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

QPLIB: a Library of Quadratic Programming Instances

Fabio Furini · Emiliano Traversi · Alper Atamturk · Pietro Belotti · Pierre Bonami · Samuel Burer · 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

Samuel Burer

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

Nick Gould

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

Andrea Lodi

Ruth Misener

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 result 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 on 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 "interesting" 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,

Hans Mittelmann

Nick Sahinidis

Frederic Roupin

Stefan Vigerske

Angelika Wiegele

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.

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 significant 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

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, 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?**).

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?**).

but nonetheless

larger

what are we suggesting here? Is the problem "in" or "out"?

hyphen ✗

over the last 60 years.

again

quote my QP bibliography?

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

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

what about equality constraints?

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.

I often convenient to allow fixed variables.

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 under the form of binary variables, could always be "hidden" under the presence of (nonconvex) quadratic constraints due to 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 might be introduced by reformulation techniques in order to extract (and algorithmically exploit) specific properties.

Should this be left to "presolve"?

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 into 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

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

Hence, their solution only require 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

this may not be trivial, and isn't just linear algebra.

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.

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

what if the constraint implicitly bounds x? e.g. $x^\top x \leq 1$

is not only $\mathcal{NP}$-hard, but downright undecidable [5]. Hence so are the "harder" {C,Q}IQ.

It is important to notice 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.

where? give it a number

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

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

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 (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 by 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

this becomes an issue for large problems; also ordering of rows/cols of Q^i can lead to different L^i

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 explicit 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: these 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 these 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 can be 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 onto *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 relaxations. The general view is that CGC are not, all other things being equal, not 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

the trust-region subproblem is a very common case with diagonal constraint Hessian

(often called active-set methods) Mike Best's book?

← data can be recovered by matrix-vector products, but this is inefficient

some of "my" problems are from SQP methods for which only local optimality is required.

checking that a saddle pt is a local minimizer is NP hard

Another issue: sometimes QCQ has hidden convexity, and some solvers can detect this.

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 distinguishing 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 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 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 the most difficult, 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?**).

most

This description is purposely coarse; each of these classes can be subdivided in several other ~~fines~~ on the grounds of more detailed information about structures present in their constraints/objective function. These can have a significant ~~significant~~ algorithmic impact, and therefore be of interest for 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 structure 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 don't attempt such a more refined classification, and stick to the coarser one described in this paragraph.

into

what are these?

to individual researchers

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 they exist; 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"; as the latter condition is not clearly defined we elected to disregard this category.

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) we mean that it is so the largest problem class that the solver naturally targets, although it may be complete on specific subclasses. 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

more generally $x^{k+1} = x^k + d^k(x^k)$ where $d^k(x)$ is an arc satisfying $d^k(0) = 0$ .... some solvers do this.

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 tricks 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 iterate at iteration $k+1$ 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 improve optimality while ensuring feasibility. Local search methods for *I* are also iterative, but since derivatives are harder to define in presence of integer variables, d^k and α_k are usually computed in different ways, depending on the application at hand.

used to be called phase-1 in the simplex method.

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). The latter, KNitro, was only recently extended to also solve CGC optimally, and therefore represents an exception—we shall comment on this KNitro option below. Traditionally, however, KNitro is also known as a local (nonconvex) NLP solver. All of these solvers implement essentially two types of local search methods for NLP:

There are many others, see e.g. the lists on wikipedia, so why these specifically (many are commercial!)

- 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.

none in problem on p 2. sometimes exclude some (e.g. if they are dependent) or have dependent gradients

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*

The general scheme works as the step is chosen so that the objective decreases, and an active set cannot repeat

a third class of methods, gradient projection, are implemented in some software ... not represented in our list?

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 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 do away with 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: