y with the same domain as x (dependency problem of interval arithmetic, see Section 5.2).

Problem P1 can be handled by new algorithms performing traversals of tree-representations of constraints [Benhamou *et al.*, 1999] [Granvilliers and Benhamou, 2001], described in the next section. Problem P2 is partially handled by box consistency [Benhamou *et al.*, 1994] [Benhamou *et al.*, 1999] (see Section 5.6).

Example 15 *A narrowing operator enforcing hull consistency for the constraint* $c : x_1 + x_2 = x_3$ *and domains* $\boldsymbol{b}_1$, $\boldsymbol{b}_2$, *and* $\boldsymbol{b}_3$, *computes the following reductions:*

$$\left\{\begin{array}{lcl} \boldsymbol{b}_1 & := & \boldsymbol{b}_1 \cap (\boldsymbol{b}_3 - \boldsymbol{b}_2) \\ \boldsymbol{b}_2 & := & \boldsymbol{b}_2 \cap (\boldsymbol{b}_3 - \boldsymbol{b}_1) \\ \boldsymbol{b}_3 & := & \boldsymbol{b}_3 \cap (\boldsymbol{b}_1 + \boldsymbol{b}_2) \end{array}\right.$$

As pointed out by Van Emden [Van Emden, 1997], the projection operations for a constraint of the form $x_1 \diamond x_2 = x_3$ may all be computed even if the function $\diamond$ has no inverse. For example, consider the case where $\diamond$ stands for the multiplication over $\mathbb{I}$: a new domain for x_1 is obtained as the smallest (union of) interval(s) containing the set $\{a_1 \in \boldsymbol{b}_1 \mid \exists a_2 \in \boldsymbol{b}_2, \exists a_3 \in \boldsymbol{b}_3 : a_1 \times a_2 = a_3\}$. Hence, it is defined even when $0 \in \boldsymbol{b}_2$.

Given a CSP, hull consistency is enforced by the constraint propagation algorithm presented in Section 5.4. The input set of narrowing operators contains one operator for each primitive constraint in the decomposition of each initial constraint from the CSP.

5.5.3 The chain rule for constraint projections

The decomposition process of constraints is a preprocessing for a particular implementation of hull consistency. Filterings and propagation just result in approximations of constraint projections. We show here that the computation of constraint projections can be described in terms of a chain rule [Granvilliers and Benhamou, 2001]. Moreover this chain rule can be implemented by two traversals of the tree-structured representation of functions [Benhamou *et al.*, 1999], that is similar to the process for computing partial derivatives in backward mode [Griewank, 1989, Bliek, 1992].

Notations. In the following a term is represented as a *directed acyclic graph* (DAG). A DAG $G = (N, A)$ is a directed graph whose set of vertices is N, whose set of arcs is $A \subseteq \{(u,v) \mid u, v \in N\}$ and such that no sequence of arcs from A is a cycle. Given a node $u \in N$, u^+ is the set of parents of u, that is $u^+ = \{v \mid \exists(v,u) \in A\}$. Given a sub-term u occurring in a constraint c (resp. a term f) and a variable z, let $c[u \leftarrow z]$ (resp. $f[u \leftarrow z]$) denote the constraint (resp. the term) obtained from c (resp. f) by replacing u with z; ∂_u^f stands for the derivative of f with respect to u, that is $(\partial f[u \leftarrow z]/\partial z)[z \leftarrow u]$.

We remark that constraint projections can be evaluated by performing two traversals of the tree-structured representation of functions. This process has been first described in [Benhamou *et al.*, 1999], where the new narrowing algorithm is called `HC4revise`. Intuitively, a bottom-up traversal evaluates (over $\mathbb{I}$) the ranges of functions for the current variable domains; then, a top-down traversal (retro-propagation process) performs elementary projection operations (basically the operations for inverting primitive constraints).

Without loss of generality[2], we may consider a constraint $c : f \bowtie 0$ where the relation symbol $\bowtie$ lies in $\{=, \leqslant, \geqslant\}$. As shown in [Griewank, 1989], the computation of partial derivatives ∂_u^f of f is based on the following chain rule:

$$\partial_u^f = \sum_{v \in u^+} \partial_v^f \partial_u^v$$

In the same spirit, we have proposed to define the following chain rule for computing the partial projections $\pi_u(c)$ of c, where z_v is a fresh variable:

$$\pi_u(c) \subseteq \bigcap_{v \in u^+} \pi_u(v = z_v)$$

The key point is that the conjunction of constraints $c[v \leftarrow z_v] \wedge v = z_v$, where z_v is a fresh variable of domain $[-\infty, +\infty]$, is equivalent to c. It follows that $v = z_v$ is more general than c. Thus, an outer approximation of the projection of c over u can be computed by projecting the new constraint $v = z_v$ over u. Doing so for each parent v of u, a tight approximation of $\pi_u(c)$ can be obtained as the intersection of all projections $\pi_u(v = z_v)$ (see the following example).

Example 16 *Let $c : x_1^2 = f(x_2)$ be a constraint with $x_1 \in \boldsymbol{b}_1$, $x_2 \in \boldsymbol{b}_2$ and $f(x_2) \in \boldsymbol{d}$, and let z be a fresh variable. The decomposition of c leads to $z =$*

[2]Every constraint $g \bowtie h$ can be transformed beforehand into $g - h \bowtie 0$.

$f(x_2) \wedge x_1^2 = z$. *We trivially have* $\pi_{x_1}(c, \boldsymbol{b}_1 \times \boldsymbol{b}_2) \subseteq \pi_{x_1}(x_1^2 = z, \boldsymbol{b}_1 \times [-\infty, +\infty]$ *since* $\boldsymbol{d}$ *is necessary included in the domain of* z.

The main results are given below [Granvilliers and Benhamou, 2001]:

- the chain rule for constraint projections is correct;
- For any constraint c containing only single occurrences of variables, the chain rule for projections can be replaced with an equality, that is:

$$\pi_u(c) = \bigcap_{v \in u^+} \pi_u(v = z_v)$$

The algorithmic process is detailed now (it is also compared with the evaluation of derivatives). The interval evaluation of every sub-term u of f is a superset of the image of u, which is guaranteed by the fundamental theorem of interval arithmetic [Moore, 1966]. The first stage then consists in a bottom-up traversal of f for computing an outer approximation of the range of every of its sub-term; the range of a term is obtained by applying the corresponding interval operation over the ranges of its sub-terms. The second stage sets either the partial derivative ∂_f^f to 1 or the projection $\pi_f(c)$, which is obtained from the interpretation of relation symbols over $\mathbb{I}$, as follows:

$$\begin{array}{ll} c : f = 0 & \pi_f(c) = [0, 0] \\ c : f \leqslant 0 & \pi_f(c) = [-\infty, 0] \\ c : f \geqslant 0 & \pi_f(c) = [0, +\infty] \end{array}$$

Finally, the retro-propagation algorithm operates a top-down traversal of f implementing the chain rules. Figure 5.4 illustrates an elementary operation at one node of f containing an operation symbol $\diamond$. Let us describe the projection operation: the projection $\pi_{u \diamond v}(c)$ results from the intersection of parent node w. The contribution of this node to $\pi_u(c)$ (resp. $\pi_v(c)$) is then $\pi_u(u \diamond v = z)$ (resp. $\pi_v(u \diamond v = z)$) where z ranges over $\pi_{u \diamond v}(c)$.

The projections of $u \diamond v = z$ (resp. derivatives of $u \diamond v$) are computed by evaluating the inverse operations of $\diamond$ (resp. elementary derivation rule). The correctness of inverse operations is proved in [Cleary, 1987]. Some of those operations are detailed now (bold letters denote the interval evaluations of the corresponding expressions):

$$\begin{array}{lcllcl} \partial_u^{u+v} & = & 1 & \pi_u(u + v = z) & = & \boldsymbol{z} - \boldsymbol{v} \\ \partial_v^{u+v} & = & 1 & \pi_v(u + v = z) & = & \boldsymbol{z} - \boldsymbol{u} \\ \partial_u^{u-v} & = & 1 & \pi_u(u - v = z) & = & \boldsymbol{z} + \boldsymbol{v} \\ \partial_v^{u-v} & = & -1 & \pi_v(u - v = z) & = & \boldsymbol{u} - \boldsymbol{z} \\ \partial_u^{\log(u)} & = & 1/\boldsymbol{u} & \pi_u(\log(u) = z) & = & \exp(\boldsymbol{z}) \\ \partial_u^{\exp(u)} & = & \exp(\boldsymbol{u}) & \pi_u(\exp(u) = z) & = & \log(\boldsymbol{z}) \end{array}$$

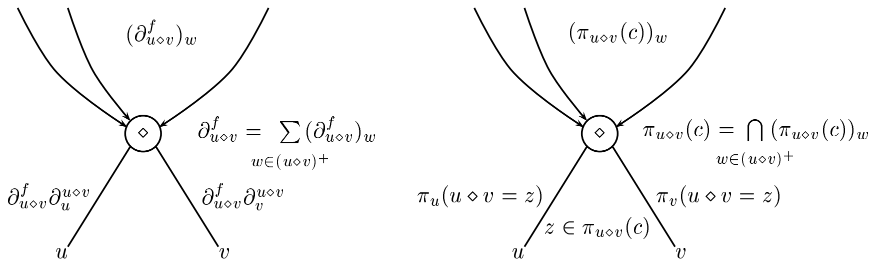


Figure 5.4: Backward evaluation of partial derivatives and constraint projections.

The final interesting results are located at variable nodes x_i, namely reliable approximations for $\partial^f_{x_i}$ and for $\pi_{x_i}(c)$. Partial derivatives are summed up while projections are intersected. Given $x_i \in \boldsymbol{b}_i$, the projection of c over x_i is computed as:

$$\pi_{x_i}(c,\mathbf{b}) = \Box[\bigcap_{v\in x_i^+} \pi_{x_i}(v = z) \cap \boldsymbol{b}_i]$$

Since $\pi_{x_i}(v = z)$ can be a union of intervals (given non interval convex operations), the hull of the intersection permits to restrict a domain to be an interval. If the new domain of x_i is empty then this constraint has no solution in the current variable domains. To sum up, this algorithm computes a tighter evaluation for the domain of x_i after a filtering through constraint c (see the following example).

Example 17 *Let $c : 2x_1 = x_2 - 1$ be a constraint given $x_1 \in [1,8]$ and $x_2 \in [-2,5]$. Figure 5.5 presents the interval evaluation and retro-propagation processes in the term $f : 2x_1 - (x_2 - 1)$.*

The computed approximations for the projections (before intersection with current domains) are $\pi_{x_1}(c) = [-1.5, 2]$ and $\pi_{x_2}(c) = [3, 17]$. The new domains are then $[1,2] = [1,8] \cap \pi_{x_1}(c)$ for x_1 and $[3,5] = [-2,5] \cap \pi_{x_2}(c)$ for x_2.

Retro-propagation at node $\times$ is detailed now: the projection over x_1 of $2\times x_1 = z$ is computed (the same algorithm processes primitive constraints), where z is a fresh variable ranging over $[-3,4]$ and $x_1 \in [1,8]$. Constraint $2 \times x_1 = z$ is inverted (a symbolic view of this process is the generation of a new constraint $x_1 = z/2$) and then $\pi_{x_1}(c) = [1,8] \cap ([-3,4]/2) = [1,2]$.

Let us remark that the projections of a constraint containing only single occurrences of variables are exactly computed by retro-propagation (modulo rounding errors). The reasons are twofold: the projection rule computes an equality in such a case, and the elementary projection operations are exactly implemented since interval evaluation computes the exact ranges of all terms of c (theorem

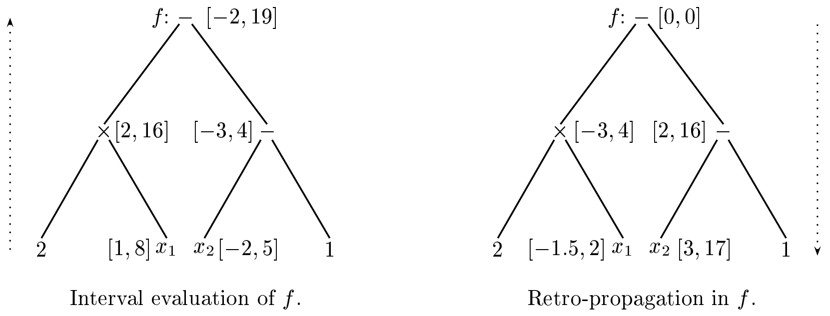


Figure 5.5: Computation of constraint projections.

from Moore [Moore, 1966]). As a consequence, only one retro-propagation process permits the computation of all projections, *i.e.*, to reach the fixed-point of the algorithm (idempotence property).

For efficiency reasons, interval evaluation of terms can be done during the backward process whenever it is needed (lazy evaluation), as is done by the algorithm computing derivatives in forward mode. More precisely, given the term $f : u \diamond v$ considered during retro-propagation on f, the interval evaluation $\boldsymbol{u}$ of u is computed if the three following conditions are verified:

C1) Term v contains some variables;

C2) Term u contains some operations;

C3) $\boldsymbol{u}$ has not been already computed.

C1 means that $\pi_v(c)$ has to be computed. The codes for elementary projection operations suggest that the value $\boldsymbol{u}$ is required (C2 and C3). C1 and C2 can be implemented setting one flag at each node of constraint terms. C3 is dynamically evaluated only using one flag during the backward traversal of each term.

The efficiency of lazy evaluation intrinsically depends on the number of variable occurrences in the constraint and their locations in terms. Example 18 illustrates the case where a lazy evaluation is more efficient.

Example 18 *Let $c : (x_1 + 1)^3 - 8 = 0$ be a constraint. Retro-propagation successively operates the projection over z_i of the following constraints: $z_1 - 8 = 0$, $z_2^3 = 8$, $z_3 + 1 = 2$. We remark that the interval evaluations of terms of c are not required (see above-mentioned conditions), since all right sub-terms are variable-free. A lazy evaluation then prevents unnecessary computation (here, this is one evaluation of the plus/minus/power operations).*

5.6 Box consistency

Box consistency [Benhamou *et al.*, 1994] has been introduced to avoid decomposing constraints, thus tackling the dependency problem for variables with many occurrences, and to model existing interval methods [Van Hentenryck *et al.*, 1997a], like the Krawczyk operator [Krawczyk, 1986] or the Hansen-Sengupta operator [Hansen and Sengupta, 1981]. With respect to hull consistency, the main difference is that box consistency avoids considering real numbers, operating a shift from an uncomputable real relation to an interval relation.

5.6.1 Classical definition

Box consistency has been defined in [Benhamou *et al.*, 1994]. Note that the definition is parametrized with an interval form of the constraint.

Definition 8 (box consistency) *Consider a constraint* $c(x_1, \ldots, x_n)$*, an interval form* Γ *of* c*, and a search space* $\mathbf{b}$*. Let* $i \in \{1, \ldots, n\}$ *be a natural. The* i*-th projection of* c *is* box consistent *w.r.t.* $\mathbf{b}$ *and* Γ *if*

$$\boldsymbol{b}_i = \Box\{a_i \in \boldsymbol{b}_i \mid (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{i-1}, \Box\{a_i\}, \boldsymbol{b}_{i+1}, \ldots, \boldsymbol{b}_n) \in \Gamma\}.$$

A constraint is box consistent if each of its projections is box consistent. A CSP is box consistent if all constraints are box consistent.

This definition precisely shows the difference with respect to hull consistency: the existence of a support is replaced with an interval reasoning, where each variable but x_i is replaced with its domain. An equivalent definition, considering only the bounds of x_i, can be given as follows: if $\boldsymbol{b}_i$ is not an interval point, box consistency is defined by:

$$(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{i-1}, [\overline{\boldsymbol{b}_i}^{-}, \overline{\boldsymbol{b}_i}], \boldsymbol{b}_{i+1}, \ldots, \boldsymbol{b}_n) \in \Gamma$$

$$\wedge\ (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{i-1}, [\underline{\boldsymbol{b}_i}, \underline{\boldsymbol{b}_i}^{+}], \boldsymbol{b}_{i+1}, \ldots, \boldsymbol{b}_n) \in \Gamma$$

Otherwise, it is defined by:

$$(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{i-1}, \boldsymbol{b}_i, \boldsymbol{b}_{i+1}, \ldots, \boldsymbol{b}_n) \in \Gamma$$

In other words, box consistency combines interval evaluation (see Section 5.4) and a search process on the domain of x_i. The effects resulting from the search process are twofold: parts of $\boldsymbol{b}_i$ can be declared inconsistent even if interval evaluation declares consistent the whole domain $\boldsymbol{b}_i$; the dependency problem originating from the multiple occurrences of x_i is efficiently handled.

The classical narrowing algorithm for box consistency, called `BC3revise`, implements a dichotomous search that discards sub-domains at the bounds of the given variable domain (see Table 5.6.1). Interval evaluation is used to reject these sub-domains. Under some conditions, the search can also be accelerated by Newton-like methods [Benhamou *et al.*, 1994].

In [Van Hentenryck *et al.*, 1997a], box consistency is computed for different interval forms (natural form, distributed form, mean value form). Nevertheless it is often more efficient to consider the best form for a given projection, thus preventing useless redundant computations. This has led to the definition of box$_\Gamma$ consistency (Section 5.6.2). Moreover, filterings may be subject to slow convergences. To handle such phenomena, the notion of box$_\varphi$ consistency has been introduced (Section 5.6.3).

5.6.2 Box$_\Gamma$ consistency

Box$_\Gamma$ consistency originates from [Benhamou *et al.*, 1999]. It extends box consistency at the constraint projection level, by permitting the use of different interval forms for constraint projections.

Definition 9 (box$_\Gamma$ consistency) *Let c be a real n-ary constraint, Γ a set $\{\Gamma_1, \ldots, \Gamma_n\}$ of n interval forms of c, and $\mathbf{b} = \boldsymbol{b}_1 \times \cdots \times \boldsymbol{b}_n$ the Cartesian product of domains. The constraint c is box consistent w.r.t.* $\mathbf{b}$ *and* Γ *if for every $i = 1, \ldots, n$ the i-th projection of c is box consistent w.r.t.* $\mathbf{b}$ *and* Γ_i.

As is said in [Benhamou *et al.*, 1994], box consistency computation is far more effective than hull consistency computation when dealing with complex constraints involving the same variables many times, since the global processing of these constraints avoids losing some useful information. Nevertheless, finding the outermost consistent sub-domains is computationally expensive. Moreover, it must be done for every variable occurring in the constraint. Therefore, box consistency is generally not the solution of choice when a complex constraint involves many different variables. These remarks have led to the combination of box consistency and hull consistency, through the generation of a particular set of narrowing operators to be processed by the constraint propagation algorithm.

- for each constraint c and each variable x_i occurring more than once in c, the i-th projection of c is processed by the narrowing operator `BC3revise` for box consistency operating on a given interval form of c;
- for each constraint c that contains at least one variable occurring once, a narrowing operator `HC4revise` for hull consistency implementing the chain rule is generated.

```
BC3revise(Γ: interval form ; i: natural ; b: box): interval
begin
  d_l := NarrowLeft(Γ,i,b)        % leftmost consistent canonical interval
  if d_l = ∅ then return ∅ fi
  b_i := [d_l, b̄_i]               % optimisation: [b_i, d_l[ can be discarded
  d_r := NarrowRight(Γ,i,b)       % rightmost consistent canonical interval
  b_i := [d_l, d̄_r]               % narrowing of b_i
  return b
end

NarrowLeft(Γ: interval form ; i: natural ; b: box): interval
begin
  Push b_i in S                   % stack for the search process
  d, e := b
  repeat
    d_i := Pop from S
    if d ∈ Γ then                 % consistent sub-domain
      if d_i is canonical then
        return d_i                % leftmost canonical consistent sub-domain
      else
        e_i := [d_i, d_i^+]       % canonical interval at the left bound of d_i
        if e ∈ Γ then
          return e_i              % leftmost canonical consistent sub-domain
        else
          r := center of [d_i^+, d̄_i] % [d_i, d_i^+[ can be discarded
          Push [r, d̄_i] in S
          Push [d_i^+, r] in S
        fi
      fi
    fi
  until S is empty
  return ∅
end

NarrowRight is similar to NarrowLeft
```

Table 5.3: Narrowing Operator for Box Consistency.

In the literature, the constraint propagation algorithm taking as input these sets of narrowing operators is called BC4.

5.6.3 Box$_\varphi$ consistency

Box consistency effectiveness comes from its ability to locally cancel the dependency problem of interval arithmetic. However, its classical implementation suffers from some drawbacks: first, applying a `BC3revise` narrowing operator is computationally expensive and leads to the unnecessary computation of a local fixed-point per operator; second, `BC3revise` narrowing operators do not ensure in practice the same amount of domain tightening. To address these problems, box$_\varphi$ consistency [Granvilliers *et al.*, 1999] has been defined as a weakening of box consistency. Similar ideas have been first developped in [Lhomme, 1993], leading to the definition of $k\mathrm{B}(w)$ consistency.

More precisely, box consistency ensures a property on canonical intervals at bounds of the variable domains. In contrast, box$_\varphi$ consistency imposes a weaker condition by replacing the canonical intervals by intervals of width φ.

Definition 10 (box$_\varphi$ consistency) *Consider a constraint $c(x_1, \ldots, x_n)$, an interval form Γ of c, a real $\varphi \geqslant 0$, and a search space* $\mathbf{b}$. *Let $i \in \{1, \ldots, n\}$ be a natural. The i-th projection of c is* box$_\varphi$ consistent *w.r.t.* $\mathbf{b}$ *and* Γ *if*

$$\boldsymbol{b}_i = \Box\{a_i \in \boldsymbol{b}_i \mid (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{i-1}, \Box\{a_i + \varphi\}, \boldsymbol{b}_{i+1}, \ldots, \boldsymbol{b}_n) \in \Gamma\}.$$

A constraint is box consistent if each of its projections is box consistent. A CSP is box consistent if all constraints are box consistent.

The narrowing operators for box$_\varphi$ consistency are implemented like `BC3revise` except that the dichotomous search for the consistent outermost sub-domains is stopped when the width of the considered sub-domain is smaller than φ. Figure 5.6 describes the process for a real function given an initial domain $[a, b]$: a narrowing operator for box consistency will compute $[c, d]$; a narrowing operator for box$_\varphi$ consistency will compute $[c', d']$ providing the width of both light-grey boxes is smaller than φ.

Box$_\varphi$ consistency then allows one to devise a new adaptive propagation algorithm for achieving box consistency that iterates two steps: a selection of the most contracting pruning functions enforcing box$_\varphi$ consistency; and their application in sequence without any propagation. The key point is that the algorithm increases the precision of the computed consistency by decreasing the φ parameter the more variable domains are tightened (that is, φ is an estimate of the distance to the global fixed-point). Hence, the computed consistency is eventually box consistency if the final value of φ is 0. Note that the selection strategy of the best narrowing operators has been introduced in [Lhomme *et al.*, 1998].

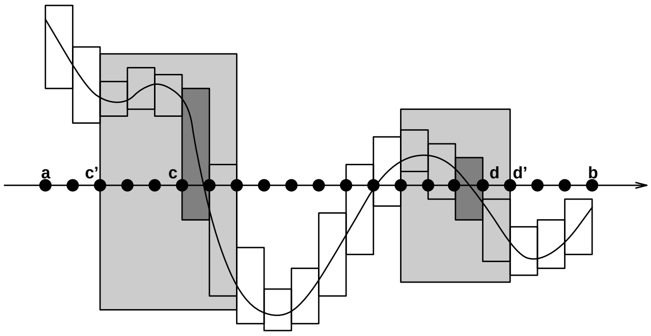


Figure 5.6: Searching for the external zeros of a real function in $[a, b]$.

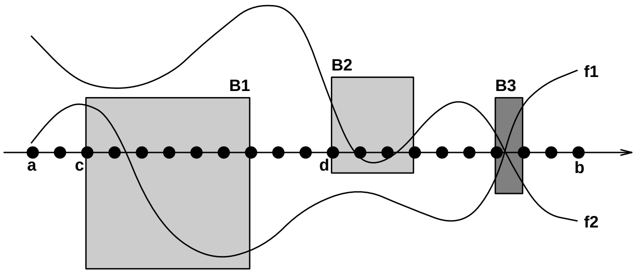


Figure 5.7: Searching for the common leftmost zero of two real functions in $[a, b]$.

The superiority of box$_\varphi$ consistency over box consistency is illustrated in Figure 5.7 that describes the search for the leftmost common zero of two real functions in the interval $[a, b]$. The idea is to enclose a local zero by an interval of width $\varphi > 0$ (the box B1 is first computed from $[a, b]$ using f1, and then the box B2 from $[c, d]$ using f2). However, a common zero (in Box B3 computed from $[d, b]$ using f1) must be more tightly approximated.

5.7 Bound consistency

The efficiency of box consistency lies in the implementation of a search process in the domain of one variable. This idea can be generalized in order to locally process more than one domain. This idea is at the basis of strong consistencies: k consistency [Freuder, 1978] for finite domains, kB consistency [Lhomme, 1993] for continuous domains (see Chapter 7). Note that arc consistency corresponds to a 2 consistency, and that hull consistency and box consistency are variants of 2B consistency.

In [Puget and Van Hentenryck, 1998] the notion of bound consistency (variant of 3B consistency) is implemented to efficiently process a well-known circuit design problem [Ebers and Moll, 1954] [Ratschek and Rokne, 1993]. Basically it consists in combining a search process in the domain of one variable with a constraint propagation algorithm taking as input a set of narrowing operators for box consistency.

Definition 11 (bound consistency) *Consider a CSP $(C, \mathbf{b})$, a set of interval forms Γ associated with all constraint projections from C and a real $w \geqslant 0$. The CSP is bound consistent w.r.t. Γ, w and $\mathbf{b}$ if for every $i = 1, \ldots, n$,*

- *the CSP $(C, \boldsymbol{b}_1 \times \cdots \times \boldsymbol{b}_{i-1} \times (\boldsymbol{b}_i \cap [\underline{\boldsymbol{b}_i}, (\underline{\boldsymbol{b}_i} + w)\!\uparrow]) \times \boldsymbol{b}_{i+1} \times \cdots \times \boldsymbol{b}_n)$ is box$_\Gamma$ consistent w.r.t. $\mathbf{b}$;*
- *the CSP $(C, \boldsymbol{b}_1 \times \cdots \times \boldsymbol{b}_{i-1} \times (\boldsymbol{b}_i \cap [(\overline{\boldsymbol{b}_i} - w)\!\downarrow, \overline{\boldsymbol{b}_i}]) \times \boldsymbol{b}_{i+1} \times \cdots \times \boldsymbol{b}_n)$ is box$_\Gamma$ consistent w.r.t. $\mathbf{b}$.*

The new constraint propagation algorithm is described in Table 5.7. The main operation is to prove the inconsistency of a small interval at one bound of one variable domain ($[a, a + w]$ or $[b - w, b]$) deriving the empty domains by `Prune`. In that case, this subdomain can be removed due to the completeness property of `Prune`.

5.8 Linearization and Cooperation of Pruning Techniques

The cooperation of interval constraint propagation techniques has been devised in [Van Hentenryck *et al.*, 1997a]. This has led to the implementation in the systems `Newton` and `Numerica` of an efficient constraint solver combining box consistency for different interval forms of constraints. These ideas have been extended in [Benhamou *et al.*, 1999] [Granvilliers *et al.*, 1999] [Granvilliers, 2001] in order to avoid useless redundant computations and to accelerate local computations. The solvers essentially combine box consistency over natural forms, box consistency over mean value forms (see the following section), and hull consistency.

```
StrongPrune(S: set of narrowing operators ; b: box ; w: ℝ⁺): box
% w is the precision of the algorithm
begin
  modified := true
  while modified and b ≠ ∅ do
    b := Prune(S,b)                  % pruning of b by constraint propagation
    modified := false
    for i := 1 to n do               % search process for contracting b_i
      let b_i = [a, b]
      d := b_i                       % the new domain is computed in d
      b_i := [a, (a + w)↑] ∩ d
      if Prune(S,b)=∅ then
        d := d ∩ [(a + w)↑, b]       % [a, (a + w)↑ [ is removed from b_i
        modified := true
      fi
      b_i := [(b − w)↓, b] ∩ d
      if Prune(S,b)=∅ then
        d := d ∩ [a, (b − w)↓]       % ](b − w)↓, b] is removed from b_i
        modified := true
      fi
      b_i := d
    od
  od
  return b
end
```

Table 5.4: Constraint Propagation Algorithm for Bound Consistency.

5.8.1 Linearization

In `Numerica`, a linear system is obtained from a system of equations $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ through the mean value form (extended to interval vectors), as follows:

$$\mathbf{f}(\mathbf{a}) + \mathbf{J}(\mathbf{b})(\mathbf{x} - \mathbf{a}) = \mathbf{0}$$

where $\mathbf{a}$ is the center of the variable domains $\mathbf{b}$, and $\mathbf{J}$ is the Jacobian matrix of $\mathbf{f}$. Given $\mathbf{y} = \mathbf{x} - \mathbf{a}$, the linear system is transformed into

$$\mathbf{J}(\mathbf{b})\mathbf{y} = -\mathbf{f}(\mathbf{a})$$

where $\mathbf{y}$ lies in $\mathbf{b} - \mathbf{a}$. This linear system can be handled by the interval Gauss-Seidel method. The algorithm iterates the contraction of the domain of $\boldsymbol{y}_i$ using the row i, as follows:

$$\boldsymbol{y}_i := \boldsymbol{y}_i \cap [(-\mathbf{f}(\mathbf{a})_i - \sum_{j=1}^{i-1} \mathbf{J}(\mathbf{b})_{ij}\boldsymbol{y}_j - \sum_{j=i+1}^{n} \mathbf{J}(\mathbf{b})_{ij}\boldsymbol{y}_j)/\mathbf{J}(\mathbf{b})_{ii}]$$

The variable domains are then updated using the equality $\mathbf{x} = \mathbf{y} + \mathbf{a}$. This method can be iterated until reaching a fixed-point. Unfortunately, this operation does not succeed each time an interval $\mathbf{J}(\mathbf{b})_{ii}$ contains 0 (in this case, the interval evaluation gives $[-\infty, +\infty]$).

To handle this problem, a classical technique is to transform the linear system by multiplying it by a matrix called a *preconditioner*. Hansen [Hansen, 1992] suggests to use the approximate inverse of the center of the Jacobian matrix when the system is not diagonally dominant (and the Jacobian is not singular), i.e., when there exists a row i of the system such that

$$\min(|\underline{\mathbf{J}(\mathbf{b})_{ii}}|, |\overline{\mathbf{J}(\mathbf{b})_{ii}}|) < \sum_{j=1, j\neq i}^{n} \max(|\underline{\mathbf{J}(\mathbf{b})_{ij}}|, |\overline{\mathbf{J}(\mathbf{b})_{ij}}|).$$

Kearfott and Shi [Kearfott and Shi, 1996] have devised optimal preconditioners. Van Hentenryck et al. [Van Hentenryck *et al.*, 1997a] have presented this technique as a particular case of linear combinations of constraints. Van Emden [Van Emden, 1999] has discussed the use of centered forms to generate redundancies from systems of nonlinear equations. These techniques have also been implemented by Hickey to enhance the power of the `CLIP` solver [Hickey, 2000].

In fact, as remarked in [Van Hentenryck *et al.*, 1997a], this technique corresponds to box consistency processing mean value forms of constraints.

5.8.2 Motivation of Cooperation

The solving of a constraint system can generally be described as the interaction of a set of elementary solvers (narrowing operators), each one being associated with a subset of constraints from the system, through the domains of the shared variables [Benhamou, 1996] [Van Hentenryck *et al.*, 1997a] [Apt, 1999]. The domain pruning (constraint propagation) algorithm which combines the narrowing operators is contracting, confluent (the output domains are characterized in terms of a common fixed-point of the solvers), and terminates in finite time. In practice, efficient domain pruning depends on the constraint propagation strategy, that is the order of application of solvers (in order to minimize the computation time for reaching the fixed-point) and the generation of the set of elementary solvers to be combined (in order to maximize the precision). The decomposition of a complex solver in a set of narrowing operators determines the grain of a cooperative strategy. For instance, domain pruning using box consistency can be seen as an indivisible algorithm, or as the combination of narrowing operators associated with either constraints or constraint-variable couples.

Numerical algorithms generally compute a sequence of domains, applying a solver on one constraint to derive a new domain which is then intersected with the current domain. For instance, the Gauss-Seidel method operates in this way, where a row of the linear system can be seen as a constraint. Unfortunately, this kind of independence of computations often leads to a weakening of the precision of computed domains, since the intersection of the computed approximations for each constraint is generally much less precise than the approximation of the conjunction of constraints (the solution set). As a consequence, the preprocessing of a system (e.g., preconditioning step for Gauss-Seidel, generation of cutting planes in integer programming, etc.) and the generation of the set of elementary solvers to be combined (combination of interval extensions, etc.) is a central procedure. In fact, each new constraint (e.g., cutting planes) must be as close as possible to the frontier of the solution set (e.g., polyhedron), and it must be associated with a solver which is effective for this kind of constraint.

Once the set of narrowing operators is known, they must be combined using a fixed-point constraint propagation strategy. First, we remark that a common (global) fixed-point of the narrowing operators must be computed, i.e., the aim is to collectively converge to the fixed-point while minimizing the total computation time. In other words, the solvers must be regularly applied (fair strategy) and the slow convergences occurring in local computations of each solver must be tackled (cooperative strategy). Second, two solvers associated with one constraint possibly derive the same informations (i.e., one can be useless). Hence, two situations has to be distinguished: static (entailment) *vs* dynamic detection of redundancy. Static detection of useless redundant solvers generally derives from theoretic properties, for instance the subdistributivity law of interval arithmetic. Dynamic detection may implement heuristics based on the behavior of a solver with respect to the current domains, for instance by considering that a centered extension is generally more precise than the natural one for tight domains, or ad hoc techniques like the application of solvers to select the most contracting ones for each domain, in order to dismiss the other solvers during a few iterations (see [Lhomme *et al.*, 1998]).

To sum up, the required properties to efficiently solve a constraint system implementing a combination of solvers are the following: adequacy between constraints (expressions) and solvers (techniques), cooperative and fair constraint propagation strategy, and detection of useless redundancy of solvers/constraints. A more difficult issue is to handle a conjunction of constraints as a whole (*e.g.* using preconditioning) instead of intersecting local computations. A possible method is to symbolically generate redundant constraints, e.g., using Gröbner bases [Benhamou and Granvilliers, 1997] [Granvilliers, 1998b], or to enforce strong consistency techniques [Lhomme, 1993].

Following this approach, an efficient domain pruning algorithm combining redundant interval methods is presented in the next section.

5.8.3 Cooperation of interval tightening procedures

On the one hand, hull consistency and box consistency have been combined in Algorithm `BC4` [Benhamou *et al.*, 1999]. Box consistency has been shown to be more precise than hull consistency since it permits avoiding the dependency problem of interval arithmetic over the considered variable (see Examples 19 and 20 below). Furthermore, the precision is the same if this variable occurs once in the constraint (using the chain rule for hull consistency). Nevertheless, the tighter the enclosure, the slower the computation.

Example 19 *Let* $c : x_1^2 - x_2 + x_1 = 0$ *be a constraint,* $\mathbf{b} = ([1,5],[1,4])^T$ *the variable domains, and denote* x_{11} *the first occurrence of* x_1 *in* c*, and* x_{12} *the second one. We report the computations of hull consistency and box consistency for both variables* x_1 *and* x_2 *in* c*, each one over the initial domains:*

$$\begin{array}{llllll}
\textit{hull consistency} & /\, x_{11} & : & \boldsymbol{b}_1 \leftarrow & \boldsymbol{b}_1 \cap \sqrt{\boldsymbol{b}_2 - \boldsymbol{b}_1} & = & [1, 1.74] \\
\textit{hull consistency} & /\, x_{12} & : & \boldsymbol{b}_1 \leftarrow & \boldsymbol{b}_1 \cap (\boldsymbol{b}_2 - \boldsymbol{b}_1^2) & = & [1, 3] \\
\textit{box consistency} & /\, x_1 & : & \boldsymbol{b}_1 \leftarrow & & & [1, 1.57] \\
\textit{hull consistency} & /\, x_2 & : & \boldsymbol{b}_2 \leftarrow & \boldsymbol{b}_2 \cap (\boldsymbol{b}_1^2 + \boldsymbol{b}_1) & = & [2, 4] \\
\textit{satisfaction} & & : & (\boldsymbol{b}_1^2 - \boldsymbol{b}_2 + \boldsymbol{b}_1) \cap [0,0] & & \neq & \varnothing
\end{array}$$

Box consistency for x_1 *in* c *is more precise than hull consistency but the computation is slower since an iteration of splitting/evaluation of* $\boldsymbol{d}_1$ *is performed, though hull consistency only needs one interval evaluation of an inverted expression of* c *(remark that this operation can be numerically performed, see [Benhamou* et al.*, 1999] [Granvilliers and Benhamou, 2001]).*

Example 20 *Let* $C : (\{x_2^2 = x_1^2, x_2^2 = 2x_1^2\}, ([-1,1],[-1,1])^T)$ *be a constraint system. The computation of box consistency for the natural forms of constraints from* S *in our implementation requires* 5409 *calls to a narrowing operator to compute a precise approximation of the solution (there is an asymptotic convergence from* 1 *to* 0*). The computation of hull consistency needs enforcing* 2150 *narrowing operators and is* 20 *times faster.*

On the other hand, it has been shown in [Lhomme, 1993] [Lhomme *et al.*, 1998] that it is not efficient to combine a set of narrowing operators which compute a fixed-point of an iterative process, which strategy may result in slow convergence. In particular, the computation of box consistency can be stopped if a desired precision of the domain to be contracted is obtained [Granvilliers *et al.*, 1999] (see Section 5.6.3). The following example illustrates this assumption.

Example 21 *Let $c_1 : x_1^3 + x_1 = 0$ and $c_2 : x_1^2 = 0$ be two constraints and consider the constraint system $(\{c_1, c_2\}, ([-1,1])^T)$. The computation of box consistency for c_1 needs 4750 evaluation/splitting steps though considering c_2 leads to immediately contracting the domain of x_1, and obtaining the solution of C. Since the contraction power of narrowing operators cannot be statically compared in general, a dynamic criterion has to be implemented for detecting slow convergence. In this example, if the computation of box consistency for c_1 stops when the contraction of domain width is no more than 10% between two consecutive steps of the iterative process, then the number of evaluation/splitting steps dramatically decreases to 24.*

Finally, it is well-known that the Taylor extension is more precise than the natural one given tight domains, and less precise given large domains (see Example 22). Box consistency over the natural and first-order Taylor extensions (interval Newton) have been combined in the solver of `Numerica` [Van Hentenryck *et al.*, 1997a], where the natural extensions are always processed before the Taylor extensions.

Example 22 *Consider the function $f(x) = x^2 - x$. Let $\boldsymbol{f}$ be its natural interval form, $\boldsymbol{m}$ its mean value form, and $\boldsymbol{f}'$ the natural interval form of its derivative. We have:*

$$\begin{array}{rlrl} \boldsymbol{f}^u([0,10]) & = [-0.25, 90] & \boldsymbol{f}^u([0.4,0.6]) & = [-0.25, -0.24] \\ \boldsymbol{f}([0,10]) & = [-10, 100] & \boldsymbol{f}([0.4,0.6]) & = [-0.44, -0.04] \\ \boldsymbol{m}([0,10]) & = [-75, 115] & \boldsymbol{m}([0.4,0.6]) & = [-0.27, -0.23] \end{array}$$

where $\boldsymbol{f}^u(\boldsymbol{x})$ is the range of f over $\boldsymbol{x}$. Let us remark that the range of f can be evaluated using the natural extension of the equivalent expression $(x-0.5)^2-0.25$ where x occurs only once.

We remark that the existing solvers implement a coarse-grained cooperation of constraint propagation algorithms. More precisely, each one is based on a fixed-point strategy, and the whole computation ends with a global fixed-point. Hence a new constraint propagation strategy combining hull consistency, box consistency and interval Newton can be defined:

- the narrowing operators for local consistencies are associated with constraint-variable couples, and interleaved using a propagation queue; hence, the strategy is fine-grained;
- as in `Numerica`, interval Newton is finally applied in order to refine a tight domain enclosing a solution.

Let us remark that it is in general better to bisect a domain rather than executing further domain pruning, i.e., it is useless to reuse local consistencies after interval Newton: if interval Newton is useful then the natural extensions of constraints are probably useless; otherwise, they have just been considered.

The mechanisms to handle slow convergences are the following: each narrowing operator for box consistency is stopped before reaching a fixed-point (parameter φ); the propagation step is executed only if a domain has been reduced enough (parameter w); only one iteration of Gauss-Seidel is performed. The aim is to stop the computation of a sequence of included domains when the size of domain contraction becomes too weak.

The domain pruning algorithm is contracting (the combination of contracting operators is contracting), complete (no solution of the constraint system is lost), and terminates in finite time (since each step is contracting and $\mathbb{I}$ is finite). Nevertheless, it is not idempotent, *i.e.* the output domains may not be a fixed-point, since the propagation process can be stopped when the reductions become smaller than a given $w \geqslant 0$.

This combination of hull and box consistency just ensures to associate the best narrowing operators with constraints (individual criterion). Interleaving these operators (see Example 23) is a step towards the implementation of the best strategy for minimizing the total computation time (which is due to obvious reasons of cooperation for collectively solving a problem).

Example 23 *Let* $c_1 : \sum_{i=1}^{20}(x_1 - i) = 0$*,* $c_2 : \sum_{i=1}^{20}(x_1 - i - 0.5) = 0$*, and* $c_3 : x_1 \leqslant 1$ *be three constraints, and* $\{c_1, c_2, c_3\}, ([-100, 100])^T)$ *a constraint system. A constraint propagation algorithm which alternatively enforces box consistency over* c_1 *and* c_2 *must compute the sequence of domains* $[0, 20] \subset [0, 19.5] \subset \cdots \subset [0, 0]$*. If* c_3 *is taken into account during propagation, for example just after the use of* c_1 *and* c_2*, the sequence becomes* $[0, 20] \subset [0, 19.5] \subset [0, 1] \subset [0, 0.5] \subset [0, 0]$*. This suggests to regularly apply a narrowing operator which belongs to the propagation queue.*

In practice, the use of interval Newton generally permits the separation of solutions (tight domains which certainly contain a solution) from quasi-solutions (tight domains which possibly contain a solution). Problem `Cyclohexane`, modeling the 3-dimensional structure of the cyclohexane molecule, typically illustrates this situation (see Example 24).

Example 24 *The system is composed of three constraints of the form* $13 + x_i^2(1 + x_j^2) + x_j(x_j - 24x_i) = 0$ *where* $(i, j) = (1, 2), (2, 3), (3, 1)$*; each domain is equal to* $[-10^8, 10^8]$ *and there are* 16 *real solutions. Box consistency alone, embedded in a bisection algorithm, results in* 61 *output domains (remark that* 45 *domains do not contain any real solution) after* 1864 *bisection steps and* 40938 *calls to a narrowing operator. Box consistency combined with interval Newton only requires* 355 *bisections to compute* 16 *domains, after* 7168 *calls to a narrowing operator for box consistency,* 464 *applications of interval Newton, and the whole process is six times faster.*

Chapter 6

Symbolic-Interval Cooperation

Laurent Granvilliers
Eric Monfroy
Frédéric Benhamou
Institut de Recherche en Informatique de Nantes (IRIN)
Université de Nantes
44322 Nantes cedex 3, France

This chapter surveys the field of cooperative constraint solving for a constraint programming perspective with an emphasis on combinations of symbolic and interval methods. On the one hand, symbolic methods provide adapted representations of the constraint expressions. On the other hand, interval methods compute verified enclosures of solution sets. Using cooperation of solvers, one can take advantage of both techniques in a unified framework: symbolic algorithms generally need to be combined with root extraction methods, and the efficiency of interval algorithms strongly depends on constraint expressions.

6.1 Introduction

In the constraint programming [Van Hentenryck *et al.*, 1996] community, solver cooperation is now well-known as a concept for improving efficiency and performance of constraint solvers. Generic solvers are generally far too inefficient for solving numerous real-life problems. However, a large part of these problems can often be handled by specific and efficient (but possibly not complete, i.e., that cannot prove inconsistency) solvers. For instance, an efficient algorithm

S	Deduced constraints	T	Result
1		50	$y = 0$, $x \approx -0.4999999851$
2		100	$x \in [-0.50, -0.49]$, $y \in [0.99, 1.01]$
3	$x(y-1) = 0$	20	$x \in [-0.50, -0.49]$, $y = 1$
4	$y - 1 = 0$	4	$x \in [-0.50, -0.49]$, $y = 1$
5	$y = 1 \wedge x^{25} + 2x + 1 = 0$	2	$x \in [-0.50, -0.49]$, $y = 1$
6	$x(y-1) = 0$ $\wedge(y - 1 = 0 \vee x = 0)$	1	$x \in [-0.50, -0.49]$, $y = 1$

In this table T represents time units, and S represents the following solvers respectively:

1: Maple

2: Interval

3: Factorize–Interval

4: Gröbner–Interval

5: Gröbner–Substitute–Interval

6: Factorize–Substitute–Interval–Split

Table 6.1: Improving efficiency with cooperation.

such as the Simplex can process a linear sub-problem extracted from a nonlinear constraint system. In this case, cooperation consists in using the result of Simplex within a nonlinear solver.

The goal of cooperation is to share and exchange data between solvers in order to tackle new types of problems (for which there is no solver), to improve the representation of solutions, and/or to speed-up computation. However, three major problems arise when dealing with cooperation of solvers:

- *Theoretical issues* Does/How a solver cooperation preserves properties (such as correctness, completeness, and termination) of the involved solvers?
- *Communication* Which types of data must be shared and exchanged between solvers? Which protocol of communication should be considered?
- *Strategies of cooperation* Which solvers should be used? How should they be combined (sequentially, concurrently, in parallel)? How to control the solving process?

These requirements are generic and hold for every type of cooperations. Nevertheless, there is no universal framework for overcoming these problems. The usual approach is thus to tackle restricted or specific fields of cooperation. Since both symbolic computation and interval solvers are relevant to the constraint programming community, and *a fortiori* their cooperation, we propose to overview systems, frameworks, and languages realized for the specific case of symbolic-interval cooperations for nonlinear systems defined by a set of equations and inequations:

$$\begin{cases} f_1(x_1,\ldots,x_n) = 0 \\ \quad\vdots \\ f_m(x_1,\ldots,x_n) = 0 \\ g_1(x_1,\ldots,x_n) \leqslant 0 \\ \quad\vdots \\ g_p(x_1,\ldots,x_n) \leqslant 0 \end{cases}$$

where each variable lies in a domain, i.e., the set of possible values for the variable. By *solving* we mean computing a representation of the solution set.

The field of symbolic-numeric algorithms has much grown during the last decade. When the numerical processing is an interval computation [Moore, 1966], the computed approximations are guaranteed to be reliable: the real solutions are enclosed by the computed intervals. Reliability (or completeness) is an important property for many applications (especially for security assessment, disaster prevention, etc.). Consequently, the study of constraint solvers which combine symbolic and interval methods is particularly relevant.

So far, Gröbner basis computation [Buchberger, 1985] and quantifier elimination [Tarski, 1951, Collins, 1975] are the two main symbolic methods used in combination with interval constraint solving. Both are algebraic methods bringing various advantages. A Gröbner basis is a particular basis of a system of polynomial equations from which a lot of information can be deduced, such as the existence of complex solutions, or the finiteness of the solution set. But Gröbner bases are also essential to the efficiency of the solving process: by introducing redundancies (i.e., S-polynomials) and inferring univariate constraints, they significantly speed-up interval solvers. On the other hand, quantifier elimination in the first order theory of real closed fields is a decision procedure of solvability of constraints over real numbers. It can also be used for eliminating non-query variables (variables generated in programs) from answer constraints, and to process the solution set in order to provide the user with convenient solutions.

More generally, there are two categories of cooperative methods for constraint solving: the master-slave approach and the full cooperation. The first one consists in using a solver to enhance the power of another solver. Here are some examples:

- the precision of a numeric algorithm can be improved by using a better representation of the constraint expressions obtained from a symbolic computation;
- slow convergences can be handled by generating redundant constraints;
- or, a numeric algorithm may derive numerical approximations from a constraint system in normal form (e.g., a Gröbner basis).

The second approach views the cooperative method as a new solver in itself. The main motivation is in general to extend the class of solvable problems of each individual solver.

We illustrate the main characteristics of cooperative solvers by means of three examples. The first one is made of two equations together with the domains of the variables involved:

$$\begin{cases} xy - x = 0 \\ x^{25} + 2xy + 1 = 0 \\ x \in [-10, 10] \\ y \in [-10, 10] \end{cases}$$

There is only one solution: $y = 1$ and $x \in [-0.50, -0.49]$. Different kinds of solvers can be implemented:

- Primitive `solve` of `Maple` [Geddes *et al.*, 1991] deduces that $y = 1$ and x is root of $z^{25} + 2z + 1$. Then, `fsolve` computes a floating-point number (incorrect value) approximating the true value.
- An interval solver, iterating splitting and contractions of the variable domains, is able to enclose the solution by a couple of intervals for x and y. Note that the exact value for y is not computed.
- If equation $xy - x = 0$ is first transformed into $x(y-1) = 0$, then the same interval solver computes the exact value for y (the precision is enhanced since there is only one occurrence of x in the constraint), and a reliable approximation for x.
- A standard Gröbner basis computation derives the redundant constraint $y - 1 = 0$. The computation of the interval solver is then 20 times faster than the solving of the initial system, and 5 times faster than the solving of the system with $x(y-1) = 0$.
- Given equation $y - 1 = 0$, y can be substituted with 1 in the second equation. Once again, the computation is faster (factor 2).

Solvers	Deduced constraints	Result
1		$x = 0$
2		$x \in [-0.33, 1.01]$, $y \in [-2.01, 2.01]$, and $z \in [-2.01, 2.01]$
3	$x = 0$	$x = 0$, $y \in [-1, 1]$, and $z \in [-1, 1]$
4	$x = 0$, $y \leqslant 0$	$x = 0$, $y = -1$, $z = 1$

In this table S represents the following solvers respectively:

1: Simplex

2: Interval

3: Simplex–Interval

4: Simplex–Symbolic–Interval

Table 6.2: Refining results with cooperation.

- Finally, $x(y-1) = 0$ can be split into $x = 0$ or $y - 1 = 0$, which leads to a disjunction of two systems. Over both systems, a set of transformations (substitutions, etc.) can be applied. The symbolic computation itself allows one to declare that the first system has no solution. The solution can be numerically derived on the second system (interval solving of $x^{25} + 2x + 1 = 0$). The whole process is the fastest on this problem.

This example is illustrated and summarized in Table 6.1.

The second example is a conjunction of two constraints together with variable domains:

$$\begin{cases} xy = 0 \\ y - x^2 = 0 \\ x \in [-1, 1] \\ y \in [-1, 1] \end{cases}$$

A specific interval solver based on box consistency [Benhamou *et al.*, 1994] computes a new domain for y, that is $[0, 1]$, and no reduction on the domain of x is performed. When it is combined with domain splitting, then there is a slow convergence of the numeric process, and a couple of intervals enclosing the solution $(0, 0)$ is derived after numerous splitting operations. The trivial symbolic computation (either using substitution or Gröbner basis computation) of $x^3 = 0$ improves the numerical solving, and the exact solution is computed without splitting.

The third example illustrates the use of a linear solver. The constraint system is a conjunction of four linear constraints, $x + y + z = 0$, $y + z \leqslant 0$, $x \leqslant y + z$, and $z \geqslant 0$, and two nonlinear equations, $3x - z^2 + 1 = 0$, and $-6x^2 + y^2 + z^2 = 2$, with $x, y, z \in [-10, 10]$. The Simplex algorithm generates $x = 0$ from the inequalities. The interval solver alone (the one previously mentioned) contracts the domains, $x \in [-0.33, 1.01]$, $y \in [-2.01, 2.01]$, and $z \in [-2.01, 2.01]$. If both solvers cooperate through the sharing of the bounds of domains, then the new domains are narrower, $x = 0$, $y \in [-1, 1]$, and $z \in [-1, 1]$. Now, if the redundant constraint $y \leqslant 0$ is first generated (by combining $x \leqslant y + z$ and $z \geqslant 0$), the solution $(0, -1, 1)$ is derived. See Table 6.2 for a summary of this example.

This chapter is structured as follows. First, we introduce designs of interval solvers based on constraint propagation techniques and interval arithmetic (Section 6.2). In order to improve solvers efficiency and precision, more subtle interval solvers are designed as low level intra-cooperations of symbolic (mainly syntactic) transformations and standard propagation techniques (Section 6.3). In the constraint programming community, the symbolic process aims at finding "good" representations of constraint systems. By "good", we mean representations on which interval solvers are more efficient than on the input system (e.g., redundant constraints, univariate constraints, linearized constraints, factorized constraints, or inverted constraints). We present briefly major symbolic techniques (e.g., Gröbner basis and quantifier elimination) that are usual to the constraint programming community (Section 6.4). The framework of cooperative solvers is presented in Section 6.5: firstly some attempts of generic frameworks for cooperation, then some practical realizations of specific symbolic-interval cooperations, and finally some systems for managing interactions between solvers in order to design high level cooperations. Some directions for future research are given in Section 6.6.

6.2 Interval Constraints

In the following, let $\mathbb{R}$ be the set of real numbers.

6.2.1 Interval Arithmetic

Interval arithmetic [Moore, 1966] is implemented in constraint propagation to compute reliable bounds of real expressions. Section 5.2 of Chapter 5 describes interval arithmetic.

6.2.2 Constraints

Let $\Sigma = (\mathbb{R}, \{+, -, \times, \mathrm{div}, \log, \exp, \sin, \cos\}, \{=, \leqslant, \geqslant\})$ be a real-based structure and let consider its usual interpretation: $\mathbb{R}$ is the set of reals, $+$ is the mapping $(x, y) \mapsto x + y$, etc. Let V be the set of real-valued variables $\{x_1, x_2, \ldots, x_n\}$. Given $\mathrm{r} = (r_1, \ldots, r_n) \in \mathbb{R}^n$ and a set of variables $X = \{x_{i_1}, \ldots, x_{i_j}\}$ such that $1 \leqslant i_1 < \cdots < i_j \leqslant n$, let $\mathrm{r}_{|X}$ denote the vector $(r_{i_1}, \ldots, r_{i_j})$ (projection of r over X).

A *term* is a syntactic expression built from real numbers, operations and variables in the usual way. A *constraint* is a conjunction of atomic formulas made from terms and relation symbols, e.g., a nonlinear equation/inequation. Given a k-ary constraint c (constraint over k variables), let ρ_c denote the set of elements of $\mathbb{R}^k$ verifying c in the usual interpretation of Σ, that is the *relation* defined by c. Two constraints defining the same relation are said to be *equivalent*. Given a constraint c, every element of ρ_c is said to satisfy —to verify or to be consistent with— c.

A *constraint system* C is given by a conjunction of constraints $c_1 \wedge \cdots \wedge c_m$ and a vector of domains $I = (I_1, \ldots, I_n)$ from $\mathbb{I}^n$ where I_k is the domain of x_k for $k = 1, \ldots, n$. Let Var_k denote the set of variables occurring in c_k for $k = 1, \ldots, m$. A *solution of* C is an element r of $\mathbb{R}^n$ included in I such that $\mathrm{r}_{|\,Var_k}$ verifies c_k for $k = 1, \ldots, m$. A constraint system is said to be *inconsistent* if its solution set is empty; otherwise, it is consistent.

Example 25 *Let $C = (x \geqslant y \wedge x^2 + y^2 = 1, (I_1, I_2))$ be a constraint system. The solution set of C is the set $\{(r_1, r_2) \in I_1 \times I_2 \mid r_1 \geqslant r_2 \wedge r_1^2 + r_2^2 = 1\}$. Given $I_1 = [0, 2]$ and $I_2 = [-1, 1]$, the point $(1, 0)$ is a solution to C.*

6.2.3 Constraint Solving

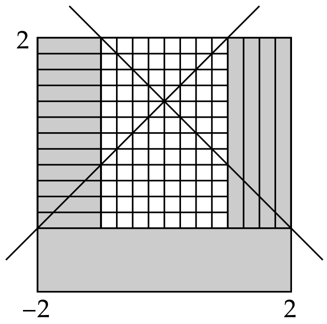
Constraint satisfaction over the real numbers —computation of a constraint relation or more generally a solution set of a constraint system— is generally intractable due to the limitations of machine arithmetic. Cleary [Cleary, 1987] has proposed to implement constraint satisfaction over the intervals, which is based on the notion of interval form of a constraint relation. Constraint solving is presented in Section 5.4 of Chapter 5.

6.3 Combining Interval Forms

The efficiency of pruning algorithms strongly depends on the precision of interval forms. A possible answer to this problem is combination of interval forms which can be seen as a type of internal cooperation used for designing solvers. Combining Interval Forms is presented in Section 5.2 of Chapter 5.

6.4 Combining Constraints

The aim is to enclose the solution set of a constraint system, that is to compute an approximation of the intersection of the constraint relations and the variable domains. Nevertheless the common procedure is to compute the intersection of an approximation of each relation, which can reveal much less precise (see Figure 6.4).



The constraint system is the following:

$$\begin{cases} y + x = 1 \ \wedge \ y - x = 1 \\ -2 \leqslant x, y \leqslant 2 \end{cases}$$

The gray boxes can be discarded. The white one cannot be contracted while removing elements of each relation taken independently.

Figure 6.1: Independence of domain contractions.

The motivation is then to combine constraints, that is to transform their syntactic expressions to be numerically processed. Combining both constraints appearing in Figure 6.4 leads to $y = 1$. This constraint is said to be *redundant*.

Definition 12 *Let $S = (C, \mathrm{I})$ and $S' = (C \wedge c, \mathrm{I})$ be two constraint systems. If S and S' have the same solution set, then the constraint c is said to be* redundant *with respect to S. In a dual manner, S can be seen as a* simplification *of S'.*

In practice, the aim is either to generate redundant constraints to perform stronger approximations, or to simplify a system by removing useless constraints (for numerical computations). Only the condition to preserve the solution set is required.

6.4.1 Linear constraints

There exist many algorithms that first transform (approximate) a nonlinear system into a linear system before applying methods from linear algebra (see e.g., [Alefeld and Herzberger, 1983, Rump, 1992]).

From an initial arbitrary system of equations $\mathrm{f}(\mathrm{x}) = 0$, the linear system can be obtained by using the first-order Taylor form, that is

$$\mathrm{f}(\mathrm{x}_0) + \mathrm{f}'(\mathrm{I})(\mathrm{X} - \mathrm{x}_0) = 0$$

where x_0 is the center of the variable domains I. Given $\mathrm{Y} = \mathrm{X} - \mathrm{x}_0$, the linear system is

$$\mathrm{f}'(\mathrm{I})\mathrm{Y} = -f(\mathrm{x}_0)$$

where Y lies in $\mathrm{I} - \mathrm{x}_0$. Since the Gaussian elimination algorithm is not so precise with interval coefficients [Beaumont, 1998], the Gauss-Seidel iterative method is often used. Let us write the system $\mathrm{AY} = \mathrm{b}$. Then each row i of the system is used to contract the domain of Y_i by means of the following instruction:

$$Y_i := Y_i \cap [(b_i - \sum_{j=1}^{i-1} A_{ij} Y_j - \sum_{j=i+1}^{n} A_{ij} Y_j)/A_{ii}]$$

The variable domains are then updated using the equality $\mathrm{X} = \mathrm{Y} + \mathrm{x}_0$. This method can be iterated until reaching a fixed-point. Unfortunately, this operation does not succeed each time an interval A_{ii} contains 0 (in this case, the interval evaluation gives $[-\infty, +\infty]$). A classical technique is to *precondition* the linear system, i.e., to multiply it by a matrix called a *preconditioner*. Hansen [Hansen, 1992] suggests to use the approximate inverse of the center of the Jacobian $\mathrm{f}'(\mathrm{I})$ when the system is not diagonally dominant (and the Jacobian is not singular), i.e., when there exists a row i of the system such that

$$\min(|\underline{A_{ii}}|, |\overline{A_{ii}}|) < \sum_{j=1, j\neq i}^{n} \max(|\underline{A_{ij}}|, |\overline{A_{ij}}|).$$

Kearfott and Shi [Kearfott and Shi, 1996] have devised optimal preconditioners. Van Hentenryck et al. [Van Hentenryck *et al.*, 1997a] have presented this technique as a particular case of linear combinations of constraints. Van Emden [Van Emden, 1999] has discussed the use of centered forms to generate redundancies from systems of nonlinear equations. These techniques have also been implemented by Hickey [Hickey, 2000] to enhance the power of the `CLIP` solver.

The same way, one may generate cutting planes [Barth and Bockmayr, 1995] for deriving a tighter enclosure of a polyhedron defined by a set of linear inequalities.

From a quasi-linear system (nonlinear system where most terms are linear), Yamamura et al. [Yamamura *et al.*, 1998] suggest to replace each nonlinear term with a variable lying in its interval evaluation on the current domains, and then to apply the Simplex algorithm. Let

$$a_1 x_1 + \cdots + a_n x_n + h(x_1, \ldots, x_n) \bowtie 0$$

be such a constraint where $h(x_1, \ldots, x_n)$ is nonlinear. Let $[\alpha, \beta]$ be the evaluation of the natural form of h over the current domains. Then the following three constraints are added in the set of constraints to be processed by the Simplex, z being a new variable:

$$\begin{cases} a_1 x_1 + \cdots + a_n x_n + z \bowtie 0 \\ z \leqslant \beta \\ z \geqslant \alpha \end{cases}$$

In fact the nonlinear term is just surrounded by a rectangle.

Finally, Yamamura [Yamamura, 2000] also proposes to linearly combine nonlinear constraints. For instance from $f = 0 \wedge g = 0$ the redundant constraint $f - g = 0$ is generated. Nevertheless this kind of transformation seems to be useful only for quasi-linear systems, i.e., when the expression $f - g$ can be sufficiently simplified with respect to the expressions of f and g. Some more powerful (and general) combinations will be introduced in the next section.

6.4.2 Nonlinear combinations

A *Gröbner basis* is a particular basis of the polynomial ideal generated by a set of polynomials. From an algorithmic point of view [Buchberger, 1965, Buchberger, 1985] a Gröbner basis is obtained (in finite time) by iteratively generating and reducing redundant polynomials called *S-polynomials*. Given a total and Noetherian order on monomials which is compatible with the multiplication, a S-polynomial of a pair of polynomials p and q is just a nonlinear combination of p and q that cancels their maximal (leading) terms (with respect to the given order), that is

$$S(p,q) = \frac{Lcm(Lm(p), Lm(q))}{Lt(p)} \cdot p - \frac{Lcm(Lm(p), Lm(q))}{Lt(q)} \cdot q$$

where Lt is the leading term of a polynomial, Lm is the leading monomial (a monomial is a power product of variables, a term is the multiplication of a coefficient and a monomial), and Lcm stands for the least common multiple of two monomials.

The reduction process iterates the Euclidean division of the S-polynomial with respect to the current set of polynomials. The result is either 0 or a nonzero polynomial which is added to the set.

Given a set of polynomial equations, redundant constraints can be added by Gröbner basis computations [Monfroy, 1992, Benhamou and Granvilliers, 1997]. In fact, $S(p,q) = 0$ (which is trivially redundant) can be generated from $p = 0$ and $q = 0$ (see Example 26).

Example 26 *Let the following system of equations*

$$\begin{cases} x + y + z^2 & = 1 \\ x + y^2 + z & = 1 \\ x^2 + y + z & = 1 \end{cases}$$

to be processed by Gröbner basis computations. Using the pure lexicographic order, the following redundant equations are generated:

$$\begin{cases} y^2 - y - z^2 + z & = 0 \\ z^6 - 4z^4 + 4z^3 - z^2 & = 0 \end{cases}$$

The main problem is to control the amount of symbolic computations. We just remark that the full Gröbner basis is no more required, and efficient stopping criteria are necessary to make this technique practicable. This aspect will be discussed in Section 6.5.

6.4.3 Quantifier elimination

Many mathematical problems (such as the solvability of systems of polynomial equations) can be formulated in the first order theory of real closed fields: they are stated as quantified ($\exists$, for all) formulas composed of conjunctions and disjunctions of equations, disequations, and inequations between polynomials. A decision procedure for this theory, i.e., a *quantifier elimination* algorithm was first devised by Tarsky [Tarski, 1951]. Since then, various algorithm were designed to improve the original method, such as Collins' *cylindrical algebraic decomposition* algorithm [Collins, 1975], or Hong's *partial cylindrical algebraic decomposition* algorithm [Collins and Hong, 1991].

The results of a quantifier elimination procedure depends on the existence of free variables in the given formula. If the formula contains free variables, the result is a quantifier free formula equivalent to the input formula, which describes a relation between the free variables. In the other case (no free variable), the result of eliminating variables is just *True* of *False*.

Let $\hat{F} = (Q_{f+1}X_{f+1}) \dots (Q_m X_m) F(X_1, \dots, X_m)$ be a prenex formula in the first order theory of real closed fields where Q_i are quantifiers, F is a quantifier free formula in which terms are polynomials with rational coefficients, m is the number of variables of $\hat{F}$, and f is the number of free variables ($0 \leqslant f < m$). The essential idea of Collins' method is a certain decomposition of $\mathbb{R}^m$ such that in each cell of the decomposition, the polynomials occurring in the formula F have constant signs. Collins' algorithm based on cylindrical algebraic decomposition (CAD) proceeds as follows:

1. Projection: the m-variate polynomials are *projected* into a set of $(m-1)$-variate polynomials. This process is iterated until univariate polynomials are obtained.

2. Base: by applying a real root isolation method on the univariate projection polynomials, a CAD of 1-space is realized.

3. Extension: successive extensions are computed until a CAD of m-space is constructed: a CAD of i-space is realized by building "stacks" of cells over the cells of the CAD of $(i-1)$-space.

4. Evaluation: the truth value of F is evaluated (by checking the signs of the polynomials on the cell) on each cell of the CAD of m-space.

5. Propagation: truth values on the cells of the CAD of $i-1$-space are determined from the truth values on the CAD of i-space. Finally, the union of the *True* cells of the CAD of $\mathbb{R}^f$ constitutes the solution set of $\hat{F}$.

6. Solution: a quantifier free formula defining the solution set (i.e., a disjunction of defining formulas for each true cell) is built.

6.4.4 Symbolic simplification

The simplification of the constraint expressions aims at handling the dependency problem of interval arithmetic. Intuitively, this means decreasing either the number of occurrences of variables in the system, or the number of variables in some constraints. One can identify the following operations: factorization; replacement of a ground variable (the domain is reduced to a real number) with its domain; substitution from an equation $f = g$, where f is replaced with g in another constraint, such that the new constraint contains less occurrences of variables; etc.

A main condition that can be required for the simplification rules is that the numerical computation over the new systems is more precise. As a consequence, a redundant constraint can be added each time this condition cannot be guaranteed (but it may lead to a slower computation).

6.4.5 Constraint splitting

In [Granvilliers, 1998b], the dependency problem of interval arithmetic is partly handled by means of constraint splitting. More precisely, given a constraint system $(C \wedge f \times g = 0, \mathrm{I})$, the union of both systems $(C \wedge f = 0, \mathrm{I})$ and $(C \wedge g = 0, \mathrm{I})$ is generated. The solution set of the initial system is equivalent to the union of the solution sets of the new systems.

Example 27 *Using interval arithmetic, solving* $(x + y = 0 \wedge xy = 0, [-1, 1]^2)$ *cannot contract the box, because of the independence of interval contractions. If the constraint* $xy = 0$ *is split into* $x = 0 \vee y = 0$*, then the solution* $(0, 0)$ *is trivially computed.*

This transformation rule is combined with simplification rules. The symbolic-numeric algorithm first generates a tree of constraint systems which is then numerically processed.

6.4.6 Abstraction

An *abstraction* is an operation that consists in replacing an expression f shared by several constraints with a variable x and to add the equation $x = f$ in the system. The generalization of this transformation is the creation of the directed acyclic graph of all constraints in the system. The main improvement is to speed up the numerical computation since less expressions need to be evaluated. Another —more subtle— effect can be encountered. In fact, the new system is symbolically different from the initial system, which may influence some techniques such as the preconditioning of a linear system. Unfortunately, no general condition is stated with respect to the quality of the new system.

6.5 Cooperative Solvers

This section focuses on solver cooperation, which is now well-known as a concept for improving performances of solvers in the constraint programming community. Three major problems arise when dealing with cooperation of solvers: theoretical issues, communication, and strategies.

Theoretical issues Properties such as correctness, completeness, and termination are relevant to solver cooperations. They are generally well-known for single solvers, but when several solvers cooperate, one has to demonstrate that these properties are preserved. Completeness ensures that no solution is lost by the solving process. On the other hand, correctness ensures that the solution space after resolution is included in the one of the input system. It is generally difficult to obtain this last property. A solver, and more especially a solver cooperation preserving these two properties is the ideal case. Let C be the initial constraint system, and S_c be the cooperative solving process. We denote by $\mathcal{S}ol(C)$ the solution of C:

$$\begin{array}{rlcl} \text{Correctness:} & \mathcal{S}ol(S_c(C)) & \subseteq & \mathcal{S}ol(C) \\ \text{Completeness:} & \mathcal{S}ol(C) & \subseteq & \mathcal{S}ol(S_c(C)) \\ \text{Ideal cooperation:} & \mathcal{S}ol(C) & = & \mathcal{S}ol(S_c(C)) \end{array}$$

Communication The (type of) data that must be shared and exchanged between solvers have to be determined; an underlying issue consists in identifying sub-parts of problems such as the linear part of a constraint system. Similarly, auxiliary information such as semantic information (convexity, linearity, primitive constraints) can be communicated with data. A protocol must be considered to fix the type of communication: should a solver send new data as soon as they are deduced, or should it wait until it terminates? The same problem arises at the other side of the communication: integrating new data as soon as they arrive, or collecting them when the other solver has finished. In other terms, the operational semantics of the cooperation must be specified.

Strategies of cooperation Strategies consists in determining which solvers should be used and for which purpose. For instance, Gröbner basis can change constraint representation, and consequently, speed-up an interval solver. But there are also other concerns: for instance, Gröbner basis can linearize constraints, but, it is generally not worth applying a nonlinear solver to accelerate a linear one. When solvers have been selected (let say A and B), other choices of strategies arise:

- sequential solving process: run first A and then B;
- parallel solving process: run A and B in parallel, and compose both results to obtain the result of the cooperation;
- concurrent solving process: run A and B in parallel, and keep the result of only one solver;
- fixed-point solving process: run A, then B, then A again and B, and so on until no solver is able to deduce new information.

The next three subsections present systems and models that proposes possible answers to each of the three major issues.

6.5.1 Theoretical frameworks of cooperations

Among the attempts that have been made to provide a theoretical framework for cooperative solvers, one can cite the theoretical model of Prolog III and Prolog IV by Colmerauer [Colmerauer, 1990, PrologIA, 1996], a framework based on narrowing operators proposed by one of the authors [Benhamou, 1996] and the use of chaotic iterations [Cousot and Cousot, 1977] to describe propagation-oriented approximations [Apt, 1999].

Since we are interested here in the cooperation of numeric and symbolic methods, we will only focus on the part of these works that is dedicated to this aspect.

The theoretical model of Prolog III introduced in some basic sense this type of cooperation by introducing *delayed constraints*. Delayed constraints are constraints that are syntactically acceptable by the system, but are too complex to be processed by the main solvers (e.g., non linear constraints with a linear solver). If at any step of the computation a sufficient enough amount of information is extracted from the constraint store (i.e., the current constraint system to be solved) to transform these delayed constraints into constraints that can be processed by the main solver, the symbolic transformation is achieved, and the modified constraint is added to the store. In Prolog IV, the cooperation is more complex, since, while keeping this notion of delayed constraints, the system will constantly move constraints from an interval-constraint based propagation solver to an infinite precision-based linear solver, as soon as variables are bounded to constant values.

The idea on which the theory of cooperation relies in this case consists in: a) considering a unique constraint structure, general enough to incorporate the specific domains of every solver, and b) giving the axioms describing the solvers in such a way that there exists a non-standard model for this structure. On this structure, the global cooperating solver is a terminating, correct, complete, and confluent decision procedure for constraint satisfiability. A specific example, for multiplication, is given in [Colmerauer, 1993]. A second example, involving — among others— interval constraints, linear constraints and delayed constraints is given in [PrologIA, 1996]. Interestingly enough, the consistency of the set of axioms given to model the cooperation behavior for Prolog IV is still an open problem.

Another attempt to provide a theoretical framework for cooperation was proposed by one of the authors in [Benhamou, 1996]. The idea is to extend the notion of narrowing operators to take into account the (symbolic) processing of the store (viewed as a set of formulas). Contracting operators are then applied to pairs of the form (X, S), where X is a representation of variables' domains (e.g., a box) and S is a set of formulas (constraints). Numerical approximation is described with domain contractions, while symbolic transformations are expressed with rewriting rules or first order axioms. One of the problems is then to define a partial order on these pairs, in order to build a complete lattice and to express the cooperation as a fixed point computation over this lattice. The paper studies the approach over some examples, including delayed multiplications and exchanges of numerical constants between a Simplex-like algorithm over perfect precision rational numbers and a floating point interval constraint-based propagation and splitting methods.

As pointed out in this paragraph, even though some attempts to provide a uniform framework to describe the main static properties of numeric-symbolic cooperation schemes have been contributed in the last few years, none of them is general enough to provide a set of minimal properties over the solvers and over the cooperation scheme which would guarantee, in most cases, termination, correctness, completeness, and confluence. Nevertheless, as remarked in [Granvilliers, 1998b], only termination and completeness are guaranteed for most of the symbolic-numeric solvers implemented in existing constraint programming systems.

6.5.2 Ad hoc cooperations

This type of systems are based on one cooperation concept (e.g., sequential solving process, or concurrent communication) and one strategy of resolution: the solvers are fixed, and the routing of constraints through the solvers is fixed *a priori*. The basic idea is: first, pre-process the system of constraints with a symbolic solver; then, use a numeric solver to find the roots. However, the cooperation can be more complex, and the numeric solver can become a pre-process to the symbolic one (such as using an interval solver to tighten domains before using a Simplex solver). Correctness and termination of the cooperation is generally straightforward since the solvers and their properties are well-known.

We first present basic components of cooperations that are then the bricks that compose more complex ad hoc cooperations.

Basic cooperations

Each of the following basic cooperations describes a one to one cooperation between a solver A and a solver B: A is applied to a constraint system, and B is then run on (a filtered part of) the resulting constraint system. The final result of the cooperation is the conjunction of the system B processed and the result of A that B did not handled because of the filter. Filters can be mandatory (e.g., when B cannot handle all types of constraints A can return), or just a choice of the designer of the cooperation.

Cooperation Gröbner to (nonlinear) interval The goal is to speed-up interval reduction by introducing redundancies (i.e., S-polynomials) computed by the Gröbner basis solver. When using a pure lexicographic order, Gröbner basis of system having finitely many solutions are in the following triangular form, i.e.,

$$\begin{cases} p_1(x_1) \\ \dots \\ p_i(x_1, x_2) \\ \dots \\ p_j(x_1, x_2, x_3) \\ \dots \\ p_l(x_1, \dots, x_n) \end{cases}$$

Using this type of polynomial systems, interval solvers are highly efficient. Propagation can be performed from x_1 to x_n.

However, computing Gröbner basis is very costly in time, and the fully computed Gröbner basis is generally not required. A trade-off between symbolic computation and the speed-up brought to the numeric solver has to be determined. Several solutions have been tried to control the amount of symbolic computations [Benhamou and Granvilliers, 1997]. The first one consists in stopping Gröbner basis computation as soon as a univariate polynomial $p(x)$ is deduced. Indeed, a constraint of the type $p(x) = 0$ extremely speeds-up the interval solver by facilitating reduction of the domain of x. Another attempt is to compute Gröbner basis down to a given depth: each level consists in computing every S-polynomials of the current system and in performing every subsequent reductions. Hence, some redundancies (the ones computed till the given level) can be forwarded to the interval solver. The last solution is to partition into sub-systems in order to compute a Gröbner basis of each partial systems (see Figure 6.5.2). Using a "good" partitioning, Gröbner basis generated this way are sufficient to significantly speed-up the interval solver.

Note than when using a linear solver, only linear and linearized (by Gröbner basis computation) constraints are given to the interval solver. The same is true when combining Gröbner basis and another linear solver such as Simplex: the goal is then to linearize equations with Gröbner basis.

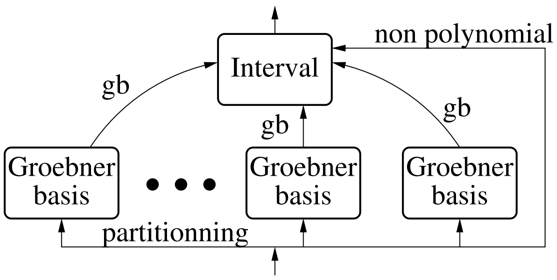


Figure 6.2: Partitioning of constraint systems.

Simplex $\Longrightarrow$ (nonlinear) interval The main goal of this cooperation is to check for the consistency of the set of linear constraints. Moreover, in few architectures the linear solver is used for computing the actual bounds of each variable. For this purpose, the objective function in the Simplex is *minimize* x (respectively *maximize* x) in order to send to the interval solver a new lower bound for x (respectively a new upper bound).

Example 28 *Consider the constraint system*

$$\begin{cases} y & \leqslant & x+1 \\ y & \geqslant & -x+1 \\ x & \leqslant & 2 \end{cases}$$

Optimizing min y *and then* max y *in the Simplex, a reduced domain can be deduced for* y*:* $y \in [-1, 3]$.

Note that to fully use Simplex (i.e., to compute new lower and upper bounds for each n variables), it must be run $2 \times n$ times. Note also that the inverse cooperation is also possible: a (possibly nonlinear) interval solver tightens domains of variables to feed Simplex with more precise bounds (in term of inequations).

Constraint inversion In [Ceberio and Granvilliers, 2000], the symbolic part of the solving process inverses constraints (using cross nested forms of quasi-polynomial expressions) in order to weaken the variable dependency problem by reducing multiplicities of variables: constraints of the form $g(x, y_1, \ldots, y_n) = 0$ are transformed into constraints $x = f(y_1, \ldots, y_n)$ with x a variable different from each y_i. For instance, inverting $x^3 - y = 2$ with respect to x leads to $x = \sqrt[3]{2+y}$ where $\sqrt[3]{\ }$ corresponds to the inverse of the power relation. These new representations of constraints can be easily processed by the interval solver to contract domains of the considered variables.

More complex cooperations

We now describe more complex cooperations involving several solvers or based on more complicated communication protocols. However, we cannot present here all realized cooperations, such as [Colmerauer, 1993], [Jaffar *et al.*, 1992], [Aiba *et al.*, 1988], [Aiba *et al.*, 1995], [Buchberger and Hong, 1991], and also [McPhee *et al.*, 1984].

Linear and interval solver The system of [Beringer and De Backer, 1995] is devoted to linear constraints, where the Simplex-like solver is used to incrementally check for the consistency of the set of linear constraints each time a variable domain is reduced. It may also detect fixed variables (see Figure 6.5.2). As soon as new information is deduced by one of the solver, it is communicated to the other one. The process terminates when a fixed-point is reached, i.e., none of the solver is able to deduce new facts anymore.

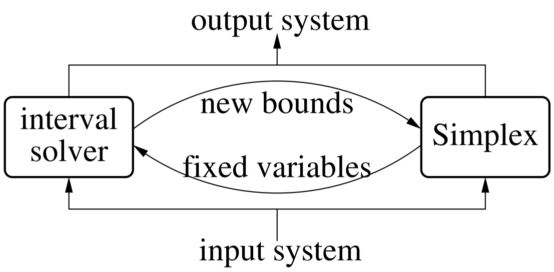


Figure 6.3: Simplex and interval solver fixed-point.

In the cooperation described in [Chiu and Lee, 1995] the input system is distributed to a linear solver (based on Gauss-Seidel method) and a nonlinear interval solver (that implements hull consistency [Lhomme, 1993]). The two solvers exchange new bounds of shared variables until they reach a fixed-point.

Concurrent solvers In [Marti and Rueher, 1995], the authors proposed a cooperative architecture around concurrent (i.e., cooperative concurrent) heterogeneous solvers (see Figure 6.5.2) for handling constraints over the real numbers. Their architecture is based upon agents that communicate via asynchronous message passing. The system of constraints is partitioned and duplicated with respect to abilities of solvers: polynomial equations are given to the Gröbner basis module, the linear sub-part is given to the Simplex implementation, and the whole system of constraints is given to the nonlinear interval solver. As soon as a solver deduces a new fact, it is communicated to a monitor that forwards it to the solvers with respect to their ability to handle it. Gröbner basis is computed once, while a fixed-point of the two other solvers is awaited. A combination of the results of each solver compose the output system of constraints where domains of variables have been contracted as much as possible.

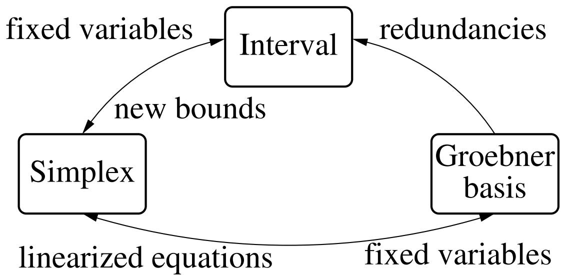


Figure 6.4: Concurrent architecture.

Multiple cooperations CoSAc [Monfroy *et al.*, 1996] is different from the previous cooperations: the linear solver (Gaussian-like elimination and treatment of inequations) is used both as a pre-process simplifying constraints for the nonlinear solver and as a final process propagating newly computed values of variables to eliminate other variables and simplify again the system. The complete solving process is the following: 1) each constraint is pre-processed in the transformation solver (simplifications of expressions, trigonometric simplifications, and transformation of constraints); 2) polynomial nonlinear equations are given to Gröbner basis and linear (in)equations to the linear solver; 3) linear equations deduced in the linear solver are sent to Gröbner basis; 4) univariate constraints (respectively linearized constraints) computed by the Gröbner basis solver are sent to the root extraction solver (respectively linear solver); 5) values of variables are forwarded to the linear solver that can hence verify their validity with respect to to inequations and possibly eliminate new variables.

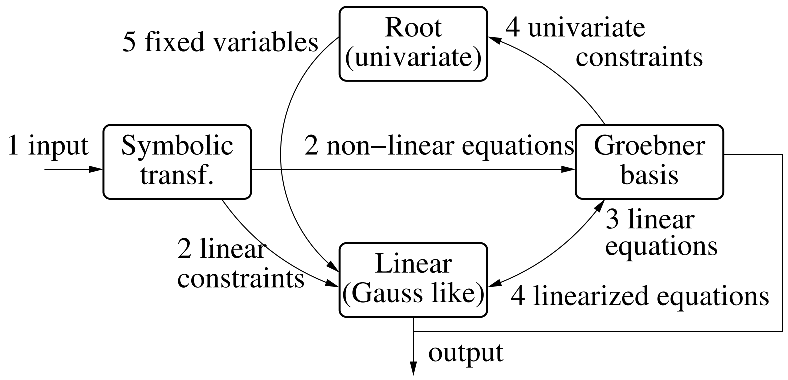


Figure 6.5: Multiple cooperations.

Quantifier elimination and Gröbner basis RISC-CLP(Real) (see [Hong, 1992]) is a constraint logic programming language that deals with non-linear constraints over real numbers. RISC-CLP(Real) mainly tackles two non-linear problems: 1) the decision of solvability of constraints, and 2) the elimination of non-query variables from answers. The solver is a cooperation between Gröbner basis computation (one of the techniques used by RISC-CLP(Real) to simplify the input constraints), and partial CAD (decision of solvability). Partial CAD is also used in the answer processor to remove non-query variables from answer constraints. CAD restricts its input to polynomial constraints with rational coefficients, while some applications (e.g., camera control) involve systems with arbitrary forms where time is the only universally quantified variable. Hence, in the system of [Benhamou and Goualard, 2000], the handling of universally quantified variables is related to the computation of inner-approximation of real solutions. More generally, the solving of first order constraints in the algebra of finite and infinite trees is discussed in [Colmerauer and Dao Thi, 2000].

Redundant linearized equations and combination of interval forms In both `Numerica` [Van Hentenryck *et al.*, 1997c] and `CLIP` [Hickey, 2000], redundancies are generated by linearizing constraints using Taylor developments. Linear combination is then used as a pre-process to simplify the constraints. Finally, different interval forms cooperate in the interval solver: natural forms for wide domains and Taylor forms for tight domains. A more complete discussion on redundant constraints can be found in [Van Emden, 1999].

6.5.3 Languages for solver cooperation

In order to realize solver cooperations at a higher level than ad hoc cooperations, two types of languages can be considered: constraint programming platforms and cooperation languages. One of their advantages over ad hoc cooperation is that they simplify the implementation. Whereas ad hoc cooperation is generally implemented from scratch, languages for solver cooperations provide one with cooperation features (e.g., functions for communication and strategies) and "encapsulation" mechanisms that enable one to re-use solvers from the shelves.

Constraint programming platforms

The first type of solver cooperation languages is composed of constraint programming platforms/languages in which several solvers have been integrated as libraries. Cooperations can be programmed calling these different modules and instructions for controlling the constraints solving process. However, this requires going in depth in the cooperation mechanisms and in the management of the constraint stores.

`Prolog IV` [Benhamou and Touraivane, 1995, PrologIA, 1996] and ECL^{i}PS^e [Wallace *et al.*, 1997] are similar platforms. They are both based on constraint logic programming: a Prolog engine is extended with constraint manipulation primitives such as delay mechanism, and functionalities for managing the constraint store. Prolog IV is provided with a nonlinear interval solver, linear solvers (Simplex-like and Gauss), and a symbolic constraint solver on terms. Among the solvers provided in ECL^{i}PS^e, we can cite: a nonlinear solver [Yakhno and Petrov, 1996] which is based on mechanisms similar to `Unicalc` [Telerman and Ushakov, 1996], an interval solver based on sub-definite computations [Narin'yani, 1983], a linear rational constraint solver, interface to external Simplex solvers, a finite domain solver, and many other solvers provided as libraries.

The `CHR` language ([Frühwirth, 1995] or [Frühwirth, 1998] for a more recent and complete overview on Constraint Handling Rules) is a part of the ECL^{i}PS^e system. `CHR` is a declarative language that allows one to write guarded rules for rewriting constraints. `CHR` provides two types of rules: simplification rules to replace a constraint by a simpler one, and propagation rules to add redundant constraints. The rules have a precedence over the usual resolution step of logic programming, and they are repeatedly applied until a fixed-point is reached. These rules are predominantly used to write constraint solvers (examples of provided libraries are a Gaussian-like elimination, and a Simplex-like solver) and to manage the solving process (see Example 29 that presents a CHR program for the and-gate).

Example 29 *The CHR program for the boolean and-gate is as follows:*

```
and(X,Y,Z) <=> X=0 | Z=0.      and(X,Y,Z) <=> Y=0 | Z=0.
and(X,Y,Z) <=> X=1 | Y=Z.      and(X,Y,Z) <=> Y=1 | X=Z.
and(X,Y,Z) <=> Z=1 | X=1, Y=1.
and(X,Y,Z1), and(X,Y,Z2) ==> Z1=Z2.
```

The last rule is a propagation rule that generates the redundant constraint `Z1=Z2`.

Solvers of the underlying language, i.e., ECL^{i}PS^e, can still be considered in the rules, and thus, the `CHR` language is well-adapted for realizing solver cooperations: for instance, CoSAc (see Section 6.5.2) has been developed with the `CHR` language.

On the other hand, `Ilog solver` [Ilog, 2001] is based around some `C++` libraries implementing solvers and providing the user with functionalities for managing constraints. Among the provided solvers, one can find an interval solver, Simplex, and finite domain solvers. Communications with `Numerica` (which is already an ad hoc cooperation, see Section 6.5.2) can also be established. The `C++` functionalities enable the user programming his own solving process and defining his own propagation mechanisms, and thus, his own solver cooperations.

`Oz` [Smolka, 1995, Mehl *et al.*, 1995] and `Mozart` [Mozart consortium, 1999] (a distributed version of `Oz`) can be seen as implementations of a concurrent constraint programming language [Saraswat, 1993]. The paradigm of concurrent constraint programming (CCP) is based on the concept of communication and control with partial information (constraints). In this formal framework, agents communicate by imposing (*Tell* primitive) and verifying (*Ask* primitive) constraints on shared variables. The concept of agents and shared information is well suited for solver cooperation. The system of [Rueher and Solnon, 1997] has been implemented with `Oz`.

Cooperation languages and models

Recently, there has been a new interest in specific languages and models for solver cooperation. This type of languages has been designed to overcome major issues of solver cooperation. By providing encapsulation mechanisms, they help integrating solvers of the shelves as callable modules. They provide cooperation and control primitives to design (and, as it is the case for BALI, to automatically implement) solver cooperations. They generally offer different cooperation paradigms (such as sequential, parallel, and concurrent solving process) and several control primitives (such as fixed-point, iterations, conditionals) to design complex strategies of cooperation. Note that this is the case for BALI and the system of [Castro and Monfroy, 2000].

BALI [Monfroy, 2000] is a general scheme for integration, and re-usability of heterogeneous solvers, together with some strategies and a language for realizing their collaborations. The following expressions are two examples of cooperation:

$$\begin{aligned} E_1 &= gb\,;\, interval \\ E_2 &= st\,;\, linear\,;\, gb\,;\, f\_p(root\_uni; linear) \end{aligned}$$

E_1 is a simple sequential cooperation: it first applies Gröbner basis on polynomial equations of the input constraint system; the intermediate result is the combination of the computed Gröbner basis and of the constraints that were not treated by the *gb* solver; then, the interval solver is applied on the intermediate solver and provides the final result. Note that the filters are integrated in the encapsulation of solvers. E_2 is a simulation of CoSAc where *st* is the symbolic transformation solver, *gb* Gröbner basis computation, *root_uni* the solver that extracts roots of univariate polynomial, and *linear* the linear solver of CoSAc. The ";" primitive represents sequential solving, and "*f_p*" fixed point solving. Note that as soon as a solver is encapsulated, it can be used in a cooperation expression. The cooperation is automatically implemented and can be used directly to solve a constraint system. We can cite here three systems similar to BALI: 1) the system described in [Hofstedt, 2000] where the underlying host language is a functional logic language, 2) the system of [Borovansky and Castro, 1998] based on rewriting logic, and 3) GMACS [Semenov *et al.*, 2000] which is a multi-component system with a script language for building cooperative solvers.

Extending ideas of BALI, the framework of [Castro and Monfroy, 2000] considers cooperations of solvers as strategies that specify the application of component solvers. The control capabilities for specifying strategies is very low-grain, and the filter mechanisms are part of the system. Thus, solvers and solver cooperations can be designed with the same language. The expression

$$\wedge\_\mathtt{p}(\mathtt{dc}(gb, \varphi_{poly\_eq}), \delta_{part}); int$$

realize the partitioning strategy of the cooperation described in Section 6.5.2. The constraint system is partitioned (partitioner δ_{part}) into sub-systems. Gröbner basis of polynomial equations (filtered with the filter φ_{poly_eq}) occurring in each of the sub-systems are computed (solver *gb* applied by the don't-care primitive dc) in parallel ($\wedge$_p parallel solving primitive). Finally, the interval solver *int* is applied on the union of the resulting Gröbner basis and on the constraints that are not rewritten (e.g., inequalities and non-polynomial constraints).

SoleX [Monfroy and Ringeissen, 1999] is a domain-independent framework to enrich solvers with symbolic computation so that they can handle constraints involving new function symbols, i.e., *alien constraints*. SoleX consists of a set of symbolic rule-based transformations together with a scheduler to control their application and the execution of the solver to be extended. Symbolic transformations add and deduce syntactical as well as semantic information related to alien constraints, complete the computation domain, and purify constraints in order to allow solvers to cope with alien constraints. The solver extensions are either built-in rules, or user-rules derived from some standard features/properties of the domain of computation. These transformations can be seen as elementary solvers, and thus, SoleX is a cooperation of these several solvers with the initial solver. *Abstraction* is one of SoleX rules. Consider a solver S that cannot handle trigonometric functions. Then, the *Abstraction* built-in rule transforms the constraint

$$\sin^2(x+y)+\cos^2(x+y)=1-\sin^3(2x)\times(\sin(x+y)+\cos(x+y))$$

into

$$\begin{cases} X^2+Y^2=1-Z^3\times(X+Y) \\ X=\sin(x+y) \\ Y=\cos(x+y) \\ Z=\sin(2x) \end{cases}$$

The *Inter-Reduction* built-in rule performs replacement of a term by a smaller (with respect to an ordering $\succ$ on terms) one, provided that there is an equation between these two terms. For instance, in the following example, it will replace S by S':

$$S=\begin{cases} y\leqslant \sin 2x+\sin z \\ \sin z=1 \\ \sin 2x=0 \end{cases} \qquad S'=\begin{cases} y\leqslant 0+1 \\ \sin z=1 \\ \sin 2x=0 \end{cases}$$

provided that $\sin z \succ 1$ and $\sin 2x \succ 0$. In a similar way, symbolic transformations can add semantic information. Consider again solver S. Then, a user-rule can add valid facts such as the constraint $\sin^2(X)+\cos^2(X)=1$ for each X appearing in a sine or cosine. After abstraction, this enriches information given as input to the solver A.

6.6 Current and future research directions

Three categories of symbolic-interval techniques for improving the solving of a nonlinear system have been described: the simplification of a constraint expression in order to tackle the dependency problem of interval arithmetic; the combination of constraint expressions for handling the independence of computations; the cooperation of solvers to collectively solve a constraint system.

However, controlling the amount of symbolic computation with respect to numeric computation remains an issue. For instance, Gaussian elimination is universal and can be computed at nearly no cost. Thus, it could be used in every cooperation as a pre-process. On the other hand, Gröbner basis computations are very costly and one must be careful when using them. Although some strategies were proposed in [Benhamou and Granvilliers, 1997] (see Section 6.5.2) it is not possible to decide *a priori* which one will be the most efficient for solving a given constraint system. To overcome these problems, several research directions are currently explored or could be explored.

Symbolic solvers So far, only few symbolic techniques have been tried in symbolic-interval constraint solving: mainly, Gröbner basis computation, factorization, Simplex, Gaussian elimination, substitution, and abstraction. However, numerous methods have been developed in the field of symbolic algebraic computation (such as Wu's method [Chou, 1988]), and we think that symbolic-interval constraint solving could largely benefit from these techniques.

Interval solvers Since they were introduced, the performance of interval solvers has been significantly improved using combination of interval forms [Van Hentenryck *et al.*, 1997c], strategies for the scheduling of variable reductions [Lhomme *et al.*, 1998], several consistencies [Benhamou *et al.*, 1999], or different types of preconditionings [Kearfott and Shi, 1996], parallel computation [Granvilliers and Hains, 2000], and also, distributed computation such as [Monfroy and Réty, 1999]. In most cases, a new algorithm was developed to introduce one new concept, and combining several of these concepts was generally impossible (or difficult to tackle). Recently, generic algorithms (such as [Granvilliers and Monfroy, 2000]) were proposed to allow one combining several improvements in one solver. We think that more elaborated models together with implemented systems could largely help designing and testing more efficient interval solvers.

Symbolic-interval cooperations Cooperations are designed for generic systems of constraints. To our knowledge, one cannot say "there is a best cooperation". Three directions could help adapting cooperations to problems. The first one consists in the identification of classes of constraint systems with the cooperations that work perfectly with them. Then, some cooperations could be tested, and the best one for a given class could be determined. However, classifying problem is a very old and very difficult problem. The second approach consists in focusing on classes of real-life applications (such as scheduling problems, or chemical problems). The structure of constraint systems for specific application is generally fixed. This way, we could think of specific and thus, more efficient and adapted cooperations. The last approach is to develop some framework to dynamically build cooperation. For instance, after some substitutions, one could see that computing Gröbner basis is not worth, and that interval solvers have enough information to be efficient.

A generic symbolic-numeric theoretical framework The theoretical frameworks of Benhamou [Benhamou, 1996] and Apt [Apt, 1999] are very well suited for interval propagation and represent valuable frameworks. However, when dealing with symbolic-numeric cooperation, both of these frameworks require an ordering on the manipulated elements. Unfortunately, defining this ordering is a big issue, if not impossible in numerous cases. This claims for the need of a better adapted theoretical framework where one could add symbolic computation without caring about issues such as termination, correctness, ... Furthermore, this framework should allow mixing symbolic and numeric elementary steps of computation (e.g., computation of a S-polynomial, or application of one narrowing operator), and not be restricted to symbolic pre-process.

Adapted cooperation languages and models Models and languages presented in Section 6.5.3 are very generic. Adapting such models to symbolic-numeric cooperation could provide very good tools for designing and testing new cooperations. Among possible improvements to these models, we can cite:

- integrated and evolving libraries of solvers;
- primitives more adapted to symbolic computation;
- a common language for exchanging data (such as [Abbott *et al.*, 1996] that could solve interoperability issues) and thus, a better communication;
- use of coordination models –such as the IWIM (Ideal Worker, Ideal Manager) control-driven coordination model [Arbab, 1996]– and languages to better decouple computation (i.e., solvers) from control (i.e., cooperations and strategies).

This would lead to fast prototyping of more complex cooperation involving elaborated strategies that could be quickly tested.

Chapter 7

Exhaustive Search for Numerical CSPs

Djamila Sam-Haroud
Boi Faltings
Artificial Intelligence Laboratory
Computer Science Department
Swiss Federal Institute of Technology (EPFL)
Lausanne, Switzerland

Many problems, ranging from resource allocation and scheduling to fault diagnosis and design, involve numerical constraint satisfaction as an essential component. A constraint satisfaction problem (CSP) can have one, several or no solutions. In many applications, it is embedded in a larger decision process where the numerical values of parameters serve as the basis for decisions and impact the nature and the quality of results. In this case, it may be desirable to compute the space of *all* solutions. This has the advantage that no optimization criterion must be formulated beforehand, and that the space of possibilities can be explored systematically. In computing the complete solution space for a CSP, the main problem is how to represent it in a compact manner. This paper introduces some complete search techniques addressing this type of problems.

7.1 Introduction

A wide range of practical problems require solving numerical constraint satisfaction problems (CSP). A numerical CSP is stated as a set of variables taking their values in domains over the reals and subject to constraints. In practice, the constraints can be equalities or inequalities of arbitrary type and arity, usually expressed using arithmetic expressions. The goal is to assign values to the variables so that all the constraints are satisfied. Such an assignment is then called a solution. Most commonly used constraint solvers take as input a numerical CSP, where the domains of the variables are intervals over the reals, and generate a set of interval boxes which conservatively enclose each punctual solution. This fits well the needs inherent to systems with discrete solution sets, typically equality systems. The approach can be less adequate for more general systems, especially when inequalities are involved. Inequalities can be used to state tolerances, like for example $\textit{beam-height} = k \pm \varepsilon$, and it is then admissible to solve them using punctual solvers. However, in their most general form, they rather express spectra of relevant alternatives that need to be identified as precisely and exhaustively as possible. Such inequalities will for example define the possible moving areas of a mobile robot, the collision regions between objects in mechanical assembly, or different alternatives of shapes for the components of a kinematic chain. In all that cases, it is not acceptable to arbitrarily focus on a specific solution, especially when this choice is forced by idiosyncrasies of the used solver.

When solving a numerical CSP is a step in a larger decision process and when its solutions impact the decisions to be taken, the ability to identify all solutions is a crucial advantage. This is recognized in many practical applications [Jaulin, 1994, Jaulin *et al.*, 2000, Jaulin, 2001, Kieffer *et al.*, 2000, Didrit, 1997, Lottaz, 2000a, Lottaz *et al.*, 1999, Lottaz *et al.*, 1998, Benhamou and Goualard, 2000]), where identifying all solutions is shown to improve robustness and efficiency, to avoid useless conflict resolution and to support least commitment decision strategies.

Exhaustive search methods are by essence complete. We therefore start by giving an overview of the principal complete search techniques for numerical CSPs. The issue of computing a compact representation of all solutions for a numerical CSP is then addressed: in section 7.3 for the convex case, and in section 7.4 for the general case.

7.2 Background: Search and Local Consistency

Backtrack search, along with its numerous variants and enhancements, is the principal complete mechanism for solving a CSP. In order to limit the risks of combinatorial explosion, it is usually combined with local consistency techniques which prunes from the domains of variables the values that are locally inconsistent with the constraints, thus reducing the subsequent search effort. One can distinguish different orders of consistency according to the size of subproblems taken into consideration: 1-consistency will check the consistency of each subproblem with one variable,2-consistency each subproblem with two variables, and in general k-consistency each subproblem with k variables. A k-consistency algorithm assigns to each subset of k-1 variables a *label* which contains the values or value combinations satisfying the constraints of a k-dimensional subproblem. In continuous domains, the domains generally take the form of intervals [Davis, 1987, Cleary, 1987, Older and Vellino, 1993, Benhamou *et al.*, 1994, Van Hentenryck *et al.*, 1997a, Hyvönen, 1992], and the labels that of Cartesian products of intervals (interval boxes). The set of labels constructed for a CSP by a consistency algorithms is called its labeling. For discrete domains, the algorithms for computing k-consistent labelings typically have a runtime polynomial with degree k [Freuder, 1978, Cooper, 1989]. In practice the most commonly used local consistency algorithms are therefore those ensuring low degrees of consistency, in reason of their low time and space requirements. Typically the algorithms used in practice enforce 2-consistency (also called arc-consistency [Waltz, 1975b, Mackworth, 1977, Mackworth and Freuder, 1985, Mohr and Henderson, 1986, Deville and van Hentenryck, 1991, Bessière, 1994]) or, at most, 3-consistency (also called path-consistency [Mackworth, 1977, Mackworth and Freuder, 1985, Mohr and Henderson, 1986, Bessière, 1996, Debruyne and Bessière, 1997, Bliek and Sam-Haroud, 1999]).

It is shown for CSPs that local consistency, applied during backtrack search can significantly reduce the thrashing behavior [Waltz, 1975b, Mackworth, 1977, Montanari, 1974]. After each value instantiation local consistency is applied to the future, not yet instantiated variables. For discrete CSPs, existing search algorithms embedding local consistency vary in the degree of filtering: Forward checking (FC) [Haralick and Elliot, 1980] prunes labels of the variables connected by constraints to the currently instantiated variable, directional arc-consistency lookahead (DAC-L) and bi-directional arc-consistency lookahead (BDAC-L) [Tsang, 1998] apply one pass of arc-consistency to all future variables, and maintaining arc-consistency (MAC) [Sabin and Freuder, 1994, Bessière and Régin, 1996] apply arc-consistency among all the variables. Interleaving path-consistency with backtracking can also be profitable in certain cases [Bessière, 1996, Debruyne and Bessière, 1997].

The direct counterparts of these techniques in continuous domains [Benhamou *et al.*, 1994, Van Hentenryck, 1997, Van Hentenryck *et al.*, 1997c, Van Hentenryck *et al.*, 1997a], interleave a branch and bound mechanism with weaker forms of arc-consistency such as box-consistency, kB-consistency and variants [Lhomme, 1993, Lhomme, 1994, Benhamou *et al.*, 1994, Granvilliers, 1998a, Benhamou *et al.*, 1999, Collavizza *et al.*, 1998]. The notion of arc-consistency cannot be computed exactly due to the finite nature of computers. It is replaced by that of interval-consistency. Informally, an interval-consistent label is the smallest union of floating point intervals containing the arc-consistent labels. The local consistency algorithms in continuous domain are often referred to as interval narrowing techniques and generally produce convex interval labels. In most algorithms, splitting the interval labels is deliberately avoided during local consistency enforcement in order to avoid

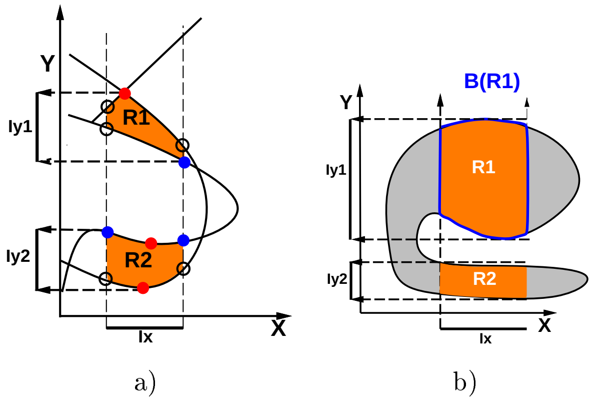


Figure 7.1: *Two examples of total constraints. The constraint on the left consists of the two feasible regions $V1$ and $V2$. When propagating from X to Y, the interval Ix generates the restricted regions $R1$ and $R2$ which project into interval $Iy1$ and $Iy2$. The example on the right shows that multiple restricted regions $R1$ and $R2$ can result from a single consistent region $V1$.*

combinatorial problems.

In a typical branch and bound approach, the constraint region is subdivided into a finite number of sub-regions (sets of intervals) that have to be tested for optimality or solution existence using, for instance, interval analysis. If a test fails on an interval region, it is guaranteed not to contain a solution and can be discarded. Consistency algorithms with a good pruning capability are thus important in order to reduce the number of necessary splittings.

Other approaches using local consistency in continuous domains for solving numeric CSPs include CIAL [Chiu and Lee, 1994], tolerance propagation [Hyvönen, 1992], dynamic backtracking [Jussien and Lhomme, 1998], and backtracking for mixed discrete/continuous constraints [Gelle and Faltings, 2001].

7.2.1 Local consistency using real analysis

Most of the classical consistency techniques in numeric domains rely on interval analysis and/or relational arithmetic. These techniques are described in Chapter 5. In [Faltings, 1994, Faltings and Gelle, 1997, Gelle, 1998] are also proposed interval-consistency techniques based on real analysis. In [Faltings, 1994] it is shown that interval-consistency for binary numeric constraints can indeed be reached by propagating all constraints between the same variables simultaneously. The difficulty is that the conjunction of several constraints, called the *total constraint*, can form complex region structures (see Figure 7.1). One propagation step must compute the projections of subsets of these regions. In

[Faltings, 1994], a simple operator is defined which can carry out this propagation without explicitly generating the structure of the total constraint. It is based on counting the succession of local *extrema* and *intersections* of the constraint curves. Three types of extrema are considered:

a) intersections between constraints and interval bounds

b) constraint intersections

c) local extrema of constraint curves

The propagation of the label of a variable X to that of a variable Y through a restricted region of a constraint between X and Y is then enforced as follows. Let $B(R)$ be the continuous boundary of a restricted region R. Extrema are in this case the local maxima and minima in Y of $B(R)$ with respect to X. Let $max_Y(B(R), y_0)$ be true if and only if $B(R)$ has a local maximum in Y with Y-coordinate y_0, and $min_Y(B(R), y_0)$ if it has a local minimum. If R is bounded by a set of constraint curves, the extrema of its boundary $B(R)$ are a subset of the extrema and intersections of these curves. Only those which satisfy all other constraints bounding R are valid, the others are ignored. The idea is that the set of extrema can thus be found by local considerations only. Let the function $\alpha_Y(R, y)$ be the difference in the number of maxima and the number of minima at y-coordinates greater than or equal to y:

$$\begin{aligned}\alpha_Y(R, y) = |\{y_0 | max_Y(B(R), y_0) \wedge y_0 > y\}| \quad - \\ |\{y_0 | min_Y(B(R), y_0) \wedge y_0 > y\}|\end{aligned}$$

It is then proved that:

- if there is a point $(x^*, y^*) \in R$, $\alpha_Y(R, y^*) > 0$, (Lemma 2.9) and
- if there is no point $(x^*, y^*) \in R$, $\alpha_Y(R, y^*) = 0$ (Lemma 2.7)

The propagation rule should eliminate a value y^* from the label L_Y if and only if it is in no region, i.e. $\alpha_Y(R_i, y^*) = 0$ for all R_i. Since $\alpha_Y(R_i, ...) \geqslant 0$ for any R_i, this can be conveniently expressed as $\sum_i \alpha_Y(R_i, y^*) = 0$. By rewriting this as:

$$\begin{aligned}\sum_i \alpha_Y(R_i, y^*) = \sum_i |\{y_0 | max_Y(B(R_i), y_0) \wedge y_0 > y^*\}| \\ - \sum_i |\{y_0 | min_Y(B(R_i), y_0) \wedge y_0 > y^*\}|\end{aligned}$$

It is only required that the total number of extrema in *all* regions taken together, without any consideration of the regions they belong to. The set of illegal y^*, and conversely the set of legal y, can thus be characterized without knowing which extrema belong to which region!

