Mathematical Programming Computation manuscript No.
(will be inserted by the editor)

QPLIB: A Library of Quadratic Programming Instances

Fabio Furini · Emiliano Traversi · Pietro Belotti · Antonio Frangioni · Ambros Gleixner · Nick Gould · Leo Liberti · Andrea Lodi · Ruth Misener · Hans Mittelmann · Nick Sahinidis · Stefan Vigerske · Angelika Wiegele

Received: date / Accepted: date

Fabio Furini
LAMSADE, Université Paris Dauphine, 75775 Paris, France. E-mail: fabio.furini@dauphine.fr

Emiliano Traversi
LIPN, Université de Paris 13, 93430 Villetaneuse, France. E-mail: emiliano.traversi@lipn.fr

Pietro Belotti
Xpress-Optimizer team, FICO, Birmingham UK E-mail: pietrobelotti@fico.com

Antonio Frangioni
Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo 2, 56127 Pisa, Italy, E-mail: frangio@di.unipi.it

Ambros Gleixner
Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany. E-mail: gleixner@zib.de

Nick Gould
STFC-Rutherford Appleton Laboratory, Chilton, Oxfordshire OX11 0QX, England. E-mail: nick.gould@stfc.ac.uk

Leo Liberti
CNRS LIX, Ecole Polytechnique, 91128 Palaiseau, France. E-mail: liberti@lix.polytechnique.fr

Andrea Lodi
Ecole Polytechnique de Montral, Canada. E-mail: andrea.lodi@polymtl.ca

Ruth Misener
Department of Computing,Imperial College London, UK. E-mail: r.misener@imperial.ac.uk

Hans Mittelmann
School of Mathematical and Statistical Sciences, Arizona State University, Tempe, AZ 85287-1804, U.S.A. E-mail: mittelmann@asu.edu

Nick Sahinidis
Chemical Engineering, Carnegie Mellon University, U.S.A. E-mail: sahinidis@cmu.edu

Stefan Vigerske
GAMS Software GmbH, P.O. Box 40 59,50216 Frechen, Germany. E-mail: stefan@gams.com

Angelika Wiegele

Abstract This paper describes a new instance library for Quadratic Programming (QP). QP is a very "varied" class, comprising sub-classes of problems ranging from trivial to undecidable. Solution methods for QP are very diverse, ranging from entirely combinatorial ones to completely continuous ones, including many for which both aspects are fundamental. Selecting a set of instances of QP that is at the same time not overwhelmingly numerous and sufficiently challenging for the many different interested communities is therefore important. We propose a simple taxonomy for QP instances that leads to a systematic problem selection mechanism. We then briefly survey the field of QP, giving an overview of theory, methods and solvers. Finally we describe how the library was put together, and detail its final contents.

Keywords Instance Library, Quadratic Programming

Mathematics Subject Classification (2000) 90C06 · 90C25

1. Introduction

Quadratic Programming (QP) problems, where either the objective function [90], or the constraints [91], or both contain (at most) expressions in the variables of degree 2, include a surprisingly diverse set of rather different instances. This is not surprising, given the vast scope of practical applications of these problems, and of solution methods designed to solve them [41]. According to the fine details of the formulation, solving a QP may require employing either fundamentally combinatorial techniques, or ideas rooted in nonlinear optimization principles, or a mix of the two. In this sense, QP is likely one of the classes of problems where the collaboration between the communities interested in combinatorial and nonlinear optimization is more necessary, and potentially fruitful.

However, this diversity also implies that QP means very different things to different researchers. This is illustrated by the simple fact that the class of problems that we simply refer to here as "QP" is called in different ways, among which Mixed-Integer Quadratically Constrained Quadratic Problems (MI-QCQP). It is perhaps therefore not surprising that, unlike for "simpler" problems classes [52], so far there has never been a single library containing all different kinds of instances of QP. Several libraries devoted to special cases of QP are indeed available; however, each of them is either focussed on one application (a specific problem that can be modeled as QP), or on QPs with specific structural properties that make them suitable to be solved with some given class of algorithmic approaches. To the best of our knowledge, collecting a set of QP instances that is at the same time not overwhelmingly numerous and significant for the many different interested communities has not been attempted, yet. This work constitutes a first step in this direction.

Institut für Mathematik, Alpen-Adria-Universität Klagenfurt, 9020 Klagenfurt am Wörthersee, Austria. E-mail: angelika.wiegele@aau.at

In this paper we report the steps that have been done to collect a (hopefully) significant library of QP instances, filtering the large set of available (or specifically provided) instances in order to end up with a manageable set that still contains a meaningful sample of all possible QP types. A particularly thorny issue in this process is how to select instances that are "interesting". Our choice has been to take this to mean "challenging for a significant set of solution methods". Our filtering process has then been in part based on the idea that if a significant fraction of the solvers that can solve a QP instance do so in a "short" time, then the instance is not challenging enough to be included in the library. Yet, if very few (maybe one) solvers can solve it very efficiently by exploiting some specific structure, but most other approaches cannot, then the instance can still be deemed "interesting". Putting this approach in practice requires a nontrivial number of technical steps and decisions that are detailed in the paper. We hope that our work can provide useful guidelines for other interested researchers.

A consequence of our focus is that this paper is *not* concerned about the different performance of the very diverse QP solvers; we will *not* report any data comparing them. The only reason why solvers are used (and, therefore, described) in this context is to ensure that the instances of the library are nontrivial at least for a significant fraction of the current solution methods; providing guidance about which solver is most suited to some specific class of QPs is entirely outside the scope of our work.

1.1 Motivation

Optimization problems with quadratic constraints and/or objective function (QP) have been the subject of a considerable amount of research over the last 60 years. At least some of the rationale for this interest is likely due to the fact that they are the "least-nonlinear nonlinear problems". Hence, in particular for the convex case, tools and techniques that have been honed during decades of research for Linear Programming (LP), typically with integrality constraints (MILP), can often be extended to the quadratic case with at least less effort than what would be required for tackling general Non Linear Programming (NLP) problems, without or with integrality constraints (MINLP). This has indeed happened over and over again with different algorithmic techniques, such as Interior Point methods, active-set methods (of which the simplex method is a prototypical example), enumeration methods, cut-generation techniques, reformulation techniques, and many others (**citations?**). Similarly, nonconvex continuous QP are perhaps the "simplest" class of problems that require features like spatial enumeration techniques to be solved. Hence, they are both the natural basis for the development of general techniques for nonconvex NLP, and a very specific class so that specialized approaches can be developed [29, 18].

On the other hand, (MI)QP is, in some sense, a considerably more expressive class than (MI)LP. Quadratic expressions are found, either naturally

or after appropriate reformulations, in very many optimization problems [53]. Table 1 provides a certainly non-exhaustive collection of applications that lead to formulations with either quadratic constraints, ot quadratic objective function, or both. Also, in general any continuous function can be approximated with arbitrary accuracy (over a compact set) by a polynomial of arbitrary degree; in turn, every polynomial can be broken down to a system of quadratic expressions. Hence, (MI)QP is, in some sense, roughly as expressive as the whole of (MI)NLP. Of course this is, in principle, true for (MI)LP as well, but at the cost of much larger and much more complex formulations (**citations?**). Hence, for many applications QP may represent the "sweet spot" between the effectiveness, but lower expressive power, of (MI)LP and the higher expressive power, but much higher computational cost, of (MI)NLP.

Table 1: Application Domains of (MI)QP

Problem	Discrete	Contributions
Classical Problems that can be formulated as MIQP		
Quadratic Assignment Problem (QAP)‡	✓	[4,62]
Max-Cut	✓	[78]
Maximum clique‡	✓	[15]
Computational chemistry & Molecular biology		
Zeolites		[42]
Computational geometry		
Layout design	✓	[3,20,27]
Maximizing polygon dimensions		[5–9]
Packing circles‡		[45,83]
Nesting polygons		[49,77]
Cutting ellipses		[50]
Finance: Portfolio optimization	✓	[25,48,61,63,73,79, 31,34–36]
Process Networks		
Crude oil scheduling	✓	[56–58,68,69]
Data reconciliation	✓	[80]
Multi-commodity flow	✓	[84]
Quadratic network design	✓	[31,37]
Multi-period blending	✓	[54,55]
Natural gas networks	✓	[43,59,60]

‡Applications with many manuscripts cite reviews and recent works

continued

Table 1 (Application Domains of (MI)QP) continued

Problem	Discrete	Contributions
Pooling‡	✓	[2,21,24,30,66,67, 72,74,81,88]
Open-pit mine scheduling	✓	[12]
Reverse osmosis	✓	[82]
Supply chain	✓	[71]
Water networks‡	✓	[1,10,16,22,38,40, 47,51,75,87]
Robotics		
TSP⋆ with Neighborhoods	✓	[39]
Telecommunications		
Delay-constrained routing	✓	[32,33]
Energy		
Unit-commitment problem	✓	[85]

‡Applications with many manuscripts cite reviews and recent works. ⋆TSP ≡ Traveling Salesman Problem

The structure of this paper is the following. In § 2 we review the basic notions about QP. In particular, § 2.1 sets out the notation, § 2.2 proposes a—to the best of our knowledge, new—taxonomy of QP that helps us in discussing the (very) different classes of QPs, § 2.3 very briefly reviews the solution methods for QP upon which the solvers we have employed are based, and § 2.4 describes the solvers. Then, § 3 describes the process used to obtain the library and its results; the software tools that we have used, and that are freely released together with the library, are discussed in § 4. Some conclusions are drawn in § 5, after which in the Appendix a complete description of all the instances of the library is provided.

2. Quadratic Programming in a nutshell

2.1 Notation

In mathematical optimization, a Quadratic Program (QP) is an optimization problem in which either the objective function, or some of the constraints, or

both, are quadratic functions. More specifically, the problem has the form

$$\begin{aligned} \min\ & x^\top Q^0 x + b^0 x & \\ & x^\top Q^i x + b^i x \le c^i & i \in \mathcal{M} \\ & l_j \le x_j \le u_j & j \in \mathcal{N} \\ & x_j \in \mathbb{Z} & j \in \mathcal{Z} \end{aligned}$$

where:

- $\mathcal{N} = \{1, \ldots, n\}$ is the set of (indices) of variables, and $\mathcal{M} = \{1, \ldots, m\}$ is the set of (indices) of constraints;
- $x = [x_j]_{j=1}^n \in \mathbb{R}^n$ is a finite vector of real variables;
- Q^i for $i \in \{0\} \cup \mathcal{M}$ are symmetric $n \times n$ real matrices: because one is always only interested in the value of quadratic functions of the type $x^\top Q^i x$, symmetry can be assumed without loss of generality by just replacing both Q^i_{hk} and Q^i_{kh} with their average $(Q^i_{hk} + Q^i_{kh})/2$;
- b^i and c^i for $i \in \{0\} \cup \mathcal{M}$ are, respectively, real n-vectors and real constants;
- $-\infty \le l_j \le u_j \le \infty$ are the (extended) real upper and lower bounds on each variable x_j for $j \in \mathcal{N}$;
- $\mathcal{M} = \mathcal{Q} \cup \mathcal{L}$ where $Q^i = 0$ for all $i \in \mathcal{L}$ (i.e., these are the linear constraints, as opposed to the truly quadratic ones);
- the variables in $\mathcal{Z} \subseteq \mathcal{M}$ are restricted to only attain integer values.

Due to the presence of integral requirements on the variables, this class of problems is often referred to as Mixed-Integer Quadratic Program (MIQP). It will be sometimes useful to refer to the (sub)set $\mathcal{B} = \{\, i \in \mathcal{Z} : l_j = 0, u_j = 1 \,\} \subseteq \mathcal{Z}$ of the binary variables, and to $\mathcal{R} = \mathcal{N} \setminus \mathcal{Z}$ as the set of continuous ones. Similarly, it will be sometimes useful to distinguish the (sub)set $\mathcal{X} = \{\, j : l_j > -\infty \vee u_j < \infty \,\}$ of the box-constrained variables from $\mathcal{U} = \mathcal{N} \setminus \mathcal{X}$ of the unconstrained ones (in the sense that finite bounds are not explicitly provided in the data of the problem, although they may be implied by the other constraints).

The relative flexibility offered by quadratic functions, as opposed e.g. to linear ones, allows several reformulation techniques to be applicable to this family of problems in order to emphasize different properties of the various components. Some of these reformulation techniques will be commented later on; here we remark that, for instance, integrality requirements, in particular in the form of binary variables could always be "hidden" by introducing (non convex) quadratic constraints utilizing the celebrated relationship $x_j \in \{0, 1\} \iff x_j^2 = x_j$. Therefore, when discussing these problems an effort has to be made to distinguish between features that come from the original model, and those that can be introduced by reformulation techniques in order to extract (and algorithmically exploit) specific properties.

2.2 Classification

Despite the apparent simplicity of the definition given in § 2.1, Quadratic Programming instances can be of several rather different "types" in practice, depending on the fine details of the data. In particular, many algorithmic approaches can only be applied to QP when the data of the problem has specific properties. A taxonomy of QP instances should therefore strive to identify the set of properties that an instance should have in order to apply the most relevant computational methods. However, the sheer number of different existing approaches, and the fact that new ones are proposed, makes it hard to provide a taxonomy that is both simple and covers all possible special cases. This is why, in this paper, we propose an approach that aims at finding a good balance between simplicity and coverage of the main families of computational methods.

