FICO® Xpress Global

47.01

USER GUIDE

FICO® Xpress Optimization

FICO

©1983–2026 Fair Isaac Corporation. All rights reserved. This documentation is the property of Fair Isaac Corporation ("FICO"). Receipt or possession of this documentation does not convey rights to disclose, reproduce, make derivative works, use, or allow others to use it except solely for internal evaluation purposes to determine whether to purchase a license to the software described in this documentation, or as otherwise set forth in a written software license agreement between you and FICO (or a FICO affiliate). Use of this documentation and the software described in it must conform strictly to the foregoing permitted uses, and no other use is permitted.

The information in this documentation is subject to change without notice. If you find any problems in this documentation, please report them to us in writing. Neither FICO nor its affiliates warrant that this documentation is error-free, nor are there any other warranties with respect to the documentation except as may be provided in the license agreement. FICO and its affiliates specifically disclaim any warranties, express or implied, including, but not limited to, non-infringement, merchantability and fitness for a particular purpose. Portions of this documentation and the software described in it may contain copyright of various authors and may be licensed under certain third-party licenses identified in the software, documentation, or both.

In no event shall FICO or its affiliates be liable to any person for direct, indirect, special, incidental, or consequential damages, including lost profits, arising out of the use of this documentation or the software described in it, even if FICO or its affiliates have been advised of the possibility of such damage. FICO and its affiliates have no obligation to provide maintenance, support, updates, enhancements, or modifications except as required to licensed users under a license agreement.

FICO is a registered trademark of Fair Isaac Corporation in the United States and may be a registered trademark of Fair Isaac Corporation in other countries. Other product and company names herein may be trademarks of their respective owners.

Patent(s): www.fico.com/en/patents

Xpress Optimizer 47.01 (FICO® Xpress 9.9)

Deliverable Version: A

Last Revised: 15 May, 2026

Contents

1 Introduction and Disambiguation 1
1.1 Current restrictions . . . 1

2 Basic Usage 3
2.1 Using FICO Xpress Global inside the Console Optimizer . . . 3
2.2 Using FICO Xpress Global from the callable library . . . 4
2.2.1 Init and Free . . . 4
2.2.2 The Problem Pointer . . . 5
2.2.3 Optimizing and querying a problem . . . 5
2.2.4 Loading a non-convex quadratic optimization problem . . . 6
2.2.5 Loading a nonlinear optimization problem . . . 7
2.3 Modelling via Mosel or Python . . . 8
2.4 Dealing with unboundedness and infeasibility . . . 9
2.5 File Formats for MINLP problems . . . 10

3 Problem Types 12
3.1 Linear vs. Nonlinear Problems . . . 12
3.2 Convex vs. Non-Convex Problems . . . 12
3.3 Continuous vs. Mixed-Integer Problems . . . 14

4 Solution Methods 15
4.1 Reformulation of an MINLP . . . 15
4.1.1 Factorable functions . . . 15
4.1.2 Reformulation and auxiliary variables . . . 16
4.2 Creation of an LP relaxation of an MINLP . . . 17
4.2.1 Refining the relaxation . . . 17
4.3 Running branch-and-bound on an MINLP . . . 18
4.3.1 Spatial Branching . . . 18
4.4 Bound reduction . . . 19
4.4.1 Feasibility-based bound reduction . . . 19
4.4.2 Optimization-Based Bound Tightening . . . 20
4.5 Heuristics . . . 20

5 References 21

Appendix 22

A Contacting FICO 22
FICO Customer Support . . . 22
Documentation . . . 22
FICO Learning . . . 23

Sales and maintenance 23
About FICO 23

Index **24**

CHAPTER 1

Introduction and Disambiguation

FICO Xpress is a powerful mathematical optimization framework well-suited to a broad range of optimization problems.

The present user guide addresses FICO Xpress Global, a solver capable of solving general mixed-integer nonlinear optimization problems (MINLP) to proven global optimality, including problems with non-convex constraints or objective.

For solving linear programs, convex quadratic programs, convex quadratically constrained programs, or their respective mixed-integer counterparts, we refer to the main FICO Xpress Optimizer manual.

For solving nonlinear programs to local optimality, in particular when seeking dual information associated with local optima, we refer to the FICO Xpress Nonlinear manual. FICO Xpress Nonlinear can also be used as a heuristic for mixed-integer nonlinear problems. Furthermore, using user functions for black-box optimization is only supported by FICO Xpress Nonlinear, and not by FICO Xpress Global.

1.1 Current restrictions

Kindly be aware that some advanced functionalities of the FICO Xpress Optimizer are presently not available in FICO Xpress Global.

This comprises the following functions:

- XPRScalcobjective
- XPRScalcreducedcosts
- XPRScalcslacks
- XPRScalcsolinfo
- XPRSgetinfeas
- XPRSgetlpsol
- XPRSgetlpsolval
- XPRSgetmipsol
- XPRSgetmipsolval
- XPRSgetpresolvemap
- XPRSgetscaledinfeas
- XPRSloadmipsol
- XPRSpostsolvesol

- XPRSpresolvecut
- XPRSpresolverow

None of these is currently available at any stage of a FICO Xpress Global run. Although all Xpress Optimizer callback types are supported during a global solve, the usability of some of them is limited due to the absence of the mentioned functions. For example, it is not possible to query a relaxation solution or presolve a cut, which would be necessary for a customized branch-and-cut code for nonlinear problems.

CHAPTER 2

Basic Usage

2.1 Using FICO Xpress Global inside the Console Optimizer

You can use FICO Xpress Global directly within the Console Optimizer, an interactive command line interface to the Optimizer. The Console Optimizer is started from the command line using the following syntax:

```
C:\> optimizer
```

Note that the example shows the command as if it were input from the Windows Command Prompt (i.e., it is prefixed with the command prompt string `C:\>`). Windows users can also start the Console Optimizer by typing `optimizer` into the "Run ..." dialog box in the Start menu.

The Console Optimizer provides a quick and convenient interface for operating on a single problem loaded into the Optimizer. Currently, the Console Optimizer supports reading nonlinear problems from NL files, files in extended MPS format and LP format (for quadratic problems). From the Console Optimizer, you can easily (i) set controls for handling and solving the problem and (ii) query attributes of the problem and its solution information.

Useful features of the Console Optimizer include a help system, auto-completion of command names, and integration of system commands.

Typing "`help`" will list the various options for getting help. Typing "`help commands`" will list available commands. Typing "`help attributes`" and "`help controls`" will list the available attributes and controls, respectively. Typing "`help`" followed by a command name or control/attribute name will list the help for this item.

Nonlinear instances should be read by the READPROB command and solved to global optimality by the OPTIMIZE or the NLPOPTIMIZE command.

Let's consider an example that reads a nonlinear problem in NL format, solves it, prints the optimal objective value on screen, and writes the solution vector to a file.

```
[xpress C:\] readprob myMinlp.nl
[xpress C:\] optimize
[xpress C:\] NLPOBJVAL
42
[xpress C:\] writeslxsol myNonlinearSol.slx
[xpress C:\] quit
```

Note that you can either pass different flags to the (nlp)optimize command or use the NLPSOLVER control to switch between solving a problem to global or local optimality, provided both Xpress Global and Xpress Nonlinear are licensed. Per default, Xpress Global will be called. In the following example, the first call to `nlpoptimize` will solve to local optimality, the second to global optimality, the third one

will be the same as the first one.

```
[xpress C:\] readprob myMinlp.nl
[xpress C:\] nlpoptimize -s
[xpress C:\] NLPOBJVAL
50
[xpress C:\] readprob myMinlp.nl
[xpress C:\] nlpoptimize
[xpress C:\] NLPOBJVAL
42
[xpress C:\] NLPSOLVER=1
[xpress C:\] readprob myMinlp.nl
[xpress C:\] nlpoptimize
[xpress C:\] NLPOBJVAL
[xpress C:\] quit
```

For more details on using the Console Optimizer, we refer to the corresponding section in the FICO Xpress Optimizer documentation.

2.2 Using FICO Xpress Global from the callable library

In the previous section, we looked at the Console Optimizer interface and introduced some essential functions that all FICO Xpress Optimizer users should be familiar with. In this section, we expand on the discussion and include some essential functions of the library interface.

2.2.1 Init and Free

Before you can use FICO Xpress Global from any of the interfaces, the Optimizer library must be initialized, and the licensing status successfully verified.

When the Console Optimizer is started from the command line, the initialization and licensing security checks happen immediately, and the results are displayed with the banner in the console window. For the library interface users, the initialization and licensing are triggered by a call to the library function XPRSinit, which must be made before any of the other Optimizer library routines can be successfully called. If the licensing security checks fail to *check out* a license, then library users can obtain a string message explaining the issue using the function XPRSgetlicerrmsg.

Once the Optimizer functionality is no longer required, the user should release the license and any system resources the Optimizer holds. The Console Optimizer releases these automatically when the user exits the Console Optimizer with the QUIT or STOP command. For library users the Optimizer can be triggered to release its resources with a call to the routine XPRSfree, which will *free* the license checked out in the earlier call to XPRSinit.

```
if (XPRSinit(NULL)) {
  char message[512];
  XPRSgetlicerrmsg(message, sizeof(message));
  printf("Problem with XPRSinit, licensing error: %s\n", message);
  exit(-1);
}

[...] /* Code goes here */

XPRSfree();
```

In general, library users will call XPRSinit once when their application starts and then call XPRSfree before it exits.

2.2.2 The Problem Pointer

The Optimizer provides a container or problem pointer to contain an optimization problem and its associated controls, attributes, and any other resources the user may attach to help construct and solve the problem. The Console Optimizer has one of these problem pointers to provide the user with loading and solving functionality. This problem pointer is automatically initialized when the Console Optimizer is started and released when it stops.

Unlike the Console Optimizer, library interface users can have multiple problem pointers coexisting simultaneously in a process. The library user creates and destroys a problem pointer using the routines XPRScreateprob and XPRSdestroyprob, respectively. In the C library interface, the user passes the problem pointer as the first argument in routines that are used to operate on the problem pointer's data. Note that it is recommended that the library user destroy all problem pointers before calling XPRSfree.

```
XPRSprob prob;
XPRScreateprob(&prob);
XPRSreadprob(prob, "myMINLP.nl", "");
XPRSoptimize(prob,"",NULL,NULL);
XPRSdestroyprob(prob);
```

Controls can be set and queried, and attributes can get queried with the typical FICO Xpress Optimizer functions. The following example again reads a problem but now accesses some attributes on the problem size and the optimal solution value. Additionally, it uses the output arguments of XPRSoptimize to print information about the solution process and the found solution.

```
XPRSprob prob;
int solvestatus, solstatus;
int ncol, nrow;

/* create an environment and read a problem */
XPRSinit(NULL);
XPRScreateprob(&prob);
XPRSreadprob(prob, "myMINLP.nl", "");

/* get and print matrix dimensions */
XPRSgetintattrib(prob, XPRS_INPUTCOLS, &ncol);
XPRSgetintattrib(prob, XPRS_INPUTROWS, &nrows);

/* solve and print results */
XPRSoptimize(prob,"",&solvestatus,&solstatus);
if (solvestatus == XPRS_SOLVESTATUS_COMPLETED)
  printf("Optimization finished successfully\n");
else
  printf("Optimization terminated with status %d", solvestatus);
if (solstatus >= XPRS_SOLSTATUS_OPTIMAL || solstatus == XPRS_SOLSTATUS_FEASIBLE) {
  double objval;
  XPRSgetdblattrib(prob, XPRS_NLPOBJVAL, &objval);
  printf("We found a solution with objective value %f\n", objval);
}

/* Goodbye! */
XPRSdestroyprob(prob);
XPRSfree();
```

2.2.3 Optimizing and querying a problem

The following example contains a code snippet that will query the solution values of an (MI)NLP that has been solved with FICO Xpress Global. Note that you should use XPRSgetsolution to get a nonlinear solution, — not `XPRSgetlpsol` or `XPRSgetmipsol`. In addition, one should not use the `..mip..` methods, but always use the corresponding `..nlp..` methods (or those without qualifiers), even for MINLP problems.

```
int cols;
int sol_available;
double *x;
...
XPRSoptimize(prob,"");
XPRSgetintattrib(prob,XPRS_INPUTCOLS,&cols);
x = malloc(cols*sizeof(double));
XPRSgetsolution(prob, &sol_available, x, 0, cols-1);
if (sol_available == XPRS_SOLAVAILABLE_OPTIMAL || sol_available == XPRS_SOLAVAILABLE_FEASIBLE) {
    printf("We found a solution\n");
}
```

2.2.4 Loading a non-convex quadratic optimization problem

Another example shows how non-convex quadratic problems can be loaded via the API functions XPRSloadqp, XPRSloadqcqp, XPRSloadmiqp, or XPRSloadmiqcqp. The latter is the most comprehensive. The others are "derivatives" with fewer input arguments since specific structures like discrete variables or quadratic constraints are not present in the respective problem classes. We use XPRSloadmiqp to create an optimization problem with non-convex objective $x^2 - 3xy - y^2 - 6x$ (presented as $0.5x^TQx + c^Tx$, where $Q = (2, -3; -3, -2)$ of which the upper triangular submatrix is given), subject to a linear constraint $x + y \leqslant 1.9$ and the requirement that x must be integer.

```
/* Row data */
int nrows        = 1;            /* Number of constraints */
char rowtype[]   = {'L'};        /* Row types */
double rhs[]     = {1.9};        /* Right-hand side values */
double *range    = NULL;         /* Range values - none in this example */

/* Column data */
ncols            = 2;            /* Number of variables */
double objcoef[] = {-6, 0};      /* Linear objective function coefficients */
double lbound[]  = {0, 0};       /* Lower bounds on the columns */
double ubound[]  = {XPRS_PLUSINFINITY, XPRS_PLUSINFINITY}; /* Upper bounds (all +infinity) */

/* Data for the linear part of the constraints */
int start[]      = {0, 1, 2};  /* Start positions of the columns in matcoef.
                                  There are nVar+1 entries, the last
                                  indicating where the next column would
                                  start if there was one. Here x1 and x2
                                  both appear once linearly. */
int *collen      = NULL;         /* Number of elements in each column - not
                                  needed thanks to the last (optional) entry
                                  in nColStart */
int rowind[]     = {0, 0};       /* Row indices for the matrix elements */
double rowcoef[] = {1, 1};       /* Matrix elements - arranged by column */

/* Quadratic data */
nquad = 3;                       /* Number of elements in the upper triangular of Q */
int objqcol1[] = {0, 0, 1};      /* Index of first variable in quadratic term Q */
int objqcol2[] = {0, 1, 1};      /* Index of second variable in quadratic term Q */
double dquad[] = {2, -3, -2};    /* Values of upper triangular entries of Q, notice factor of 2 for diagor

/* Integrality data */
int nentities = 1;               /* Number of MIP entities (i.e. discrete variables) */
int entind[] = {0};              /* Indices of MIP entities (i.e. discrete variables) */
char coltype[]={'I'};            /* Integrality type, in this case integer */
nsets = 0;                       /* No special ordered sets */

...

XPRSloadmiqp(prob, "myprob", ncols, nrows, rowtype, rhs,
               range, objcoef, start, collen, rowind,
               rowcoef, lbound, ubound, nquad, objqcol1, objqcol2,
               dquad, nentities, nsets, coltype, entind, NULL,
```

```
                NULL, NULL, NULL, NULL);
```