In [Faltings, 1994], the following propagation rule is shown to compute the maximal intervals of admissible y and thus achieves interval-consistency:

1. $I_y \leftarrow \{\}$
2. compute the set MAX of local maxima and the set MIN of local minima of all restricted regions $r \in R(Q, I_x), Q \in C^t(x, y)$.
3. Filter both MAX and MIN to keep only those extrema which satisfy all other constraints, i.e. actually bound a legal region. Order the two sets of points according to their y-coordinate (these sets are obtained directly by consideration of individual constraints $c(x, y)$).
4. set $index \leftarrow 0$, consider the extrema e in MIN and MAX in decreasing order of their y-coordinate, where elements of MAX are always considered before elements of MIN, and do:
   (a) if $e \in MAX$, $index \leftarrow index + 1$. If $e \in MIN, index \leftarrow index - 1$.
   (b) if $index$ has just changed from 0 to 1, set $y_{lower} \leftarrow y - coordinate(e)$.
   (c) if $index$ has just changed from 1 to 0, set $y_{upper} \leftarrow y - coordinate(e)$ and add the interval $[y_{lower}..y_{upper}]$ to I_y.
5. **return** I_y.

With the exception of intersections with interval bounds, all types of extrema can be preprocessed and the propagation is thus just as efficient as propagating one constraint at a time.

An extension of this propagation rule to ternary constraints is presented in [Faltings and Gelle, 1997, Gelle, 1998]. Total constraints for ternary constraints having two variables in common but differing in the third one are then considered. It has been shown, [Gelle, 1998], that for achieving interval-consistency, the ternary operator proposed in [Faltings and Gelle, 1997, Gelle, 1998] computes the correct projection of a 3-dimensional region on a single variable as long as the feasible regions are topologically simply connected. The idea behind generalizing to ternary constraints is that any arithmetic constraint with an arbitrary number of variables can be decomposed into an equivalent network of ternary constraints. Hence, for constraints with higher arity, either the binary operator can be applied during search when all but two variables of the constraint have been instantiated or the extension to ternary constraints proposed in [Faltings and Gelle, 1997], [Gelle, 1998] can be used.

7.2.2 Local consistency using discretization

The aforementioned consistency techniques build labels of dimension one. The representation of the labels is in that case easy and generally takes the form

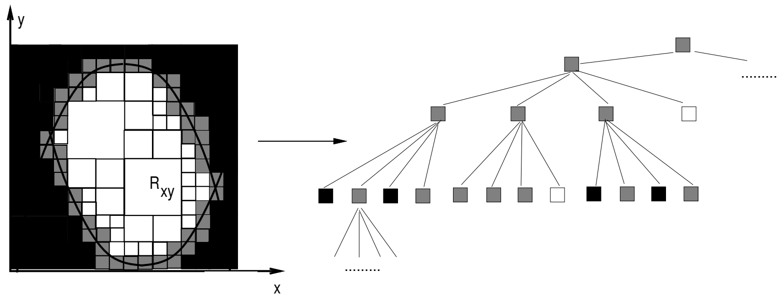


Figure 7.2: Quadtree representation for a binary constraint.

of convex or non-convex intervals. A small amount of work has been done for higher dimensions. This is due to the fact that representing and manipulating k-dimensional labels requires solving k-dimensional systems of equations/inequations of arbitrary types and storing the output in a compact and easy to handle representation.

Discretization using 2^k-trees One way of overcoming these difficulty is to approximate the constraints by a hierarchical decomposition of their solution space into 2^k-trees [Sam-Haroud and Faltings, 1996, Jaulin, 1994, Lottaz, 2000a], a representation commonly used in computer vision [Samet, 1990]. For binary relations, these are called quadtrees, for ternary ones, octrees. Figure 7.2 shows an example of a quadtree representing a binary relation. Such a hierarchical decomposition can be computed as long as each variable domain is restricted to a bounded interval and that for each variable, there is a maximum precision with which results can be used. Both conditions are considered to hold in a wide range of practical engineering applications. Each node in a 2^k-tree represents a k-dimensional sub-region of the original domain (i.e. the domain over which the decomposition is carried out). A node is white if all points within the region are consistent, grey, if some points are consistent and some are inconsistent, and black if all points are inconsistent. Each grey k-dimensional cube is decomposed into 2^k smaller ones whose sides are one half the length of the original cube, and which form the children of the grey node in the hierarchy. When a node is black or white, the recursive division stops. This is also the case for the grey nodes with a smaller size than the maximal precision needed. In [Sam, 1995] strict equality constraint are translated into a weaker form where the final grey nodes are all replaced by white rather than black nodes. This amounts to replacing each equality by two inequalities close to each other. Since the 2^k-tree structure is exponential in k, the authors in [Sam-Haroud and Faltings, 1996, Lottaz, 2000a] limit themselves to quad/ and octrees by ternarizing the original CSP, each k-ary constraint being rewritten as a network of ternary constraints using additional variables.

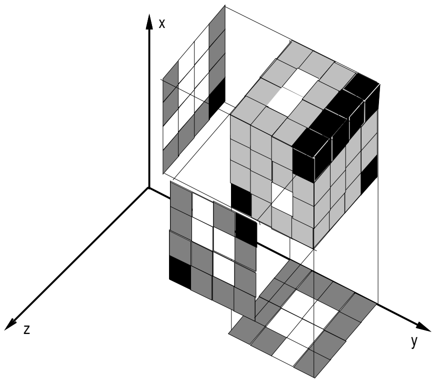


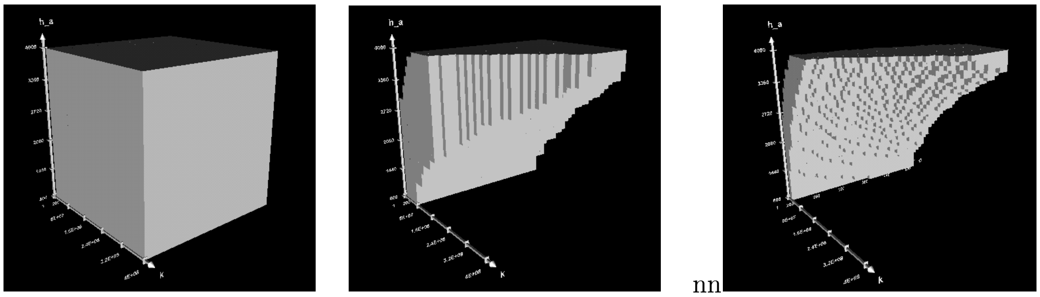
Figure 7.3: *Composing, intersecting and projecting constraints relations is done using logical operations on 2^k-tree nodes.*

K-consistency algorithms using 2^k-trees The 2^k-tree constraint representation allows applying consistency algorithms such as `AC-3` [Davis, 1987] for arc-consistency, `PC-1` [Montanari, 1974] and `PC-2` [Mackworth, 1977]) for path-consistency as well as their generalizations for higher degrees of consistency. Algorithms for computing a k-consistent labeling [Freuder, 1978, Mackworth, 1977, Montanari, 1974] compute labels for each subset of k-1 variables. Each label is computed so that it admits only value combinations of the $k-1$ variables which are consistent with a least one value in the label of a kth variable, for any kth variable. In the standard k-consistency algorithms, the label for the set of variables $x_1, x_2, ..., x_{k-1}$ is computed as the intersection ($\oplus$) of any existing constraint between $x_1, x_2, ..., x_{k-1}$ and all *induced* constraints involving these variables and one additional k-th variable. An induced constraint is given by the *composition* ($\otimes$) of a pair of (k-1)-ary labels, each of which involves all but one of the k-1 variables $x_1, .., x_{k-1}$, and one additional variable x_k. Composition involves extending the (k-1)-ary constraints into the k-dimensional space spanned by $x_1, .., x_{k-1}$ together with x_k, intersecting the resulting volumes, and *projecting* the intersection into the subspace $x_1, .., x_{k-1}$. Figure 7.3 illustrates this process for k=3. Using the 2^k-tree representation, all three operators can be implemented using *logical* operations. Given the ordering *white* $<$ *grey* $<$ *black*, rules for determining the feasibility of a node obtained by one of these operators can be expressed as follows:

i. $color(node_1 \oplus node_2) = Max(color(node_1), color(node_2))$

ii. $color(node_1 \otimes node_2) = Max(color(node_1), color(node_2))$

iii. $color(\prod(node)) = Min(color(node_i))$
where $node_i$ are all nodes having $\prod(node)$ as facet.



nn

a) b) c)

Figure 7.4: *3d-projections onto the same variables of a) the arc-, b) the path-, and c) the 3,2-relational consistent (variant of 5-consistent) space of the same problem.*

This makes it possible to implement, at least in theory, k-consistency algorithm for any k [Sam-Haroud and Faltings, 1996].

The 2^k-tree representation of constraints is well adapted to inequality constraints with small arity and has been used successfully for solving non-linear problems with inequalities involved in robotics and engineering design [Jaulin, 1994, Lottaz, 2000a, Jaulin *et al.*, 2000, Jaulin, 2001, Kieffer *et al.*, 2000, Didrit, 1997, Lottaz *et al.*, 1999, Lottaz *et al.*, 1998]. Figure 7.4 shows different constraint approximations build by Space-Solver [Lottaz, 2000b], the 2^k-tree based solver used for the engineering design applications. However, for the general case, 2^k-trees are prohibitively space consuming for high arities and precisions.

7.3 Convex CSPs and Backtrack-free search

From a general perspective, local consistency techniques can be seen as means of computing *approximate* descriptions of the complete solution space for a CSP. These approximations are complete in the sense that they contain all solutions. The labels computed by local consistency, when low dimensional, provide a *compact* representation of such complete approximations. When the labeling is *globally consistent*, each label contains only values or value combinations which occur in at least one solution. A globally consistent, also called decomposable, labeling is *sound* in the sense that the labeling never admits any value which does not lead to a solution. It is *complete* in the sense that all solutions are represented in it.

Using a globally consistent labeling, any partial consistent instantiation of a subset of variables can be extended to a *solution* with no backtracking [Dechter, 1992], a process which can generally be carried out in linear time. Minimality as defined in [Montanari, 1974] for binary networks, guarantees that each pair of values allowed by the constraints participates in at least one solution. A decomposable network being always minimal while the opposite is not true, so minimality is a weaker property. Extracting a particular solution from a consistent labeling is an iteration of two steps in which values are assigned to variables sequentially. In the first step, an unassigned variable is selected and assigned a value within its label. In the second step, the labels of all remaining unassigned variables are updated so that they contain only values which are consistent with those already assigned. If the initial labeling is globally consistent and non-empty, every partial assignment of variables can be extended to a full solution. Consequently, the assignment procedure will never require any backtracking.

In general, a globally consistent labeling may require explicitly representing constraints for all variables in the problem, a task which has exponential time complexity in the worst case. However, under certain conditions local consistency algorithms are sufficient to compute a globally consistent solution space or at least to bound the remaining search effort. For example, it has been shown, [Freuder, 1982], for discrete constraints that when the constraint network is a tree, arc-consistency also guarantees global consistency. This is an example of how the topology of the constraint *network* can simplify the computation of a globally consistent labeling [Freuder, 1978]. Similarly, path-consistency is sufficient to ensure backtrack-free search in networks whose topology is *series-parallel*. For discrete and temporal constraints[1] it has also been shown, using Helly's theorem in one dimension, that when all constraints satisfy a *convexity property* and are binary, path-consistency ensures global consistency irrespective of the topology of the network ([Dechter *et al.*, 1990],[Van Beek and Dechter, 1997],[Deville *et al.*, 1997]).
This result has been extended to numeric binary CSPs in [Haroud and Faltings, 1994]:

Theorem 1 *A binary constraint network which is convex and path-consistent is minimal and decomposable.*

The convexity property is conservative with respect to composition, intersection and projection of constraints. Hence, if a continuous CSP is convex its path-consistent counterpart remains so. The following a priori result is therefore stated for convex continuous CSP:

Corollary 1 *Path-consistency computes the minimal and decomposable network representation of any convex binary numerical CSP.*

[1] In temporal CSPs, the variables represent time points and the constraint bounded differences between time points.

```
Procedure 3-2-rel-con(V,C,D)
   repeat
   changed ← false
   for each pair (u,v), u,v ∈ V do
      for each ternary tuple (i,j,k), i,j,k ∈ V do
         begin
         c'(i,j,k)
         ← c(i,j,k) ⊕ ∏_(i,j,k) c(i,u,v) ⊗ c(j,u,v) ⊗ c(k,u,v)
         if c'(i,j,k) ≠ c(i,j,k) then
            begin
            c(i,j,k) ← c'(i,j,k)
            changed ← true
            end
         end
   until changed = false
```

Figure 7.5: *Algorithm for computing a (3-2)-relationally consistent labeling.*

The generalization to constraints of arity r [Sam-Haroud and Faltings, 1996] relies on the notion of $(r, r-1)$-relational consistency which guarantees that each set of r relations having $r-1$ variables in common has a non-null intersection.

Definition 13 *Let $\mathcal{P}$ be a network of relations over a set of variables X, of arity r. Let $R_{I_1}(x_1, y_1, ..., y_{r-1}), \dots R_{I_r}(x_r, y_1, ..., y_{r-1})$ be r relations of N sharing the r-1 variables $\{y_1, \dots y_{r-1}\}$. The relations are (r,r-1)-relationally consistent relative to the shared variables if and only if any consistent instantiation of the variables in $\{x_1, ..., x_r\}$ has an extension to $\{y_1, \dots y_{r-1}\}$ that satisfies all relations simultaneously. The network $\mathcal{P}$ is relationally (r,r-1) consistent if and only if all relations are (r,r-1)-consistent with respect to all subsets of shared variables.*

The result is proved using Helly's theorem in $r-1$ dimensions and states that :

Theorem 2 *Let $\mathcal{P}$ be a convex constraint network of arity r at most. (r,r-1)-relationally consistency computes the minimal and decomposable network representation for $\mathcal{P}$.*

A $(r, r-1)$-relational consistency algorithm computes labels of dimension r at most. It is polynomial with degree $2r-1$. For this reason, the result has in practice been applied only on ternarization of numeric CSPs [Sam-Haroud and Faltings, 1996, Lottaz, 2000a, Lottaz *et al.*, 1999] using the quadtree/octree representation of constraints. In that case, the notion of (3,2)-relational consistency is applied.

Theorem 3 ([Sam-Haroud and Faltings, 1996]) *For any convex ternary network $\mathcal{P}$, (3,2)-relational-consistency will either:*

- *decide that the network is inconsistent by generating an empty label, or*
- *compute an equivalent globally consistent labeling of $\mathcal{P}$*

in time $O(n^5)$ where n is the number of variables of $\mathcal{P}$.

(3,2)-relational consistency can be computed by the algorithm shown in Figure 7.5. It takes as input a continuous CSP, $\mathcal{P} = (V, C, D)$, where V is the set of variables, C the set of constraints, and D is the set of variable domains. c denotes the label of a relation in C, and the algorithm terminates with a (3-2)-relational consistent set of labels. These labels are of dimension 3 at most.

Hence, for convex problems the 3-dimensional labels computed by (3,2)-relational provide a compact, sound and complete representation of the solution space.

Partial convexity conditions The aforementioned results have been extended to a less restrictive form of convexity, called $(x_1, \ldots x_k)$-convexity:

Definition 14 $(x_1, \ldots x_k)$-Convexity
Let R be an n-ary relation between n variables $x_1 \ldots x_n$. R is said to be $(x_1, \ldots x_k)$-convex in the domains $D_1 \times \ldots D_{x_k}$ if, for any two points q_1 and q_2 in the region r defined by R, such that the segment $\overline{q_1 q_2}$ is on a plane parallel to $x_1 \cdots \times x_k$, $\overline{q_1 q_2}$ is entirely contained in r

Informally, this means that a relation is $(x_1, \ldots x_k)$-convex if any subprojection over the subset $(x_1, \ldots x_k)$ yields a convex k-ary region.

The following results are then stated:

Theorem 4 ([Sam-Haroud and Faltings, 1996]) *Let N be a path-consistent binary constraint network. If there exists an ordering of the variables $x_1, \ldots x_n$ such that each relation of N R_{x_i, x_j}, $1 \leqslant j \leqslant i$, is (x_i)-convex, then a consistent instantiation can be found without backtracking.*

Theorem 5 ([Sam-Haroud and Faltings, 1996]) *Let $\mathcal{N}$ be a (3,2)-relationally consistent ternary constraint network. If there exists an ordering of the variables $x_1, \ldots x_n$ such that for any $i, j, k : 1 \leqslant i < j \leqslant k \leqslant n$, $R(x_i, x_j, x_k)$ is (x_j, x_k)-convex, then the network is globally consistent and a consistent instantiation can be found without backtracking.*

However, even if the initial problem is $(x_1 \ldots x_k)$-convex, consistency algorithms may not preserve this property since intersecting two non-convex regions may results in an arbitrary number of distinct sub-regions. For these reasons, the result of Theorems 4, 5 is valid only on the a posteriori networks computed by consistency algorithms.

Ternarization Although ternarizing numeric CSPs has the potential to accelerate consistency algorithms considerably, the gain in performance strongly depends on the number of variables in the transformed CSP. The work presented in [Lottaz, 1999] addresses this issue and proposes heuristics for optimizing the number of auxiliary variables and for removing unnecessary intermediary variables from the original CSP.

7.4 Search for general CSPs

For general CSPs with non discrete solution sets, one of the natural alternatives to the punctual solving approach is to try to cover the spectrum of solutions using a reduced number of subsets from $\mathbb{R}^n$. Usually, these subsets are chosen with known and simple properties (interval boxes, polytopes, ellipsoid,..) [Jaulin, 1994]. In recent years, several authors have proposed set covering algorithms with *intervals boxes* [Jaulin, 1994, Garloff and Graf, 1999, Sam-Haroud and Faltings, 1996, Lottaz, 2000a, Benhamou and Goualard, 2000, Silaghi *et al.*, 2001, Ratschan, 2001]. In [Jaulin, 1994], a recursive dichotomous split is performed on the variable domains. Each box obtained by splitting is tested for inclusion using interval arithmetic tools. The boxes obtained are hierarchically structured as 2^k-trees. The underlying algorithms are exponential in the number of variables but can be made practical in some complex applications from robotics and robust control by the use of domain-dependent heuristics and by a hierarchical handling of precision. The heuristics notably allow for restricting the splittings to particular regions of the search space [Jaulin, 2001]. In [Sam-Haroud and Faltings, 1996], the results on convex CSPs (see section 7.3) are coupled with the 2^k-tree representation of constraints and provide polynomial time set covering algorithms with bounded space requirements for a restricted class of CSPs. In [Sam, 1995], are proposed techniques for approximating non-convex CSPs as unions of convex ones. Each convex sub-problem is then solved during search using local consistency algorithms. In depth empirical evaluation is still required to state the practicality of the approach. In [Lottaz, 2000a, Lottaz, 2000b], 2^k-tree coverings are also used. The covering is built by local consistency algorithms and is seen as a complete but unsound approximation which must be further explored using interactive search algorithms. The technique proposed in [Garloff and Graf, 1999] constructs a interval boxes covering algebraically using Bernstein polynomials, which give formal guarantees on the result of the inclusion test. The approach is restricted to polynomial constraints and is exponential in the number of variables. In [Ratschan, 2001], are proposed heuristics for improving box-covering strategies.

Interval-based search techniques for CSPs with equalities and inequalities are essentially dichotomous. Variables are instantiated using intervals. When the search reaches an interval that contains no solutions it backtracks, otherwise the interval is recursively split in two halves up to an established resolution. In practice this process is enhanced by applying interval narrowing operator to the overall constraint system, after each split. In most dichotomous search algorithms, equalities and inequalities are treated the same way. Splitting is performed until canonical intervals are reached and as long as the error inherent to the interval narrowing operator is smaller than the interval to split. This policy, works generally well for equality systems but leaves place for improvement when inequalities are involved. Instead, the technique proposed in [Benhamou and Goualard, 2000] uses a feasibility (soundness) test for boxes, which allows better splitting decisions. Given a constraint and a box, the feasibility test checks whether all the points in the box satisfy the constraint. Given a constraint c and a box $\mathbf{B}$, it consists of proving that $\{r \in \mathbf{B} \mid r \in \neg c\} = \varnothing$. The proof is done by construction using complete search on $\neg c$. In [Benhamou and Goualard, 2000], this feasibility test is used to construct an inner approximation of universally quantified inequality constraints. An extension of this approach to general mixed equality/inequality constraints is proposed in [Silaghi *et al.*, 2001] which combines the classical dichotomous search for equalities with a set covering strategy similar to that of [Benhamou and Goualard, 2000], for inequalities. The proposed interval box covering algorithm improves the compactness of the results and the efficiency of search in some cases.

7.5 Conclusion

Search methods using interval-based techniques have been used extensively in non-linear optimization and in solving equation systems resulting in single solutions. Most of the algorithms are mainly thought for constraint systems the solution set of which is discrete, i.e. in which the numeric solutions are isolated and for finding global optima in a constrained problem. For constraint systems with non-discrete solution sets, the problem is not only to compute the solution set efficiently but also to represent it as compactly as possible. Alternatives to the punctual solving approach, such as set covering strategies, can be proposed. Although such approaches can be proved tractable in the convex case, much work remains to be done in order to make the techniques practical and to extend them to the non-convex case.

Bibliography

The number(s) at the end of each reference give the page number(s) where the reference is cited.

[Aarts and Lenstra, 1997] E.H.L. Aarts and J.K. Lenstra, editors. *Local Search in Combinatorial Optimization.* Wiley, Chichester, 1997. [70]

[Abbott *et al.*, 1996] J. Abbott, A. Van Leeuwen, and A. Strotmann. Objectives of OpenMath. Technical Report 12, RIACA (Research Institute for Applications of Computer Algebra), June 1996. [175]

[Adjiman *et al.*, 1996] C.S. Adjiman, I.P. Androulakis, C.D. Maranas, and C.A. Floudas. A global optimization method αbb for process design. *Comput. Chem. Engin.*, 20:419–424, 1996. [69, 85, 110]

[Adjiman *et al.*, 1998a] C.S. Adjiman, I.P. Androulakis, and C.A. Floudas. A global optimization method, αbb, for general twice-differentiable constrained nlps – ii. implementation and computational results. *Comput. Chem. Engin.*, 22:1159–1179, 1998. [69, 110]

[Adjiman *et al.*, 1998b] C.S. Adjiman, S. Dallwig, C.A. Floudas, and A. Neumaier. A global optimization method, αbb, for general twice-differentiable constrained nlps – i. theoretical advances. *Comput. Chem. Engin.*, 22:1137–1158, 1998. [69, 85, 110]

[Aiba *et al.*, 1988] A. Aiba, K. Sakai, Y. Sato, D. J. Hawley, and R. Hasegawa. Constraint Logic Programming Language CAL. In *Proceedings of the International Conference on Fifth Generation Computer Systems, Tokyo (Japan)*, pages 263–276. ICOT Research Center, December 1988. [167]

[Aiba *et al.*, 1995] A. Aiba, K. Yokota, and H. Tsuda. Heterogeneous Distributed Cooperative Problem Solving System Helios and its Cooperation Mechanisms. *International Journal of Cooperative Information Systems*, 4(4):369–385, 1995. [167]

[Akrotirianakis and Rustem, 1997] I. Akrotirianakis and B. Rustem. A globally convergent interior-point algorithm for general nonlinear programming problems. Technical Report 97/14, Department of Computing, Imperial College, London., 1997. [41]

[Al-Baali, 1985] M Al-Baali. Descent property and global convergence of the Fletcher-Reeves method with inexact line searches. *J.I.M.A. Num. Anal.*, 5:121–124, 1985. [15]

[Al-Khayyal and Falk, 1983] F.A. Al-Khayyal and J.E. Falk. Jointly constrained biconvex programming. *Math. Operations Res.*, 8:273–286, 1983. [110]

[Alefeld and Herzberger, 1983] Götz Alefeld and Jürgen Herzberger. *Introduction to Interval Computations.* Academic Press, 1983. [118, 158]

[Anderson and Anderson, 1995] E.D. Anderson and K.D. Anderson. Presolving in linear programming. *Math. Program.*, 71:221–245, 1995. [85]

[Androulakis *et al.*, 1995] I.P. Androulakis, C.D. Maranas, and C.A. Floudas. αbb: a global optimization method for general constrained nonconvex problems. *J. Global Optim.*, 7:337–363, 1995. [69, 85, 110]

[Apt, 1999] Krzysztof R. Apt. The Essence of Constraint Propagation. *Theoretical Computer Science*, 221(1-2):179–210, 1999. [115, 130, 145, 163, 174]

[Arbab, 1996] F. Arbab. The IWIM model for coordination of concurrent activities. In Paolo Ciancarini and Chris Hankin, editors, *Coordination Languages and Models*, volume 1061 of *Lecture Notes in Computer Science*, pages 34–56. Springer-Verlag, April 1996. [175]

[Averick and Moré, 1994] B. M. Averick and J.J. Moré. Evaluation of large-scale optimization problems on vector and parallel architectures. *SIOPT*, 4:708–721, 1994. Software in info.mcs.anl.gov: /pub/MINPACK-2. [18]

[Babichev *et al.*, 1993] A.B. Babichev, O.B. Kadyrova, T.P. Kashevarova, A.S. Leshchenko, and A.L. Semenov. UniCalc, a novel approach to solving systems of algebraic equations. *Interval Computations*, 3:29–47, 1993. [85]

[Barr *et al.*, 1995] R.S. Barr, B.L. Golden, J.P. Kelly, M.G.C. Resende, and W.R. Stewart. Designing and reporting on computational experiments with heuristic methods. *J. Heuristics*, 1:9–32, 1995. [112]

[Barth and Bockmayr, 1995] Peter Barth and Alexander Bockmayr. Finite domain and cutting plane techniques in CLP($\mathcal{PB}$). In *Procs. of International Conference of Logic Programming*, Tokyo, 1995. [158]

[Bartholomew-Biggs and Hernandez, 1994] M. C. Bartholomew-Biggs and F.G. Hernandez. Modifications to the subroutine OPALQP for dealing with large problems. *J. Econ. Dyn. Control*, 18:185–203, 1994. [28, 29]

[Bartholomew-Biggs and Hernandez, 1995] M. C. Bartholomew-Biggs and F.G. Hernandez. Using the KKT-matrix in the augmented lagrangian SQP method for sparse constrained optimization. *J.O.T.A.*, 85:201–220, 1995. [29, 32]

[Baur and Strassen, 1983] W. Baur and V. Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22:317–330, 1983. [49]

[Beale and Forrest, 1976] E.M.L. Beale and J.J.H. Forrest. Global optimization using special ordered sets. *Math. Program.*, 10:52–69, 1976. [103, 106]

[Beale and Tomlin, 1970] E.M.L. Beale and J.A. Tomlin. Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. In J. Lawrence, editor, *OR 69: Proc. Fifth Int. Conf. Oper. Res.*, pages 447–454. Tavistock Publications, London, 1970. [103]

[Beale, 1988] E.M.L. Beale. *Introduction to Optimization.* Wiley, Chichester, 1988. [103, 107]

[Beaumont, 1998] Olivier Beaumont. Solving Interval Linear Systems with Linear Programming Techniques. *Linear Algebra and Its Applications*, 281(1-3):293–309, 1998. [158]

[Ben-Ayed, 1993] O. Ben-Ayed. Bilevel linear programming. *Comput. Oper. Res.*, 20:485–501, 1993. [102]

[Benhamou and Goualard, 2000] F. Benhamou and F. Goualard. Universally quantified interval constraints. In R. Dechter, editor, *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming*, volume 1894 of *Lecture Notes in Computer Science*, pages 67–82, Singapore, 2000. Springer-Verlag. [169, 177, 188, 189]

[Benhamou and Granvilliers, 1997] Frédéric Benhamou and Laurent Granvilliers. Automatic Generation of Numerical Redundancies for Non-Linear Constraint Solving. *Reliable Computing*, 3(3):335–344, 1997. [130, 146, 159, 166, 173]

[Benhamou and Older, 1997] F. Benhamou and W.J. Older. Applying Interval Arithmetic to real, integer, and boolean constraints. *Journal of Logic Programming*, 32(1):1–24, 1997. Extension of a Bell Northern Research research report published in 1992. [85, 115, 127, 131]

[Benhamou and Touraivane, 1995] F. Benhamou and Touraivane. Prolog IV: Langage et algorithmes. In *IV mes Journées Francophones de Programmation en Logique, JFPL'95*, pages 51–65, Dijon (France), May 1995. In french. [170]

[Benhamou *et al.*, 1994] Frédéric Benhamou, David McAllester, and Pascal Van Hentenryck. CLP(Intervals) Revisited. In Maurice Bruynooghe, editor, *Proceedings of ILPS'94, International Logic Programming Symposium*, pages 124–138, Ithaca, NY, USA, 1994. MIT Press. [85, 115, 131, 132, 137, 138, 154, 178]

[Benhamou *et al.*, 1999] Frédéric Benhamou, Frédéric Goualard, Laurent Granvilliers, and Jean-Franccois Puget. Revising Hull and Box Consistency. In Danny De Schreye, editor, *Proceedings of ICLP'99, International Conference on Logic Programming*, pages 230–244, Las Cruces, NM, USA, 1999. MIT Press. [116, 132, 133, 138, 142, 146, 174, 178]

[Benhamou, 1996] Frédéric Benhamou. Heterogeneous Constraint Solving. In Michael Hanus and Mario Rodrguez-Artalejo, editors, *Proceedings of ALP'96, 5th International Conference on Algebraic and Logic Programming*, volume 1139 of *Lecture Notes in Computer Science*, pages 62–76, Aachen, Germany, 1996. Springer-Verlag. [115, 128, 145, 163, 164, 174]

[Beringer and De Backer, 1995] H. Beringer and B. De Backer. Combinatorial problem solving in constraint logic programming with cooperative solvers. In *Logic programming: formal methods and practical applications*. Elsevier Science Publisher B.V., 1995. [167]

[Berner, 1996] S. Berner. Parallel methods for verified global optimization: practice and theory. *J. Global Optim.*, 9:1–22, 1996. [107]

[Berz and Hoffsttter, 1998] Martin Berz and Georg Hoffsttter. Computation and Application of Taylor Polynomials with Interval Remainder Bounds. *Reliable Computing*, 4:83–97, 1998. [124]

[Bessière and Régin, 1996] C. Bessière and J-C. Régin. Mac and combined heuritics: two reasons to forsake jc (and cbj). In E.C. Freuder and M. Jampel, editors, *Lecture notes in computer Science 1118: Principles and Practice of constraint programming*. Springer Verlag, 1996. [178]

[Bessière, 1994] C. Bessière. Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65(1), 1994. [178]

[Bessière, 1996] C. Bessière. A simple way to improve path consistency processing in interval algebra networks. In *AAAI-96*, pages 375–380, Portland, Oregon, 1996. [178]

[Betts and Frank, 1994] J.T. Betts and P.D. Frank. A sparse nonlinear optimization algorithm. *J.O.T.A.*, 82:519–541, 1994. [28]

[Biegler *et al.*, 1995] L.T Biegler, J. Nocedal, and C. Schmid. A reduced Hessian method for large-scale constrained optimization. *SIOPT*, 5:314–347, 1995. [28, 30]

[Biegler *et al.*, 2000] L.T Biegler, J. Nocedal, C. Schmid, and D. Ternet. Numerical experience with a reduced Hessian method for large scale constrained optimization. *J. Comput. Optim. Appl*, 15:45–67, 2000. [30]

[Billups and Murty, 2000] A.C. Billups and K.G. Murty. Complementarity problems. *J. Comput. Appl. Math.*, 124:303–318, 2000. [102]

[Bischof and Haghighat, 1996] C. Bischof and M. Haghighat. Hierarchical approaches to automatic differentiation. In M. Berz, C. Bischof, G. Corliss, and A. Griewank, editors, *Computational Differentiation: Techniques, Applications, and Tools*, pages 83–94. SIAM, Philadelphia, Pennsylvania, 1996. [51]

[Bixby *et al.*, 2000] R.E. Bixby, M. Fenelon, Z. Gu, E. Rothberg, and R. Wunderling. Mip: theory and practice – closing the gap. In M.J.D. Powell and S. Scholtes, editors, *System Modelling and Optimization: Methods, Theory and Applications*, pages 19–49. Kluwer, Dordrecht, 2000. [59, 66]

[Bjoerck, 1997] A. Bjoerck. Numerical stability of methods for solving augmented systems. *American Mathematical Society. Contemp. Math.*, 204:51–60, 1997. [29]

[Bliek and Sam-Haroud, 1999] C. Bliek and D. Sam-Haroud. Path consistency for triangulated constraint graphs. In *Proceedings of IJCAI'99*, 1999. [178]

[Bliek, 1992] C. Bliek. *Computer Methods for Design Automation*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, July 1992. [50, 51, 52, 124, 133]

[Bliek, 1997] Christian Bliek. Fast Evaluation of Partial Derivatives and Interval Slopes. *Reliable Computing*, 3(3):259–268, 1997. [52, 124]

[Boender *et al.*, 1982] C.G.E. Boender, A.H.G. Rinnooy Kan, L. Strougie, and G.T. Timmer. A stochastic method for global optimization. *Math. Program.*, 22:125–140, 1982. [70]

[Boggs *et al.*, 1994] P. T. Boggs, A. J. Kearsley, and J. W. Tolle. A truncated SQP algorithm for large-scale nonlinear programming problems. In S. Gomez and J.P. Hennart, editors, *Advances in Optimization and Numerical Analysis*, pages 69–77. Kluwer Academic Publishers, 1994. [33]

[Boggs *et al.*, 1996] P.T. Boggs, P. T. Domich, and J.E. Rogers. An interior-point method for large-scale quadratic programming problems. *Ann. Oper. Res*, 62:419–437, 1996. [33]

[Boggs *et al.*, 1999] P. T. Boggs, A. J. Kearsley, and J. W. Tolle. A practical algorithm for general large-scale nonlinear optimization problems. *SIOPT*, 9:755–778, 1999. [33, 35]

[Bomze *et al.*, 1996] I.M. Bomze, T. Csendes, R. Horst, and P.M. Pardalos, editors. *Developments in Global Optimization*. Kluwer, Dordrecht, 1996. [61]

[Borovansky and Castro, 1998] P. Borovansky and C. Castro. Cooperation of constraint solvers: Using the new process control facilities of elan. *Electronic Notes in Theoretical Computer Science*, 15:379–398, 1998. in Proceedings of The Second International Workshop on Rewriting Logic and its Applications, RWLW'98, Pont-à-Mousson, France. [172]

[Branch *et al.*, 1999] M. A. Branch, Th. F. Coleman, and Y. Li. A subspace, interior and conjugate gradient method for large-scale bound-constrained minimization problems. *SIAM. J. Sci. Comp.*, 21:1–23, 1999. [19]

[Breitfeld and Shanno, 1994] M. G. Breitfeld and D. Shanno. Preliminary computational experience with modified log-barrier functions for large-scale nonlinear programming. In W. W. Hager, editor, *Large scale optimization. State of the art*, pages 45–67. Kluwer Academic Publishers, 1994. [38]

[Buchberger and Hong, 1991] B. Buchberger and H. Hong. Speeding-up Quantifier Elimination by Gröbner Bases. Technical report 91-06.0, RISC (Research Institute for Symbolic Computation), Linz (Austria), February 1991. [167]

[Buchberger, 1965] Bruno Buchberger. *An Algorithm for Finding a Basis for the Residue Class Ring of a Zero-Dimensional Polynomial Ideal*. PhD thesis, University of Innsbruck, 1965. (in German). [159]

[Buchberger, 1985] Bruno Buchberger. Gröbner Bases: an Algorithmic Method in Polynomial Ideal Theory. In *Multidimensional Systems Theory*, pages 184–232. D. Reidel Publishing Company, 1985. [152, 159]

[Buckley and Lenir, 1983] A. Buckley and L. Lenir. Qn-like variable storage conjugate gradients. *Math. Prog.*, 27:155–175, 1983. [13]

[Burke, 1991] J.V. Burke. An exact penalization viewpoint of constrained optimization. *SIAM J. Control and Optimization*, 29:968–998, 1991. [11, 94]

[Burke, 1992] J.V. Burke. A robust trust-region method for constrained nonlinear programming problems. *SIOPT*, 2:325–347, 1992. [33]

[Byrd and Nocedal, 1989] R.H. Byrd and J. Nocedal. A tool for the analysis of quasi-newton methods with applications to unconstrained minimization. *J. SIAM Numer. Anal.*, 26:727–739, 1989. [16]

[Byrd *et al.*, 1994] R.H. Byrd, J. Nocedal, and R.B. Schnable. Representations of quasi-newton matrices and their use in limited memory methods. *Math. Prog.*, 63:129–156, 1994. [13, 16]

[Byrd *et al.*, 1995] R.H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comp.*, 16:1190–1208, 1995. [17, 18, 23]

[Byrd *et al.*, 1996] R.H. Byrd, J.Ch. Gilbert, and J. Nocedal. A trust-region method based on interior-point techniques for nonlinear programming. Technical Report 96/05, Optimization Technology Center, Northwestern University, Evanston, 1996. [39]

[Byrd *et al.*, 1999] R.H. Byrd, M.E. Hribar, and J. Nocedal. An interior-point algorithm for large-scale nonlinear programming. *SIOPT*, 9:877–900, 1999. [40]

[Caprani and Madsen, 1980] Ole Caprani and Kaj Madsen. Mean value forms in interval analysis. *Computing*, 25:147–154, 1980. [124]

[Castro and Monfroy, 2000] C. Castro and E. Monfroy. Basic Operators for Solving Constraints via Collaboration of Solvers. In *Proceedings of the 5th International Conference on Artificial Intelligence and Symbolic Computation*, volume 1930 of *Lecture Notes in Artificial Intelligence*, Madrid, Spain, 2000. Springer. To appear. [171, 172]

[Ceberio and Granvilliers, 2000] Martine Ceberio and Laurent Granvilliers. Solving Nonlinear Systems by Constraint Inversion and Interval Arithmetic. In John A. Campbell and Eugenio Roanes-Lozano, editors, *Proceedings of AISC'2000, 5th International Conference on Artificial Intelligence and Symbolic Computation*, volume 1930 of *Lecture Notes in Artificial Intelligence*, pages 127–141, Madrid, Spain, 2000. Springer-Verlag. [123, 167]

[Chen and van Emden, 1995] H.M. Chen and M.H. van Emden. Adding interval constraints to the Moore–Skelboe global optimization algorithm. In V. Kreinovich, editor, *Reliable Computing (Supplement), Extended Abstracts of APIC'95, International Workshop on Applications of Interval Computations,*, pages 54–57, 1995. [85]

[Chiu and Lee, 1994] C.K. Chiu and T.W. Lee. Towards practical interval constraint solving in logic programming. In *Proceedings of the 11th International Logic Programming Symposium*, 1994. [179]

[Chiu and Lee, 1995] C. K. Chiu and J. H. M. Lee. Interval Linear Constraint Solving using the Preconditionned Interval Gauss-Seidel Method. In *Proceedings of the 12-th International Conference on Logic Programming*, pages 17–32, Kanagawa, Japan, 1995. MIT Press. [168]

[Chou, 1988] S. C. Chou. *Mechanical Geometry Theorem Proving*. D. Reidel Publishing Company, 1988. [174]

[Cleary, 1987] J.G. Cleary. Logical arithmetic. *Future Computing Systems*, 2(2):125–149, 1987. [85, 115, 127, 131, 132, 134, 156, 178]

[coconut, 2001] COCONUT, continuous constraints – updating the technology. http://solon.cma.univie.ac.at/~neum/glopt/coconut.html, 2001. [56]

[Coleman and Verma, 1998] T.F. Coleman and A. Verma. The efficient computation of sparse jacobian matrices using automatic differentiation. *SIAM Journal of Scientific Computation*, 19(4):1210–1233, July 1998. [53, 54]

[Collavizza *et al.*, 1998] H. Collavizza, F. Delobel, and M. Rueher. A note on partial consistencies over continuous domains. In *Proceedings CP'98*, 1998. [178]

[Collavizza *et al.*, 1999] Hélène Collavizza, Franois Delobel, and Michel Rueher. Comparing Partial Consistencies. *Reliable Computing*, 5(3):213–228, 1999. [115]

[Collins and Hong, 1991] G. E. Collins and Hoon Hong. Partial Cylindrical Algebraic Decomposition for Quantifier Elimination. *Journal of Symbolic Computation*, 12:299–328, 1991. [160]

[Collins, 1975] G. E. Collins. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In *Proceedings of the Second GI Conference on Automata Theory and Formal Languages*, volume 33 of *Lecture Notes in Computer Science*, pages 515–532. Springer-Verlag, 1975. [152, 160]

[Colmerauer and Dao Thi, 2000] A. Colmerauer and B. Dao Thi. Expressiveness of full first order constraints in the algebra of finite and infinite trees. In R. Dechter, editor, *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming*, volume 1894 of *Lecture Notes in Computer Science*, pages 172–186, Singapore, 2000. Springer-Verlag. [169]

[Colmerauer, 1990] A. Colmerauer. An Introduction to Prolog III. *Communications of the ACM*, 33(7):69–90, 1990. [163]

[Colmerauer, 1993] A. Colmerauer. Naive solving of non-linear constraints. In Frédéric Benhamou and Alain Colmerauer, editors, *Constraint Logic Programming: Selected Research*, pages 89–112. MIT Press, 1993. [164, 167]

[Conn *et al.*, 1988a] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Global convergence of a class of trust-region algorithms for optimization with simple bounds. *SINUM*, 25:433–460, 1988. [18]

[Conn *et al.*, 1988b] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Testing a class of methods for solving minimization problems with simple bounds on the variables. *Math. Comp.*, 50:399–430, 1988. [18]

[Conn *et al.*, 1989] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Global convergence of a class of trust-region algorithms for optimization with simple bounds. *SINUM*, 26:764–767, 1989. [18]

[Conn *et al.*, 1991] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. A globally convergent lagrangian algorithm for optimization with general constraints and simple bounds. *SINUM*, 28:545–572, 1991. [30]

[Conn *et al.*, 1994] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Large scale constrained optimization: A current survey. In E. Spedicato, editor, *Algorithms for Continuous Optimization – The State of the Art*, volume 434 of *NATO Advanced Science Institute Series C: Mathematical and Physical Sciences*, pages 287–332. Kluwer Academic Publishers, Dordrecht, 1994. Proceedings of the NATO Advanced Study Institute Conference, Il Ciocco, Barga, Italy, September 5-18, 1993. [9]

[Conn *et al.*, 1996] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. A primal-dual algorithm for minimizing a nonconvex function subject to bound and linear equality constraints. Technical Report 96/9, RAL, 1996. [30, 32, 37]

[Conn *et al.*, 1997] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. A globally convergent lagrangian barrier algorithm for optimization with general inequality constraints and simple bounds. *Math. Comp.*, 66:261–288, 1997. [31]

[Cooper, 1989] M.C. Cooper. An optimal k-consistency algorithm. *Artificial Intelligence*, 41, 1989. [178]

[Cornelius and Lohner, 1984] H. Cornelius and R. Lohner. Computing the range of values of real functions with accuracy higher than second order. *Computing*, 33:331–347, 1984. [96]

[Cousot and Cousot, 1977] P. Cousot and R. Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proceedings of ACM Symposium on Principles of Programming Languages (POPL)*, pages 238–252, 1977. [163]

[Crowder *et al.*, 1979] H.P. Crowder, R.S. Dembo, and J.M. Mulvey. On reporting computational experiments with mathematical software. *ACM Trans. Math. Software*, 5:193–203, 1979. [112]

[Csendes and Ratz, 1997] T. Csendes and D. Ratz. Subdivision direction selection in interval methods for global optimization. *SIAM J. Numer. Anal.*, 34:922–938, 1997. [109]

[Dallwig *et al.*, 1997] S. Dallwig, A. Neumaier, and H. Schichl. Glopt – a program for constrained global optimization. In I. Bomze *et al.*, editor, *Developments in Global Optimization*, pages 19–36. Kluwer, Dordrecht, 1997. [74, 85, 90]

[Dantzig *et al.*, 1958] G.B. Dantzig, S. Johnson, and W. White. A linear programming approach to the chemical equilibrium problem. *Management Science*, 5:38–43, 1958. [103]

[Dantzig, 1960] G.B. Dantzig. On the significance of solving linear programming problems with some integer variables. *Econometrica*, 28:30–44, 1960. [104]

[Davis, 1987] Ernest Davis. Constraint Propagation with Interval Labels. *Artificial Intelligence*, 32:281–331, 1987. [115, 178, 183]

[Debruyne and Bessière, 1997] R. Debruyne and C. Bessière. From restricted path consistency to max-restricted path consistency. In *CP-97*, pages 312–326, Linz, Austria, 1997. [178]

[Dechter *et al.*, 1990] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3), 1990. [185]

[Dechter, 1992] R. Dechter. From local to global consistency. *Artificial Intelligence*, 55, 1992. [185]

[Deville and van Hentenryck, 1991] Y. Deville and P. van Hentenryck. An efficient arc consistency algorithm for a class of CSP problems. In *Proceedings of the 12th International Joint Conference on AI*, 1991. [178]

[Deville *et al.*, 1997] Y. Deville, O. Barette, and P. Van Hentenryck. Constraint satisfaction over connected row convex constraints. In *IJCAI-97*, pages 405–410, Nagoya, Japan, 1997. [185]

[Di Pillo *et al.*, 2000] G. Di Pillo, S. Lucidi, and L. Palagi. A superlinearly convergent primal-dual algorithm model for constrained optimization problems with bounded variables. *OMS*, 14:49–73, 2000. [19]

[Didrit, 1997] O. Didrit. *Analyse par intervalles pour l'automatique: résolution globale et garantie de problèmes non-linéaires en robotique.* PhD thesis, Université Paris-Sud, Orsay, 1997. [177, 184]

[Dixon and Szegö, 1975] L.C.W. Dixon and G.P. Szegö. *Towards global optimization.* Elsevier, New York, 1975. [61, 70, 111]

[Dolan and Moré, 2000] E.D. Dolan and J.J. Moré. Benchmarking optimization software with cops. Technical Report ANL/MCS-246, Argonne National Laboratory Mathematics and Computer Science Division, November 2000. [43]

[Dorigo *et al.*, 1991] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: optimization by a colony of cooperating agents. *IEEE Trans. Systems, Man, Cyber. Part B*, 26:29–41, 1991. [70]

[Dueck and Scheuer, 1990] G. Dueck and T. Scheuer. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *J. Comp. Physics*, 90:161–175, 1990. [70]

[Duff and Reid, 1983] I.S. Duff and J.K. Reid. The multifrontal solution of indefinite sparse symmetric linear systems. *TOMS*, 9:302–323, 1983. [26]

[Duff and Reid, 1993] I.S. Duff and J.K. Reid. Ma47: A set of fortran subroutines for solving sparse symmetric sets of linear equations. Technical report, Rutherford Appleton Laboratory, Chilton, 1993. [26]

[Duran and Grossmann, 1986] M.A. Duran and I.E. Grossmann. An outer approximation algorithm for a class of mixed integer nonlinear programs. *Math. Program.*, 36:307–339, 1986. [69]

[Durazzi, 2000] C. Durazzi. On the newton interior-point method for nonlinear programming problems. *J.O.T.A.*, 104:73–90, 2000. [42]

[Ebers and Moll, 1954] J. J. Ebers and J. L. Moll. Large-Scale Behaviour of Junction Transistors. In *IEE. Proc.*, volume 42, pages 1761–1772, 1954. [142]

[El-Bakry *et al.*, 1996] A.S. El-Bakry, R.A. Tapia, T. Tsuchiya, and Y. Zhang. On the formulation and theory of the newton interior-point method for nonlinear programming. *J.O.T.A.*, 89:507–541, 1996. [42]

[Epelly *et al.*, 2000] O. Epelly, J. Gondzio, and J.-P. Vial. An interior-point solver for smooth convex optimization with an application to environmental-energy-economic models. Technical Report 2000.8, Department of Management Studies, University of Geneva, Geneva, Switzerland, July 2000. [27, 43]

[Epperly and Pistikopoulos, 1997] T.G.W. Epperly and E.N. Pistikopoulos. A reduced space branch and bound algorithm for global optimization. *J. Global Optim.*, 11:287–311, 1997. [107]

[Epperly and Swaney, 1996] T.G.W. Epperly and R.E. Swaney. Branch and bound for global nlp. In I.E. Grossman, editor, *Global Optimization in Engineering Design*, chapter 1–2. Kluwer, Dordrecht, 1996. [107]

[Falk and Soland, 1969] J.E. Falk and R.M. Soland. An algorithm for separable nonconvex programming. *Management Sci.*, 15:550–569, 1969. [104]

[Faltings and Gelle, 1997] Boi Faltings and Esther Gelle. Local Consistency for Ternary Numeric Constraints. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 392–397, Nagoya, Japan, 1997. [115, 179, 181]

[Faltings, 1994] Boi Faltings. Arc Consistency for Continuous Variables. *Artificial Intelligence*, 65(2):363–376, 1994. [115, 179, 180, 181]

[Farouki and Rajan, 1987] Rida T. Farouki and V. T. Rajan. On the Numerical Condition of Polynomials in Bernstein Form. *Computer Aided Geometric Design*, 4:191–216, 1987. [123]

[Felkel and Spellucci, 1996] R. Felkel and P. Spellucci. Numerical experiments with an exact penalty function for convex inequality constrained qp-problems. Technical Report 1815, TUD Dept. Math., 1996. http://www.mathematik.th-darmstadt.de/ags/ag8/felkel/felkel-d.html. [23]

[Felkel, 1996] R. Felkel. An interior-point method for large-scale qp problems. Technical Report 1850, TUD Dept. Math., 1996. http://www.mathematik.th-darmstadt.de/ags/ag8/felkel/felkel-d.html. [36]

[Felkel, 1999] R. Felkel. *On solving large-scale nonlinear programming problems using iterative methods.* Shaker, Aachen, 1999. ISBN 3-8265-6797-8. [18]

[Fiacco and McCormick, 1990a] A. Fiacco and G.P. McCormick. *Sequential Unconstrained Minimization Techniques.* Number 4 in Classics in Applied Mathematics. SIAM, Philadelphia, 1990. [73]

[Fiacco and McCormick, 1990b] A.V. Fiacco and G.P. McCormick. *Nonlinear Programming. Sequential unconstrained minimization techniques.* Wiley, New York, 1990. Reprinted by SIAM publications. [26]

[Fletcher and Leyffer, 1994] R. Fletcher and S. Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Math. Program.*, 66:327–349, 1994. [69]

[Fletcher and Leyffer, 1997] R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. Num. Anal. NA/171, Dept. of Math., University of Dundee, 1997. [36]

[Fletcher and Leyffer, 1998a] R. Fletcher and S. Leyffer. Numerical experience with lower bounds for miqp branch–and–bound. *SIAM J. Optim.*, 8:604–616, 1998. [69]

[Fletcher and Leyffer, 1998b] R. Fletcher and S. Leyffer. On the global convergence of an slp-filter algorithm. Technical Report 98/13, Dept. of Math., FUNDP, Namur, Belgium, 1998. [36]

[Fletcher and Leyffer, 1998c] R. Fletcher and S. Leyffer. User manual for filtersqp. Num. Anal. NA/181, Dept. of Math., University of Dundee, 1998. [36]

[Fletcher and Sainz de la Maza, 1989] R. Fletcher and E. Sainz de la Maza. Nonlinear programming and nonsmooth optimization by successive linear programming. *Math. Prog.*, 43:235–256, 1989. [32]

[Fletcher *et al.*, 1999] R. Fletcher, N.I.M. Gould, S. Leyffer, and Ph. L. Toint. Trust-region sqp-filter algorithms for general nonlinear programming. Technical Report 99/03, Dept. of Math., FUNDP, Namur, Belgium, 1999. [36]

[Fletcher, 1971] R. Fletcher. A general quadratic programming algorithm. *J.I.M.A.*, 7:76–91, 1971. [20]

[Floudas and Pardalos, 1990] C. A. Floudas and P.M. Pardalos. *A collection of test problems for constrained global optimization algorithms.* Number 455 in Lecture Notes Comp. Sci. Springer, Berlin, 1990. [58]

[Floudas and Pardalos, 1992] C.A. Floudas and P.M. Pardalos, editors. *Recent Advances In Global Optimization.* Princeton Univ. Press, Princeton, 1992. [61]

[Floudas and Pardalos, 2000] C.A. Floudas and P.M. Pardalos. *Optimization in Computational Chemistry and Molecular Biology: Local and Global Approaches.* Kluwer, Dordrecht, 2000. [61]

