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

QPLIB: a Library of a Quadratic Programming Instances

Fabio Furini · Emiliano Traversi · Alper Atamturk · Pietro Belotti · Pierre Bonami · Sourour Elloumi · Antonio Frangioni · Ambros Gleixner · Nick Gould · Leo Liberti · Andrea Lodi · Ruth Misener · Hans Mittelmann · Nick Sahinidis · Frederic Roupin · 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

Alper Atamturk

Pietro Belotti

Pierre Bonami

Sourour Elloumi

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

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

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

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 indecidable. Correspondingly, solution methods for QP are very diverse, ranging from entirely combinatorial ones to completely continuous ones, to many where both aspects are fundamental. Selecting a set of instances of QP that is at the same time not overwhelmingly numerous and significant for the many different interested communities is therefore challenging. In order to help our selection process we propose a simple taxonomy for QP instances. 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 how the final results look.

Keywords Instance Library, Quadratic Programming

Mathematics Subject Classification (2000) 90C06 · 90C25

1. Introduction

Quadratic Programming (QP) problems, where both the objective function and the constraints contain (at most) expressions in the variables of degree 2, contain a surprisingly diverse set of rather different problems. 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. It is perhaps therefore not surprising that, unlike for "simpler" problems classes [6], 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 instances of QP 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.

Nick Sahinidis

Frederic Roupin

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

Angelika Wiegele

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 solution is 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. This also takes into account the fact that if very few (maybe one) solver 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

$\boxed{\text{TASK 2 : write motivation}}$

Optimization problems with quadratic constraints and/or objective function (QP) have been the subject of a considerable amount of research in the last decade. 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 (**citations, like copositive/standard QP?**).

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 practical problems (**citations?**). 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.

...

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

$$\begin{array}{lll} \min & x^\top Q^0 x + L^0 x & \\ & x^\top Q^i x + L^i x \leq C^i & i \in \mathcal{M} \\ & l_j \leq x_j \leq u_j & j \in \mathcal{N} \\ & x_j \in \mathbb{Z} & j \in \mathcal{Z} \end{array}$$

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 $(Q^i_{hk} + Q^i_{kh})/2$;

- L^i and C^i for $i \in \{0\} \cup \mathcal{M}$ are, respectively, real n-vectors and real constants;
- $-\infty \leq l_j < u_j \leq \infty$ are the (extended) real upper and lower bounds on each variable x_j for $j \in \mathcal{N}$, assumed different because otherwise the variable is fixed;
- $\mathcal{M} = \mathcal{Q} \cup \mathcal{L}$ 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 "*FVC*", where F 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 threee types);
- constraints: (N)one, (B)ox, (L)inear, (C)onvex quadratic, nonconvex (Q)uadratic.

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 F 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 SDP 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

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

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

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: this can be attained trivially in $O(1)$ 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 $L^0 = 0$) will not be discussed here. Similarly, CCB (and, a fortiori, 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.

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

is not only $\mathcal{NP}$-hard, but downright undecidable [5]. 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 example in the introduction, 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

$$\begin{array}{lll} \min & x^0 & \\ & x^\top Q^0 x + L^0 x \le x^0 & \\ & x^\top Q^i x + L^i x \le C^i & i \in \mathcal{M} \\ & l_j \le x_j \le u_j & j \in \mathcal{N} \\ & x_j \in \mathbb{Z} & j \in \mathcal{Z} \end{array}$$

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 (SDP), i.e., the corresponding constraint/objective function is convex, general quadratic constraints are in fact equivalent to diagonal ones. In fact, every SDP 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. Conversely, there is usually little gain in distinguishing between **C problems and "**D" ones where the constraints are restricted to be diagonal, as there are much fewer cases where this naturally occurs (**to the best of my knowledge**). This is why we did not include a "D" option for the constraints.

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 [6], so they won't be explicitly addressed here.
- *Convex Continuous Quadratic Programs* CCC can be solved in polynomial time by Interior-Point techniques; the simpler CCL can also be solved by means of "simplex-like" techniques (**citations?**). 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 [4]. 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 local optima (or at least saddle points) of the CQC, but providing global guarantees about the quality of the obtained solutions is challenging. 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 (**some citations? some specific problem to mention apart from Max-Cut?**).
– *Nononvex Integer Quadratic Programs* QGQ is the more general, and therefore is the most general, 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 (**citations?**).

This description is purposely coarse; each of these classes can be subdivided into several other ones 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 of some 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 1). 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 1 can, following [9], 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 [9], *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 to find a feasible starting solution if there is one (the so-called "phase 0").

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. 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 1 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 two types of local search methods for NLP:

– active set methods (ConOpt, Minos, SnOpt);
– interior point methods (IpOpt, KNitro).

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 subject to all 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 subproblem is defined in such a way as to be (a) easier to solve than the original problem, and (b) helping the sequence x^k to converge to a Karush-Kuhn-Tucker (KKT) point.

Interior point methods. These approaches can be seen as recasting the original constrained problem QP as a parametrized family of unconstrained ones

QP$_\mu$, 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$_\mu$ 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$_\mu$ 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.

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 1 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 [5]; and yet, there exists a general decision method for deciding feasibility of systems of polynomial equations and inequalities [11], 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 [3]. 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 [9].

Branch-and-Bound. Nearly all of the complete solvers in Table 1 that address $\mathcal{NP}$-hard problems (i.e. those in QGQ$\smallsetminus$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 [7,1], 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 $\hat{t} \leq \overline{t}(x)$. 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 [8,2].

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 1 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, which implement complete BB algorithms for CGC. **how about mosek?**

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 [2].

	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	I	I	I	I	C	I
KNitro_bb	C	?	C		C	I
lindo	C	C	C	C	C	C
lindoglobal	C	C	C	C	C	C
SCIP	C	C	C	C	C	C
oqnlp	?	?	?	?	C	A
AlphaECP	C	C	C	C	C	C
BonMin	C	?	C	?	C	?
DICOpt	C	C	C	C	C	C
sBB	C	?	C	?	C	?
ConOpt					C	I
IpOpt					C	I
lgo_bb	C	C	C	C	C	C
lgo_gars	?	?	?	?	C	A
lgo_ms	?	?	?	?	C	A
Minos	?	?	?	?	C	I
msnlp	?	?	?	?	C	I
SnOpt	?	?	?	?	C	I
XPress	C	?	C	?	C	?
GuRoBi	C	?	C		C	?
CPLEX	C	C	C		C	I
mosek	C	?	C	?	C	?

Table 1 Families of QP problems that can be tackled each solver

Cutting plane approaches . The two remaining solvers in Table 1 are AlphaECP [12] and DICOpt [10], 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

We now provide a succinct list of the solvers we have tested, using the approaches described in §2.3. In Table 1, 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 library. In particular we describe the first set of gathered instances (Section 3.1), we

Starting set	$\approx$ 8500 Instances	
	$\Downarrow$	
	$\approx$ 6000 Discr. Inst.	$\approx$ 2500 Cont. inst.
First Filter	$\Downarrow$	
	$\approx$ 3000 Discr. Inst.	$\Downarrow$
Second Filter	$\Downarrow$	
	600 Discr. Inst.	250 Cont. inst.

Table 2 Instance filter steps

discuss the issues concerning the format of the instances (Section 3.2), we present the feature used to classified the instances (Section 3.3), and finally we then describe the selection process that we have used to filter the instances in order to construct the final library (Section 3.4).

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 to the main international mailing lists of the mathematical optimization and numerical analysis community, in order to reach the largest possible set of interested researchers, both in academia and in industry. The call remained open for 10 months, during which we received a large number of contributions with different characteristics. The instances we received are both artificial and coming from real-world applications.

In addition to spontaneous contribution we scanned the other known libraries of instances and we selected all the QP ones. In particular we cite here the libraries of "generic" QP instances from which we draw material:

– `POLIP` `http://polip.zib.de/pipformat.php`
– `MINLP` `http://www.gamsworld.org/minlp/minlplib.htm`
– `MacMINLP` `https://wiki.mcs.anl.gov/leyffer/index.php/MacMINLP`
– `Meszaros` `http://www.doc.ic.ac.uk/~im/00README.QP`
– `BARON Library` `http://www.minlp.com/nlp-and-minlp-test-problems`
– `GAMS Library` `http://www.gamsworld.org/performance/performlib.htm`

Other QP instances were found in libraries devoted to specific optimization problems that can be modeled as QP, such as Max-Cut, the Quadratic Assignment Problem, Portfolio Optimization problems and several others; for reason of space we refrain from listing all these sources.

At the end of this process we had gathered more than eight thousand instances. Three fourths of them contained discrete variables, while the remaining ones contained only continuous variables.

Obj. Fun.	Variables	Constraints	#
Linear	Binary	Quadratic	71
	Mixed	Convex	5
		Quadratic	277
	General	Convex	22
		Quadratic	8
Convex	Binary	Linear	12
	Mixed	Linear	37
		Quadratic	3
	General	Linear	1
Quadratic	Binary	None	29
		Linear	180
	Mixed	Linear	12
		Quadratic	9
	Integer	Linear	2
	General	Linear	1
		Quadratic	2
Total			600

Table 3 Classification of the final set of discrete instances

First Instances Filter. As already mentioned an important characteristic of an instances is its computational difficulty, i.e., the CPU time needed by a complete solver (cf. §2.3) to solve the instance to global optimality. Accordingly, for each of the gathered instance we run the solvers in `GAMS` (see Table **??**) able to solve it to global optimality. The number of solvers depends on the category of the instances under consideration. Then in order to reduce the number of instances we performed a first filter based on a relative measure of computational difficulty, i.e., we discarded all "easy" instances which are solved by at least 30% of the complete solvers within a time limit of 30 seconds. Thanks to this first filter we obtained what we call the *filtered test bed.*

The characteristics of the instances in the final library are presented in Table 3.1 for *discrete* instances (*{B,M,I,G}*) and in Table 3.1 for *continuous* ones (*C*).

3.2 Instance Format

$\boxed{\text{TASK X : write}}$ In the call for instances no specific formats 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 solvers. For this reason, we decided to translate all instances into the `.gms` gams format. In a preliminary phase, all the instances received were divided according to their format and subsequently translated. In Sect. the tools used to translate an instance from a given format to the `.gms` format are described

Obj. Fun.	Constraints	#
Linear	Convex	14
	Quadratic	71
Convex	None	6
	Box	6
	Linear	50
	Convex	3
	Quadratic	6
Quadratic	Box	8
	Linear	19
	Convex	27
	Quadratic	40
Total		250

Table 4 Classification of the final set of continuous instances

more in detail.

As second format, we introduced a specific format `.qplib`. This new format is capable to describe 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. ToDo: describe `.qplib` format. Maybe in an appendix?

3.3 Instance Features

TASK X : write

For each instance of the starting set, the following features have been collected:

- Objective function characteristics:
  - Type of objective function: Linear, Convex or Quadratic.
  - Density of the objective function, i.e. the percentage of nonzero entries of Q_0.
  - Percentage of negative eigenvalues of Q_0.
- Variables characteristics:
  - Number of Continuous, Binary and Integer variables.
- Constraints characteristics:
  - Number of Linear, Convex and Quadratic constraints.
  - Density of the constraints, i.e. the percentage of nonzero entries of the coefficients of the Linear, Continuous and Quadratic constraints.

The mentioned features

- constraint-types: big-M constraints? box-constraints? combinatorial constraints? linear network?
- nonconvex: whether they can be turned into cones through appropriate decomposition

- quadratic: are discrete variables contained in the quadratic part or in the linear part only?
- note: some remark like ”max-cut”, ”quadratic knapsack”, etc.

Solver-dependent characteristics:

- time (?) [how measure time? what architecture will be used for tests? how to deal with parallel machines/algorithms?]
- number of nodes in case of branch-and-bound algorithms
- memory (?)
- matrix-free access (?)
- lower bound at root node
- time of finding first feasible solution

- The static analysis is not enough to identify the hardness of the instances.
- An empirical way for testing the hardness of one instance is the time needed to solve it.
- We decided to use a broad set of solvers to test the computational hardness of the instances.

3.4 Instance Selection

$\boxed{\text{TASK X : write}}$

4. Software tools

4.1 instance translator

GAMS–LP–QPFORMAT $\boxed{\text{TASK X : write}}$

4.2 code that computes the features of an instance

$\boxed{\text{TASK X : write}}$

4.3 code that selects subsets of instances

$\boxed{\text{TASK X : write}}$

4.4 website, instance collector

Select subset or categories of instances (EXTRACT FROM THE LIBRARY A SUBSET OF INSTANCES WITH SPECIFIC CHARACTERISTICS) $\boxed{\text{TASK X : write}}$

4.5 testing environment

RUN GAMS USING A SUBSET OF SOLVERS $\boxed{\text{TASK X : write}}$

5. Conclusions

6. Acknowledgements

References

1. 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)
2. 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)
3. 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)
4. 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)
5. Jeroslow, R.: There cannot be any algorithm for integer programming with quadratic constraints. Operations Research **21**(1), 221–224 (1973)
6. 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)
7. McCormick, G.: Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems. Mathematical Programming **10**, 146–175 (1976)
8. Misener, R., Floudas, C.: GloMIQO: Global Mixed-Integer Quadratic Optimizer. Journal of Global Optimization **57**, 3–50 (2013)
9. Neumaier, A.: Complete search in continuous global optimization and constraint satisfaction. Acta Numerica **13**, 271–369 (2004)
10. Quesada, I., Grossmann, I.: Global optimization of bilinear process networks and multicomponent flows. Computers & Chemical Engineering **19**(12), 1219–1242 (1995)
11. Tarski, A.: A decision method for elementary algebra and geometry. Tech. Rep. R-109, Rand Corporation (1951)
12. Westerlund, T., Pörn, R.: Solving pseudo-convex mixed integer optimization problems by cutting plane techniques. Optimization and Engineering **3**, 235–280 (2002)

Appendix

In this Appendix we provide detailed data on all the instances of the final library. This is done in Tables 6–6 for *discrete* instances (*{B,M,I,G}*) and in Tables 6–6 for *continuous* ones (*C*). In the former, the features of the instances are described by three sets of columns. The first ("% eig") 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**), "0.000" 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 following three columns describe the variables by reporting the number of binary ones ("# bin"), general integer ones ("# int"), and continuous ones ("# cont"). Finally, the last three columns describe the constraints reporting the number of linear ones ("# lin"), nonconvex quadratic ones ("# quad"), and convex quadratic ones ("# conv"). Tables 6–6 are similarly structured except that all variables are continuous, and hence only one column is present.

			Variables			Constraints			
name	% eig	% dens	# bin	# int	# cont	# lin	# quad	# conv	# box
qplib_1	2e-1	2e-1	30	0	30	32	0	0	30
qplib_2	3e-1	2e-1	50	0	50	52	0	0	50
qplib_3	5e-1	3e-1	90	0	0	1	0	0	90
qplib_4	5e-1	5e-1	100	0	0	1	0	0	100
qplib_5	5e-1	5e-1	90	0	0	1	0	0	90
qplib_6	5e-1	7e-1	100	0	0	1	0	0	100
qplib_7	5e-1	5e-1	70	0	0	1	0	0	70
qplib_8	5e-1	9e-1	90	0	0	1	0	0	90
qplib_9	4e-1	7e-1	50	0	0	1	0	0	50
qplib_10	5e-1	7e-1	80	0	0	1	0	0	80
qplib_11	5e-1	9e-1	80	0	0	1	0	0	80
qplib_12	5e-1	1e-1	144	0	0	24	0	0	144
qplib_13	3e-1	1e-2	536	0	0	138	0	0	536
qplib_14	5e-1	1e-2	750	0	0	253	0	0	750
qplib_15	5e-1	4e-2	400	0	0	104	0	0	400
qplib_16	5e-1	4e-2	300	0	0	103	0	0	300
qplib_17	5e-1	3e-2	300	0	0	103	0	0	300
qplib_18	5e-1	3e-2	300	0	0	103	0	0	300
qplib_19	2e-1	1e-2	402	0	0	137	0	0	402
qplib_20	5e-1	2e-2	496	0	0	128	0	0	496
qplib_21	5e-1	3e-2	300	0	0	103	0	0	300
qplib_22	5e-1	3e-2	400	0	0	104	0	0	400
qplib_23	5e-1	4e-2	400	0	0	104	0	0	400
qplib_24	5e-1	3e-2	400	0	0	104	0	0	400
qplib_25	4e-1	6e-3	1056	0	0	355	0	0	1056
qplib_26	5e-1	3e-2	400	0	0	104	0	0	400
qplib_27	3e-1	1e-2	670	0	0	139	0	0	670
qplib_28	4e-1	2e-2	372	0	0	127	0	0	372
qplib_29	5e-1	4e-2	372	0	0	127	0	0	372
qplib_30	5e-1	9e-1	50	0	0	1	0	0	50
qplib_31	6e-1	1e+0	50	0	0	1	0	0	50
qplib_32	5e-1	2e-1	50	0	0	1	0	0	50
qplib_33	5e-1	7e-1	50	0	0	1	0	0	50
qplib_34	5e-1	2e-1	75	0	0	1	0	0	75
qplib_35	5e-1	9e-1	75	0	0	1	0	0	75
qplib_36	6e-1	9e-1	75	0	0	1	0	0	75
qplib_37	5e-1	1e+0	75	0	0	1	0	0	75
qplib_38	5e-1	7e-1	75	0	0	1	0	0	75
qplib_39	5e-1	7e-1	50	0	0	1	0	0	50
qplib_40	6e-1	9e-1	75	0	0	1	0	0	75
qplib_41	5e-1	9e-1	50	0	0	1	0	0	50
qplib_42	5e-1	9e-1	75	0	0	1	0	0	75
qplib_43	6e-1	1e+0	75	0	0	1	0	0	75
qplib_44	5e-1	2e-1	50	0	0	1	0	0	50
qplib_45	5e-1	7e-1	75	0	0	1	0	0	75
qplib_46			9600	0	6497	8417	960	480	10337
qplib_47			87	0	205	730	48	0	246
qplib_48			87	0	299	1002	96	0	294
qplib_49			104	0	328	1032	96	0	336
qplib_50			124	0	412	1508	128	0	408
qplib_51			304	0	760	2868	192	0	872
qplib_52			760	0	2220	8196	640	0	2340
qplib_53			760	0	2540	9348	800	0	2500
qplib_54			2040	0	5500	28256	1600	0	5940
qplib_55			792	0	1436	11924	288	0	1940
qplib_56			6520	0	13340	128792	3200	0	16660
qplib_57			187	0	240	423	33	0	394
qplib_58			55	0	78	141	15	0	118
qplib_59	0e+0	3e-4	720	0	240	5329	0	0	730
qplib_60	5e-1	1e-1	250	0	0	1	0	0	250