2.2.1 Classification

Our taxonomy is based on a three-fields code of the form "OVC", where O indicates the objective function, V the variables, and C the constraints of the problem. The fields can be given the following values:

- objective function: (L)inear, (D)iagonal convex quadratic, (C)onvex quadratic, nonconvex (Q)uadratic;
- variables: (C)ontinuous only, (B)inary only, (M)ixed binary and continuous, (I)nteger only, (G)eneral (all three types);
- constraints: (N)one, (B)ox, (L)inear, (D)iagonal convex quadratic, (C)onvex quadratic, nonconvex (Q)ua-dratic.

The wildcard "*" will be used to indicate any possible choice, and lists of the form "{X, Y, Z}" will indicate that the value of the given field can freely attain any of the specified values.

The ordering of the values in the previous lists is not irrelevant; in general, problems become "harder" when going from left to right. More specifically, for the O and C fields the order is that of *strict* containment between problem classes: for instance, Linear objective functions are strictly contained in Diagonal convex quadratic ones (by just allowing the diagonal elements to be all-zero), which are strictly contained into general Convex quadratic ones (by allowing the off-diagonal elements to be all-zero), which in turn are strictly contained into general nonconvex Quadratic ones (by allowing any symmetric Q^0, hence possibly positive semidefinite ones as well). The only field for which the containment relationship is not a total order is V, for which only the partial orderings

$$\mathrm{C} \subset \mathrm{M} \subset \mathrm{G} \quad , \quad \mathrm{B} \subset \mathrm{M} \subset \mathrm{G} \quad , \quad \mathrm{B} \subset \mathrm{I} \subset \mathrm{G}$$

hold. In the following discussion we will repeatedly exploit this by assuming that, unless otherwise mentioned, when a method can be applied to a given problem, it can be applied as well to all simpler problems where the value of each field is arbitrarily replaced with a value denoting a less-general class.

2.2.2 Examples and reformulations

Ambros: maybe switch this with the next section? the next one contains more familiar material to the reader and might be an easier first introduction to the notation.

We will now provide a first general discussion about the different problem classes that our proposed taxonomy defines. Some of them are actually "too simple" to make sense in our context. For instance, D*B problems have only diagonal quadratic (hence separable) objective function and bound constraints; as such, they read

$$\min \left\{ \sum_{j\in\mathcal{N}} \left(Q^0_j x_j^2 + b^0_j x_j \right) \ : \ l_j \le x_j \le u_j \quad j \in \mathcal{N} \ , \ x_j \in \mathbb{Z} \quad j \in \mathcal{Z} \right\} .$$

Hence, their solution only requires the independent minimization of a convex quadratic univariate function in each single variable x_j over a box constraint and possibly integrality requirements, which can be attained trivially in $O(1)$ (per variable) by closed-form formulæ, projection and rounding arguments. *A fortiori*, the even simpler cases L*B, D*N and L*N (the latter obviously unbounded unless $b^0 = 0$) will not be discussed here. Similarly, CCN are immediately solved by linear algebra techniques, and therefore are of no interest in this context. On the other end of the spectrum, in general QP is a hard problem. Actually, LIQ—linear objective function and quadratic constraints in integer variables with no finite bounds, i.e.

$$\min \left\{ b^0 x \ : \ x^\top Q^i x + b^i x \le c^i \quad i \in \mathcal{M} \ , \ x_j \in \mathbb{Z} \quad j \in \mathcal{N} \right\} ,$$

is not only $\mathcal{NP}$-hard, but downright undecidable [46]. Hence so are the "harder" {C,Q}IQ.

It is important to note that the relationships between the different classes can be somehow blurred because reformulation techniques may allow to move one instance from one class to the other. The already recalled fact that $x^2 = x \iff x \in \{0,1\}$, for instance, says that *M*—instances with only binary and continuous variables—can be recast as *CQ: nonconvex quadratic constraints can always take the place of binary variables. Actually, this is also true for *G* as long as $\mathcal{U} = \emptyset$, as bounded general integer variables can be represented by binary ones.

Another relevant reformulation trick concerns the fact that, as soon as quadratic constraints are allowed, then a linear objective function can be assumed w.l.o.g.. Indeed, any Q** (C*C) problem can always be rewritten as

$$\begin{aligned} \min \ & x^0 & \\ & x^\top Q^0 x + b^0 x \le x^0 & \\ & x^\top Q^i x + b^i x \le c^i & i \in \mathcal{M} \\ & l_j \le x_j \le u_j & j \in \mathcal{N} \\ & x_j \in \mathbb{Z} & j \in \mathcal{Z} \end{aligned}$$

i.e., a L*Q (L*C). Hence, it is clear that quadratic constraints are, in a well-defined sense, the most general situation (cf. also the result above about hardness of LIQ).

When a Q^i is positive semidefinite (PSD), i.e., the corresponding constraint/objective function is convex, general quadratic constraints are in fact equivalent to diagonal ones. In fact, every PSD matrix can be factorized as $Q^i = L^i(L^i)^\top$, e.g. by the (incomplete) Cholesky factorization, $f^i(x) = x^\top Q^i x = \sum_{j\in\mathcal{N}} z_j^2 z$ where $z = x^\top L^i$. Hence, one could think that D** problems need not be distinguished from C** ones; however, this is true only for "complicated" constraints, but not for "simple" ones, because the above reformulation technique introduces linear constraints. Indeed, while C*L (and, a fortiori, C*{C,Q}) can always be brought to D*L (D*{C,Q}), using the same technique C*B becomes D*L, which is significantly different from D*B. In practice, a diagonal convex objective function under linear constraints is found in many applications (**citations?**), so that D*L still makes sense to distinguish the case where the objective function is "naturally" separable from that where separability is artificially introduced, although this is in theory always possible.

2.2.3 QP classes

The proposed taxonomy can then be used to describe the main classes of QP according to the type of algorithms that can be applied for their solution:

- *Linear Programs* LCL and *Mixed-Integer Linear Programs* LGL have been subject of an enormous amount of research and have their well-established instance libraries [52], so they won't be explicitly addressed here.
- *Convex Continuous Quadratic Programs* CCC can be solved in polynomial time by Interior-Point techniques [92]; the simpler CCL can also be solved by means of "simplex-like" techniques, usually referred to as active-set methods [28]. Actually, a slightly larger class of problems can be solved with Interior-Point methods: those that can be represented by Second-Order Cone Programs. When written as QPs the corresponding Q^i may not be positive semidefinite, but still the problems can be efficiently solved. Of course, these problems can still require considerable time, like LCL, when the size of the instance grows. In this sense, like in the linear case, a significant divide is from solvers that need all the data of QP to work, and those that are "matrix-free", i.e., only require the solution of simple operations (typically, matrix-vector products) with the data of the problem to work (**citations?**). While in our library we have never exploited such a characteristic, which is not suitable to the use of standard modeling tools, this may be relevant for the solution of very-large-scale CIC.
- *Nonconvex Continuous Quadratic Programs* QCQ are instead in general $\mathcal{NP}$-hard, even if the constraints are very specific (QCB) and only one eigenvalue of Q^0 is negative [44]. They therefore require enumerative techniques like the spatial Branch&Bound (**citations?**), to be solved to optimality. Of course, local approaches are available that are able to efficiently provide saddle points (hopefully, local optima) of the CQC, but providing global guarantees about the quality of the obtained solutions is challeng-

ing. In our library we have specifically focussed on *exact* solution of the instances.

- *Convex Integer Quadratic Programs* CGC are in general $\mathcal{NP}$-hard, and therefore require enumerative techniques to be solved. However, convexity of the objective function and constraints implies that efficient techniques (see CCC) can be used at least to solve the continuous relaxation. The general view is that CGC are not, all other things being equal, substantially more difficult than LGL to solve, especially if the objective function and/or the constraints have specific properties (e.g., DGL, CGL). Often integer variables are in fact binary ones, so several CCC models are C{B,M}C ones. In practice binary variables are considered to lead to somewhat easier problems than general integer ones (cf. the cited result about hardness of unbounded integer quadratic programs), and several algorithmic techniques have been specifically developed for this special case. However, the general approaches for CBC are basically the same as for CGC, so there is seldom the need to distinguish between the two classes as far as solvability is concerned, although matters can be different regarding actual solution cost. Programs with only binary variables CBC can be easier than mixed-binary or integer ones C{M,I}C because some techniques are specifically known for the binary-only case, cf. the next point (**citations?**). Absence of continuous variables, even in the presence of integer ones CIC, can also lead to specific techniques (**citations??**).
- *Nonconvex Binary Quadratic Programs* QB{B, N, L} obviously are $\mathcal{NP}$-hard. However, the special nature of binary variables combined with quadratic forms allows for quite specific techniques to be developed, among which is the reformulation of the problem as a LBL. Also, many well-known combinatorial problems can be naturally reformulated as problems of this class, and therefore a considerable number of results have been obtained by exploiting specific properties of the set of constraints ([78], **some citations? some specific problem to mention apart from Max-Cut?**).
- *Nonconvex Integer Quadratic Programs* QGQ is the most general, and therefore is the most difficul, class. Due to the lack of convexity even when integrality requirements are removed, solution methods must typically combine several algorithmic ideas, such as enumeration (distinguishing the role of integral variables from that of continuous ones involved into nonconvex terms) and techniques (e.g., outer approximation, SDP relaxation, ...) that allow to efficiently compute bounds. As in the convex case, QBQ, QMQ, and QIQ can benefit from more specific properties of the variables ([17,26], **citations?**).

This description is purposely coarse; each of these classes can be subdivided into several others on the grounds of more detailed information about structures present in their constraints/objective function. These can have a significant algorithmic impact, and therefore can be of interest to researchers. Common structures are, e.g., network flow or knapsack-type linear constraints, semi-continuous variables, or the fact that a nonconvex quadratic objective

function/constraint can be reformulated as a second-order cone (hence, convex) one. It would be rather hard to collect a comprehensive list of all types of structures that may be of interest to any individual researcher, since these are as varied as the different possible approaches for specialized sub-classes of QP. For this reason we do not attempt such a more refined classification, and limit ourselves to the coarser one described in this paragraph.

2.3 Solution Methods

In this section we provide a quick overview of existing solution methods for QP, restricting ourselves to these implemented by the set of solvers considered in this paper (Table 2). For each approach we briefly describe the main algorithmic ideas and point out the formulation they address according to the classification set out in Sect. 2.2. We remark that many solvers implement more than one algorithm, among which the user can choose at runtime. Moreover, algorithms are typically implemented by different solvers in different ways, so that the same conceptual algorithm can sometimes yield wildly different results or performance measures on the same instances.

The methods implemented by the solvers in Table 2 can, following [70], be broadly organized in three categories: *incomplete*, *asymptotically complete*, and *complete*. Incomplete methods are only able to identify solutions, often locally optimal according to a suitable notion, and may even fail to find one even when one exists; in particular, they are typically not able to determine that an instance is empty. Asymptotically complete methods can find a globally optimal solution with probability one in infinite time, but again they cannot prove that a given instance is infeasible. Complete methods find an approximate globally optimal solution to within a prescribed optimality tolerance within finite time, or prove that none such exists (but see Sect. 2.3.3 below); they are often referred to as *exact* methods in the computational optimization community. According to [70], *rigorous* methods also exist which find global within given tolerances even in the presence of rounding errors, except in "near-degenerate cases". Since it is debatable whether any of the solver we are using can be classified as rigorous, we have elected to limit ourselves to declaring solvers complete.

Incomplete methods are usually realized as local search algorithms, asymptotically complete methods are usually realized by meta-heuristic methods such as multi-start or simulated annealing, and complete methods for $\mathcal{NP}$-hard problems such as QP are typically realized as implicit exhaustive exploration algorithms. However, these three categories may exhibit some overlap. For example, any deterministic method for solving QCQ locally is incomplete in general, but becomes complete for CCC, since any local optimum of a convex QP is also global. Therefore, when we state that a given algorithm is incomplete or (asymptotically) complete we mean that it is so the largest problem class that the solver naturally targets, although it may be complete on specific sub-classes. For example, SnOpt naturally targets continuous nonconvex

NLPs and it is incomplete on this class, and therefore on QCQ, but becomes complete for CCC. In general, all complete methods for a problem P must be complete for any problem $Q \subseteq P$, while a complete method for P might be incomplete for $R \supset P$.

2.3.1 Incomplete methods

Local search methods typically require as an input a solution x' and attempt to improve it—either towards feasibility, or towards optimality, or both—using only information that is available in a neighborhood of x'. In general, local search methods are incomplete. As remarked above, however, local search methods deployed on a convex problem behave like complete methods, possibly via devices that that allow one to find a feasible starting solution if there is one (like what was used to be called "phase -1" in the simplex method).

Most local search methods for *C* are iterative in nature, and need a feasible starting point x^0 as input. The $(k+1)$-st iterate is obtained as $x^{k+1} = x^k + \alpha_k d^k$, where α_k (a scalar) is the *step length*, and d^k (a vector) is the *search direction*, e.g. belonging to a tangent manifold and having a negative directional derivative at x^k, in order to improving optimality while reducing infeasibility. Alternatively, $x^{k+1} = x^k + d^k(\alpha_k)$ where $d^k(\cdot)$ is an arc satisfying $d^k(0) = 0$. Local search methods for *I* are also iterative, but since defining a useful notion of descent direction is harder in presence of integer variables, d^k and α_k are usually computed in different ways, depending on the application at hand.

The solvers in Table 2 which implement incomplete methods are ConOpt, IpOpt, Minos, SnOpt and KNitro. The former four solvers implement local search algorithms for continuous nonconvex NLPs (a problem class containing QCQ). Note that traditionally KNitro used to be a local (nonconvex) NLP solver. All of these solvers implement essentially three types of local search methods for NLP:

- active set methods (ConOpt, Minos, SnOpt) [28];
- interior point methods (IpOpt, KNitro) [92];
- projected gradient methods [19,23].

Active set and interior point methods have been defined for *C* but also for general (continuous) NLPs. In fact, all of the solvers named above target this larger problem class.

Active set methods. At each iteration k the algorithm forms the *active set* $\mathcal{A}^k$ containing the (indices of the) *active constraints*, i.e., all of the equality constraints as well as the inequality constraints that are satisfied at equality at the current iterate x^k. A subproblem, consisting of a minimization of a certain auxiliary objective function and including only active constraints, is then solved to identify a good search direction d^k. An appropriate step length α_k is then found by checking for the inactive constraints (since these must be satisfied too). If at x^{k+1} some of the previously inactive constraints have

become active, or vice versa, $\mathcal{A}^k$ is updated accordingly. This general scheme works because the step is chosen so that the objective decreases, and the active set cannot repeat.

Interior point methods. These approaches can be seen as recasting the original constrained problem QP as a parametrized family of unconstrained ones QP_μ, where the constraints are moved in the objective function via a *barrier term*—that goes to $+\infty$ as the boundary of the feasible region is approached—weighted with the *barrier parameter* $\mu \in (0, \infty)$. In the convex case, the *central path*—the continuous line formed from the optimal solutions of QP_μ for all varying μ, typically unique e.g. if the classical barrier based on the logarithmic function is employed—leads to a (central) optimal solution of QP when $\mu \to 0$. Starting from an appropriately constructed "central" point (close to the solution of QP_μ for "large" μ), these algorithms strive to follow the central path by performing $O(1)$ Newton steps before (substantially) reducing μ. The algorithms can be shown to converge in a small number of iterations, each of which can be costly due to the need of solving an appropriately modified version of the KKT conditions for QP ("slackened" with μ), a (possibly) large-scale linear system. Actually, nowadays the algorithm is most often implemented in the primal-dual version, where the nonlinear KKT system is iteratively solved with Newton-like iterations; this has the extra advantage of allowing to remove the need of a feasible (central) starting point.

Projected gradient methods. These approaches are similar to active-set methods in that at each iteration they consider the gradient projected onto the set of active constraints, in order to (try to) guarantee feasibility of the next iterate. They differ from active-set methods in that the set of active constraints can change dramatically on each iteration, and they have stronger convergence properties, more akin to those of interior-point methods.

2.3.2 Asymptotically complete methods

Asymptotically complete methods do not usually require a starting point, and, if given sufficient time (infinite in the worst case) will identify a globally optimal solution with probability one. Most often, these methods are meta-heuristics, involving an element of random choice, which exploit a given (heuristic) local search procedure.

The solvers in Table 2 which implement asymptotically complete methods are OQNLP, MSNLP and certain sub-solvers of LGO. Specifically, we consider the following methods:

- global adaptive random search (LGO_GARS);
- multi-start (LGO_MS, MSNLP, OQNLP); specifically, the former two apply to QCQ whereas the latter to QGQ.

Global Adaptive Random Search. This is a modification of an algorithm called *pure random search*, which consists in sampling a random point x' from a given compact set known to contain a global optimum, and then sampling a new candidate solution y in a neighborhood of x', setting $x' \leftarrow y$ if y improves x', and repeating as long as a termination condition is not satisfied. The adaptivity stems from changing the distribution for sampling y at run-time, depending on the quality of the solutions identified by the method. Since this method only depends on sampling and function evaluation, it is usually fast. In the LGO_GARS solver, it provides a useful starting point for a subsequent local search procedure. Asymptotic global convergence is attained by restarting the random search from different initial points x'.

Multi-start. Multi-start methods define a loop around a given local search procedure so that it starts from many different starting points, perform local search, and record the best optimum found so far as they explore the search space randomly. For example, any of the methods described in Sect. 2.3.1 can be embedded in a multi-start framework as follows:

1. initialize a "best solution so far" x^*
2. sample a starting point x' uniformly at random from a given compact set known to contain a global optimum;
3. run a local search method from x' to yield an improved (feasible) point x
4. if x improves on x^* with respect to the objective function value, replace x^* with x
5. repeat from Step 2 until a given termination condition is satisfied.

The method is asymptotically complete if the termination condition in Step 2 is a certificate of global optimality for x^*, which is usually hard to obtain. However, typically some bound on the total CPU time, or number of function evaluations, or any other criteria that makes sense for the application at hand, is needed, which renders the method incomplete.

In general, the applicability of meta-heuristics to a given problem depends on whether the local search they utilize addresses that problem or not. **Antonio: the concept of "addresses that problem" is not very clear to me, this sentence may benefit from rephrasing (or the paragraph from deleting if it's not deemed to be utterly necessary).** Depending on the local search employed, Multi-start methods can address MINLP of the most general class.

2.3.3 Complete methods

Complete methods are often referred to as *exact* in a large part of the mathematical optimization community. This nomenclature has to be used with care, as it implicitly makes assumptions on the underlying computational model that may not be acceptable in all cases. To see that, consider that, as already mentioned, QPs (more precisely, LIQ) are generally undecidable [46];

and yet, there exists a general decision method for deciding feasibility of systems of polynomial equations and inequalities [86], including the solution of LCQ with zero objective function. This apparent contradiction is due to the fact that the two statements refer to different computational models: the former is based on the Turing Machine (TM), whereas the latter is based on the Real RAM (RRAM) machine [14]. Due to the potentially infinite nature of exact real arithmetic computations, exact computations on the RRAM necessarily end up being approximate on the TM. Analogously, a complete method may reasonably be called "exact" on a RRAM; however, the computers we use in practice are more akin to TMs than RRAMs, and therefore calling *exact* a solver that employs floating point computations is, technically speaking, stretching the meaning of the word. However, because the term is well understood in the computational optimization community, in the following we shall loosen the distinction between complete and exact methods, with either properties intended to mean "complete" in the sense of [70].

Branch-and-Bound. Nearly all of the complete solvers in Table 2 that address $\mathcal{NP}$-hard problems (i.e. those in QGQ$\setminus$CCC) are based on Branch-and-Bound (BB). This is an implicit but exhaustive search process based on exploring a *branching tree* of the problem, where each node in the tree represents a subset of the feasible region. Guaranteed lower and upper bounds to the objective function value relative to nodes are computed in various ways. Nodes are discarded when: (a) they can be shown to be empty; (b) their bound in the optimization direction is worse than an opposite global bound; (c) a global optimum limited to the node can be found (this happens when the two bounds are closer than a given ε tolerance); (d) they are selected for branching, which means expanding the tree constructing at least two new nodes, children of the current one. Branching takes place by identifying one or more branching directions, which are usually a coordinate axes, and one or more branching point per direction, in various common sense fashions. The algorithm is driven by a queue of active nodes, usually endowed with a priority to select the most promising node from which to continue exploration of the tree (such as "most promising bound"); the BB algorithm terminates when the queue is empty.

Typically, bounds in the optimization direction are computed by means of convex relaxations [64,11], which replace nonconvex terms $t(x)$ with linearization variables $\hat{t}$, and then replace the corresponding defining constraints $\hat{t} = t(x)$ by means of lower and upper (respectively, convex and concave) bounding functions $\hat{t} \geq \underline{t}(x)$ and $\hat{t} \leq \bar{t}(x)$. This is actually where finite (and tight) bounds on the variables are crucial, which differentiate also in practice the bounded case from the unbounded one. Different strategies are used when the nonconvexities are only quadratic [65,13].

When the BB algorithm is allowed to select coordinate directions corresponding to continuous variables, it is called *spatial* BB (sBB). Branching on continuous (rather than integer or binary) variables becomes necessary in the presence of nonconvex nonlinearities, as it happens e.g. in QCQ, since the

quality of the bounds improves as the feasible set in the current node gets smaller.

BB algorithms are exponential time in the worst case, and their exponential behavior unfortunately often shows up in practice. They can also be used heuristically (forsaking their completeness guarantee) by either terminating them early, or by using non-guaranteed bounds.

The solvers in Table 2 BB type methods are:

- Antigone, Baron, Couenne, Lindo, LindoGlobal, SCIP, lgo_bb, which implement complete BB algorithms for QGQ;
- CPLEX, which implements a complete BB algorithm for QGL;
- Knitro_bb, BonMin, sBB, Xpress, Gurobi, and Mosek which implement complete BB algorithms for CGC.

We remark that the latter category can be used as incomplete solvers for QGQ. We also remark that CPLEX can currently only target problems with linear constraints when the objective function is nonconvex [13].

Cutting plane approaches . The two remaining solvers in Table 2 are AlphaECP [89] and DICOpt [76], which are complete cutting plane methods based on different principles. Characteristic of both solvers is that they need to employ a complete method for solving MILP (LIL) sub-problems at each iteration; in turn, this is typically based on BB, which is therefore a crucial technique also for this class of approaches. Additionally, incomplete methods can be used to provide local solutions. Other solvers, like BonMin, also offer this kind of approach among their algorithmic options.

2.4 Solvers

Ambros: merge with Section 2.3.3 or put list of solvers in Section 2.3.3 here

We now provide a succinct list of the solvers we have tested, using the approaches described in §2.3. In Table 2, we mark with "I" a pair (solver, problem) if the solver accepts the problem in input but it is an incomplete solver for the problem, with "A" if it is asymptotically complete, with "C" if it is complete, and leave it blank if the solver won't accept the problem in input.

The table has to be checked, as I've extrapolated from the text but I'm not 100% sure. Also, "?"s have to be removed.

3. Library Construction

In this section we present all the steps performed in order to build the new library. In §3.1, we describe the set of gathered instances. In §3.2 we present the features used to classify the instances and we discuss the issues concerning the format of the instances. Finally, in §3.3, we describe the selection process used to filter the instances and we graphically present the main features of the selected instances.

	CGL	QGL	CGC	QGQ	CCC	QCQ
Antigone	C	C	C	C	C	C
Baron	C	C	C	C	C	C
Couenne	C	C	C	C	C	C
KNitro	C	I	C	I	C	A
lindo api	C	C	C	C	C	C
SCIP	C	C	C	C	C	C
oqnlp	A	A	A	A	C	A
AlphaECP	C	I	C	I	C	I
BonMin	C	I	C	I	C	I
DICOpt	C	I	C	I	C	I
sBB	C	I	C	I	C	I
ConOpt					C	I
IpOpt					C	I
lgo					C?	A
Minos					C	I
msnlp					C	A
SnOpt					C	I
XPress	C		C		C	?
Gurobi	C		C		C	?
CPLEX	C	C	C		C	I?
mosek	C		C		C	

Table 2 Families of QP problems that can be tackled each solver

3.1 Instance Collection

In this section we describe the procedure we adopted to gather the instances. In January 2014, we issued an online call for instances using the main international mailing lists of the mathematical optimization and numerical analysis communities, reaching in this way the largest possible set of interested researchers and practitioners. The call remained open for 10 months, during which we received a large number of contributions of different nature. The instances we gathered come both from theoretical studies as well as from real-world applications.

In addition to spontaneous contribution we analysed the other generic libraries of instances available on internet and containing QP instances. Namely, the libraries from which we gathered instances are

- the `BARON` library `http://www.minlp.com/nlp-and-minlp-test-problems`;
- the `CUTEst` library `https://ccpforge.cse.rl.ac.uk/gf/project/cutest/wiki`;
- the `GAMS` library `http://www.gamsworld.org/performance/performlib.htm`;
- the `MacMINLP` library `https://wiki.mcs.anl.gov/leyffer/index.php/MacMINLP`;
- the `Meszaros` library `http://www.doc.ic.ac.uk/~im/00README.QP`;
- the `MINLP` library `http://www.gamsworld.org/minlp/minlplib.htm`;
- the `POLIP` library `http://polip.zib.de/pipformat.php`.

Other quadratic instances were found in online libraries devoted to specific QP problems as Max-Cut, Quadratic Assignment, Portfolio Optimization, and several others.

At the end of this process we gathered more than eight thousand instances. Three fourths of them contained discrete variables, while the remaining ones contained only continuous variables. More in details, we gathered $\approx$ 1800 Quadratic Binary Linear (QBL) instances, $\approx$ 2000 Quadratic Continuous Quadratic (QCQ) instances, and and $\approx$ 2500 Quadratic General Quadratic (QGQ) instances. W also gathered $\approx$ 1000 Convex General Convex (CGC) instances. We gathered relatively fewer Quadratic Binary Quadratic (QBQ), Convex Continuous Convex (CCC) and Convex Mixed Convex (CMC) instances, i.e., $\approx$ 150 instances, $\approx$ 200 instances and $\approx$ 200 instances respectively. Finally, we gathered only 17 Quadratic Mixed Linear (QML) instances. In the call for instances, no specific formats requirements were imposed for the submissions.

To evaluate the instances we decided, for practical reasons, to use Gams as common platform for all the experiments involving commercial and non-commercial solvers. For this reason, we decided to translate all instances into the Gams format (`.gms`). In addition, we have introduced a new, specific `.qplib` format. This new format is capable of describing all the instances of the library in a sparse form. In comparison to a more *high level* format like `.gms`, the new format presents two advantages: it is easier to read by a self-made parser and it produces smaller files. See the Appendix for more details.

For each instance of the starting set, we collected important characteristics which allowed us to classify the instances into the QP categories described in Section 2. As far as the variable types are concerned, we collected the following information:

- number of binary variables;
- number of integer variables;
- number of continuous variables.

In case at least one binary or integer variable is present, then the instance is categorized as *discrete*, otherwise it is categorized as *continuous*. As fas as the objective function is concerned, we gathered the following information:

- percentage of negative eigenvalues of the Q^0 matrix;
- density of the Q^0 matrix (number of nonzero entries over the total number).

The number of negative eigenvalues of Q^0 allows us to identify the objective function type, as in presence of at least one negative eigenvalue the objective function is non convex. Finally, as far as the constraint types are concerned, we collected the following information:

- number of linear constraints,
- number of quadratic constraints,
- number of convex constraints,
- number of box constraints.

A constraint is considered quadratic if it contains at least one nonzero in the quadratic term. Among the quadratic constraints, the ones with only non-negative eigenvalues are classified as convex constraints. All this information

allowed us to analyze the gathered instances and to perform the filters described in the the next paragraph.

3.2 Instance Selection

During the development of the library, a discussion ensued about what the expected gols that we wanted to achieve. The following four goals where finally identified:

1. represent as much as possible all the different categories of QP problems;
2. gather "challenging" instances, i.e., ones which can not be easily solved by state-of-the-art solvers;
3. include for each of the categories a set of well-diversified instances;
4. obtain a set of instances which is neither too small, so as to preserve statistical relevance, nor too large so as to being computationally manageable.

To achieve the aforementioned goals, we performed the following two filters, applied in cascade.

- *First Instances Filter.*
  The first filter was designed to drastically reduce the number of instances by eliminating the "easy" ones. An empirical measure for the hardness of an instance is the CPU time needed by a complete solver (cf. §2.3) to solve it to global optimality. Accordingly, for each of the gathered instance we ran the complete solvers in `Gams` (cf. Table 2) capable of solving it, whose number depends on the category of the instance under consideration. We then filtered according to a relative measure of computational difficulty, i.e., we discarded all instances that are solved by at least 30% of the complete solvers within a time limit of 30 seconds.
- *Second Instances Filter.*
  The goal of the second filter was to eliminate "similar" instances. We carefully analysed the instances one by one, and we clustered them according to their features; for each cluster we kept only a few representatives. **Comment: we may want to be a tad more specific about what "features" exactly meant there**. Finally, in order to only keep computationally challenging instances we ran a general purpose solver able to tackle all QP categories with a time limit of 120 seconds; all the instances which have been solved to proven optimality within this time limit were discarded. **Comment: this might actually be best characterized as a third filter.**

In Table 3 we summarize the two filter steps, which allowed us to identify the final set of 251 discrete instances and 116 continuous instances.

Starting set	≈ 8500 Instances	
	⇓	
	≈ 6000 Discr. Inst.	≈ 2500 Cont. inst.
First Filter	⇓	⇓
	≈ 3000 Discr. Inst.	≈ 1000 Cont. Inst.
Second Filter	⇓	⇓
	251 Discr. Inst.	116 Cont. inst.

Table 3 Instance filter steps

3.3 Analysis of the final set of instances

We now analyse the features of the instances selected to be part of the library. The characteristics of the instances are presented in Table 4 for *discrete* instances (*{B,M,I,G}*) and in Table 5 for *continuous* ones (*C*). For each category, the Tables report in column # the corresponding number of instances. It can be seen that the final set well respects the original distribution of the gathered instances among the different categories. Indeed, the discrete categories (LMQ) or (QBL) are well represented by 118 and 59 instances, respectively. Similarly, the continuous categories (LCQ) and (QCQ) are well represented by 50 and 17 instances, respectively. Moreover, the library actually covers the large majority of all possible categories of instances.

We now report some graphs that help in illustrating the main features of the instences. In Figure **??** we plot the number of variables (horizontal axis) versus the number of constrains (vertical axis), both in logarithmic scale. Continuous instances are marked with "+", and discrete ones with "×". The figure shows that the library contains a quite diverse set of instances of different in terms of number of variables and constraints. The number of constraints goes from few units up to one hundred thousands constraints, while the number of variables goes up to almost forty-five thousands.

Figure 2 describes how discrete and continuous variables are distributed within the instances. The instances are sorted accordingly to the total number of variables. For each instance we report the total number of variables with a "+", and the total number of discrete variables (binary or general integer) with a "×". The pictures clearly show that instances with different percentages of integer and continuous variables are present in the library, and well distributed across the whole spectrum of variable sizes.

Similarly, Figures 3 describes how linear and quadratic constraints are distributed within the instances. The instances are sorted accordingly to the total number of constraints For each instances we report the total number of constraints with a "+" and the total number of (either convex of nonconvex) quadratic constraints with a "×". Also in this case, different percentages of linear and quadratic constraints are present and well-distributed across the spectrum of constraint sizes, although both medium- and large-size instances

Obj. Fun.	Variables	Constraints	#
Linear	Binary	Quadratic	9
	Mixed	Convex	2
		Quadratic	118
	Integer	Quadratic	2
	General	Quadratic	3
Convex	Binary	Linear	2
	Mixed	Linear	13
		Quadratic	1
Quadratic	Binary	None	23
		Linear	59
		Quadratic	5
	Mixed	Linear	10
		Quadratic	1
	Integer	Linear	2
	General	Quadratic	1
Total			251

Table 4 Classification of the final set of discrete instances

Obj. Fun.	Constraints	#
Linear	Convex	11
	Quadratic	50
Convex	Box	3
	Linear	12
	Convex	2
	Quadratic	5
Quadratic	Linear	5
	Convex	11
	Quadratic	17
Total		116

Table 5 Classification of the final set of continuous instances

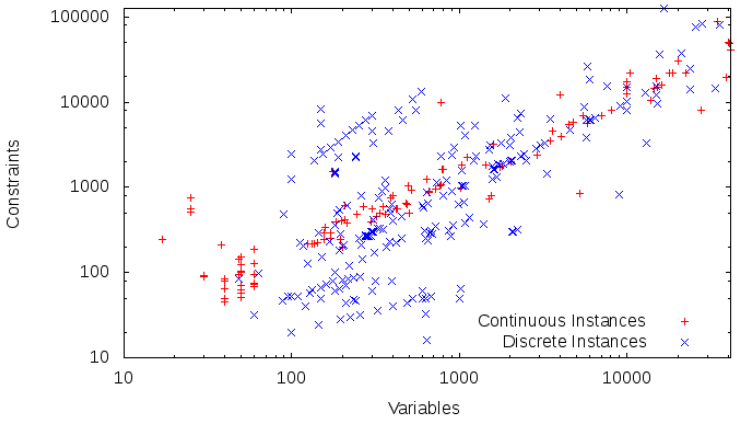

Fig. 1 Distribution of variables and constrains of the qplib instances

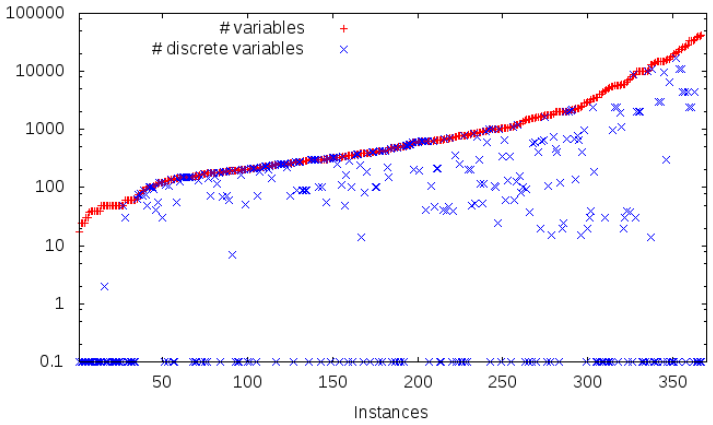

Fig. 2 Number of variables

show a prevalence of lower percentages of quadratic constraints. In particular, from Figure 3 we learn that while the maximum number of linear constraints exceeds 100000, the maximum number of quadratic constraints tops up at around 10000. This is, however, reasonable, considering how quadratic constraints can, in general, be expected to be much more computationally challenging than linear ones, especially if nonconvex.

Figure 4 shows the instances with at least a quadratic constraints sorted according to the number of quadratic constraints (vertical axis). For each instances we report the total number of constraints with a "+" and the total number of nonconvex quadratic constraints with a "×".

Speaking of nonconvexity, Figure 5 focuses on the instances with a quadratic objective function, ordered by the the percentage of negative eigenvalues (vertical axis). Although almost all percentages are represented, the instances are

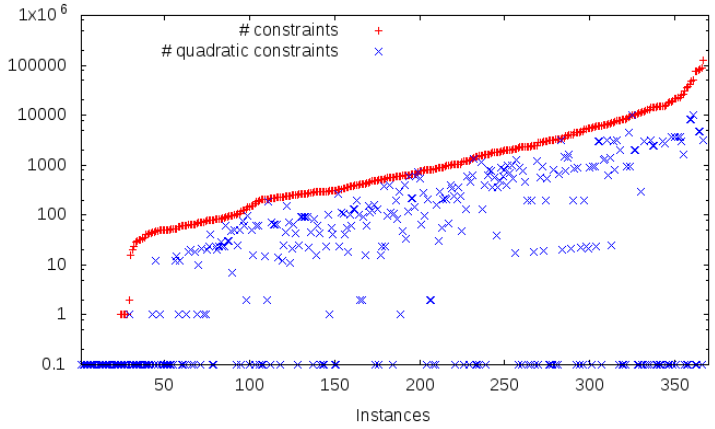

Fig. 3 Number of constraints

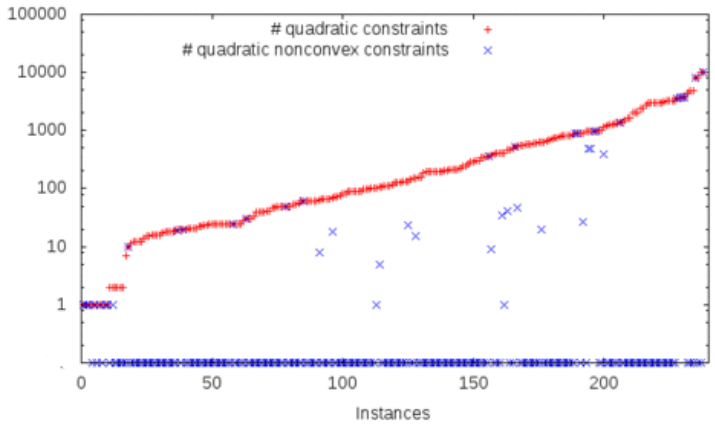

Fig. 4 Quadratic constraints

mostly clustered around two values. About 25% of the instances are convex, i.e., they have 0% of negative eigenvalues. Among the others, a vast majority have around 50% of negative eigenvalues. However, instances with high or low percentages of negative eigenvalues are present.

Similarly, Figure 6 shows the instances with a quadratic objective function sorted according to the density of the Q^0 matrix (vertical axis). The majority of the instances have either a very low or a rather high density: indeed, about 30% of the instances have density smaller that 5%, and about 30% of the instances have density larger that 50%. However, also intermediate values are present.

Additional details on the instance features can be found in the Appendix A.

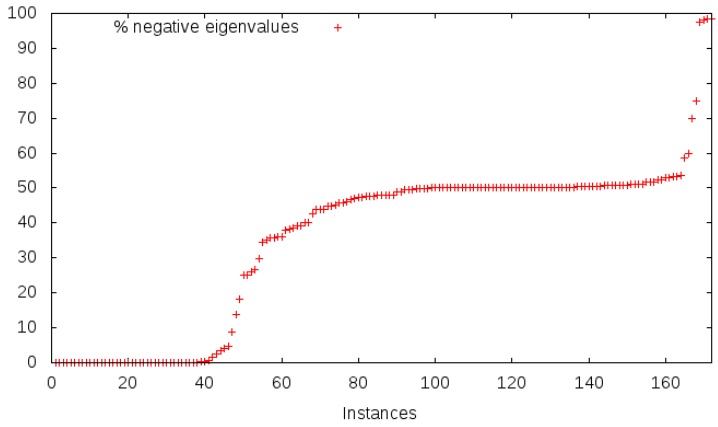

Fig. 5 Negative eigenvalues

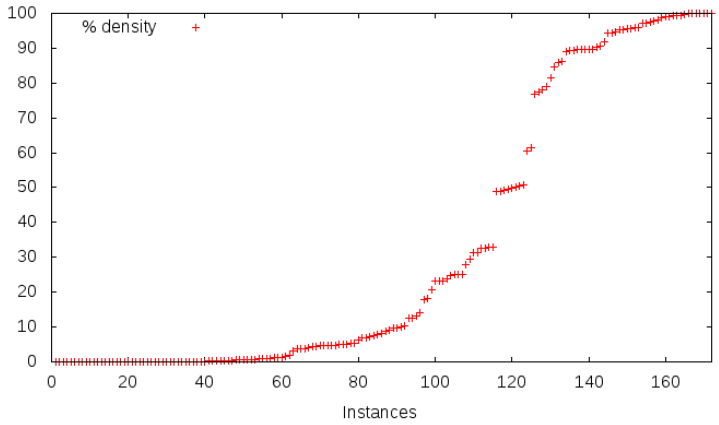

Fig. 6 Density

4. Software tools

4.1 instance translator

GAMS–LP–QPFORMAT TASK X : write

4.2 code that computes the features of an instance

TASK X : write

4.3 code that selects subsets of instances

TASK X : write

4.4 website, instance collector

Select subset or categories of instances (EXTRACT FROM THE LIBRARY A SUBSET OF INSTANCES WITH SPECIFIC CHARACTERISTICS) TASK X : write

4.5 testing environment

RUN GAMS USING A SUBSET OF SOLVERS TASK X : write

5. Conclusions

This manuscript describes the first comprehensive library of instances for Quadratic Programming (QP). Since QP comprises different and 'varied" categories of problems, we propose a classification and then we describe the principal solution methods representing the state-of-the-art algorithms for QP.

We then describe the steps of the adopted process used to filter the gathered instances in order to build the new library. The goals we purposed are of different natures and we tried to build a library which is computationally challenging and as broad as possible, i.e., it represents the largest possible categories of QP problems.

We want to stress once again that we intentionally avoid to perform a computational comparison of the performances of the different solvers. Our goal is instead to provide a common test-bed of instances for practitioners and researchers in the field. This new library will hopefully serve as a point of reference to test new ideas and algorithms for QP problems.

6. Acknowledgements

References

1. Ahmetović, E., Grossmann, I.E.: Global superstructure optimization for the design of integrated process water networks. AIChE J. **57**(2), 434–457 (2011)
2. Alfaki, M., Haugland, D.: A multi-commodity flow formulation for the generalized pooling problem. J. Glob. Optim. **56**(3), 917–937 (2013)
3. Anjos, M.F., Liers, F.: Global approaches for facility layout and VLSI floorplanning. In: M.F. Anjos, J.B. Lasserre (eds.) Handbook on Semidefinite, Conic and Polynomial Optimization, *International Series in Operations Research & Management Science*, vol. 166, pp. 849–877. Springer US (2012)
4. Anstreicher, K.M.: Recent advances in the solution of quadratic assignment problems. Math. Program. **97**(1-2), 27–42 (2003)

5. Audet, C., Guillou, A., Hansen, P., Messine, F., Perron, S.: The small hexagon and heptagon with maximum sum of distances between vertices. J. Glob. Optim. **49**(3), 467–480 (2011)
6. Audet, C., Hansen, P., Messine, F.: The small octagon with longest perimeter. J. Combin. Theory Ser. A **114**(1), 135 – 150 (2007)
7. Audet, C., Hansen, P., Messine, F.: Simple polygons of maximum perimeter contained in a unit disk. Discrete Comput. Geom. **41**(2), 208–215 (2009)
8. Audet, C., Hansen, P., Messine, F., Xiong, J.: The largest small octagon. J. Combin. Theory Ser. A **98**(1), 46 – 59 (2002)
9. Audet, C., Ninin, J.: Maximal perimeter, diameter and area of equilateral unit-width convex polygons. J. Glob. Optim. **56**(3), 1007–1016 (2013)
10. Bagajewicz, M.: A review of recent design procedures for water networks in refineries and process plants. Comput. Chem. Eng. **24**, 2093 – 2113 (2000)
11. Belotti, P., Lee, J., Liberti, L., Margot, F., Wächter, A.: Branching and bounds tightening techniques for non-convex MINLP. Optimization Methods and Software **24**(4), 597–634 (2009)
12. Bley, A., Gleixner, A.M., Koch, T., Vigerske, S.: Comparing MIQCP solvers to a specialised algorithm for mine production scheduling. In: H.G. Bock, X.P. Hoang, R. Rannacher, J.P. Schlöder (eds.) Modeling, Simulation and Optimization of Complex Processes, pp. 25–39. Springer Berlin Heidelberg (2012)
13. Bliek, C., Bonami, P., Lodi, A.: Solving mixed-integer quadratic programming problems with IBM-CPLEX: a progress report. In: Proceedings of the RAMP Symposium, *RAMP*, vol. 26, pp. 171–180. Hosei University, Tokyo (2014)
14. Blum, L., Shub, M., Smale, S.: On a theory of computation and complexity over the real numbers: *NP*-completeness, recursive functions, and universal machines. Bulletin of the AMS **21**(1), 1–46 (1989)
15. Bomze, I.M., Budinich, M., Pardalos, P.M., Pelillo, M.: The maximum clique problem. In: D.Z. Du, P.M. Pardalos (eds.) Handbook of Combinatorial Optimization, pp. 1–74. Springer US (1999)
16. Bragalli, C., D'Ambrosio, C., Lee, J., Lodi, A., Toth, P.: On the optimal design of water distribution networks: A practical MINLP approach. Optim. Eng. **13**, 219–246 (2012)
17. Buchheim, C., Wiegele, A.: Semidefinite relaxations for non-convex quadratic mixed-integer programming. Mathematical Programming **141**(1), 435–452 (2013)
18. Burer, S.: Copositive Programming, pp. 201–218. Springer US, Boston, MA (2012)
19. Calamai, P.H., Moré, J.J.: Projected gradient methods for linearly constrained problems. Mathematical Programming **39**(1), 93–116 (1987)
20. Castillo, I., Westerlund, J., Emet, S., Westerlund, T.: Optimization of block layout design problems with unequal areas: A comparison of MILP and MINLP optimization methods. Comput. Chem. Eng. **30**(1), 54 – 69 (2005)
21. Castillo, P.A.C., Mahalec, V., Kelly, J.D.: Inventory pinch algorithm for gasoline blend planning. AIChE J. **59**(10), 3748–3766 (2013)
22. Castro, P.M., Teles, J.P.: Comparison of global optimization algorithms for the design of water-using networks. Comput. Chem. Eng. **52**, 249 – 261 (2013)
23. Chen, P., Gui, C.: Linear convergence analysis of the use of gradient projection methods on total variation problems. Computational Optimization and Applications **54**(2), 283–315 (1987)
24. D'Ambrosio, C., Linderoth, J., Luedtke, J.: Valid inequalities for the pooling problem with binary variables. In: O. Günlük, G.J. Woeginger (eds.) Integer Programming and Combinatoral Optimization, *Lecture Notes in Computer Science*, vol. 6655, pp. 117–129. Springer Berlin Heidelberg (2011)
25. Deng, Z., Bai, Y., Fang, S.C., Tian, Y., Xing, W.: A branch-and-cut approach to portfolio selection with marginal risk control in a linear conic programming framework. J. Sys. Sci. Sys. Eng. **22**(4), 385–400 (2013)
26. Dong, H.: Relaxing nonconvex quadratic functions by multiple adaptive diagonal perturbations. SIAM Journal on Optimization (to appear) (2016). URL `http://arxiv.org/abs/1403.5317`. ArXiv:1403.5317 [math.OC]
27. Dorneich, M.C., Sahinidis, N.V.: Global optimization algorithms for chip layout and compaction. Eng. Optim. **25**, 131–154 (1995)

28. Dostál, Z.: Optimal Quadratic Programming Algorithms: With Applications to Variational Inequalities. Springer Verlag, Heidelberg, Berlin, New York (2009)
29. Dür, M.: Copositive Programming – a Survey, pp. 3–20. Springer, Berlin, Heidelberg (2010)
30. Faria, D.C., Bagajewicz, M.J.: A new approach for global optimization of a class of minlp problems with applications to water management and pooling problems. AIChE J. **58**(8), 2320–2335 (2012)
31. Frangioni, A., Furini, F., Gentile, C.: Approximated Perspective Relaxations: a Project&Lift Approach. Computational Optimization and Applications **63**(3), 705–735 (2016)
32. Frangioni, A., Galli, L., Scutellà, M.: Delay-Constrained Shortest Paths: Approximation Algorithms and Second-Order Cone Models. Journal of Optimization Theory and Applications **164**(3), 1051–1077 (2015)
33. Frangioni, A., Galli, L., Stea, G.: Delay-constrained routing problems: Accurate scheduling models and admission control. Computers & Operations Research **to appear** (2017)
34. Frangioni, A., Gentile, C.: Perspective Cuts for a Class of Convex 0–1 Mixed Integer Programs. Mathematical Programming **106**(2), 225–236 (2006)
35. Frangioni, A., Gentile, C.: SDP Diagonalizations and Perspective Cuts for a Class of Nonseparable MIQP. Operations Research Letters **35**(2), 181–185 (2007)
36. Frangioni, A., Gentile, C.: A Computational Comparison of Reformulations of the Perspective Relaxation: SOCP vs. Cutting Planes. Operations Research Letters **37**(3), 206–210 (2009)
37. Frangioni, A., Gentile, C., Grande, E., Pacifici, A.: Projected Perspective Reformulations with Applications in Design Problems. Operations Research **59**(5), 1225–1232 (2011)
38. Geissler, B., Morsi, A., Schewe, L.: A new algorithm for MINLP applied to gas transport energy cost minimization. In: M. Jünger, G. Reinelt (eds.) Facets of Combinatorial Optimization, pp. 321–353. Springer Berlin Heidelberg (2013)
39. Gentilini, I., Margot, F., Shimada, K.: The travelling salesman problem with neighbourhoods: MINLP solution. Optim Met Softw **28**(2), 364–378 (2013)
40. Gleixner, A.M., Held, H., Huang, W., Vigerske, S.: Towards globally optimal operation of water supply networks. Numerical Algebra, Control and Optimization **2**(4), 695–711 (2012)
41. Gould, N.I.M., , Toint, Ph.L.: A quadratic programming bibliography. Numerical Analysis Group Internal Report 2000-1, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England (2000)
42. Gounaris, C.E., First, E.L., Floudas, C.A.: Estimation of diffusion anisotropy in microporous crystalline materials and optimization of crystal orientation in membranes. J. Chem. Phys. **139**(12), 124,703 (2013)
43. Hasan, M.M.F., Karimi, I.A., Avison, C.M.: Preliminary synthesis of fuel gas networks to conserve energy and preserve the environment. Ind. Eng. Chem. Res. **50**(12), 7414–7427 (2011)
44. Hemmecke, R., Köppe, M., Lee, J., Weismantel, R.: Nonlinear integer programming. In: M. Jünger, M.T. Liebling, D. Naddef, L.G. Nemhauser, R.W. Pulleyblank, G. Reinelt, G. Rinaldi, A.L. Wolsey (eds.) 50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art, pp. 561–618. Springer Berlin Heidelberg (2010)
45. Hifi, M., M'Hallah, R.: A literature review on circle and sphere packing problems: Models and methodologies. Advances in Operations Research **2009**, 22 (2009)
46. Jeroslow, R.: There cannot be any algorithm for integer programming with quadratic constraints. Operations Research **21**(1), 221–224 (1973)
47. Jeżowski, J.: Review of water network design methods with literature annotations. Ind. Eng. Chem. Res. **49**(10), 4475 – 4516 (2010)
48. Kallrath, J.: Exact computation of global minima of a nonconvex portfolio optimization problem. In: C.A. Floudas, P.M. Pardalos (eds.) Frontiers in Global Optimization, pp. 237–254. Kluwer Academic Publishers (2003)
49. Kallrath, J.: Cutting circles and polygons from area-minimizing rectangles. J. Glob. Optim. **43**, 299 – 328 (2009)
50. Kallrath, J., Rebennack, S.: Cutting ellipses from area-minimizing rectangles. J. Glob. Optim. **59**(2-3), 405–437 (2014)

51. Khor, C.S., Chachuat, B., Shah, N.: Fixed-flowrate total water network synthesis under uncertainty with risk management. J. Clean. Prod. **77**(0), 79 – 93 (2014)
52. Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R.E., Danna, E., Gamrath, G., Gleixner, A.M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D.E., Wolter, K.: Miplib 2010. Mathematical Programming Computation **3**(2), 103–163 (2011)
53. Kochenberger, G., Hao, J.K., Glover, F., Lewis, M., Lü, Z., Wang, H., Wang, Y.: The unconstrained binary quadratic programming problem: a survey. Journal of Combinatorial Optimization **28**(1), 58–81 (2014)
54. Kolodziej, S.P., Castro, P.M., Grossmann, I.E.: Global optimization of bilinear programs with a multiparametric disaggregation technique. J. Glob. Optim. **57**(4), 1039–1063 (2013)
55. Kolodziej, S.P., Grossmann, I.E., Furman, K.C., Sawaya, N.W.: A discretization-based approach for the optimization of the multiperiod blend scheduling problem. Comput. Chem. Eng. **53**, 122 – 142 (2013)
56. Li, J., Li, A., Karimi, I.A., Srinivasan, R.: Improving the robustness and efficiency of crude scheduling algorithms. AIChE J. **53**(10), 2659–2680 (2007)
57. Li, J., Misener, R., Floudas, C.A.: Continuous-time modeling and global optimization approach for scheduling of crude oil operations. AIChE J. **58**(1), 205–226 (2012)
58. Li, J., Misener, R., Floudas, C.A.: Scheduling of crude oil operations under demand uncertainty: A robust optimization framework coupled with global optimization. AIChE J. **58**(8), 2373–2396 (2012)
59. Li, X., Armagan, E., Tomasgard, A., Barton, P.I.: Stochastic pooling problem for natural gas production network design and operation under uncertainty. AIChE J. **57**(8), 2120–2135 (2011)
60. Li, X., Tomasgard, A., Barton, P.I.: Decomposition strategy for the stochastic pooling problem. J. Glob. Optim. **54**(4), 765–790 (2012)
61. Lin, X., Floudas, C.A., Kallrath, J.: Global solution approach for a nonconvex MINLP problem in product portfolio optimization. J. Glob. Optim. **32**, 417–431 (2005)
62. Loiola, E.M., de Abreu, N.M.M., Boaventura-Netto, P.O., Hahn, P., Querido, T.: A survey for the quadratic assignment problem. Eur. J. Oper. Res. **176**(2), 657 – 690 (2007)
63. Maranas, C.D., Androulakis, I.P., Floudas, C.A., Berger, A.J., Mulvey, J.M.: Solving long-term financial planning problems via global optimization. Journal of Economic Dynamics and Control **21**(8-9), 1405 – 1425 (1997)
64. McCormick, G.: Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems. Mathematical Programming **10**, 146–175 (1976)
65. Misener, R., Floudas, C.: GloMIQO: Global Mixed-Integer Quadratic Optimizer. Journal of Global Optimization **57**, 3–50 (2013)
66. Misener, R., Floudas, C.A.: Advances for the pooling problem: Modeling, global optimization, and computational studies. Applied and Computational Mathematics **8**(1), 3 – 22 (2009)
67. Misener, R., Floudas, C.A.: Global optimization of large-scale pooling problems: Quadratically constrained MINLP models. Ind. Eng. Chem. Res. **49**(11), 5424 – 5438 (2010)
68. Mouret, S., Grossmann, I.E., Pestiaux, P.: A novel priority-slot based continuous-time formulation for crude-oil scheduling problems. Ind. Eng. Chem. Res. **48**(18), 8515–8528 (2009)
69. Mouret, S., Grossmann, I.E., Pestiaux, P.: A new Lagrangian decomposition approach applied to the integration of refinery planning and crude-oil scheduling. Comput. Chem. Eng. **35**(12), 2750–2766 (2011)
70. Neumaier, A.: Complete search in continuous global optimization and constraint satisfaction. Acta Numerica **13**, 271–369 (2004)
71. Nyberg, A., Grossmann, I.E., Westerlund, T.: The optimal design of a three-echelon supply chain with inventories under uncertainty (2012). Available from CyberInfrastructure for MINLP [`www.minlp.org`, a collaboration of Carnegie Mellon University and IBM Research] at: `www.minlp.org/library/problem/index.php?i=157`