2.2.5 Loading a nonlinear optimization problem

Our final example shows the creation of an MINLP problem with both quadratic and nonlinear expressions by creating variables and using them in string-defined expressions. The problem is

$$
\begin{array}{ll}
\min & 2x_1 + \frac{1}{2}7x_2^2 \\
\text{s.t.} & x_1^4 + x_2^4 - 100x_3 \leqslant 3 \\
& x_1 + 2x_2 + 5x_3 \geqslant 1 \\
& \sqrt{x_1 - x_2} + \log x_3 \geqslant 3 \\
& x_1 \in \mathbb{Z} \\
& x_1, x_2, x_3 \geqslant 0 \\
& x_2 \leqslant 200
\end{array}
$$

and it contains linear, quadratic, and nonlinear elements. While the linear and quadratic elements can be introduced in a single call, one needs to use other functions to add nonlinear expressions to the problem.

```
XPRSprob prob = NULL;

/* Initialise problem object */
XPRScreateprob(&prob);

/* Row data */
int nRow       = 3;                  /* Number of constraints */
char rowType[] = {'L','G','G'};      /* Row types */
double rhs[]   = {3,1,3};            /* Right-hand side values */
double *range  = NULL;               /* Range values - none in this example */
char rowName[] = "con1\0con2\0con3"; /* Constraint names */

/* Column data */
int nVar     = 3;                   /* Number of variables */
double obj[] = {2,0,0};             /* Linear objective function coefficients */
double lb[]  = {0,0,0};             /* Lower bounds on the columns */
double ub[]  = {XPRS_PLUSINFINITY,
                200,
                XPRS_PLUSINFINITY}; /* Upper bounds */
char varName[] = "x1\0x2\0x3";      /* Variable names */

/* Data for the linear part of the constraints */
int nColStart[]  = {0,1,2,4};    /* Start positions of the columns in matcoef.
                                    There are nVar+1 entries, the last
                                    indicating where the next column would
                                    start if there was one. Here x1 and x2
                                    appear once linearly, x3 appears twice. */
int *nColElem    = NULL;         /* Number of elements in each column - not
                                    needed thanks to the last (optional) entry
                                    in nColStart */
int rowInd[]     = {1,1,0,1};    /* Row indices for the matrix elements */
double matcoef[] = {1,2,-100,5}; /* Matrix elements - arranged by column */
```

This first part declares only the linear part of the problem. For the quadratic elements (only one appears in the objective function, i.e., $\frac{1}{2}7x_2^2$) and the integer variable x_1 more data is necessary before the call to XPRSloadmiqp, after which we also assign names to variables and constraints:

```
int nquad = 1;
int objqcol1[] = {1}; /* First index (i.e. x2) in the quadratic term in the objective*/
int objqcol2[] = {1}; /* Second index (x2 here as well) */
double quadcoef[] = {7}; /* Coefficient of quadratic term */
```

```
int nentities = 1, nsets = 0;  /* This problem has one MIP entity
                                  (i.e. discrete variable)
                                  and no special ordered sets */

int entind[] = {0};  /* The only discrete variable is x1 */

char coltype[]={'I'};

XPRSloadmiqp(prob, "myMINLP", nVar, nRow, rowType, rhs, range,
             obj, nColStart, nColElem, rowInd, matcoef,
             lb, ub,
             nquad, objqcol1, objqcol2, quadcoef,
             nentities, nsets, coltype, entind, NULL, NULL, NULL, NULL, NULL);

/* Add constraints and variable names */
XPRSaddnames(prob, 1, rowName, 0, 2);
XPRSaddnames(prob, 2, varName, 0, 2);
```