Table 5 Discrete Instance Feature 1-60

			Variables			Constraints			
name	% eig	% dens	# bin	# int	# cont	# lin	# quad	# conv	# box
qplib_61	5e-1	1e-1	250	0	0	1	0	0	250
qplib_62	5e-1	1e-1	250	0	0	1	0	0	250
qplib_63	5e-1	1e-1	250	0	0	1	0	0	250
qplib_64	5e-1	1e-1	250	0	0	1	0	0	250
qplib_65	3e-1	6e-2	152	0	17	153	16	0	152
qplib_66	4e-1	5e-2	252	0	22	253	21	0	252
qplib_67	4e-1	4e-2	275	0	23	276	22	0	275
qplib_68	4e-1	4e-2	299	0	24	300	23	0	299
qplib_69	4e-1	4e-2	299	0	24	300	23	0	299
qplib_70	4e-1	4e-2	324	0	25	325	24	0	324
qplib_71	4e-1	4e-2	324	0	25	325	24	0	324
qplib_72	4e-1	4e-2	324	0	25	325	24	0	324
qplib_73			136	0	17	2057	17	0	136
qplib_74			153	0	18	2466	18	0	153
qplib_75			153	0	18	2466	18	0	153
qplib_76			153	0	18	2466	18	0	153
qplib_77			171	0	19	2926	19	0	171
qplib_78			171	0	19	2926	19	0	171
qplib_79			171	0	19	2926	19	0	171
qplib_80			171	0	19	2926	19	0	171
qplib_81			171	0	19	2926	19	0	171
qplib_82			190	0	20	3440	20	0	190
qplib_83			190	0	20	3440	20	0	190
qplib_84			190	0	20	3440	20	0	190
qplib_85			190	0	20	3440	20	0	190
qplib_86			190	0	20	3440	20	0	190
qplib_87			190	0	20	3440	20	0	190
qplib_88			210	0	21	4011	21	0	210
qplib_89			210	0	21	4011	21	0	210
qplib_90			210	0	21	4011	21	0	210
qplib_91			231	0	22	4642	22	0	231
qplib_92			253	0	23	5336	23	0	253
qplib_93			253	0	23	5336	23	0	253
qplib_94			253	0	23	5336	23	0	253
qplib_95			276	0	24	6096	24	0	276
qplib_96			276	0	24	6096	24	0	276
qplib_97			300	0	25	6925	25	0	300
qplib_98			300	0	25	6925	25	0	300
qplib_99			300	0	25	6925	25	0	300
qplib_100			192	0	2	65	1	0	192
qplib_101			683	0	1376	1366	683	0	683
qplib_102			345	0	697	690	345	0	345
qplib_103			61	0	131	122	61	0	61
qplib_104			214	0	438	428	214	0	214
qplib_105			297	0	608	594	297	0	297
qplib_106			351	0	736	702	351	0	351
qplib_107			150	0	305	300	150	0	150
qplib_108			150	0	305	300	150	0	150
qplib_109			215	0	436	430	215	0	215
qplib_110			768	0	1545	1536	768	0	768
qplib_111			90	0	190	180	90	0	90
qplib_112			90	0	195	180	90	0	90
qplib_113			90	0	200	180	90	0	90
qplib_114			90	0	205	180	90	0	90
qplib_115			90	0	185	180	90	0	90
qplib_116			100	0	205	200	100	0	100
qplib_117			110	0	225	220	110	0	110
qplib_118			958	0	1926	1916	958	0	958
qplib_119			194	0	421	388	194	0	194
qplib_120			0	100	2	4	1	1	0

Table 6 Discrete Instance Feature 61-120

			Variables			Constraints			
name	% eig	% dens	# bin	# int	# cont	# lin	# quad	# conv	# box
qplib_121			0	100	2	4	1	1	0
qplib_122			0	100	2	4	1	1	0
qplib_123			0	100	2	4	1	1	0
qplib_124			0	100	2	4	1	1	0
qplib_125			0	100	2	4	1	1	0
qplib_126			0	100	2	4	1	1	0
qplib_127			595	0	2	13091	1	0	595
qplib_128			435	0	2	8121	1	0	435
qplib_129			240	0	2	2241	1	0	240
qplib_130			240	0	2	2241	1	0	240
qplib_131			240	0	2	2241	1	0	240
qplib_132			240	0	2	2241	1	0	240
qplib_133			147	0	95	2241	1	0	240
qplib_134			240	0	2	2241	1	0	240
qplib_135			240	0	2	2241	1	0	240
qplib_136			240	0	2	2241	1	0	240
qplib_137			240	0	2	2241	1	0	240
qplib_138			306	0	2	3265	1	0	306
qplib_139			306	0	2	3265	1	0	306
qplib_140			190	0	2	2281	1	0	190
qplib_141	0e+0	3e-4	28	0	477	576	0	0	56
qplib_142	5e-1	9e-1	676	0	0	52	0	0	676
qplib_143	4e-1	9e-1	196	0	0	28	0	0	196
qplib_144	6e-1	3e-1	256	0	0	32	0	0	256
qplib_145	5e-1	8e-1	100	0	0	20	0	0	100
qplib_146			42	0	86	211	14	0	58
qplib_147			42	0	86	211	14	0	58
qplib_148			25	0	38	31	26	26	30
qplib_149	1e-2	2e-4	0	2213	4191	483	0	0	2213
qplib_150	5e-1	6e-1	625	0	0	50	0	0	625
qplib_151	4e-1	2e-1	1024	0	0	64	0	0	1024
qplib_152	3e-1	1e-1	256	0	0	32	0	0	256
qplib_153	5e-2	1e-2	259	0	1	212	0	0	259
qplib_154			108	0	568	369	30	0	661
qplib_155	4e-1	9e-1	324	0	0	36	0	0	324
qplib_156	5e-1	7e-2	625	0	0	50	0	0	625
qplib_157			30	0	68	157	12	0	44
qplib_158			42	0	86	211	14	0	58
qplib_159	3e-3	2e-4	252	0	1499	1913	0	0	1714
qplib_160	5e-1	9e-1	625	0	0	50	0	0	625
qplib_161			56	0	106	273	16	0	74
qplib_162	0e+0	2e-2	150	0	1	68	0	0	151
qplib_163	5e-1	3e-1	225	0	0	30	0	0	225
qplib_164			2	0	33	31	6	0	29
qplib_165			25	0	368	298	24	0	120
qplib_166	7e-1	6e-1	256	0	0	32	0	0	256
qplib_167			72	0	128	343	18	0	92
qplib_168	5e-1	6e-1	484	0	0	44	0	0	484
qplib_169			42	0	86	211	14	0	58
qplib_170	5e-1	9e-1	225	0	0	30	0	0	225
qplib_171	2e-2	3e-3	256	0	256	296	0	0	256
qplib_172	5e-1	1e-1	1024	0	0	64	0	0	1024
qplib_173			24	0	189	213	48	30	40
qplib_174	5e-1	9e-1	2500	0	0	100	0	0	2500
qplib_175	5e-1	4e-1	144	0	0	24	0	0	144
qplib_176	2e-3	2e-4	48	0	792	1192	0	0	48
qplib_177	4e-1	8e-1	144	0	0	24	0	0	144
qplib_178	5e-1	9e-1	676	0	0	52	0	0	676
qplib_179	2e-2	4e-3	169	0	169	195	0	0	169
qplib_180	5e-1	1e-1	144	0	0	24	0	0	144