72. Papageorgiou, D.J., Toriello, A., Nemhauser, G.L., Savelsbergh, M.W.P.: Fixed-charge transportation with product blending. Transport. Sci. **46**(2), 281–295 (2012)
73. Parpas, P., Rustem, B.: Global optimization of the scenario generation and portfolio selection problems. In: M. Gavrilova, O. Gervasi, V. Kumar, C. Tan, D. Taniar, A. Laganá, Y. Mun, H. Choo (eds.) Computational Science and Its Applications - ICCSA 2006, *Lecture Notes in Computer Science*, vol. 3982, pp. 908–917. Springer-Verlag (2006)
74. Pham, V., Laird, C., El-Halwagi, M.: Convex hull discretization approach to the global optimization of pooling problems. Ind. Eng. Chem. Res. **48**, 1973 – 1979 (2009)
75. Ponce-Ortega, J.M., El-Halwagi, M.M., Jiménez-Gutiérrez, A.: Global optimization for the synthesis of property-based recycle and reuse networks including environmental constraints. Comput. Chem. Eng. **34**(3), 318 – 330 (2010)
76. Quesada, I., Grossmann, I.: Global optimization of bilinear process networks and multicomponent flows. Computers & Chemical Engineering **19**(12), 1219–1242 (1995)
77. Rebennack, S., Kallrath, J., Pardalos, P.M.: Column enumeration based decomposition techniques for a class of non-convex MINLP problems. J. Glob. Optim. **43**(2-3), 277–297 (2009)
78. Rendl, F., Rinaldi, G., Wiegele, A.: Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. Mathematical Programming **121**(2), 307–335 (2008)
79. Rios, L.M., Sahinidis, N.V.: Portfolio optimization for wealth-dependent risk preferences. Annals of Oper. Res. **177**, 63–90 (2010)
80. Ruiz, J.P., Grossmann, I.E.: Exploiting vector space properties to strengthen the relaxation of bilinear programs arising in the global optimization of process networks. Optim. Lett. **5**, 1–11 (2011)
81. Ruiz, M., Briant, O., Clochard, J.M., Penz, B.: Large-scale standard pooling problems with constrained pools and fixed demands. J. Glob. Optim. **56**(3), 939–956 (2013)
82. Saif, Y., Elkamel, A., Pritzker, M.: Global optimization of reverse osmosis network for wastewater treatment and minimization. Ind. Eng. Chem. Res. **47**(9), 3060 – 3070 (2008)
83. Szabó, P.G., Markót, C.M., Csendes, T.: Global optimization in geometry circle packing into the square. In: C. Audet, P. Hansen, G. Savard (eds.) Essays and Surveys in Global Optimization, pp. 233–265. Springer, New York (2005)
84. Tadayon, B., Smith, J.C.: Algorithms for an integer multicommodity network flow problem with node reliability considerations. J. Optim. Theory Appl. **161**, 506–532 (2013)
85. Tahanan, M., van Ackooij, W., Frangioni, A., Lacalandra, F.: Large-scale Unit Commitment under uncertainty. 4OR **13**(2), 115–171 (2015)
86. Tarski, A.: A decision method for elementary algebra and geometry. Tech. Rep. R-109, Rand Corporation (1951)
87. Teles, J.P., Castro, P.M., Matos, H.A.: Global optimization of water networks design using multiparametric disaggregation. Comput. Chem. Eng. **40**, 132 – 147 (2012)
88. Visweswaran, V.: MINLP: Applications in blending and pooling. In: C.A. Floudas, P.M. Pardalos (eds.) Encyclopedia of Optimization, 2 edn., pp. 2114 – 2121. Springer Science (2009)
89. Westerlund, T., Pörn, R.: Solving pseudo-convex mixed integer optimization problems by cutting plane techniques. Optimization and Engineering **3**, 235–280 (2002)
90. Wikipedia: Quadratic programming (2016). URL `https://en.wikipedia.org/wiki/Quadratic_programming`. [Online; accessed 29-July-2016]
91. Wikipedia: Quadratically constrained quadratic program (2016). URL `https://en.wikipedia.org/wiki/Quadratically_constrained_quadratic_program`. [Online; accessed 29-July-2016]
92. Wright, S.: Primal-Dual Interior-Point Methods. SIAM, Philadelphia (1997)

A. Instance details

In this Appendix we provide detailed data on all the instances of the final library. This is done in Tables **??**–**??** for *discrete* instances (*{B,M,I,G}*) and in Tables **??**–**??** for *continuous*

ones (*C*). In the former, the features of the instances are described by three sets of columns. The first ("% n.e.") describes the objective function by reporting the fraction of eigenvalues of Q^0 that are negative: a positive number implies that Q^0 is not SDP (hence, the instance is a Q**), "0.0" implies that Q^0 is SDP (hence, the instance is a C**), a blank implies that $Q^0 = 0$, i.e., the objective function is linear (hence, the instance is a L**). The second ("% dens") describes the density of the Q^0 matrix; a blank implies that the corresponding instance has a linear objective function. For both columns ("% n.e." and "% d.") we report 0.1 in case an instance has smaller positive value. The following three columns describe the variables by reporting the number of binary ones ("# b."), general integer ones ("# i."), and continuous ones ("# c."). Finally, the last three columns describe the constraints reporting the number of linear ones ("# l."), nonconvex quadratic ones ("# q."), and convex quadratic ones ("# c."). Tables **??**–**??** are similarly structured except that all variables are continuous, and hence only one column is present.

	Q^0		Variables			Constraints			
name	% n.e.	% d.	# b.	# i.	# c.	# l.	# q.	# c.	# b.
qplib_0018	48.0	100.0	0	0	50	1	0	0	0
qplib_0031	18.4	25.0	30	0	30	32	0	0	0
qplib_0032	25.0	25.0	50	0	50	52	0	0	0
qplib_0067	47.5	88.9	80	0	0	1	0	0	0
qplib_0343	48.0	100.0	0	0	50	1	0	0	50
qplib_0633	58.7	98.7	75	0	0	1	0	0	0
qplib_0678			9600	0	5537	7457	960	480	737
qplib_0681			72	0	143	419	48	0	143
qplib_0682			71	0	190	501	96	0	190
qplib_0684			101	0	260	815	128	0	260
qplib_0685			256	0	519	1603	192	0	519
qplib_0686			692	0	1512	4440	640	0	1512
qplib_0687			672	0	1651	4875	800	0	1651
qplib_0688			1964	0	3824	20568	1600	0	3824
qplib_0689			756	0	1112	9800	288	0	1112
qplib_0690			6428	0	10048	112400	3200	0	10048
qplib_0696			187	0	207	390	33	0	207
qplib_0698			55	0	63	126	15	0	63
qplib_0752	50.0	10.0	250	0	0	1	0	0	0
qplib_0911	44.0	50.5	0	0	50	0	50	50	50
qplib_0975	50.0	50.7	0	0	50	0	10	10	50
qplib_1055	50.0	100.0	0	0	40	0	20	19	40
qplib_1143	50.0	97.2	0	0	40	4	20	0	40
qplib_1157	25.0	94.5	0	0	40	8	1	1	40
qplib_1353	26.0	95.8	0	0	50	5	1	1	50
qplib_1423	75.0	95.4	0	0	40	4	20	20	40
qplib_1437	50.0	95.6	0	0	50	10	1	1	50
qplib_1451	50.0	49.1	0	0	60	6	60	0	60
qplib_1493	50.0	97.4	0	0	40	4	1	0	40
qplib_1507	26.7	95.8	0	0	30	3	30	30	30
qplib_1535	50.0	94.4	0	0	60	6	60	60	60
qplib_1619	50.0	95.6	0	0	50	5	25	25	50
qplib_1661	50.0	95.4	0	0	60	12	1	1	60
qplib_1675	51.7	48.8	0	0	60	12	1	0	60
qplib_1703	51.7	98.0	0	0	60	6	30	0	60
qplib_1745	50.0	48.8	0	0	50	5	50	0	50
qplib_1773	50.0	94.8	0	0	60	6	1	1	60
qplib_1886	50.0	50.0	0	0	50	0	50	0	50
qplib_1913	50.0	25.0	0	0	48	0	48	0	48
qplib_1922	50.0	49.6	0	0	30	0	60	0	30
qplib_1931	50.0	49.9	0	0	40	0	40	0	40
qplib_1940	50.0	25.0	0	0	48	0	96	0	48
qplib_1967	50.0	99.8	0	0	50	0	75	0	50
qplib_1976	38.2	7.0	152	0	0	136	16	0	0
qplib_2017	39.3	5.5	252	0	0	231	21	0	0
qplib_2022	38.6	5.3	275	0	0	253	22	0	0
qplib_2029	40.2	5.1	299	0	0	276	23	0	0
qplib_2036	39.2	4.9	324	0	0	300	24	0	0
qplib_2047			136	0	0	2040	17	0	0
qplib_2055			153	0	0	2448	18	0	0
qplib_2060			171	0	0	2907	19	0	0
qplib_2067			190	0	0	3420	20	0	0
qplib_2073			210	0	0	3990	21	0	0
qplib_2077			231	0	0	4620	22	0	0
qplib_2085			253	0	0	5313	23	0	0
qplib_2087			276	0	0	6072	24	0	0
qplib_2096			300	0	0	6900	25	0	0
qplib_2165			683	0	1376	1366	683	0	0
qplib_2166			345	0	697	690	345	0	0
qplib_2167			61	0	131	122	61	0	0
qplib_2168			214	0	438	428	214	0	0
qplib_2169			297	0	608	594	297	0	0
qplib_2170			351	0	736	702	351	0	0
qplib_2171			150	0	305	300	150	0	0
qplib_2173			215	0	436	430	215	0	0
qplib_2174			768	0	1545	1536	768	0	0