[Floudas *et al.*, 1999] C.A. Floudas, P.M. Pardalos, C.S. Adjiman, W.R. Esposito, Z.H. Gümüs, S.T. Harding, J.L. Klepeis, C.A. Meyer, and C.A. Schweiger. *Handbook of Test Problems in Local and Global Optimization.* Kluwer, Dordrecht, 1999. [111, 112]

[Floudas, 1995] C.A. Floudas. *Nonlinear and Mixed-Integer Optimization: Fundamentals and Applications*. Oxford Univ. Press, Oxford, 1995. [62, 107, 110]

[Floudas, 1999] C.A. Floudas. *Deterministic Global Optimization: Theory, Algorithms and Applications*. Kluwer, Dordrecht, 1999. [60, 110]

[Floudas, 2000] C.A. Floudas. Deterministic global optimization in design, control, and computational chemistry. Manuscript, 2000. [58]

[Forgó, 1988] F. Forgó. *Nonconvex Programming*. Akadémiai Kiadó, Budapest, 1988. [60]

[Forrest, 1993] S. Forrest. Genetic algorithms: principles of natural selection applied to computation. *Science*, 261:872–878, 1993. [70]

[Forsgren and Gill, 1998] A. Forsgren and Ph. E. Gill. Primal-dual interior methods for nonconvex nonlinear programming. *SIAM J. Optim.*, 8:1132–1152, 1998. [38]

[Forsgren and Murray, 1997] A. Forsgren and W. Murray. Newton methods for large-scale linear inequality constrained minimization. *SIOPT*, 7:162–176, 1997. [20]

[Fourer, 2001] R. Fourer. linear/nonlinear programming faq's. http://www.mcs.anl.gov/otc/Guide/faq, 2001. [27]

[Franke, 1995] R. Franke. Hqp - a solver for sparse nonlinear optimization. Internal report, TU Ilmenau dept. of autom. and systems theory, 1995. [30, 32]

[Freuder, 1978] Eugène C. Freuder. Synthesizing Constraint Expressions. *Communications of the ACM*, 21(11):958–966, 1978. [114, 131, 141, 178, 183, 185]

[Freuder, 1982] E.C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982. [185]

[Freund and Jarre, 1994] R. Freund and F. Jarre. Iterative solution of indefinite systems arising in large-scale optimization by the symmetric qmr algorithm. Presented at XVISMP in Ann Arbor, 1994. [26, 29]

[Friedlander *et al.*, 1994] A. Friedlander, J.M. Martínez, and A.S. Santos. On the resolution of linearly constrained convex minimization problems. *SIOPT*, 4:331–339, 1994. [21]

[Frühwirth, 1995] T. Frühwirth. Constraint handling rules. In A. Podelski, editor, *Constraint Programming: Basics and Trends*, volume 910 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995. [170]

[Frühwirth, 1998] T. Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming*, 37(1–3):95–138, October 1998. Special Issue on Constraint Logic Programming (P. Stuckey and K. Marriot, Eds.). [170]

[Gablonsky and Kelley, 2001] J.M. Gablonsky and C.T. Kelley. A locally-biased form of the DIRECT algorithm. *J. Global Optim.*, 2001. to appear. [68]

[Garey and Johnson, 1979] M.R. Garey and D.S. Johnson. *Computers and Intractability*. Freeman, San Francisco, CA, 1979. [55]

[Garloff and Graf, 1999] Jürgen Garloff and Brigit Graf. Solving Strict Polynomial Inequalities by Bernstein Expansion. In N. Munro, editor, *Symbolic Methods in Control System Analysis and Design*, pages 339–352, 1999. [123, 188]

[Gauvin, 1977] J. Gauvin. A necessary and sufficient regularity condition to have bounded multipliers in nonconvex programming. *Math. Prog.*, 12:136–138, 1977. [11]

[Gay *et al.*, 1998] D. Gay, M. Overton, and M.H. Wright. A primal-dual interior method for nonconvex nonlinear programming. In Yuan Ya-xiang, editor, *Advances in nonlinear programming: Proceedings of the 96 international conference. Beijing, China, September 2–5, 1996*, pages 31–56. Kluwer Academic Publishers, Dordrecht, 1998. Appl. Optim. 14. [40]

[Gay, 1984] D.M. Gay. A trust-region approach to linearly constrained optimization,. In D.F. Griffiths, editor, *Numerical Analysis*, number 1066 in Lecture Notes in Mathematics, pages 72–105. Springer, Berlin, 1984. [112]

[Geddes *et al.*, 1991] K. Geddes, G. Gonnet, and B. Leong. *Maple V: Language reference manual.* Springer Verlag, New York, Berlin, Paris, 1991. [153]

[Gelle and Faltings, 2001] E. Gelle and B. Faltings. Solving mixed and conditional constraint satisfaction problems. *Constraints*, 2001. [179]

[Gelle, 1998] E. Gelle. *On the generation of locally consistent solution spaces in mixed dynamic constraints problems.* PhD thesis, Ecole Polytechnique Fédérale de Lausanne, Switzerland, 1998. [179, 181]

[Gilbert and Lemarechal, 1989] J. Ch. Gilbert and C. Lemarechal. Some numerical experiments with variable-storage quasi-newton methods. *Math. Prog.*, 45:407–435, 1989. [17]

[Gill *et al.*, 1996] Ph. E. Gill, M. A. Saunders, and J.R. Shinnerl. On the stability of cholesky factorization for symmetric quasidefinite systems. *SIAM J. Matrix Anal. Appl.*, 17:35–46, 1996. [26]

[Gill *et al.*, 1997] Ph. E. Gill, W. Murray, and M. Saunders. User's guide for snopt (version 5.3): A fortran package for large-scale nonlinear programming. Technical Report NA 97-4, Department of mathematics, University of California at San Diego, 1997. [28]

[Glover and Laguna, 1997] F. Glover and M. Laguna. *Tabu Search.* Kluwer, Boston, 1997. [70]

[Glover, 1989] F. Glover. Tabu search, part 1. *ORSA J. Comput.*, 1:190–206, 1989. [70]

[Glover, 1990] F. Glover. Tabu search, part 2. *ORSA J. Comput.*, 2:4–32, 1990. [70]

[Goldberg, 1991] David M. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, March 1991. [118]

[Goldfarb, 1972] D. Goldfarb. Extensions of newton's method and simplex methods for solving quadratic programs. In Lootsma, editor, *Numerical methods for nonlinear optimization*, pages –. Academic Press, New York, 1972. [20]

[Goldfard and Idnani, 1983] D. Goldfard and A. Idnani. A numerically stable method for solving strictly convex quadratic programs. *Math. Prog.*, 27:1–33, 1983. [20]

[Gomory, 1960] R.E. Gomory. An algorithm for the mixed integer problem. Technical Report RM-2597, The Rand Corporation, 1960. [110]

[Goualard *et al.*, 1999] Frédéric Goualard, Frédéric Benhamou, and Laurent Granvilliers. An Extension of the WAM for Hybrid Interval Solvers. *Journal of Functional and Logic Programming*, 5(4):1–31, 1999. [127]

[Goualard, 2000] Frédéric Goualard. *Languages and Environments for Interval Constraint Programming*. PhD thesis, University of Nantes, France, 2000. In French. [123]

[Gould and Toint, 1999] N.I.M. Gould and Ph. L. Toint. SQP methods for large-scale nonlinear programming. Technical Report 99/05, Dept. of Math., FUNDP, Namur, Belgium, 1999. ftp://thales.math.fundp.ac.be/pub/reports. [9, 32]

[Gould and Tolle, 1971] F.J Gould and J.W. Tolle. A necessary and sufficient qualification for constrained optimization. *SIAM J. Appl. Math.*, 20:164–172, 1971. [10]

[Gould *et al.*, 1998] N.I.M. Gould, M.E. Hribar, and J. Nocedal. On the solution of equality constrained quadratic programming problems arising in optimization. Technical Report 09/24/98, Optimization Technology Center, Northwestern University, Evanston, 1998. [29]

[Gould, 1991] N.I.M Gould. An algorithm for large-scale quadratic programming. *J.I.M.A. Num. Anal.*, 11:299–324, 1991. [20]

[Granvilliers and Benhamou, 2001] Laurent Granvilliers and Frédéric Benhamou. Progress in the Solving of a Circuit Design Problem. *Journal of Global Optimization*, 2001. To appear. [132, 133, 134, 146]

[Granvilliers and Hains, 2000] Laurent Granvilliers and Gaétan Hains. A Conservative Scheme for Parallel Interval Narrowing. *Information Processing Letters*, 74:141–146, 2000. [174]

[Granvilliers and Monfroy, 2000] Laurent Granvilliers and Eric Monfroy. Declarative modelling of constraint propagation strategies. In *Proceedings of the First Biennial International Conference on Advances in Information Systems*, volume 1909 of *Lecture Notes in Artificial Intelligence*, Izmir, Turkey, 2000. Springer-Verlag. [174]

[Granvilliers *et al.*, 1999] Laurent Granvilliers, Frédéric Goualard, and Frédéric Benhamou. Box Consistency through Weak Box Consistency. In Weiyi Meng, editor, *Proceedings of ICTAI'99, 11th IEEE International Conference on Tools with Artificial Intelligence.* IEEE Computer Society, 1999. [140, 142, 147]

[Granvilliers, 1998a] L. Granvilliers. *Consistance locales et transformation symboliques de contraintes d'intervalles.* PhD thesis, Université d'Orl'eans, 1998. [178]

[Granvilliers, 1998b] Laurent Granvilliers. A Symbolic-Numerical Branch and Prune Algorithm for Solving Non-linear Polynomial Systems. *Journal of Universal Computer Science*, 4(2):125–146, 1998. [146, 161, 165]

[Granvilliers, 2001] Laurent Granvilliers. On the Combination of Interval Constraint Solvers. *Reliable Computing*, 7(6), 2001. To appear. [142]

[Gray *et al.*, 1997] P. Gray, W. Hart, L. Painton, C. Phillips, M. Trahan, and J. Wagner. A survey of global optimization methods. http://www.cs.sandia.gov/opt/survey/, 1997. [57, 70]

[Greenberg, 1990] H.J. Greenberg. Computational testing: why, how, and how much. *ORSA J. Computing*, 2:94–97, 1990. [112]

[Griewank and Reese, 1991] A. Griewank and S. Reese. On the calculation of Jacobian matrices by the Markowitz rule. In A. Griewank and G.F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 126–135. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1991. [50, 51]

[Griewank and Toint, 1982] A. Griewank and Ph. L. Toint. Partitioned variable metric updates for large structured optimization problems. *Num. Math.*, 39:429–448, 1982. [13]

[Griewank, 1989] A. Griewank. On automatic differentiation. In M. Iri and K. Tanabe, editors, *Mathematical Programming*, pages 83–107. KTK Scientific Publishers, Tokyo, 1989. [50, 133]

[Griewank, 1991] A. Griewank. *Automatic Evaluation of First and Higher-Derivative Vectors*, volume 97 of *International Series of Numerical Mathematics*, pages 135–148. Birkhäuser Verlag, 1991. [50]

[Griewank, 1992] A. Griewank. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and Software*, 1(1):35–54, 1992. [52]

[Grossman, 1990] I.E. Grossman. Mixed-integer nonlinear programming techniques for the synthesis of engineering systems. *Research in Engineering Design*, 1:205–228, 1990. [107]

[Grossman, 1996] I.E. Grossman, editor. *Global Optimization in Engineering Design*. Kluwer, Dordrecht, 1996. [61]

[Guddat *et al.*, 1986] J. Guddat, H.Th. Jongen, and J. Rueckmann. On stability and stationary points in nonlinear optimization. *J. Austr. Math. Soc.*, Ser. B 28:36–56, 1986. [11]

[Guddat *et al.*, 1990] J. Guddat, F. Guerra Vasquez, and H.Th. Jongen. *Parametric optimization: singularities, path following and jumps*. Wiley, Chichester, 1990. [70, 71, 90]

[Guddat *et al.*, 1997] J. Guddat, F. Guerra, and D. Nowack. On the role of the mangasarian-fromowitz constraint qualification for penalty-, exact penalty-, and lagrange multiplier methods. In A. Fiacco, editor, *Mathematical programming with data perturbations*, volume 195 of *Lect. Notes Pure Appl. Math*, pages 159–183. Marcel Dekker, New York, 1997. 17th symposium, George Washington Univ. [11, 31]

[Guignard, 1969] M. Guignard. Generalized kuhn-tucker conditions for mathematical programming in a banach space. *SIAM J. Control*, 7:232–241, 1969. [10]

[Hager, 1993] G.D. Hager. Solving large systems of nonlinear constraints with application to data modeling. *Interval Computations*, 3:169–200, 1993. [85]

[Hales, 1998] T.C. Hales. The kepler conjecture. http://www.math.lsa.umich.edu/~hales/countdown/, 1998. math.MG/9811071. [57, 111]

[Hansen and Sengupta, 1981] Eldon R. Hansen and Saumyendra Sengupta. Bounding Solutions of Systems of Equations using Interval Analysis. *BIT*, 21:203–211, 1981. [137]

[Hansen, 1992] E.R. Hansen. *Global Optimization Using Interval Analysis*. Dekker, New York, 1992. [60, 144, 158]

[Haralick and Elliot, 1980] R.M. Haralick and L.G. Elliot. Increasing tree search consistency for constraint satisfaction problems. *Artificial Intelligence*, 14, 1980. [178]

[Haroud and Faltings, 1994] D. Haroud and B. Faltings. Global consistency for continuous constraints. In Alan Borning, editor, *Lecture notes in computer Science 874: Principles and Practice of constraint programming*. Springer Verlag, 1994. [185]

[Heinz and Spellucci, 1994] J. Heinz and P. Spellucci. A successful implementation of the pantoja-mayne SQP method. *Optimization Methods and Software*, 4:1–28, 1994. [35]

[Henkin *et al.*, 1971] Leon Henkin, J. Donald Monk, and Alfred Tarski. *Cylindric algebras.* Studies in Logic. North-Holland Publishing Company, Amsterdam-London, 1971. [130]

[Hickey, 2000] Timothy J. Hickey. CLIP: a CLP(Intervals) Dialect for Metalevel Constraint Solving. In *Proceedings of the International Workshop on Practical Aspects of Declarative Languages*, volume 1753 of *Lecture Notes in Computer Science*, pages 200–214. Springer-Verlag, 2000. [127, 144, 158, 169]

[Hofstedt, 2000] P. Hofstedt. Better communication for tighter cooperation. In *Proceedings of Computational Logic, London, UK*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 342–357. Springer-Verlag, 2000. [172]

[Holland, 1973] J. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM J. Computing*, 2:88–105, 1973. [70]

[Hong and Stahl, 1994] Hoon Hong and Volker Stahl. Safe Start Region by Fixed points and Tightening. *Computing*, 53(3-4):323–335, 1994. [124]

[Hong and Stahl, 1995] Hoon Hong and Volker Stahl. Bernstein Form is Inclusion Monotone. *Computing*, 55:43–53, 1995. [123]

[Hong, 1992] H. Hong. Non-linear real constraints in constraint logic programming. In Hélène Kirchner and G. Levi, editors, *Proceedings of the 3rd International Conference on Algebraic and Logic Programming*, volume 632 of *Lecture Notes in Computer Science*, pages 201–212, Volterra (Italy), 1992. Springer-Verlag. [169]

[Horst and Pardalos, 1995] R. Horst and P.M. Pardalos. *Handbook of Global Optimization*. Kluwer, Dordrecht, 1995. [60]

[Horst and Tuy, 1990] R. Horst and H. Tuy. *Global Optimization: Deterministic Approaches*. Springer, Berlin, 2nd edition, 1990. [61]

[Horst *et al.*, 1995] R. Horst, P.M. Pardalos, and N.V. Thoai. *Introduction to Global Optimization*. Kluwer, Dordrecht, 1995. [61, 104, 107]

[Hribar, 1996] M.E. Hribar. *Large-scale constrained optimization*. PhD thesis, Northwestern University, Evanston, Illinois, 1996. [33]

[Huyer and Neumaier, 1999] W. Huyer and A. Neumaier. Global optimization by multilevel coordinate search. *J. Global Optim.*, 14:331–355, 1999. [68, 77, 111]

[Hyvönen and De Pascale, 1996] E. Hyvönen and S. De Pascale. Interval computations on the spreadsheet. In R. B. Kearfott and V. Kreinovich, editors, *Applications of Interval Computations, Applied Optimization*, pages 169–209. Kluwer, Dordrecht, 1996. [85]

[Hyvönen, 1992] Eero Hyvönen. Constraint Reasoning based on Interval Arithmetic. The Tolerance Propagation Approach. *Artificial Intelligence*, 58:71–112, 1992. [115, 178, 179]

[IEEE, 1985] IEEE. IEEE standard for binary floating-point arithmetic. Technical Report IEEE Std 754-1985, Institute of Electrical and Electronics Engineers, 1985. Reaffirmed 1990. [118, 131]

[Ilog, 2001] Ilog optimization suite – white paper, 2001. Available via http://www.ilog.fr. [171]

[Ingber, 1993] L. Ingber. Simulated annealing: Practice versus theory. *Math. Comput. Modelling*, 18:29–57, 1993. [70]

[InteriorPoint, 2001] Interior points online. http://www.mcs.anl.gov/otc/InteriorPoint/, 2001. [24]

[Iri, 1984] M. Iri. Simultaneous computation of functions, partial derivatives and estimates of rounding errors —complexity and practicality—. *Japan Journal of Applied Mathematics*, 1:223–252, 1984. [49, 53]

[Jackson *et al.*, 1990] R.H.F. Jackson, P.T. Boggs, S.G. Nash, and S. Powell. Guidelines for reporting results of computational experiments, report of the ad hoc committee. *Math. Program.*, 49:413–426, 1990. [112]

[Jaffar *et al.*, 1992] J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. The CLP($\Re$) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, July 1992. [167]

[Janka, 2000] E. Janka. A comparison of stochastic methods for global optimization. http://www.mat.univie.ac.at/~vpk/math/gopt_eng.html, 2000. [73, 111]

[Jansson and Knüppel, 1992] C. Jansson and O. Knüppel. A global minimization method: The multi-dimensional case. Technical Report 92-1, TU Hanburg-Harburg, January 1992. [107]

[Jansson and Knüppel, 1995] C. Jansson and O. Knüppel. Branch and bound algorithm for bound constrained optimization problems without derivatives. *J. Global Optim.*, 7:297–333, 1995. [111, 112]

[Jaulin *et al.*, 2000] L. Jaulin, M. Kieffer, I. Braems, and E. Walter. Guaranteed non-linear estimation using constraint propagation on sets. *International Journal of control*, 2000. [177, 184]

[Jaulin, 1994] L. Jaulin. *Solution Globale et Guarantie de problèmes ensemblistes: application à l'estimation non linéaire et à la commande robuste.* PhD thesis, Université Paris-Sud, Orsay, 1994. [177, 182, 184, 188]

[Jaulin, 2001] L. Jaulin. Path planning using intervals and graphs. *Reliable Computing*, 7(1), 2001. [177, 184, 188]

[Jeroslov, 1977] R.G. Jeroslov. Cutting plane theory: disjunctive methods. *Ann. Discrete Math.*, 1:293–330, 1977. [100]

[John, 1948] F. John. Extremum problems with inequalities as subsidiary conditions. In *Studies and Essays Presented to R. Courant on his 60th Birthday*, pages 187–204. Interscience, New York, 1948. Reprinted as pp. 543–560 of: Fritz John, Collected Papers, Vol.2 (J. Moser, ed.), Birkhäuser, Boston 1985. [63]

[Jonasson and Madsen, 1994] K. Jonasson and K. Madsen. Corrected sequential linear programming for sparse minimax optimization. *BIT*, 34:372–387, 1994. [32]

[Jones *et al.*, 1993] D.R. Jones, C.D. Perttunen, and B.E. Stuckman. Lipschitzian optimization without the lipschitz constant. *J. Optim. Th. Appl.*, 79:157–181, 1993. [68, 77]

[Jussien and Lhomme, 1998] N. Jussien and O. Lhomme. Dynamic domain splitting for numeric csps. In *Proceedings of ECAI'98*, 1998. [179]

[Kanzow, 1994] Ch. Kanzow. An unconstrained optimization technique for large-scale linearly constrained convex minimization problems. *Computing*, 53:101–117, 1994. [21]

[Karmarkar, 1984] N. Karmarkar. A new polynomial time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984. [24]

[Karush, 1939] W. Karush. Minima of functions of several variables with inequalities as side constraints. Master's thesis, Dept. of Mathematics, Univ. of Chicago, IL, 1939. [63]

[Kearfott and Du, 1994] R.B. Kearfott and K. Du. The cluster problem in multivariate global optimization. *J. Global Optim.*, 5:253–265, 1994. [95]

[Kearfott and Shi, 1996] R. Baker Kearfott and Xiaofa Shi. Optimal Preconditionners for Interval Gauss-Seidel Methods. In *Scientific Computing and Validated Numerics*, pages 173–178, 1996. [144, 158, 174]

[Kearfott, 1987] R. Baker Kearfott. Some Tests of Generalized Bisection. *ACM Transactions on Mathematical Software*, 13(3):197–220, 1987. [126]

[Kearfott, 1991] R.B. Kearfott. Decomposition of arithmetic expressions to improve the behavior of interval iteration for nonlinear systems. *Computing*, 47:169–191, 1991. [85]

[Kearfott, 1996a] R.B. Kearfott. A review of techniques in the verified solution of constrained global optimization problems. In R.B. Kearfott and V. Kreinovich, editors, *Applications of Interval Computations*, pages 23–60. Kluwer, Dordrecht, 1996. [98, 111]

[Kearfott, 1996b] R.B. Kearfott. *Rigorous Global Search: Continuous Problems.* Kluwer, Dordrecht, 1996. [60, 69, 83, 96, 98, 107, 110, 111]

[Kieffer *et al.*, 2000] M. Kieffer, L. Jaulin, E. Walter, and D. Meizel. Robust autonomous robot localization using interval analysis. *Reliable Computing*, 6, 2000. [177, 184]

[Kirkpatrick *et al.*, 1983] S. Kirkpatrick, C.D. Jr. Geddat, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983. [70]

[Kojima *et al.*, 1989] M. Kojima, S. Mizuno, and A. Yoshise. A primal-dual interior-point algorithm for linear programming. In N. Megiddo, editor, *Progress in mathematical programming: interior-point and related methods*, pages 29–47. Springer, New York, 1989. [25]

[Krawczyk, 1969] R. Krawczyk. Newton-algorithmen zur bestimmung von nullstellen mit fehlerschranken. *Computing*, 4:187–201, 1969. [83]

[Krawczyk, 1986] R. Krawczyk. A Class of Interval Newton Operators. *Computing*, 37:179–183, 1986. [137]

[Kuhn and Tucker, 1951] H.W. Kuhn and A.W. Tucker. Nonlinear programming. In J. Neyman, editor, *Proc. 2nd Berkeley Symp. Math. Stat. Prob.*, pages 481–492. Univ. of Calif. Press, Berkeley, CA, 1951. [8, 64]

[Kuhn, 1991] H.W. Kuhn. Nonlinear programming: a historical note. In J.K. *et al.* Lenstra, editor, *History of Mathematical Programming*, pages 82–96. North Holland, Amsterdam, 1991. [64]

[Lagrange, 1797] J.L. Lagrange. *Théorie des fonctions analytiques.* Impr. de la République, Paris, 1797. [64]

[Lalee *et al.*, 1998] M. Lalee, J. Nocedal, and T. Plantenga. On the implementation of an algorithm for large-scale equality constrained optimization. *SIOPT*, 8:682–706, 1998. [33]

[Land and Doig, 1960] A.H. Land and A.G. Doig. An automated method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960. [107]

[Laux, 1999] Th. Laux. Numerische untersuchungen zur primal-dualen methode von A. Forsgren und Ph. E. Gill. Diploma Thesis, TU Darmstadt, Dept. of mathematics, 1999. [39]

[Lee and Van Emden, 1993] Jimmy Lee and Maarten Van Emden. Interval Computation as Deduction in CHIP. *Journal of Logic Programming*, 16:255–276, 1993. [127]

[Levy and Montalvo, 1985] A.V. Levy and A. Montalvo. The tunneling algorithm for the global minimization of functions. *SIAM J. Sci. Stat. Comput.*, 6:15–29, 1985. [70]

[Lhomme *et al.*, 1998] Olivier Lhomme, Arnaud Gotlieb, and Michel Rueher. Dynamic Optimization of Interval Narrowing Algorithms. *Journal of Logic Programming*, 37(1–2):165–183, 1998. [140, 145, 147, 174]

[Lhomme, 1993] Olivier Lhomme. Consistency Techniques for Numeric CSPs. In Ruzena Bajcsy, editor, *Proceedings of IJCAI'93, 13th International Joint Conference on Artificial Intelligence*, pages 232–238, Chambéry, France, 1993. Morgan Kaufmann. [115, 140, 141, 146, 147, 168, 178]

[Lhomme, 1994] O. Lhomme. *Contribution à la résolution de contraintes sur les réels par propagation d'intervalles.* PhD thesis, Université de Nice-Sopjia-Antipolis, 1994. [178]

[Lin and Moré, 1999] C.-J. Lin and J.J. Moré. Newton's method for large bound-constrained optimization problems. *SIAM J. Optim.*, 9(4):1100–1127, 1999. [14, 18]

[Little *et al.*, 1963] J.D. Little, K.C. Murty, D.W. Sweeney, and C. Karel. An algorithm for the travelling salesman problem. *Operations Research*, 11:972–989, 1963. [107]

[Liu and Nocedal, 1989] D.C. Liu and J. Nocedal. On the limited memory bfgs method for large-scale optimization. *Math. Prog.*, 45:503–528, 1989. [13, 17]

[Lodwick, 1989] W.A. Lodwick. Constraint propagation, relational arithmetic in ai systems and. *mathematical programs, Ann. Oper. Res.*, 21:143–148, 1989. [85]

[Lottaz *et al.*, 1998] C. Lottaz, D. Sam-Haroud, B. Faltings, and I. Smith. Constraint techniques for collaborative design. In *Proceedings of ICTAI'98*, 1998. [177, 184]

[Lottaz *et al.*, 1999] C. Lottaz, D. Clément, B. Faltings, and I. Smith. Constraint-based support for collaboration in design and cnnstruction. *Journal of Computing in Civil Engineering*, 13:23–35, 1999. [177, 184, 186]

[Lottaz, 1999] C. Lottaz. Rewriting numeric csps for consistency algorithms. In *Proceedings of CP'99*, 1999. [188]

[Lottaz, 2000a] C. Lottaz. *Collaborative Design using Solution Spaces.* PhD thesis, Ecole Polytechnique Fédérale de Lausanne, Switzerland, 2000. [177, 182, 184, 186, 188]

[Lottaz, 2000b] C. Lottaz. Spacesolver - approximate solution spaces using an internet-based environement. http://liawww.epfl.ch/Research/research.html#ConstraintSatisfaction, 2000. [184, 188]

[Lucidi and Roma, 1997] St. Lucidi and M. Roma. Numerical experiences with new truncated newton methods in large-scale unconstrained optimization. *Comp. Opt. Appl.*, 7:71–87, 1997. [17]

[Luo *et al.*, 1996] Z.-Q. Luo, J.-S. Pang, and D. Ralph. *Mathematical Programs with Equilibrium Constraints.* Cambridge Univ. Press, Cambridge, 1996. [102]

[Lustig and Puget, 2000] I.J. Lustig and J.-F. Puget. Program $\neq$ program: constraint programming and its relationship to mathematical programming. http://www.ilog.com/products/optimization/tech/interfaces_informs.pdf, 2000. [60]

[Mackworth and Freuder, 1985] A.K. Mackworth and E.C. Freuder. The complexity of some polynomial consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25, 1985. [178]

[Mackworth, 1977] Alan K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1):99–118, 1977. [114, 130, 131, 178, 183]

[Mangasarian and Fromovitz, 1967] O.L. Mangasarian and S. Fromovitz. The fritz john necessary optimality conditions in the presence of equality and inequality constraints. *J. Math. Anal. Appl.*, 17:37–47, 1967. [10, 94]

[Markowitz and Manne, 1957] H.M. Markowitz and A.S. Manne. On the solution of discrete programming problems. *Econometrica*, 25:84–110, 1957. [104]

[Marti and Rueher, 1995] Philippe Marti and Michel Rueher. A Distributed Cooperating Constraints Solving System. *International Journal on Artificial Intelligence Tools*, 4(1–2):93–113, 1995. [127, 168]

[McCormick, 1976] G.P. McCormick. Computability of global solutions to factorable nonconvex programs: Part i – convex underestimating problems. *Math. Program.*, 10:147–175, 1976. [104]

[McDonald and Floudas, 1995] C.M. McDonald and C.A. Floudas. Global optimization for the phase and chemical equilibrium problem: application to the nrtl equation. *Comput. Chem. Eng.*, 19:1111–1139, 1995. [58]

[McPhee *et al.*, 1984] N. McPhee, S.-C. Chou, and X.-S. Gao. Mechanically proving geometry theorems using a combination of wu's method and collins' method. In Alan Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, pages 401–415, 1984. [167]

[Mehl *et al.*, 1995] M. Mehl, T. Müller, K. Popov, and R. Scheidhauer. *DFKI Oz User's manual.* DFKI, Sarrebrucken (Germany), May 1995. [171]

[Michalewicz, 1996] Z. Michalewicz. *Genetic Algorithm + Data Structures = Evolution Programs.* Springer, New York, 3rd edition, 1996. [70]

[Mittelmann and Spellucci, 2000] H.D. Mittelmann and P. Spellucci. Decision tree for optimization software. http://plato.la.asu.edu/guide.html, 2000. [8, 18, 27, 32]

[Mittelmann, 2000] H.D. Mittelmann. benchmarks for optimization software. http://plato.la.asu.edu/bench.html, 2000. [26, 27, 45]

[Mockus, 1989] J. Mockus. *Bayesian Approach to Global Optimization.* Kluwer, Dordrecht, 1989. [70]

[Mockus, 1994] J. Mockus. Application of bayesian approach to numerical methods of global and stochastic optimization. *J. Global Optim.*, 4:347–356, 1994. [70]

[Mohr and Henderson, 1986] R. Mohr and T.C. Henderson. Arc and path consistency revisted. *Artificial Intelligence*, 28:225–233, 1986. [178]

[Monfroy and Réty, 1999] E. Monfroy and J.-H. Réty. Chaotic Iteration for Distributed Constraint Propagation. In J. Carroll, H. Haddad, D. Oppenheim, B. Bryant, and G. Lamont, editors, *Proceedings of ACM Symposium on Applied Computing, Artificial Intelligence and Computational Logic Track*, pages 19–24, San Antonio, Texas, USA, March 1999. [174]

[Monfroy and Ringeissen, 1999] E. Monfroy and C. Ringeissen. An Open Automated Framework for Constraint Solver Extension: the SoleX Approach. *Fundamenta Informaticae*, 39(1–2), July 1999. Special Issue: Symbolic Computation and Artificial Intelligence. [172]

[Monfroy *et al.*, 1996] E. Monfroy, M. Rusinowitch, and R. Schott. Implementing Non-Linear Constraints with Cooperative Solvers. In K. M. George, J. H. Carroll, D. Oppenheim, and J. Hightower, editors, *Proceedings of ACM Symposium on Applied Computing*, pages 63–72, Philadelphia, PA, USA, February 1996. ACM Press. [168]

[Monfroy, 1992] Eric Monfroy. Gröbner Bases: Strategies and Applications. In *Proceedings of International Conference on Artificial Intelligence and Symbolic Mathematical Computation*, volume 737 of *Lecture Notes in Computer Science*, pages 133–151. Springer-Verlag, 1992. [159]

[Monfroy, 2000] E. Monfroy. The Constraint Solver Collaboration Language of BALI. In D.M. Gabbay and M. de Rijke, editors, *Frontiers of Combining Systems 2*, volume 7 of *Studies in Logic and Computation*, pages 211–230. Research Studies Press/Wiley, 2000. [171]

[Montanari, 1974] Ugo Montanari. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Science*, 7(2):95–132, 1974. [114, 178, 183, 185]

[Moore, 1966] Ramon E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, NJ, 1966. [115, 116, 118, 124, 134, 136, 152, 155]

[Moore, 1979] R.E. Moore. *Methods and Applications of Interval Analysis.* SIAM, Philadelphia, 1979. [81, 82]

[Moré and Toraldo, 1989] J.J. Moré and G. Toraldo. Algorithms for bound constrained quadratic programming problems. *Num. Math.*, 55:377–400, 1989. [18]

[Moré and Toraldo, 1991] J.J. Moré and G. Toraldo. On the solution of large quadratic programming problems with bound constraints. *SIOPT*, 1:93–113, 1991. [18]

[Moré and Wright, 1993] J.J. Moré and St.J. Wright. *Optimization software guide.* SIAM, Philadelphia, 1993. [27]

[Moré *et al.*, 1981] J.J. Moré, B.S. Garbow, and K.E. Hillstrom. Testing unconstrained optimization software. *ACM Trans. Math. Software*, 7:17–41, 1981. [112]

[Motzkin and Strauss, 1965] T.S. Motzkin and E.G. Strauss. Maxima for graphs and a new proof of a theorem of turan. *Canad. J. Math.*, 17:533–540, 1965. [57]

[Mozart consortium, 1999] Mozart consortium. The Mozart programming system, 1999. Available from http://www.mozart-oz.org. [171]

[Muller, 1997] Jean-Michel Muller. *Elementary Functions: Algorithms and Implementation.* Birkhäuser, 1997. [118, 119]

[Murray, 1997] W. Murray. Sequential quadratic programming methods for large-scale problems. *COAP*, 7:127–142, 1997. [32]

[Murty and Kabadi, 1987] K.G. Murty and S.N. Kabadi. Some np-complete problems in quadratic and nonlinear programming. *Math. Program.*, 39:117–129, 1987. [55]

[Narin'yani, 1983] Aleksandr Narin'yani. Sub-definiteness and Basic Means of Knowledge Representation. *Computers and Artificial Intelligence*, 5, 1983. [170]

[Nash and Sofer, 1993] St. G. Nash and A. Sofer. A barrier method for large-scale constrained optimization. *ORSA J. on Computing*, 5:40–53, 1993. [38]

[Nash, 1984] St. G. Nash. Newton type minimization via the lanczos method. *SINUM*, 21:770–788, 1984. [13]

[Nash, 2001] St. G. Nash. Code tn and tnb. http://netlib.org/opt/index.html, 2001. [15]

[Nemhauser and Wolsey, 1988] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, New York, 1988. [56, 59, 110]

[Nemhauser and Wolsey, 1989] G.L. Nemhauser and L.A. Wolsey. Integer programming. In G.L. *et al.* Nemhauser, editor, *Optimization*, number 1 in Handbooks in Operations Research and Management Science, chapter VI (pp. 447-527). North Holland, Amsterdam, 1989. [56, 59, 110]

[Neumaier, 1990] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge Univ. Press, Cambridge, 1990. [60, 81, 82, 83, 84, 91, 96, 97, 116, 118]

[Neumaier, 1996] A. Neumaier. Second-order sufficient optimality conditions for local and global nonlinear programming. *J. Global Optim.*, 9:141–151, 1996. [84, 96, 97]

[Neumaier, 1997] A. Neumaier. Molecular modeling of proteins and mathematical prediction of protein structure. *SIAM Review*, 39:407–460, 1997. [56, 58]

[Neumaier, 2001] A. Neumaier. Rational functions with prescribed global and local minimizers. Manuscript, 2001. [76]

[Ni, 1993] Q. Ni. General large-scale nonlinear programming using sequential quadratic programming methods. *Bayreuther Mathematische Schriften*, 45:133–236, 1993. [28]

[Nickel and Tolle, 1989] R.H. Nickel and J.W. Tolle. A sparse sequential quadratic programming algorithm. *J.O.T.A.*, 60:453–473, 1989. [28]

[Nocedal and Wright, 1999] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, Berlin, 1999. [11, 12, 60, 73]

[Older and Vellino, 1993] W. Older and A. Vellino. Constraint arithmetic on real intervals. In F. Benhamou and A. Colmerauer, editors, *Constrained Logic Programming: Selected Research*. MIT Press, 1993. [85, 178]

[Outrata *et al.*, 1998] J. Outrata, M. Kočvara, and J. Zowe. *Nonsmooth Approach to Optimization problems with Equilibrium Constraints*. Kluwer, Dordrecht, 1998. [102]

[Paige and Saunders, 1975] C.C. Paige and M.A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12:617–629, 1975. [29]

[Pardalos and Rosen, 1987] P.M. Pardalos and J.B. Rosen. *Constrained Global Optimization: Algorithms and Applications.* Number 268 in Lecture Notes in Computer Science. Springer, Berlin, 1987. [104]

[Pardalos and Schnitger, 1988] P.M. Pardalos and G. Schnitger. Checking local optimality in constrained quadratic programming is np-hard. *Oper. Res. Lett.*, 7:33–35, 1988. [55]

[Pinter, 1996a] J.D. Pinter. Continuous global optimization software: a brief review. *Optima*, 52:1–8, 1996. [57]

[Pinter, 1996b] J.D. Pinter. *Global Optimization in Action.* Kluwer, Dordrecht, 1996. [58, 68, 75, 77, 107]

[Pinz, 1996] W. Pinz. Ein neues verfahren zum lösen quadratischer optimierungsprobleme. Diploma Thesis, TU Darmstadt, Dept. of mathematics, 1996. [19]

[Plantenga, 1994] T.D. Plantenga. *Large-scale constrained optimization using trust-regions.* PhD thesis, Northwestern University, Evanston, Illinois, 1994. [32]

[Polyak, 1992] R. Polyak. Modified barrier functions (theory and methods). *Math. Prog.*, 54:177–222, 1992. [36]

[Potra *et al.*, 1993] F. Potra, R. Tapia, and Y. Zhang. On the superlinear convergence of interior-point algorithms for a general class of problems. *SIAM J. Optim.*, 3:413–422, 1993. [27]

[PrologIA, 1996] PrologIA. Prolog IV: constraints inside, 1996. Prolog IV reference manual. [115, 127, 163, 164, 170]

[Puget and Leconte, 1995] Jean-Franois Puget and Michel Leconte. Beyond the Glass Box: Constraints as Objects. In John Lloyd, editor, *Proceedings of ILPS'95, International Logic Programming Symposium*, pages 513–527, Portland, USA, 1995. MIT Press. [115, 127]

[Puget and Van Hentenryck, 1998] Jean-Franois Puget and Pascal Van Hentenryck. A Constraint Satisfaction Approach to a Circuit Design Problem. *J. Global Optim.*, 13(1):75–93, 1998. [142]

[Rall, 1981] L.B. Rall. *Automatic Differentiation: Techniques and Applications.* Number 120 in Springer Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1981. [48]

[Ratliff and Pierskalla, 1981] H.D. Ratliff and W. Pierskalla. Reporting computational experience in operations research. *Operations Research*, 29:xi–xiv, 1981. [112]

[Ratschan, 2001] S. Ratschan. Search heuristics for box decomposition methods. *preprint (http://www.risc.uni-linz.ac.at/people/sratscha/)*, 2001. [188]

[Ratschek and Rokne, 1980] Helmut Ratschek and John Rokne. About the centered form. *SIAM Journal on Numerical Analysis*, 17(3):333–337, 1980. [124]

[Ratschek and Rokne, 1984] H. Ratschek and J.G. Rokne. *New Computer Methods for the Range of Functions.* Ellis Horwood, Chichester, 1984. [61]

[Ratschek and Rokne, 1988] H. Ratschek and J.G. Rokne. *New Computer Methods for Global Optimization.* Wiley, New York, 1988. [61]

[Ratschek and Rokne, 1993] Helmut Ratschek and John Rokne. Experiments using Interval Analysis for Solving a Circuit Design Problem. *J. Global Optim.*, 3:501–518, 1993. [142]

[Ratschek and Rokne, 1995] Helmut Ratschek and John Rokne. Interval Methods. In R. Horst and P. M. Pardalos, editors, *Handbook of Global Optimization*, Nonconvex optimization and its applications ; vol. 2, pages 751–828. Kluwer Academic Publishers, Dordrecht, Boston, 1995. [125]

[Ratschek, 1980] Helmut Ratschek. Centered forms. *SIAM Journal on Numerical Analysis*, 17(5):656–662, 1980. [124]

[Ratz and Csendes, 1995] D. Ratz and T. Csendes. On the selection of subdivision directions in interval branch-and-bound methods for global optimization. *J. Global Optim.*, 7:183–207, 1995. [109]

[Ratz, 1996] D. Ratz. On branching rules in second-order branch-and-bound methods for global optimization. In G. Alefeld, editor, *Scientific computation and validation.* Akademie-Verlag, Berlin, 1996. [109]

[Robinson, 1972] St. Robinson. A quadratically convergent algorithm for general nonlinear programming problems. *Math. Prog.*, 1:145–156, 1972. [28]

[Robinson, 1976] St. Robinson. Stability theory for systems of inequalities: Part II. Differentiable nonlinear systems. *SIAM J. Num. Anal.*, 13:497–512, 1976. [11]

[Rockafellar, 1993] R.T. Rockafellar. Lagrange multipliers and optimality. *SIAM Rev.*, 35:183–238, 1993. [46]

[Rokne, 1979] John Rokne. A Note on the Bernstein Algorithm for Bounds for Interval Polynomial. *Computing*, 21:159–170, 1979. [123]

[Rose and Tarjan, 1978] D.J. Rose and R.E. Tarjan. Algorithmic aspects of vertex elimination on directed graphs. *SIAM Journal of Applied Mathematics*, 34(1):176–197, January 1978. [51]

[Rueher and Solnon, 1997] M. Rueher and C. Solnon. Concurrent cooperating solvers over reals. *Reliable Computing*, 3(3):325–333, 1997. [171]

[Rump, 1992] Siegfried M. Rump. On the Solution of Interval Linear Systems. *Computing*, 47:337–353, 1992. [158]

[Ryoo and Sahinidis, 1996] H.S. Ryoo and N.V. Sahinidis. A branch-and-reduce approach to global optimization. *J. Global Optim.*, 8:107–139, 1996. [68]

[Sabin and Freuder, 1994] D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In Alan Borning, editor, *Lecture notes in computer Science 874: Principles and Practice of constraint programming*. Springer Verlag, 1994. [178]

[Sahinidis, 1996] N.V. Sahinidis. Baron: A general purpose global optimization software package. *J. Global Optim.*, 8:201–205, 1996. [62, 68, 110]

[Sahni, 1974] S. Sahni. Computationally related problems. *SIAM J. Comp*, 3:262–279, 1974. [27]

[Sam-Haroud and Faltings, 1996] D. Sam-Haroud and B. Faltings. Consistency techniques for continuous constraints. *Constraints*, 1:85–118, 1996. [182, 184, 186, 187, 188]

[Sam, 1995] J. Sam. *Constraint consistency techniques for continuous domains*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, Switzerland, 1995. [182, 188]

[Samet, 1990] H. Samet. *The design and analysis of spatial data structures*. Addison-Wesley Publishing Company, 1990. [182]

[Saraswat, 1993] V. Saraswat. *Concurrent Constraint Programming*. MIT Press, Cambridge, London, 1993. [171]

[Saunders, 1994] M.A. Saunders. Recent developments to minos (minos5.4). Presented at XVISMP in Ann Arbor, 1994. [28]

[Schichl and Neumaier, 2001] H. Schichl and A. Neumaier. Tunneling and lagrangian aggregation for global optimization, 2001. in preparation. [92, 111]

[Schittkowski, 1983] K. Schittkowski. On the convergence of a sequential quadratic programming method with an augmented lagrangian line search function. *Math. Oper. and Stat. Series Opt.*, 14:197–216, 1983. [32]

[Semenov and Leshchenko, 1994] A. Semenov and A. Leshchenko. Interval and Symbolic Computations in the Unicalc Solver. In *Proceedings of INTERVAL'94, International Conference on Interval and Computer-Algebraic Methods in Science and Engineering*, pages 206–208, St-Petersburg, Russia, 1994. [127]

[Semenov *et al.*, 2000] A. Semenov, D. Petunin, and A. Kleymenov. GMACS: the general-purpose module architecture for building cooperative solvers. In *Proceedings of the ERCIM/CompulogNet Workshop on Constraint programming*, Padova, Italy, 2000. [172]

[Sherali and Shetty, 1980] H.D. Sherali and C.M. Shetty. *Optimization with disjunctive constraints.* Springer, Berlin, 1980. [100]

[Shimizu *et al.*, 1997] K. Shimizu, Y. Ishizuka, and J.F. Bard. *Nondifferentiable and Two-Level Programming.* Kluwer, Boston, 1997. [102]

[Silaghi *et al.*, 2001] M. Silaghi, D. Sam-Haroud, and B. Faltings. Search techniques for non-linear csps with inequalities. In *Proceedings of the 14th Canadian Conference on AI*, 2001. [188, 189]

[Smolka, 1995] G. Smolka. The OZ programming model. In Jan van Leeuwen, editor, *Computer Science Today*, number 1000 in Lecture Notes in Computer Science, pages 324–343, Berlin, 1995. Springer-Verlag. [171]

[Speelpenning, 1980] B. Speelpenning. *Compiling fast Partial Derivatives of Functions given by Algorithms.* PhD thesis, University of Illinois, Urbana-Champaign, Illinois, 1980. [48]

[Spellucci, 1993a] P. Spellucci. *Numerische Verfahren der nichtlinearen Optimierung.* Birkhäuser, 1993. [10, 20, 31]

[Spellucci, 1993b] P. Spellucci. Solving general convex qp problems via an exact quadratic lagrangian with bound constraints. Technical Report 1563, TUD Dept. Math., 1993. Revised version 7/94. [22]

[Spellucci, 1997] P. Spellucci. Numerical experiments with modern methods for large-scale qp problems. In Gritzmann, Horst, Sachs, and Tichatschke, editors, *Recent advances in optimization*, volume 452 of *Lecture Notes in Econ. and Math. Syst*, pages 313–335. Springer, 1997. [23]

[Stahl, 1995] Volker Stahl. *Interval Methods for Bounding the Range of Polynomials and Solving Systems of Nonlinear Equations.* PhD thesis, University of Linz, Austria, 1995. [123]

[Stillinger, 1985] F.H. Stillinger. Role of potential-energy scaling in the low-temperature relaxation behavior of amorphous materials. *Phys. Rev. B*, 32:3134–3141, 1985. [70]

[Tarski, 1951] A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, Berkeley, 1951. 2nd revised edition. [152, 160]

[Tawarmalani and Sahinidis, 2001] M. Tawarmalani and N.V. Sahinidis. Global optimization of mixed-integer nonlinear programs: a theoretical and computational study. *Mathematical Programming*, 2001. submitted. [62]

[Telerman and Ushakov, 1996] V. Telerman and D. Ushakov. Subdefinite models as a variety of constraint programing. In *Proceedings of the 8th IEEE International Conference on Tools with Artificial Intelligence*, Toulouse (France), November 1996. [170]

[Tomlin, 1970] J.A. Tomlin. Branch and bound methods for integer and nonconvex programming. In J. Abadie, editor, *Integer and Nonlinear Programming*, pages 437–450. American Elsevier Publishing Company, New York, 1970. [103]

[Törn and Žilinskas, 1989] A. Törn and A. Žilinskas. *Global Optimization*. Number 350 in Lecture Notes in Computer Science. Springer, Berlin, 1989. [76]

[Törn *et al.*, 1999] A. Törn, M. Ali, and S. Viitanen. Stochastic global optimization: problem classes and solution techniques. *J. Global Optim.*, 14:437–447, 1999. [57]

[Törn, 2000] A. Törn. Global optimization. http://www.abo.fi/~atorn/Globopt.html, 2000. [57, 70]

[Tsang, 1998] E. Tsang. No more partial and full looking ahead. *Artificial Intelligence*, 98(1-2), 1998. [178]

[Ulbrich and Ulbrich, 1996] M. Ulbrich and S. Ulbrich. Automatic differentiation: A structure-exploiting forward mode with almost optimal complexity for kantorovic trees. In H. Fischer, B. Riedmüller, and S. Schäffler, editors, *Applied Mathematics and Parallel Computing*, pages 327–357. Physica-Verlag, Heidelberg, 1996. [52, 53]

[Van Beek and Dechter, 1997] P. Van Beek and R. Dechter. On the minimality and global consistency of row-convex constraint networks. *Journal of the ACM*, 44:549–566, 1997. [185]

[Van Emden, 1997] Maarten H. Van Emden. Value Constraints in the CLP Scheme. *Constraints*, 2(2):163–183, 1997. [132]

[Van Emden, 1999] Maarten H. Van Emden. Algorithmic Power from Declarative Use of Redundant Constraints. *Constraints*, 4(4):363–381, 1999. [144, 158, 169]

[Van Hentenryck *et al.*, 1996] P. Van Hentenryck, V. Saraswat, A. Borning, A. Brodski, P. Codognet, R. Dechter, M. Dincbas, E. Freuder, M. Hermenegildo, J. Jaffar, S. Kasif, J.-L. Lassez, D. McAllester, Ken McAloon, A. Macworth, U. Montanari, W. Older, J.-F. Puget, R. Ramakrishnan, F. Rossi, G. Smolka, and R. Wachter. Strategic directions in constraint programming. *ACM Computing Surveys*, 28(4):701–726, 1996. 50th Anniversary Issue on Strategic Directions in Computer Research. [150]

[Van Hentenryck *et al.*, 1997a] P. Van Hentenryck, D. McAllester, and D. Kapur. Solving Polynomial Systems Using a Branch and Prune Approach. *SIAM Journal on Numerical Analysis*, 34(2):797–827, 1997. [124, 126, 137, 138, 142, 144, 145, 147, 158, 178]

[Van Hentenryck *et al.*, 1997b] P. Van Hentenryck, L. Michel, and F. Benhamou. Newton: constraint programming over non-linear constraints. *Sci. Program.*, 30:83–118, 1997. [85]

[Van Hentenryck *et al.*, 1997c] P. Van Hentenryck, L. Michel, and Y. Deville. *Numerica: A Modeling Language for Global Optimization.* MIT Press, Cambridge, MA, 1997. [60, 70, 83, 85, 88, 90, 96, 107, 110, 111, 115, 127, 169, 174, 178]

[Van Hentenryck, 1989] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, Cambridge, MA, 1989. [85]

[Van Hentenryck, 1997] P. Van Hentenryck. Numerica: a modeling language for global optimization. In *Proceedings of IJCAI'97*, 1997. [178]

[Van Iwaarden, 1986] R. Van Iwaarden. *An improved unconstrained global optimization algorithm.* PhD thesis, Univ. of Colorado at Denver, Denver, CO, May 1986. [98]

[Van Laarhoven and Aarts, 1987] P.J.M. Van Laarhoven and E.H.L. Aarts. *Simulated Annealing: Theory and Applications.* Kluwer, Dordrecht, 1987. [61, 70]

[Vanderbei and Shanno, 1997] R.J. Vanderbei and D.F. Shanno. An interior-point algorithm for nonconvex nonlinear programming. Technical Report SOR-97-21, Statistics and Operations Research, Princeton University, 1997. [41]

[Vanderbei, 1995] R.J. Vanderbei. Symmetric quasidefinite matrices. *SIAM J. Optim.*, 5:100–113, 1995. [26]

[Vavasis, 1990] S. Vavasis. Quadratic programming is in np. *Info. Proc. Lett.*, 36:73–77, 1990. [27]

[Vicente and Calamai, 1994] L.N. Vicente and P.H. Calamai. Bilevel and multilevel programming: a bibliography review. *J. Global Optim.*, 5:291–306, 1994. [103]

[Wallace *et al.*, 1997] Mark Wallace, Stefano Novello, and Joachim Schimpf. ECLiPSe : A Platform for Constraint Logic Programming. Technical report, IC-Parc, London, 1997. Available at the URL: http://www.icparc.ic.ac.uk/eclipse/reports/eclipse/eclipse.html. [127, 170]

[Walster *et al.*, 1985] G. Walster, E. Hansen, and S. Sengupta. Test results for a global optimization algorithm. In P.T. *et al.* Boggs, editor, *Numerical optimization 1984*, pages 272–287. SIAM, Philadelphia, 1985. [111, 112]

[Waltz, 1975a] David L. Waltz. Generating Semantic Descriptions from Drawings of Scenes with Shadows. In P. H. Winston, editor, *The Psychology of Computer Vision*. McGraw Hill, 1975. [114, 131]

[Waltz, 1975b] D.L. Waltz. Understanding line drawings of scenes with shadows. In P.W. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, 1975. [178]

[Wolsey, 1998] L.A. Wolsey. *Integer Programming*. Wiley, 1998. [59, 110]

[Wright, 1997] S.J. Wright. *Primal-Dual Interior-Point Methods*. SIAM, Philadelphia, 1997. [26, 60]

[Yagiura and Ibaraki, 2001] M. Yagiura and T. Ibaraki. On metaheuristic algorithms for combinatorial optimization problems. *Systems and Computers in Japan*, 2001. to appear. [70]

[Yakhno and Petrov, 1996] T. Yakhno and E. Petrov. LogicCalc: Integrating constraint programming and subdefinite models. In *Proceedings of Practical Application of Constraint Technology*, London (U.K.), April 1996. [170]

[Yamamura *et al.*, 1998] Kiyotaka Yamamura, Hitomi Kawata, and Ai Tokue. Interval Analysis using Linear Programming. *BIT*, 38:188–201, 1998. [158]

[Yamamura, 2000] Kiyotaka Yamamura. Finding all Solutions of Nonlinear Equations using Linear Combinations of Functions. *Reliable Computing*, 6:105–113, 2000. [159]

[Zhigljavsky, 1991] A.A. Zhigljavsky. *Theory of Global Random Search*. Kluwer, Dordrecht, 1991. [61]

[Zou *et al.*, 1993] X. Zou, I.M. Navon, M. Berger, K.H. Phua, T. Schlick, and F.X. Le Dimet. Numerical experience with limited-memory quasi-newton and truncated newton methods. *SIOPT*, 3:582–608, 1993. [17]