Table 7 Discrete Instance Feature 121-180

			Variables			Constraints			
name	% eig	% dens	# bin	# int	# cont	# lin	# quad	# conv	# box
qplib_181	0e+0	1e-4	17136	0	3988	36703	0	0	17912
qplib_182	5e-1	9e-1	1225	0	0	70	0	0	1225
qplib_183	6e-1	5e-1	256	0	0	32	0	0	256
qplib_184	3e-1	1e-1	256	0	0	32	0	0	256
qplib_185			84	0	328	200	16	0	406
qplib_186			25	0	574	378	150	0	180
qplib_187			56	0	273	170	16	0	321
qplib_188	5e-1	6e-1	256	0	0	32	0	0	256
qplib_189	5e-1	1e-1	225	0	0	30	0	0	225
qplib_190	5e-1	9e-1	676	0	0	52	0	0	676
qplib_191	5e-1	3e-1	1024	0	0	64	0	0	1024
qplib_192	4e-1	8e-1	144	0	0	24	0	0	144
qplib_193	3e-2	3e-4	8904	0	0	823	0	0	8904
qplib_194	5e-1	8e-1	144	0	0	24	0	0	144
qplib_195	5e-1	9e-2	400	0	0	40	0	0	400
qplib_196			42	0	86	211	14	0	58
qplib_197			200	56	136	687	64	8	264
qplib_198			90	0	2	481	1	0	90
qplib_199	0e+0	2e-3	10	0	400	440	0	0	10
qplib_200			10920	0	4180	2299	3130	0	10920
qplib_201			80	0	104	1457	1	0	182
qplib_202			201	0	603	605	2	0	402
qplib_203			496	0	2	1	1	0	496
qplib_204	0e+0	1e-3	25	0	625	650	0	0	25
qplib_205			2450	0	1782	990	1332	0	2450
qplib_206			101	0	303	305	2	0	202
qplib_207			105	0	977	4813	54	0	927
qplib_208			2450	0	4134	5792	1283	392	2450
qplib_209			87	0	157	622	24	0	222
qplib_210			126	0	2108	4104	882	0	720
qplib_211			15	0	1800	960	900	0	15
qplib_212			352	0	430	768	48	0	734
qplib_213			64	0	769	1739	256	0	286
qplib_214	0e+0	2e-4	180	0	111	406	0	0	200
qplib_215			112	0	1677	3405	672	0	571
qplib_216	5e-1	9e-1	600	0	0	50	0	0	600
qplib_217			42	0	630	254	42	0	42
qplib_218			155	0	29	1457	1	0	182
qplib_219			132	0	1141	3445	192	0	829
qplib_220	5e-1	5e-4	0	1662	126	91	39	0	1710
qplib_221			38	0	128	213	34	0	97
qplib_222			40	0	472	1016	160	0	172
qplib_223			38	0	2033	2253	544	0	982
qplib_224			0	100	2	4	1	1	0
qplib_225			240	0	168	201	25	0	269
qplib_226	5e-1	9e-1	525	0	0	50	0	0	525
qplib_227			10	0	800	440	400	0	10
qplib_228			9	0	60	82	20	0	49
qplib_229	5e-1	2e-2	243	0	0	81	0	0	243
qplib_230	0e+0	1e-3	20	0	800	840	0	0	20
qplib_231			70	0	1140	2102	490	0	376
qplib_232			70	0	1026	1998	420	0	340
qplib_233			44	0	48	481	1	0	90
qplib_234	0e+0	2e-6	462	0	1536	3137	0	0	462
qplib_235			650	0	1416	1709	583	175	650
qplib_236	0e+0	4e-2	14	0	19	28	0	0	26
qplib_237			36	0	78	213	12	0	102
qplib_238	0e+0	1e-3	15	0	900	960	0	0	15
qplib_239			182	0	2	1457	1	0	182
qplib_240	1e-1	2e-1	14	0	370	556	0	0	14

Table 8 Discrete Instance Feature 181-240

			Variables			Constraints			
name	% eig	% dens	# bin	# int	# cont	# lin	# quad	# conv	# box
qplib_241	5e-1	9e-1	525	0	0	50	0	0	525
qplib_242			0	100	2	4	1	1	0
qplib_243			126	0	1894	3908	756	0	648
qplib_244	6e-1	1e+0	50	0	0	1	0	0	50
qplib_245			12	36	3	54	3	0	48
qplib_246			7	56	7	42	7	0	63
qplib_247			276	0	2	1	1	0	276
qplib_248	6e-2	2e-1	12	0	54	33	0	0	66
qplib_249			182	0	2	1457	1	0	182
qplib_250			24	0	57	38	27	0	24
qplib_251			0	100	2	4	1	1	0
qplib_252			44	0	16	28	12	0	44
qplib_253			114	0	2114	3854	912	0	725
qplib_254			0	100	2	4	1	1	0
qplib_255	5e-1	4e-2	144	0	0	48	0	0	144
qplib_256			10	0	1600	880	800	0	10
qplib_257			108	0	24	45	18	0	108
qplib_258			184	0	32	60	24	0	184
qplib_259			528	0	2	10913	1	0	528
qplib_260			9	0	74	92	30	0	53
qplib_261	5e-1	1e-1	240	0	0	46	0	0	240
qplib_262			600	0	784	441	584	0	600
qplib_263	5e-1	2e-3	225	0	225	255	0	0	225
qplib_264	0e+0	3e-5	144	0	667	1045	0	0	162
qplib_265			0	100	2	4	1	1	0
qplib_266			107	0	1409	1666	132	132	624
qplib_267	5e-1	9e-1	600	0	0	50	0	0	600
qplib_268			112	0	16	45	12	0	112
qplib_269			168	0	366	233	267	70	168
qplib_270			552	0	2	8097	1	0	552
qplib_271			160	0	1268	4611	192	0	978
qplib_272			66	0	2	1	1	0	66
qplib_273			124	0	220	884	32	0	312
qplib_274			138	0	48	95	6	0	180
qplib_275			0	100	2	4	1	1	0
qplib_276	5e-1	1e-1	210	0	0	44	0	0	210
qplib_277			74	0	18	481	1	0	90
qplib_278			190	0	3602	6854	1520	0	1269
qplib_279			112	0	1866	3578	784	0	634
qplib_280			25	0	2000	1040	1000	0	25
qplib_281			120	0	2	1	1	0	120
qplib_282			40	0	6400	3280	3200	0	40
qplib_283			46	0	644	237	46	0	46
qplib_284	0e+0	1e-3	25	0	750	780	0	0	25
qplib_285			21	0	72	1	1	0	91
qplib_286			750	0	168	235	25	20	779
qplib_287	0e+0	2e-6	462	0	1179	2723	0	0	462
qplib_288			0	100	2	4	1	1	0
qplib_289	5e-1	2e-2	192	0	0	64	0	0	192
qplib_290			98	0	1460	2919	588	0	494
qplib_291			1035	0	2	1	1	0	1035
qplib_292			216	72	140	893	68	18	296
qplib_293			182	0	2	1457	1	0	182
qplib_294			101	0	303	305	2	1	202
qplib_295			20	0	2000	1050	1000	0	20
qplib_296	0e+0	1e-3	20	0	1000	1050	0	0	20
qplib_297			40	0	680	306	40	0	40
qplib_298			133	0	2486	4574	1064	0	861
qplib_299			946	0	2	1	1	0	946
qplib_300			140	0	2350	4647	980	0	806