	Q^0		Variables			Constraints			
name	% n.e.	% d.	# b.	# i.	# c.	# l.	# q.	# c.	# b.
qplib_2181			90	0	190	180	90	0	0
qplib_2187			90	0	195	180	90	0	0
qplib_2192			90	0	200	180	90	0	0
qplib_2195			90	0	205	180	90	0	0
qplib_2202			90	0	185	180	90	0	0
qplib_2203			100	0	205	200	100	0	0
qplib_2204			110	0	225	220	110	0	0
qplib_2205			958	0	1926	1916	958	0	0
qplib_2206			194	0	421	388	194	0	0
qplib_2315	44.8	7.5	595	0	0	13090	0	0	0
qplib_2353	50.0	23.8	147	0	93	2240	0	0	93
qplib_2357	50.0	7.9	240	0	0	2240	0	0	0
qplib_2359	47.1	3.8	306	0	0	3264	0	0	0
qplib_2416			0	0	25	153	533	46	25
qplib_2430			0	0	125	27	65	0	125
qplib_2445			0	0	143	14	66	0	143
qplib_2456			0	0	5477	4131	1369	1369	0
qplib_2468			0	0	14885	11203	3721	3721	0
qplib_2480			0	0	399	199	201	0	398
qplib_2482			0	0	1806	1418	361	0	0
qplib_2483			0	0	760	40	240	0	760
qplib_2492	35.8	86.3	196	0	0	28	0	0	0
qplib_2505			0	0	1039	302	480	0	1039
qplib_2512	46.0	77.4	100	0	0	20	0	0	0
qplib_2519			0	0	4806	3802	961	961	0
qplib_2540			0	0	498	341	210	0	498
qplib_2546	0.0	0.2	0	0	1015	592	400	0	0
qplib_2590			0	0	25	93	401	35	25
qplib_2626			0	0	22327	14763	3721	3721	0
qplib_2635			0	0	176	0	1154	384	0
qplib_2650			0	0	1110	228	904	27	1110
qplib_2658			0	0	184	57	133	23	184
qplib_2676			0	0	1445	1095	361	361	0
qplib_2693			0	0	791	183	631	20	791
qplib_2696	1.8	0.1	0	0	3500	1995	1500	0	0
qplib_2698			0	0	196	36	11	0	196
qplib_2702	4.7	1.2	259	0	1	212	0	0	0
qplib_2703			0	0	799	399	401	1	798
qplib_2707			0	0	634	151	466	42	586
qplib_2708			108	0	526	327	30	0	526
qplib_2712	50.0	100.0	0	0	200	1	0	0	200
qplib_2714			0	0	352	301	298	0	0
qplib_2733	40.2	89.2	324	0	0	36	0	0	0
qplib_2738			0	0	199	99	101	1	198
qplib_2758			0	0	303	139	112	0	303
qplib_2761	50.0	100.0	0	0	500	1	0	0	500
qplib_2784			0	0	4501	3680	900	900	0
qplib_2819			0	0	334	24	132	0	334
qplib_2823			0	0	390	103	283	0	374
qplib_2834			0	0	156	14	72	0	156
qplib_2862			0	0	40501	32640	8100	8100	0
qplib_2880	48.8	90.4	625	0	0	50	0	0	0
qplib_2881			0	0	1512	0	720	0	0
qplib_2882			56	0	88	257	16	0	16
qplib_2894			0	0	17	55	154	15	17
qplib_2935			72	0	108	325	18	0	18
qplib_2957	45.7	60.4	484	0	0	44	0	0	0
qplib_2958			42	0	70	197	14	0	14
qplib_2967	47.4	5.0	0	0	38	1	190	0	19
qplib_2981	0.0	0.1	0	0	2015	1192	800	0	0
qplib_2987			0	0	208	114	90	0	208
qplib_2993			0	0	266	235	84	0	266
qplib_3029			0	0	5767	3783	961	961	0
qplib_3034			0	0	780	40	240	0	780
qplib_3049	0.6	0.1	0	0	7000	3995	3000	0	0
qplib_3060	0.3	0.1	48	0	792	1192	0	0	0
qplib_3080	0.0	0.1	0	0	4015	2392	1600	0	0
qplib_3083			0	0	243	107	126	0	243
qplib_3088			0	0	3601	2780	900	900	0
qplib_3089			0	0	132	12	72	0	132
qplib_3105			0	0	18606	14802	3721	3721	0
qplib_3120			0	0	662	40	204	0	662
qplib_3122	0.0	0.1	17136	0	3988	36703	0	0	776
qplib_3147			0	0	419	32	108	0	419
qplib_3170			0	0	660	40	160	0	660
qplib_3177			0	0	1599	799	801	0	1598
qplib_3181			84	0	308	180	16	0	308
qplib_3185			0	0	18001	14560	3600	3600	0
qplib_3192			0	0	479	32	145	0	479
qplib_3225			0	0	136	14	66	0	136
qplib_3240			0	0	516	187	220	0	516
qplib_3247			0	0	361	322	156	0	0
qplib_3279			56	0	251	148	16	0	251
qplib_3297	0.0	0.1	0	0	8015	4792	3200	0	0
qplib_3307	48.1	61.6	256	0	0	32	0	0	0
qplib_3312			0	0	41406	33002	8281	0	0
qplib_3318			0	0	25	93	381	9	25
qplib_3326	2.6	0.1	0	0	1750	995	750	0	0

	Q^0		Variables			Constraints			
name	% n.e.	% d.	# b.	# i.	# c.	# l.	# q.	# c.	# b.
qplib_3334			0	0	715	40	210	0	715
qplib_3337			0	0	297	0	198	0	199
qplib_3338			0	0	320	26	110	0	320
qplib_3347	51.8	85.8	676	0	0	52	0	0	0
qplib_3358			0	0	158	66	106	5	158
qplib_3361	45.1	31.3	1024	0	0	64	0	0	0
qplib_3369			0	0	485	32	116	0	485
qplib_3380	3.5	0.1	8904	0	0	823	0	0	0
qplib_3385			0	0	155	77	60	0	155
qplib_3387			0	0	170	18	65	0	170
qplib_3402	47.3	81.5	144	0	0	24	0	0	0
qplib_3413	50.0	9.1	400	0	0	40	0	0	0
qplib_3416			0	0	424	32	96	0	424
qplib_3496			200	56	72	623	64	8	64
qplib_3502			10920	0	2090	209	3130	0	0
qplib_3505			201	0	603	605	2	0	201
qplib_3506	50.5	0.8	496	0	0	0	0	0	0
qplib_3508			2450	0	891	99	1332	0	0
qplib_3510			105	0	919	4568	21	0	786
qplib_3511			2450	0	3292	4950	1283	0	0
qplib_3512			72	0	119	403	24	0	119
qplib_3513			123	0	1897	2569	763	0	504
qplib_3514			15	0	1800	960	900	0	0
qplib_3515			352	0	382	720	48	0	382
qplib_3522			42	0	588	212	42	0	0
qplib_3523	50.0	13.2	155	0	27	1456	0	0	27
qplib_3524			132	0	949	3165	192	0	697
qplib_3525	47.6	0.1	0	1662	87	52	39	0	1710
qplib_3529			38	0	1488	1580	544	0	944
qplib_3533			240	0	143	176	25	0	29
qplib_3547	0.0	0.1	462	0	1536	3137	0	0	0
qplib_3549			650	0	1033	1326	583	0	0
qplib_3554	13.9	23.3	14	0	370	556	0	0	0
qplib_3562			7	56	0	35	7	0	56
qplib_3565	50.4	1.4	276	0	0	0	0	0	0
qplib_3580			108	0	24	45	18	0	0
qplib_3582			184	0	32	60	24	0	0
qplib_3584	44.0	8.1	528	0	0	10912	0	0	0
qplib_3587	50.0	12.7	240	0	0	46	0	0	0
qplib_3588			600	0	392	49	584	0	0
qplib_3592	50.0	0.3	225	0	225	255	0	0	0
qplib_3596			104	0	921	1054	132	0	370
qplib_3600			112	0	16	45	12	0	0
qplib_3605			160	0	1076	4315	192	0	818
qplib_3614	50.0	12.7	210	0	0	44	0	0	0
qplib_3620			187	0	3285	4071	1344	0	943
qplib_3621			109	0	1655	2213	665	0	432
qplib_3622			25	0	2000	1040	1000	0	0
qplib_3624			40	0	6400	3280	3200	0	0
qplib_3625			46	0	598	191	46	0	0
qplib_3631			750	0	143	210	25	0	29
qplib_3642	49.8	0.4	1035	0	0	0	0	0	0
qplib_3643			216	72	72	825	68	18	80
qplib_3645			101	0	302	304	2	1	101
qplib_3646			20	0	2000	1050	1000	0	0
qplib_3648			40	0	680	306	40	0	0
qplib_3650	50.5	0.5	946	0	0	0	0	0	0
qplib_3651			137	0	2139	2942	861	0	576
qplib_3659			0	960	4577	5537	960	480	1697
qplib_3661			10816	0	12997	11024	3221	0	0
qplib_3662			144	0	32	55	24	0	0
qplib_3670			54	0	864	305	54	0	0
qplib_3676			30	0	9000	4650	4500	0	0
qplib_3677			30	0	6000	3100	3000	0	0
qplib_3678			200	0	400	402	1	0	200
qplib_3680			92	0	16	40	12	0	0
qplib_3683			126	0	24	48	18	0	0
qplib_3690			20	0	6000	3150	3000	0	0
qplib_3692			128	0	1091	751	528	528	248
qplib_3693	49.4	0.4	1128	0	0	0	0	0	0
qplib_3694	0.0	0.1	40	0	3200	3280	0	0	0
qplib_3697			168	0	32	58	24	0	0
qplib_3698	0.0	0.1	30	0	3000	3100	0	0	0
qplib_3699			116	0	792	1668	192	0	541
qplib_3701			60	0	1080	377	60	0	0
qplib_3703	46.7	84.7	225	0	0	30	0	0	0
qplib_3705	50.6	1.1	378	0	0	0	0	0	0
qplib_3706	50.3	0.6	703	0	0	0	0	0	0
qplib_3708	0.0	0.1	14	0	12916	12917	0	0	0
qplib_3709	48.0	91.9	600	0	0	50	0	0	0
qplib_3713			42	0	630	254	42	0	0
qplib_3714	97.5	32.5	120	0	0	40	0	0	0
qplib_3719			133	0	28	51	21	0	0
qplib_3725			81	0	1171	1552	469	0	288
qplib_3726			116	0	816	2190	192	0	565
qplib_3727			20	0	1600	840	800	0	0
qplib_3728			72	0	16	35	12	0	0
qplib_3729			650	0	408	51	608	0	0

	Q^0		Variables			Constraints			
name	% n.e.	% d.	# b.	# i.	# c.	# l.	# q.	# c.	# b.
qplib_3733			46	0	644	237	46	0	0
qplib_3734			38	0	7533	7690	2754	0	4779
qplib_3738	49.9	0.9	435	0	0	0	0	0	0
qplib_3745	49.6	1.2	325	0	0	0	0	0	0
qplib_3748			75	0	20	37	15	0	0
qplib_3750	98.6	32.9	210	0	0	70	0	0	0
qplib_3751	98.0	32.7	150	0	0	50	0	0	0
qplib_3752	47.7	3.8	462	0	0	6160	0	0	0
qplib_3757	38.1	1.0	552	0	0	8096	0	0	0
qplib_3762	50.0	28.0	90	0	0	480	0	0	0
qplib_3772	50.0	3.9	380	0	0	4560	0	0	0
qplib_3775	98.4	32.8	180	0	0	60	0	0	0
qplib_3780			12	156	0	60	12	0	156
qplib_3785			200	0	32	62	24	0	0
qplib_3790	8.8	23.3	7	0	188	283	0	0	0
qplib_3792	0.0	0.1	20	0	3000	3150	0	0	0
qplib_3794			576	0	986	624	602	0	0
qplib_3797			48	0	296	623	56	0	223
qplib_3798			54	0	810	251	54	0	0
qplib_3803	42.7	14.1	190	0	0	2280	0	0	0
qplib_3809			224	0	32	65	24	0	0
qplib_3813			15	0	2400	1280	1200	0	0
qplib_3814	4.2	0.7	2	0	46	13	28	0	44
qplib_3815	50.0	3.2	192	0	0	64	0	0	0
qplib_3816			70	0	117	363	24	0	117
qplib_3822	49.9	0.5	861	0	0	0	0	0	0
qplib_3825			60	0	1020	317	60	0	0
qplib_3832	50.3	0.7	561	0	0	0	0	0	0
qplib_3834	60.0	98.0	50	0	0	1	0	0	0
qplib_3838	49.7	0.5	780	0	0	0	0	0	0
qplib_3840			2401	0	3334	2499	1374	0	0
qplib_3841	44.0	10.3	300	0	0	4600	0	0	0
qplib_3850	50.3	0.4	1225	0	0	0	0	0	0
qplib_3852	50.7	1.7	231	0	0	0	0	0	0
qplib_3854			40	0	640	266	40	0	0
qplib_3855			400	0	2118	791	1284	0	0
qplib_3856			168	0	183	50	267	0	0
qplib_3857			201	0	602	604	2	0	201
qplib_3859			600	0	968	1225	560	0	0
qplib_3860	44.9	8.7	435	0	0	8120	0	0	0
qplib_3861	0.0	0.1	30	0	4500	4650	0	0	0
qplib_3863			625	0	1053	675	628	0	0
qplib_3865	48.0	90.7	525	0	0	50	0	0	0
qplib_3870	49.0	23.1	116	0	66	1456	0	0	66
qplib_3871	0.0	0.1	25	0	1000	1040	0	0	0
qplib_3872			95	0	1413	1874	567	0	360
qplib_3877	50.4	0.7	630	0	0	0	0	0	0
qplib_3879			10920	0	12906	21945	3026	0	0
qplib_3883	50.0	17.8	182	0	0	1456	0	0	0
qplib_3913	0.0	100.0	300	0	0	61	0	0	0
qplib_3923	53.7	5.2	395	0	0	80	0	0	0
qplib_3931	50.4	4.5	316	0	0	80	0	0	0
qplib_3980	0.0	100.0	235	0	0	48	0	0	0
qplib_4095	0.0	4.0	400	0	1600	1603	400	0	0
qplib_4270	0.0	6.3	400	0	1200	1603	0	0	0
qplib_4455			3000	0	12000	9001	3000	0	0
qplib_4722			2000	0	8000	6001	2000	0	0
qplib_4805			2000	0	8000	6074	2000	0	2000
qplib_5023			3000	0	12000	9155	3000	0	3000
qplib_5442			2000	0	7999	6088	2000	0	1999
qplib_5527	0.0	0.1	4492	0	21117	64348	0	0	12305
qplib_5543	0.0	0.1	4514	0	21186	64096	0	0	12328
qplib_5554			4492	0	30878	64769	4800	0	12158
qplib_5573			4450	0	23692	72976	4800	0	4987
qplib_5577	0.0	0.1	1118	0	4896	15690	0	0	2703
qplib_5721	51.0	76.9	300	0	0	0	0	0	0
qplib_5725	49.9	1.8	343	0	0	0	0	0	0
qplib_5755	50.0	1.0	400	0	0	0	0	0	0
qplib_5875	50.0	78.9	200	0	0	0	0	0	0
qplib_5881	50.9	29.5	120	0	0	0	0	0	0
qplib_5882	50.7	78.2	150	0	0	0	0	0	0
qplib_5909	50.0	9.7	250	0	0	0	0	0	0
qplib_5922	50.2	9.9	500	0	0	0	0	0	0
qplib_5924	0.0	0.1	300	0	15220	36060	0	0	0
qplib_5925			100	0	1301	271	100	0	0
qplib_5926			2400	0	31201	11923	2400	0	0
qplib_5927			2400	0	31201	11963	2400	0	0
qplib_5935	51.0	99.0	100	0	0	1237	0	0	0
qplib_5944	51.0	99.0	100	0	0	2475	0	0	0
qplib_5962	50.7	99.4	150	0	0	2793	0	0	0
qplib_5971	50.7	99.4	150	0	0	5587	0	0	0
qplib_5980	50.7	99.4	150	0	0	8381	0	0	0
qplib_6287			0	0	171	36	81	0	171
qplib_6310			0	0	208	22	390	0	208
qplib_6311			0	0	212	43	128	0	212
qplib_6324	51.0	31.3	640	0	0	16	0	0	0
qplib_6487	51.0	20.9	618	0	0	309	0	0	0
qplib_6597	45.7	97.4	600	0	0	60	0	0	0

	Q^0		Variables			Constraints			
name	% n.e.	% d.	# b.	# i.	# c.	# l.	# q.	# c.	# b.
qplib_6647	70.1	7.3	627	0	0	33	0	0	0
qplib_6757	36.1	4.8	2046	0	0	297	0	0	0
qplib_6764	35.2	4.8	2071	0	0	297	0	0	0
qplib_6799	36.2	4.8	2075	0	0	297	0	0	0
qplib_6941	35.9	4.5	2203	0	0	315	0	0	0
qplib_7127	50.6	6.8	1000	0	0	50	0	0	0
qplib_7139	53.4	89.3	180	0	0	100	0	0	0
qplib_7144	53.2	89.7	220	0	0	121	0	0	0
qplib_7149	53.1	89.7	264	0	0	144	0	0	0
qplib_7154	52.9	89.7	312	0	0	169	0	0	0
qplib_7159	52.5	89.7	364	0	0	196	0	0	0
qplib_7164	52.4	89.7	420	0	0	225	0	0	0
qplib_7579			100	0	200	202	1	1	100
qplib_8009			101	0	303	305	2	0	101
qplib_8153			31	0	93	95	2	0	31
qplib_8381			51	0	153	155	2	0	51
qplib_8495	0.0	0.1	0	0	27543	8000	0	0	0
qplib_8505	0.0	0.1	0	0	20050	10001	0	0	20050
qplib_8515	0.0	0.1	0	0	16002	8002	0	0	8001
qplib_8559	0.0	0.1	0	0	10000	5000	0	0	10000
qplib_8567	0.0	0.1	0	0	10000	7500	0	0	10000
qplib_8602	0.0	0.1	0	0	34552	52983	0	0	34552
qplib_8605	0.0	0.1	0	0	5000	0	1	1	0
qplib_8616	0.0	0.1	0	0	13870	10404	0	0	4
qplib_8685	0.0	0.2	0	0	772	0	10000	0	0
qplib_8777	34.6	0.1	0	0	10000	2500	0	0	10000
qplib_8785	0.0	0.1	0	0	10399	11362	0	0	10399
qplib_8790	0.0	0.1	0	0	39204	0	0	0	19602
qplib_8792	0.0	0.1	0	0	15129	0	0	0	15129
qplib_8845	0.0	4.9	0	0	1546	777	0	0	15
qplib_8906	0.0	0.5	0	0	5223	838	0	0	0
qplib_8938	0.0	0.1	0	0	4001	11999	0	0	0
qplib_8991	0.0	0.1	0	0	14400	0	0	0	14400
qplib_9002	0.0	0.1	0	0	2890	1649	0	0	727
qplib_9004	0.0	0.1	0	0	40000	10001	10001	10001	20000
qplib_9030	0.3	0.1	0	10000	0	5000	0	0	10000
qplib_9048	29.8	18.3	0	202	0	1	0	0	202

B. The file format

Following the notation of Section 2, a generic Quadratic Program (QP) can be written as follows:

$$\begin{aligned} \min\ & x^\top Q^0 x + b^0 x + f \\ & c_l^i \le x^\top Q^i x + b^i x \le c_u^i && i \in \mathcal{M} \\ & l_j \le x_j \le u_j && j \in \mathcal{N} \\ & x_j \in \mathbb{Z} && j \in \mathcal{Z} \end{aligned}$$

where:

- $\mathcal{N} = \{1, \ldots, n\}$ is the set of (indices) of variables, and $\mathcal{M} = \{1, \ldots, m\}$ is the set of (indices) of constraints;
- $x = [x_j]_{j=1}^n \in \mathbb{R}^n$ is a finite vector of real variables;
- Q^i for $i \in \{0\} \cup \mathcal{M}$ are symmetric $n \times n$ real matrices;
- b^i, c_l^i, c_u^i for $i \in \{0\} \cup \mathcal{M}$ and f are, respectively, real n-vectors, real constants, real constants and real constant;
- $-\infty \le l_j \le u_j \le \infty$ are the (extended) real upper and lower bounds on each variable x_j for $j \in \mathcal{N}$;
- $\mathcal{M} = \mathcal{Q} \cup \mathcal{L}$ where $Q^i = 0$ for all $i \in \mathcal{L}$;
- the variables in $\mathcal{Z} \subseteq \mathcal{M}$ are restricted to only attain integer values.

The QPLIB format is defined in Table 7. The data is in free format (blanks separate values), but must occur in the order given here. Any blank lines, or lines starting with any of the characters !, % or # are ignored. Each term in the first column of Table 7 denotes a required value. Any strings beyond those required on a given line will be regarded as comments and ignored. Real values may either by in decimal or exponential formats; for the latter, the exponent may be preceded by either the character `D` or `E`, e.g. `12.56D+2` or `12.56E+2`.

Table 7: The QPLIB file format: refer to the notes after the table for more details.

data	description	note
name	problem name (character string)	
type	problem type (character string)	[1]
sense	one of the words minimize or maximize (character string, case irrelevant)	
n	number of variables (integer)	
m	number of constraints (integer)	[2]
n^{Q^0}	number of nonzeros (integer) in lower triangle of Q^0	[3]
$h \;\; k \;\; Q^0_{hk}$	row and column indices (integers) and value (real) for each nonzero entry of Q^0, if $n^{Q^0} > 0$, one triple on each line	
b^0_d	default value (real) for entries in b^0	
n^{b^0}	number of non-default entries (integer) in b^0	
$j \;\; b^0_j$	index (integer) and value (real) for each non-default term in b^0, if $n^{b^0} > 0$, one pair per line	
f	constant part of the objective function	
n^{Q^i}	number of nonzeros (integer) in lower triangle of Q^i	[2,4]
$i \;\; h \;\; k \;\; Q^i_{hk}$	constraint, row and column indices (integers) and value (real) for each entry of Q^i, if n^{Q^i}, one quadruple on each line	
n^b	number of nonzeros (integer) in b^i, $i \in \mathcal{M}$	[2]
$i \;\; j \;\; b^i_j$	row and column indices (integers) and value (real) for each nonzero entry of b^i, if $n^b > 0$, one triple on each line	
∞	value (real) for infinity for constraint or variable bounds—any bound greater than or equal to this in, absolute value, is infinite	
$c_{l,d}$	default value (real) for entries in c_l	[2]
$n^{c_{l,d}}$	number of non-default entries (integer) in c_l	[2]
$i \;\; c^i_l$	index (integer) and value (real) for each non-default term in $c_{l,d}$, if $n^{c_{l,d}} > 0$, one pair per line	[2]
$c_{u,d}$	default value (real) for entries in c_u	[2]
$n^{c_{u,d}}$	number of non-default entries (integer) in c_u	[2]
$i \;\; c^i_u$	index (integer) and value (real) for each non-default term in $c_{u,d}$, if $n^{c_{u,d}} > 0$, one pair per line	
l_d	default value (real) for entries in l	
n^{l_d}	number of non-default entries (integer) in l	
$i \;\; l_i$	index (integer) and value (real) for each non-default term in l, if $n^{l_d} > 0$, one pair per line	
u_d	default value (real) for entries in u	
n^{u_d}	number of non-default entries (integer) in u	
$i \;\; u_i$	index (integer) and value (real) for each non-default term in u, if $n^{u_d} > 0$, one pair per line	
v_d	default variable type (integer, 0 for continuous variables, 1 for integer variables)	[5]
n^v	number of non-default variables (integer)	[5]
$i \;\; v_i$	index and type (integers) for each non-default variable type, if $n^v > 0$, one pair per line	[5]
x^0_d	default value (real) for the components of the starting point x^0 for the variables x	
n^{x^0}	number of non-default starting entries (integer) in x	
$i \;\; x^0_i$	index (integer) and value (real) for each non-default starting value in x^0, if $n^{x^0} > 0$, one pair per line	
y^0_d	default value (real) for the components of the starting point y^0 for the Lagrange multipliers y for the general constraints	[2]
n^{y^0}	number of non-default starting entries (integer) in y	[2]

Table 7: The QPLIB file format (continued)

data	description	note
i y_i^0	index (integer) and value (real) for each non-default starting value in y^0, if $n^{y^0} > 0$, one pair per line	[2]
z_d^0	default value (real) for the components of the starting point z^0 for the dual variables z for the simple bound constraints	
n^{z^0}	number of non-default starting entries (integer) in z	
i z_i^0	index (integer) and value (real) for each non-default starting value in z^0, if $n^{z^0} > 0$, one pair per line	
n_d^x	number of non-default names (integer) of variables—default for variable i is the character string representing the numerical value i	
j `var_name`$_j$	index (integer) and name (character string) for each non-default variable name, if $n_d^x > 0$, one pair per line	
n_d^c	number of non-default names (integer) of general constraints—default for constraint i is the character string representing the numerical value i	
i `cons_name`$_i$	index (integer) and name (character string) for each non-default constraint name, if $n_d^c > 0$, one pair per line	

Stefan: I would suggest to make it simpler, even though that would reduce flexibility. Always fix the default value for coefficients to 0 (otherwise, how you parse this? First initialize a n x n matrix with the non-zero default and then overwrite with the given entries, which may then be zero?) and have no default bounds for variables. If you then also adapt the suggestion to use a constraint sense ($\leq$,$\geq$,$=$) and a right-hand-side instead of a left- and right-hand-side, then there is also no need to specify a value for infinity. This would leave out ranged constraints - but are they really used? One could do it like CPLEX and have an extra constraint sense for ranges and an extra section to specify the range.

[1] Fabio and Emiliano: we think that, if possible, we should follow the classification proposed in Section 2.

The problem type is represented by a character string as one of the following:

For continuous problems (i.e., all variables are continuous):

- `LP` a linear program,
- `LPQC` a linear program with quadratic constraints,
- `BQP` a bound-constrained quadratic program,
- `QP` a quadratic program, and
- `QPQC` a quadratic program with quadratic constraints.

For integer problems (i.e., all variables are integer valued):

- `ILP` an integer linear program,
- `ILPQC` an integer linear program with quadratic constraints,
- `IBQP` an integer bound-constrained quadratic program,
- `IQP` an integer quadratic program, and
- `IQPQC` an integer quadratic program with quadratic constraints.

For mixed-integer problems (i.e., there is a mix of continuous and integer variables):

- `MILP` a mixed-integer linear program,
- `MILPQC` a mixed-integer linear program with quadratic constraints,
- `MIBQP` a mixed-integer bound-constrained quadratic program,
- `MIQP` a mixed-integer quadratic program, and
- `MIQPQC` a mixed-integer quadratic program with quadratic constraints.

Stefan: I agree that leaving out information on numerical properties of the matrices and vectors (convexity, etc.) is good here. Such properties may also depend on tolerances, etc. As above, I would suggest to eliminate LPQC and rename QPQC into QCP. Having a string for "unspecified" may be nice, too, or one just uses MIQCQP in this case.

[2] for bound-constrained QPs, these sections are omitted.
[3] for linear program with quadratic constraints, this section is omitted.
[4] for problems without quadratic constraints, this section is omitted.
[5] for purely-continuous or purely-integer problems, this section is omitted.

As a simple example, consider the mixed-integer QP

$$\min_{x\in\mathbb{R}^3} x_1^2 + x_2^2 + x_3^2 - x_1x_2 - x_2x_3 - 0.2x_1 - 0.4x_2 - 0.2x_3$$
$$\text{subject to } 1 \le x_1 + x_2,\ 1 \le x_1 + x_3,\ 0 \le x_1 \le 1,\ 0 \le x_2 \le 2, \text{ and binary } x_3.$$

This may then be represent in QPLIB format as follows:

```
! ---------------
! example problem
! ---------------
MIPBAND   # problem name
MIQP      # problem is a mixed-integer quadratic program
Minimize  # minimize the objective function
3         # variables
2         # general linear constraints
5         # nonzeros in lower triangle of Q^0
1 1 2.0   5 lines of row & column index & value of nonzeros in lower triangle Q^0
2 1 -1.0  |
2 2 2.0   |
3 2 -1.0  |
3 3 2.0   |
-0.2      default value for entries in b_0
1         # non default entries in b_0
2 -0.4    1 lines of index & value of non-default values in b_0
0.0       value of f
4         # nonzeros in vectors b^i (i =0,...,m)
1 1 1.0   4 lines of row & column index & value of nonzeros in b^i (i =0,...,m)
1 2 1.0   |
2 1 1.0   |
2 3 1.0   |
1.0E+20   infinity
1.0       default value for entries in c_l
0         # non default entries in c_l
1.0E+20   default value for entries in c_u
0         # non default entries in c_u
0.0       default value for entries in l
0         # non default entries in l
1.0       default value for entries in u
1         # non default entries in u
2 2.0     1 line of non-default indices and values in u
0         default variable type is continuous
1         # non default variable types
3 1       variable 3 is integer
1.0       default value for initial values for x
0         # non default entries in x
0.0       default value for initial values for y
0         # non default entries in y
```

```
0.0       default value for initial values for z
0         # non default entries in z
0         # non default names for variables
0         # non default names for constraints
```