Table 9 Discrete Instance Feature 241-300

			Variables			Constraints			
name	% eig	% dens	# bin	# int	# cont	# lin	# quad	# conv	# box
qplib_301	0e+0	1e-3	10	0	800	880	0	0	10
qplib_302			0	960	5537	6497	960	480	1697
qplib_303			10816	0	15178	13205	3221	0	10816
qplib_304			144	0	32	55	24	0	144
qplib_305	5e-1	1e-2	300	0	0	100	0	0	300
qplib_306			120	0	216	860	32	0	304
qplib_307			0	100	2	4	1	1	0
qplib_308			54	0	864	305	54	0	54
qplib_309			462	0	2	6161	1	0	462
qplib_310			6	42	6	36	6	0	48
qplib_311			25	0	1250	650	625	0	25
qplib_312			435	0	2	8121	1	0	435
qplib_313			30	0	9000	4650	4500	0	30
qplib_314			30	0	6000	3100	3000	0	30
qplib_315			200	0	401	403	1	1	400
qplib_316			152	0	2858	5314	1216	0	997
qplib_317			92	0	16	40	12	0	92
qplib_318	5e-1	9e-1	600	0	0	50	0	0	600
qplib_319			126	0	24	48	18	0	126
qplib_320	5e-1	6e-2	108	0	0	36	0	0	108
qplib_321			72	0	868	2000	288	0	324
qplib_322			20	0	6000	3150	3000	0	20
qplib_323			128	0	2713	2506	528	0	1079
qplib_324			1128	0	2	1	1	0	1128
qplib_325	0e+0	3e-4	40	0	3200	3280	0	0	40
qplib_326			168	0	32	58	24	0	168
qplib_327	0e+0	3e-4	30	0	3000	3100	0	0	30
qplib_328			116	0	984	1900	192	0	657
qplib_329			60	0	1080	377	60	0	60
qplib_330	5e-1	8e-1	225	0	0	30	0	0	225
qplib_331			378	0	2	1	1	0	378
qplib_332			703	0	2	1	1	0	703
qplib_333	0e+0	-9e-6	14	0	89988	90997	0	0	1022
qplib_334	5e-1	9e-1	600	0	0	50	0	0	600
qplib_335	5e-1	6e-2	108	0	0	36	0	0	108
qplib_336			380	0	2	4561	1	0	380
qplib_337			42	0	630	254	42	0	42
qplib_338	1e+0	3e-1	120	0	0	40	0	0	120
qplib_339	5e-1	2e-2	243	0	0	81	0	0	243
qplib_340			90	0	2	481	1	0	90
qplib_341			133	0	28	51	21	0	133
qplib_342			80	0	967	2271	320	0	362
qplib_343			84	0	1382	2577	588	0	462
qplib_344			116	0	1008	2422	192	0	681
qplib_345			20	0	1600	840	800	0	20
qplib_346			72	0	16	35	12	0	72
qplib_347			650	0	816	459	608	0	650
qplib_348			72	0	24	55	4	0	92
qplib_349			46	0	644	237	46	0	46
qplib_350			38	0	10288	11093	2754	0	4817
qplib_351			25	0	1500	780	750	0	25
qplib_352			435	0	2	1	1	0	435
qplib_353			51	0	153	155	2	1	102
qplib_354			325	0	2	1	1	0	325
qplib_355			90	0	2	481	1	0	90
qplib_356			75	0	20	37	15	0	75
qplib_357	5e-1	7e-2	81	0	0	27	0	0	81
qplib_358	1e+0	3e-1	210	0	0	70	0	0	210
qplib_359	1e+0	3e-1	150	0	0	50	0	0	150
qplib_360			462	0	2	6161	1	0	462

Table 10 Discrete Instance Feature 301-360

			Variables			Constraints			
name	% eig	% dens	# bin	# int	# cont	# lin	# quad	# conv	# box
qplib_361			87	0	159	627	24	0	222
qplib_362			28	0	121	167	4	0	33
qplib_363			153	0	2	1	1	0	153
qplib_364			552	0	2	8097	1	0	552
qplib_365			48	0	571	1247	192	0	210
qplib_366	0e+0	3e-4	144	0	91	325	0	0	162
qplib_367			90	0	2	481	1	0	90
qplib_368			0	100	2	4	1	1	0
qplib_369			0	100	2	4	1	1	0
qplib_370			84	0	1243	2450	504	0	417
qplib_371			51	0	153	155	2	0	102
qplib_372			380	0	2	4561	1	0	380
qplib_373	1e+0	3e-1	180	0	0	60	0	0	180
qplib_374			90	0	2	481	1	0	90
qplib_375			12	156	12	72	12	0	168
qplib_376			3	0	30	30	6	0	27
qplib_377			200	0	32	62	24	0	200
qplib_378			24	0	93	135	5	1	106
qplib_379			0	100	2	4	1	1	0
qplib_380	0e+0	2e-5	180	0	831	1306	0	0	200
qplib_381	1e+0	3e-1	90	0	0	30	0	0	90
qplib_382	9e-2	2e-1	7	0	188	283	0	0	7
qplib_383	0e+0	3e-4	20	0	3000	3150	0	0	20
qplib_384			576	0	1396	1034	602	0	576
qplib_385			95	0	1742	3154	760	0	589
qplib_386			48	0	352	695	56	0	271
qplib_387			54	0	864	305	54	0	54
qplib_388			104	0	200	736	32	0	272
qplib_389			190	0	2	2281	1	0	190
qplib_390	5e-1	9e-1	525	0	0	50	0	0	525
qplib_391			5	30	5	44	5	0	35
qplib_392			30	0	47	72	8	0	30
qplib_393			224	0	32	65	24	0	224
qplib_394			0	100	2	4	1	1	0
qplib_395			56	0	670	1488	224	0	248
qplib_396			15	0	2400	1280	1200	0	15
qplib_397	3e-2	3e-3	2	0	70	37	28	0	46
qplib_398	5e-1	3e-2	192	0	0	64	0	0	192
qplib_399			87	0	159	606	24	0	222
qplib_400			36	0	199	284	147	0	86
qplib_401			116	0	1008	2416	192	0	681
qplib_402			0	100	2	4	1	1	0
qplib_403			190	0	2	1	1	0	190
qplib_404			861	0	2	1	1	0	861
qplib_405	0e+0	-7e-6	7	0	89429	89934	0	0	511
qplib_406			60	0	1080	377	60	0	60
qplib_407			182	0	2	1457	1	0	182
qplib_408	5e-1	4e-2	144	0	0	48	0	0	144
qplib_409			45	0	2	1	1	0	45
qplib_410			0	100	2	4	1	1	0
qplib_411			561	0	2	1	1	0	561
qplib_412	6e-1	1e+0	50	0	0	1	0	0	50
qplib_413	7e-2	1e-4	0	899	126	87	39	0	947
qplib_414	0e+0	4e-5	112	0	521	813	0	0	128
qplib_415			780	0	2	1	1	0	780
qplib_416	0e+0	4e-3	10	0	250	275	0	0	10
qplib_417			2401	0	4267	3432	1374	0	2401
qplib_418			300	0	2	4601	1	0	300
qplib_419	0e+0	6e-5	84	0	393	610	0	0	98
qplib_420			36	0	68	106	48	48	44

Table 11 Discrete Instance Feature 361-420

			Variables			Constraints			
name	% eig	% dens	# bin	# int	# cont	# lin	# quad	# conv	# box
qplib_421			68	0	24	55	4	0	88
qplib_422			140	0	2111	4428	840	0	725
qplib_423			1225	0	2	1	1	0	1225
qplib_424			231	0	2	1	1	0	231
qplib_425			10	0	500	275	250	0	10
qplib_426			40	0	680	306	40	0	40
qplib_427			400	0	3452	1991	1368	0	450
qplib_428			168	0	366	233	267	0	168
qplib_429			201	0	603	605	2	1	402
qplib_430			600	0	1336	1593	560	168	600
qplib_431			435	0	2	8121	1	0	435
qplib_432	0e+0	2e-4	30	0	4500	4650	0	0	30
qplib_433			171	0	3230	6074	1368	0	1133
qplib_434			625	0	1481	1103	628	0	625
qplib_435			90	0	2	481	1	0	90
qplib_436	5e-1	9e-1	525	0	0	50	0	0	525
qplib_437			116	0	68	1457	1	0	182
qplib_438	0e+0	1e-3	25	0	1000	1040	0	0	25
qplib_439			98	0	1624	3069	686	0	548
qplib_440			182	0	2	1457	1	0	182
qplib_441	0e+0	2e-5	136	0	320	666	0	0	136
qplib_442			100	0	35	74	5	0	130
qplib_443			28	0	2	1	1	0	28
qplib_444			630	0	2	1	1	0	630
qplib_445			10920	0	14892	23931	3026	936	10920
qplib_446	5e-1	3e-2	192	0	0	64	0	0	192
qplib_447			182	0	2	1457	1	0	182
qplib_448			50	0	101	103	1	1	100
qplib_449			0	100	2	4	1	1	0
qplib_450	0e+0	8e-4	15	0	1200	1280	0	0	15
qplib_451	0e+0	1e+0	300	0	0	61	0	0	300
qplib_452	5e-1	7e-2	300	0	0	61	0	0	300
qplib_453	5e-1	7e-2	300	0	0	61	0	0	300
qplib_454	5e-1	7e-2	300	0	0	61	0	0	300
qplib_455	0e+0	1e+0	300	0	0	61	0	0	300
qplib_456	0e+0	1e+0	300	0	0	61	0	0	300
qplib_457	5e-1	5e-2	395	0	0	80	0	0	395
qplib_458	5e-1	5e-2	395	0	0	80	0	0	395
qplib_459	5e-1	4e-2	316	0	0	80	0	0	316
qplib_460	5e-1	4e-2	316	0	0	80	0	0	316
qplib_461	5e-1	5e-2	395	0	0	80	0	0	395
qplib_462	5e-1	4e-2	316	0	0	80	0	0	316
qplib_463	0e+0	1e+0	235	0	0	48	0	0	235
qplib_464	5e-1	1e-1	235	0	0	48	0	0	235
qplib_465	0e+0	1e+0	235	0	0	48	0	0	235
qplib_466	5e-1	1e-1	235	0	0	48	0	0	235
qplib_467	0e+0	1e+0	235	0	0	48	0	0	235
qplib_468	0e+0	1e+0	235	0	0	48	0	0	235
qplib_469	0e+0	1e+0	235	0	0	48	0	0	235
qplib_470	5e-1	1e-1	235	0	0	48	0	0	235
qplib_471	0e+0	1e+0	235	0	0	48	0	0	235
qplib_472	0e+0	4e-2	400	0	1600	1603	400	0	400
qplib_473	0e+0	6e-2	400	0	1200	1603	0	0	400
qplib_474	0e+0	4e-2	400	0	1600	1603	400	0	400
qplib_475	0e+0	4e-2	400	0	1600	1603	400	0	400
qplib_476	0e+0	6e-2	400	0	1200	1603	0	0	400
qplib_477			2000	0	8000	6001	2000	0	2000
qplib_478			3000	0	12000	9001	3000	0	3000
qplib_479			2000	0	8000	6001	2000	0	2000
qplib_480			2000	0	8000	6074	2000	0	2000

Table 12 Discrete Instance Feature 421-480

			Variables			Constraints			
name	% eig	% dens	# bin	# int	# cont	# lin	# quad	# conv	# box
qplib_481			3000	0	12000	9155	3000	0	3000
qplib_482			2000	0	7999	6089	2000	0	2000
qplib_483	0e+0	7e-6	4639	0	21658	66685	0	0	16961
qplib_484	0e+0	7e-6	4662	0	21683	66283	0	0	16972
qplib_485			4639	0	31258	66637	4800	0	16961
qplib_486			4646	0	24042	75036	4800	0	9754
qplib_487	0e+0	3e-5	1152	0	5054	16322	0	0	3878
qplib_488	5e-1	1e+0	150	0	0	0	0	0	150
qplib_489	5e-1	8e-1	300	0	0	0	0	0	300
qplib_490	5e-1	2e-2	343	0	0	0	0	0	343
qplib_491	5e-1	2e-2	225	0	0	0	0	0	225
qplib_492	5e-1	8e-1	250	0	0	0	0	0	250
qplib_493	5e-1	1e+0	150	0	0	0	0	0	150
qplib_494	5e-1	5e-1	200	0	0	0	0	0	200
qplib_495	5e-1	1e-2	400	0	0	0	0	0	400
qplib_496	5e-1	3e-1	200	0	0	0	0	0	200
qplib_497	5e-1	5e-1	500	0	0	0	0	0	500
qplib_498	5e-1	8e-1	120	0	0	0	0	0	120
qplib_499	5e-1	1e-1	250	0	0	0	0	0	250
qplib_500	5e-1	3e-1	120	0	0	0	0	0	120
qplib_501	5e-1	3e-1	150	0	0	0	0	0	150
qplib_502	5e-1	8e-1	200	0	0	0	0	0	200
qplib_503	5e-1	8e-1	120	0	0	0	0	0	120
qplib_504	5e-1	3e-1	200	0	0	0	0	0	200
qplib_505	5e-1	3e-1	150	0	0	0	0	0	150
qplib_506	5e-1	8e-1	200	0	0	0	0	0	200
qplib_507	5e-1	1e-1	250	0	0	0	0	0	250
qplib_508	5e-1	8e-1	120	0	0	0	0	0	120
qplib_509	5e-1	8e-1	150	0	0	0	0	0	150
qplib_510	5e-1	8e-1	200	0	0	0	0	0	200
qplib_511	5e-1	3e-1	120	0	0	0	0	0	120
qplib_512	5e-1	8e-1	150	0	0	0	0	0	150
qplib_513	5e-1	1e-1	250	0	0	0	0	0	250
qplib_514	5e-1	1e-1	500	0	0	0	0	0	500
qplib_515	5e-1	1e-1	250	0	0	0	0	0	250
qplib_516	5e-1	1e-1	500	0	0	0	0	0	500
qplib_517	0e+0	3e-6	60	0	3033	7153	0	0	60
qplib_518	0e+0	6e-7	300	0	15221	36061	0	0	300
qplib_519			100	0	1301	271	100	0	100
qplib_520			2400	0	31201	11923	2400	0	2400
qplib_521			2400	0	31201	11963	2400	0	2400
qplib_522	5e-1	1e+0	100	0	0	1237	0	0	100
qplib_523	5e-1	1e+0	100	0	0	1237	0	0	100
qplib_524	5e-1	1e+0	100	0	0	1237	0	0	100
qplib_525	5e-1	1e+0	100	0	0	2475	0	0	100
qplib_526	5e-1	1e+0	100	0	0	2475	0	0	100
qplib_527	5e-1	1e+0	100	0	0	2475	0	0	100
qplib_528	5e-1	1e+0	100	0	0	3712	0	0	100
qplib_529	5e-1	1e+0	100	0	0	3712	0	0	100
qplib_530	5e-1	1e+0	100	0	0	3712	0	0	100
qplib_531	5e-1	1e+0	150	0	0	2793	0	0	150
qplib_532	5e-1	1e+0	150	0	0	2793	0	0	150
qplib_533	5e-1	1e+0	150	0	0	2793	0	0	150
qplib_534	5e-1	1e+0	150	0	0	5587	0	0	150
qplib_535	5e-1	1e+0	150	0	0	5587	0	0	150
qplib_536	5e-1	1e+0	150	0	0	5587	0	0	150
qplib_537	5e-1	1e+0	150	0	0	8381	0	0	150
qplib_538	5e-1	1e+0	150	0	0	8381	0	0	150
qplib_539	5e-1	1e+0	150	0	0	8381	0	0	150
qplib_540	5e-1	5e-2	4010	0	0	100	0	0	4010

Table 13 Discrete Instance Feature 481-540

			Variables			Constraints			
name	% eig	% dens	# bin	# int	# cont	# lin	# quad	# conv	# box
qplib_541	5e-1	5e-2	3708	0	0	100	0	0	3708
qplib_542	5e-1	3e-1	640	0	0	16	0	0	640
qplib_543	5e-1	9e-1	1738	0	0	1097130	0	0	1738
qplib_544	5e-1	9e-1	2242	0	0	2393767	0	0	2242
qplib_545	0e+0	7e-2	11726	0	0	300	0	0	11726
qplib_546	0e+0	4e-2	21994	0	0	400	0	0	21994
qplib_547	4e-1	1e-1	6745	0	0	200	0	0	6745
qplib_548	4e-1	7e-2	8821	0	0	250	0	0	8821
qplib_549	4e-1	1e+0	812	0	0	30	0	0	812
qplib_550	0e+0	7e-2	15203	0	0	350	0	0	15203
qplib_551	2e-1	4e-2	1403	0	0	1094	0	0	1403
qplib_552	5e-1	9e-1	4692	0	0	36	0	0	4692
qplib_553	5e-1	9e-1	4220	0	0	37	0	0	4220
qplib_554	5e-1	1e+0	600	0	0	60	0	0	600
qplib_555	5e-1	1e+0	1200	0	0	60	0	0	1200
qplib_556	5e-1	1e+0	600	0	0	60	0	0	600
qplib_557	5e-1	1e+0	900	0	0	60	0	0	900
qplib_558	7e-1	9e-2	405	0	0	27	0	0	405
qplib_559	7e-1	7e-2	627	0	0	33	0	0	627
qplib_560	5e-1	8e-2	316	0	0	33	0	0	316
qplib_561	5e-1	4e-2	235	0	0	47	0	0	235
qplib_562	7e-1	5e-2	120	0	0	30	0	0	120
qplib_563	5e-1	5e-2	188	0	0	47	0	0	188
qplib_564	5e-1	7e-2	141	0	0	47	0	0	141
qplib_565	7e-1	6e-2	90	0	0	30	0	0	90
qplib_566	4e-1	5e-2	2046	0	0	297	0	0	2046
qplib_567	4e-1	5e-2	2071	0	0	297	0	0	2071
qplib_568	4e-1	5e-2	2075	0	0	297	0	0	2075
qplib_569	4e-1	4e-2	2203	0	0	315	0	0	2203
qplib_570	5e-1	7e-2	1000	0	0	50	0	0	1000
qplib_571	5e-1	9e-2	1000	0	0	50	0	0	1000
qplib_572	5e-1	7e-2	1000	0	0	50	0	0	1000
qplib_573	5e-1	7e-2	1000	0	0	50	0	0	1000
qplib_574	5e-1	7e-2	1000	0	0	50	0	0	1000
qplib_575	5e-1	8e-2	1000	0	0	50	0	0	1000
qplib_576	5e-1	7e-2	1000	0	0	50	0	0	1000
qplib_577	5e-1	8e-2	316	0	0	33	0	0	316
qplib_578	5e-1	8e-2	327	0	0	33	0	0	327
qplib_579	5e-1	9e-1	180	0	0	100	0	0	180
qplib_580	5e-1	9e-1	220	0	0	121	0	0	220
qplib_581	5e-1	9e-1	264	0	0	144	0	0	264
qplib_582	5e-1	9e-1	312	0	0	169	0	0	312
qplib_583	5e-1	9e-1	364	0	0	196	0	0	364
qplib_584	5e-1	9e-1	420	0	0	225	0	0	420
qplib_585			40	0	81	83	1	0	80
qplib_586			50	0	101	103	1	0	100
qplib_587			50	0	101	103	1	0	100
qplib_588			100	0	201	203	1	0	200
qplib_589			41	0	123	125	2	0	82
qplib_590			41	0	123	125	2	0	82
qplib_591			41	0	123	125	2	0	82
qplib_592			51	0	153	155	2	0	102
qplib_593			101	0	303	305	2	0	202
qplib_594			21	0	63	65	2	0	42
qplib_595			31	0	93	95	2	0	62
qplib_596			51	0	153	155	2	0	102
qplib_597	2e-2	2e-4	252	0	1499	1913	0	0	1714
qplib_598	2e-3	7e-4	0	10000	0	5000	0	0	10000
qplib_599	0e+0	1e-3	0	151	1851	1000	0	0	1000
qplib_600	3e-1	2e-1	0	202	0	1	0	0	202

Table 14 Discrete Instance Feature 541-600

			Variables	Constraints			
name	% eig	% dens	# cont	# lin	# quad	# conv	# box
qplib_1	5e-1	1e+0	50	1	0	0	0
qplib_2	5e-1	1e+0	50	1	0	0	0
qplib_3	4e-1	1e+0	50	1	0	0	50
qplib_4	5e-1	1e+0	50	1	0	0	50
qplib_5	3e-1	1e-1	60	20	20	0	40
qplib_6	5e-1	3e-1	40	0	0	0	40
qplib_7	3e-1	2e-1	45	15	15	0	30
qplib_8	3e-1	3e-1	60	30	30	29	30
qplib_9	3e-1	2e-1	60	20	20	0	40
qplib_10	4e-1	2e-1	48	8	8	0	40
qplib_11	4e-1	4e-1	48	8	8	8	40
qplib_12	2e-1	1e-1	100	50	50	50	50
qplib_13	3e-1	3e-1	40	20	20	20	20
qplib_14	5e-1	5e-1	50	0	0	0	50
qplib_15	4e-1	4e-1	60	10	10	10	50
qplib_16	2e-1	1e-1	60	30	30	0	30
qplib_17	5e-1	1e+0	40	0	0	0	40
qplib_18	3e-1	4e-1	45	15	15	15	30
qplib_19	3e-1	4e-1	60	20	20	19	40
qplib_20	5e-1	5e-1	40	0	0	0	40
qplib_21	3e-1	4e-1	60	24	20	20	40
qplib_22	3e-1	4e-1	45	18	15	15	30
qplib_23	5e-1	5e-1	41	9	1	0	40
qplib_24	3e-1	4e-1	60	24	20	0	40
qplib_25	2e-1	9e-1	41	9	1	1	40
qplib_26	3e-1	2e-1	90	36	30	0	60
qplib_27	2e-1	6e-2	100	55	50	0	50
qplib_28	5e-1	4e-1	60	24	20	20	40
qplib_29	2e-1	4e-1	75	30	25	25	50
qplib_30	1e+0	9e-1	51	11	1	1	50
qplib_31	1e-1	2e-1	40	22	20	20	20
qplib_32	3e-1	6e-2	40	22	20	0	20
qplib_33	3e-1	9e-1	51	6	1	1	50
qplib_34	3e-1	1e-1	40	22	20	0	20
qplib_35	2e-1	1e-1	120	66	60	0	60
qplib_36	3e-1	2e-1	40	22	20	20	20
qplib_37	5e-1	4e-1	60	24	20	20	40
qplib_38	5e-1	9e-1	51	11	1	1	50
qplib_39	3e-1	1e-1	120	66	60	0	60
qplib_40	5e-1	5e-1	41	5	1	0	40
qplib_41	3e-1	2e-1	100	55	50	50	50
qplib_42	5e-1	9e-1	41	5	1	0	40
qplib_43	1e-1	2e-1	60	33	30	30	30
qplib_44	5e-1	4e-1	45	18	15	15	30
qplib_45	3e-1	2e-1	120	66	60	60	60
qplib_46	7e-1	4e-1	75	30	25	25	50
qplib_47	3e-1	2e-1	40	22	20	0	20
qplib_48	3e-1	4e-1	75	30	25	0	50
qplib_49	3e-1	2e-1	60	33	30	30	30
qplib_50	5e-1	9e-1	61	13	1	1	60

Table 15 Continuous Instance Feature 1-50

			Variables	Constraints			
name	% eig	% dens	# cont	# lin	# quad	# conv	# box
qplib_51	5e-1	5e-1	61	13	1	0	60
qplib_52	3e-1	4e-1	90	36	30	0	60
qplib_53	5e-1	9e-1	41	9	1	1	40
qplib_54	5e-1	9e-1	41	5	1	1	40
qplib_55	3e-1	1e-1	100	55	50	0	50
qplib_56	6e-1	4e-1	45	18	15	15	30
qplib_57	5e-1	9e-1	61	7	1	1	60
qplib_58	5e-1	7e-1	40	0	0	0	40
qplib_59	5e-1	3e-1	50	0	0	0	50
qplib_60	5e-1	3e-1	40	0	0	0	40
qplib_61	5e-1	8e-1	40	0	0	0	40
qplib_62	2e-1	1e-1	60	40	40	0	20
qplib_63	2e-1	3e-2	60	40	40	0	20
qplib_64	3e-1	6e-2	56	28	28	0	28
qplib_65	2e-1	4e-2	70	42	42	0	28
qplib_66	3e-1	1e-1	100	50	50	0	50
qplib_67	3e-1	6e-2	96	48	48	0	48
qplib_68	3e-1	6e-2	96	48	48	0	48
qplib_69	2e-1	6e-2	90	60	60	0	30
qplib_70	3e-1	1e-1	80	40	40	0	40
qplib_71	2e-1	3e-2	144	96	96	0	48
qplib_72	3e-1	6e-2	80	40	40	0	40
qplib_73	2e-1	2e-1	125	75	75	0	50
qplib_74			558	689	533	0	25
qplib_75	0e+0	3e-4	3203	3042	0	0	1
qplib_76			5006	4	2	0	5000
qplib_77			1256	4	2	0	1250
qplib_78			141	43	65	0	126
qplib_79	0e+0	1e-4	4725	4563	1521	0	0
qplib_80			212	135	66	0	146
qplib_81			6846	5500	1369	1369	0
qplib_82	0e+0	5e-4	1030	513	0	0	0
qplib_83			18606	14924	3721	3721	0
qplib_84			402	202	201	0	398
qplib_85			2167	1779	361	0	0
qplib_86			998	538	240	0	760
qplib_87			810	256	256	16	0
qplib_88			649	598	297	205	0
qplib_89			1519	782	480	0	1039
qplib_90			3756	4	2	0	3750
qplib_91			5767	4763	961	961	0
qplib_92			708	551	210	0	498
qplib_93	0e+0	4e-4	1816	1393	400	0	0
qplib_94	5e-1	1e+0	200	1	0	0	200
qplib_95			426	497	401	0	25
qplib_96			1397	1197	200	0	0
qplib_97	0e+0	2e-3	2500	0	0	0	0
qplib_98			2506	4	2	0	2500
qplib_99			26048	18484	3721	3721	0
qplib_100			601	486	100	100	0

Table 16 Continuous Instance Feature 51-100

			Variables	Constraints			
name	% eig	% dens	# cont	# lin	# quad	# conv	# box
qplib_101			300	102	100	0	1
qplib_102			1685	1503	1154	0	9
qplib_103			2014	1134	904	0	1110
qplib_104			316	191	133	23	184
qplib_105	0e+0	3e-4	1219	1058	0	0	1
qplib_106	0e+0	1e-3	3750	0	0	0	0
qplib_107			1806	1456	361	0	0
qplib_108	0e+0	8e-7	1600	1599	0	0	800
qplib_109			1422	816	631	20	791
qplib_110	8e-3	2e-5	6502	4996	1500	0	1
qplib_111			207	57	11	0	196
qplib_112			802	402	401	1	798
qplib_113	5e-1	1e+0	500	1	0	0	500
qplib_114			1097	616	466	0	586
qplib_115	5e-1	1e+0	200	1	0	0	200
qplib_116			650	599	298	205	0
qplib_117			202	102	101	1	198
qplib_118			528	487	157	0	2
qplib_119			415	251	112	0	303
qplib_120	5e-1	1e+0	500	1	0	0	500
qplib_121			5401	4580	900	900	0
qplib_122			467	287	132	0	336
qplib_123			671	386	283	32	374
qplib_124			230	153	72	0	158
qplib_125			48601	40740	8100	8100	0
qplib_126			398	83	16	0	382
qplib_127			11196	9596	1600	0	0
qplib_128			2232	720	720	0	0
qplib_129			2528	1768	361	0	0
qplib_130			171	218	154	7	17
qplib_131	0e+0	4e-3	1250	0	0	0	0
qplib_132			503	201	200	0	102
qplib_133	8e-2	1e-3	230	191	190	0	21
qplib_134	0e+0	7e-5	3616	2793	800	0	0
qplib_135			298	204	90	0	208
qplib_136	0e+0	1e+0	301	1	0	0	0
qplib_137			350	319	84	0	266
qplib_138	0e+0	1e-4	2402	1600	0	0	1600
qplib_139			1003	401	400	400	202
qplib_140			6728	4744	961	961	0
qplib_141			1020	540	240	0	780
qplib_142	3e-3	4e-6	13002	9996	3000	0	1
qplib_143			722	225	225	0	0
qplib_144	0e+0	4e-6	7216	5593	1600	0	0
qplib_145			369	233	126	0	243
qplib_146			4501	3680	900	0	0
qplib_147			204	162	72	0	132
qplib_148			22327	18523	3721	3721	0
qplib_149			2003	801	800	0	402
qplib_150			871	431	204	0	668

Table 17 Continuous Instance Feature 101-150

			Variables	Constraints			
name	% eig	% dens	# cont	# lin	# quad	# conv	# box
qplib_151	1e+0	1e-2	951	4	4	0	0
qplib_152			532	244	108	0	424
qplib_153	0e+0	2e-4	517	286	0	0	63
qplib_154			290	111	90	0	0
qplib_155			820	380	160	0	660
qplib_156			1602	802	801	0	1598
qplib_157			21601	18160	3600	3600	0
qplib_158			627	323	145	0	482
qplib_159			2797	2397	400	0	0
qplib_160			4003	1601	1600	1600	802
qplib_161			205	135	66	0	139
qplib_162			736	407	220	0	516
qplib_163			520	479	157	149	2
qplib_164	0e+0	1e-3	650	500	0	0	0
qplib_165			1015	827	169	169	0
qplib_166	0e+0	1e-6	14416	11193	3200	0	0
qplib_167	0e+0	1e-4	2400	2398	0	0	1598
qplib_168			652	196	196	14	0
qplib_169			49687	41283	8281	8281	0
qplib_170			406	477	381	0	25
qplib_171	0e+0	1e-4	1200	201	0	0	0
qplib_172	2e-2	6e-5	3252	2496	750	0	1
qplib_173			930	450	210	0	720
qplib_174			500	202	198	0	200
qplib_175			434	239	110	0	324
qplib_176	0e+0	1e-3	5000	0	0	0	0
qplib_177			5597	4797	800	0	0
qplib_178			264	174	106	0	158
qplib_179			604	268	116	0	488
qplib_180			215	137	60	0	155
qplib_181			240	132	65	0	175
qplib_182	0e+0	1e-4	2400	2398	0	0	1598
qplib_183			528	208	96	0	432
qplib_184	5e-1	1e+0	1000	1	0	0	1000
qplib_185			290	110	90	0	0
qplib_186	3e-1	1e+0	50	35	0	0	0
qplib_187	5e-1	7e-1	40	28	0	0	0
qplib_188	2e-1	2e-2	172	31	100	0	172
qplib_189			171	37	81	0	171
qplib_190			208	24	390	0	208
qplib_191			212	43	128	0	212
qplib_192	2e-1	2e-5	20002	15002	0	0	0
qplib_193	3e-1	3e-5	19017	14017	0	0	0
qplib_194	2e-1	2e-5	20002	15002	0	0	0
qplib_195	3e-1	3e-5	19017	14017	0	0	0
qplib_196	2e-1	2e-5	20002	15002	0	0	0
qplib_197	2e-1	2e-4	2300	1800	0	0	0
qplib_198	0e+0	1e-4	10000	0	1	1	0
qplib_199	0e+0	5e-5	20200	10000	0	0	0
qplib_200	0e+0	3e-5	27543	8000	0	0	0

Table 18 Continuous Instance Feature 151-200

			Variables	Constraints			
name	% eig	% dens	# cont	# lin	# quad	# conv	# box
qplib_201	0e+0	5e-5	20050	10001	0	0	20050
qplib_202	0e+0	1e-4	10010	5001	0	0	10010
qplib_203	0e+0	2e-4	16002	8002	0	0	8001
qplib_204	0e+0	4e-3	2003	0	0	0	2003
qplib_205	0e+0	7e-4	10000	5000	0	0	10000
qplib_206	0e+0	7e-4	10000	7500	0	0	10000
qplib_207	0e+0	6e-5	16514	405	0	0	14931
qplib_208	5e-2	5e-2	6000	320	0	0	5700
qplib_209	0e+0	3e-5	34552	52983	0	0	34552
qplib_210	0e+0	2e-4	5000	0	1	1	0
qplib_211	0e+0	7e-5	13870	10404	0	0	4
qplib_212	0e+0	2e-4	4096	5376	0	0	3564
qplib_213	0e+0	1e-4	10000	2	0	0	0
qplib_214	0e+0	3e-4	15129	0	0	0	0
qplib_215	0e+0	3e-4	15129	0	0	0	0
qplib_216	0e+0	1e-3	772	0	10000	0	0
qplib_217	0e+0	1e-4	1530	2220	0	0	407
qplib_218	0e+0	1e-4	10001	10000	0	0	0
qplib_219	0e+0	1e-4	10002	10000	0	0	0
qplib_220	0e+0	5e-4	2002	2000	0	0	0
qplib_221	0e+0	1e-4	10002	10000	0	0	0
qplib_222	0e+0	1e-4	10001	10000	0	0	0
qplib_223	0e+0	4e-4	2500	700	0	0	0
qplib_224	0e+0	2e-4	1530	2329	0	0	610
qplib_225	0e+0	8e-5	2300	3664	0	0	907
qplib_226	0e+0	2e-4	1530	2329	0	0	610
qplib_227	0e+0	2e-4	1530	2329	0	0	610
qplib_228	3e-1	7e-4	10000	2500	0	0	10000
qplib_229	0e+0	1e-4	10399	11362	0	0	10399
qplib_230	0e+0	1e-4	39204	0	0	0	19602
qplib_231	0e+0	3e-4	15129	0	0	0	15129
qplib_232	0e+0	1e-4	10201	202	0	0	10000
qplib_233	0e+0	1e-4	10000	10000	0	0	0
qplib_234	0e+0	7e-4	1489	75	0	0	0
qplib_235	0e+0	2e-3	520	8	0	0	0
qplib_236	0e+0	5e-2	1571	820	0	0	18
qplib_237	0e+0	2e-4	1066	590	0	0	258
qplib_238	0e+0	1e-3	1371	990	0	0	99
qplib_239	0e+0	6e-4	2750	397	0	0	0
qplib_240	0e+0	4e-3	5360	975	0	0	0
qplib_241	0e+0	5e-5	20000	10001	0	0	20000
qplib_242	0e+0	2e-4	4001	11999	0	0	0
qplib_243	0e+0	2e-4	4992	2464	0	0	2397
qplib_244	0e+0	5e-4	9604	0	0	0	9604
qplib_245	0e+0	1e-3	5184	0	0	0	5184
qplib_246	0e+0	3e-4	14400	0	0	0	14400
qplib_247	0e+0	3e-4	2890	1649	0	0	727
qplib_248	0e+0	1e-5	40003	10001	10001	10001	20003
qplib_249	0e+0	7e-6	45003	30000	0	0	15009
qplib_250	0e+0	5e-4	2000	2000	0	0	0

Table 19 Continuous Instance Feature 201-250