Adding nonlinear terms is possible by entering strings with variable names, constants, and operators separated by spaces. The expression $x_1^4 + x_2^4$ can be represented by the string `x1 ^ 4 + x2 ^ 4`, where `x1` and `x2` are the names assigned to x_1 and x_2, respectively, through the second call to XPRSaddnames. We can set the nonlinear terms for a constraint with the function XPRSnlpchgformulastr, as in the snippet below. The call to XPRSoptimize with option `"x"` allows for solving the problem to global optimality.

```
XPRSnlpchgformulastr(prob, 0, "x1 ^ 4 + x2 ^ 4");
XPRSnlpchgformulastr(prob, 2, "sqrt ( x1 - x2 ) + log ( x3 )");

XPRSoptimize(prob, "", NULL, NULL);

XPRSdestroyprob(prob);
```

2.3 Modelling via Mosel or Python

For an example of how to model a nonlinear problem via FICO Xpress Mosel, we refer to the FICO Xpress Nonlinear Manual.

For Python, the script below shows how to solve the classical *Kissing Number Problem*, a sphere packing problem in d dimensions where one seeks to determine if p unit spheres (i.e. spheres of radius 1) can be placed around a unit sphere at the center in such a way that each surrounding sphere touches the central one.

We solve this problem by creating variables for the coordinates of the centers of all surrounding spheres and an extra variable z, which we minimize, which represents the degree of superposition of spheres to one another. The model, which is quadratic yet non-convex, is as follows:

$$\begin{array}{lll} \min & z & \\ \text{s.t.} & \sum_{k=1}^{d}(x_{ik}-x_{jk})^2 \geqslant 4 - z & \forall i = 1,2,\ldots,p, \forall j = 1,2,\ldots,p, i < j \\ & \sum_{k=1}^{d} x_{ij}^2 = 4 & \forall i = 1,2,\ldots,p \end{array}$$

Its Python implementation is below. Note that this version is parametric in the number d of dimensions and the number p of spheres. The value of z in an optimal solution to the above is zero if and only if it is possible to arrange p spheres, otherwise no such arrangement exists.

```
import xpress as xp

d = 2  # number of dimensions
n = 6  # hypothetical number of unit spheres around a unit sphere centered at (0,0)
N = range(n)
```

```
p = xp.problem()

x = p.addVariables(n, d, lb=-2, ub=2)  # coordinates of the centers (can't be
                                       # below -2 or above 2)

z = p.addVariable()  # violation of superposition of spheres

p.setObjective(z)  # z is to be minimized

# orbital spheres don't overlap each other unless z > 0
p.addConstraint(xp.Sum((x[i,:] - x[j,:])**2) >= 4 - z for i in N for j in N if i<j)

# orbital spheres have distance two to the central one at the origin so that they don't overlap it
p.addConstraint(xp.Sum(x[i,:]**2) == 4 for i in N)

p.optimize('x')
```

Visualizing the solution for two dimensions and any value of p is easy through the `matplotlib` module with the snippet below, which results, for $p = 6$, in Figure 2.1 below.

```
import matplotlib.pyplot as plt

center = plt.Circle((0,0), 1, color='red')

# Create circles whose center is the solution obtained from the above
# problem
spheres = [plt.Circle(xp.evaluate([x[i,j] for j in range(d)],
                                  problem=p), 1, color='blue') for i in N]

fig, ax = plt.subplots()

ax.set_xlim((-3,3))
ax.set_ylim((-3,3))

ax.add_patch(center)
for i in N:
    ax.add_patch(spheres[i])

plt.show()
```

2.4 Dealing with unboundedness and infeasibility

A linear optimization problem, with or without binary or integer variables, that does not admit any feasible solution is called *infeasible*, and the FICO Xpress Optimizer has efficient algorithms to determine and investigate the infeasibility of an LP problem.

Similarly, a (mixed-integer) linear optimization problem that does not admit a finite optimum is called *unbounded*. Both for LP and MILP problems, determining unboundedness can be done efficiently and often leads to re-writing the optimization model.

In MINLP, infeasibility can be diagnosed similarly to MILP: either the LP relaxation of the original MINLP problem is infeasible or, through a branch-and-bound run, it is determined that no feasible solution exists for the MINLP.

However, unboundedness requires special care in MINLP. Unlike for an MILP problem, where unboundedness of its LP relaxation implies unboundedness of the MILP, for MINLP problems an unbounded relaxation warrants further investigation.

As an example, the problem $\min\{x : x^3 - x^2 \geqslant 0\}$, which is reformulated to $\min\{x : w_1 - w_2 \geqslant 0, w_1 = x^3, w_2 = x^2\}$, does not have lower or upper bounds on either auxiliary variable

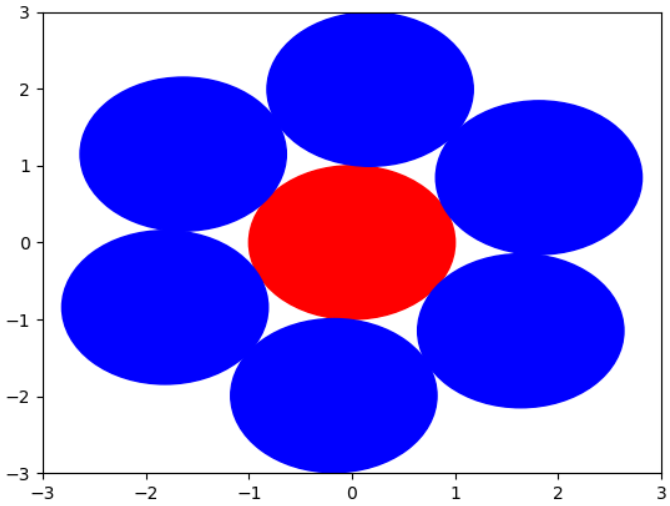


Figure 2.1: Solution of the KNP in two dimensions for p=6

due to the absence of strong bounds on x. Although it is clear that the optimal solution is $x = 0$, the first LP relaxation is unbounded. In general, even though bound reduction can help tighten the bounds on all variables and thus make the problem bounded, there are edge cases where this does not happen.

FICO Xpress Global handles this case as follows: first, if the initial root relaxation is found to be unbounded, new lower and upper bounds are imposed on every *original* (i.e., non-artificial) variable: $-B \leqslant x_i \leqslant B$, where B is determined by the control GLOBALBOUNDINGBOX and can be set by the user (and the same control can be used to disable this procedure altogether). Second, the LP relaxation is solved again. If the new (restricted) LP relaxation is still unbounded, the same bounds are imposed on the auxiliary variables as well, thus making the problem surely bounded.

Regardless of the value of GLOBALBOUNDINGBOX, if the above procedure is carried out on an MINLP, i.e., if the initial LP relaxation is found to be unbounded, this implies that the solution produced by FICO Xpress Global is *not* a guaranteed globally optimal solution, since the artificial bounds may exclude an optimal solution. If an optimal solution for the new problem is found, the solution status will be `LOCALLY_OPTIMAL` rather than `OPTIMAL`.

This remarks again how important it is to formulate an optimization problem in the correct way. In order to make unboundedness unlikely and hence retain the chance of finding an optimal solution, it is useful to impose valid bounds on as many variables as possible.

2.5 File Formats for MINLP problems

Three file formats can be used to read an MINLP problem into FICO Xpress Global: `NL`, `MPS`, and `LP`, whose file extensions are `.nl`, `.mps`, and `.lp`, respectively. Whether it is with the READPROB command on the Optimizer Console or through the XPRSreadprob and XPRSwriteprob functions in the C library, these three file formats can be used for loading or saving an MINLP problem from/to a file.

The `NL` format is universally used to input MINLP with nonlinear, non-quadratic objective functions and constraints. As for the `MPS` and `LP` formats, their extension to handle nonlinear, non-quadratic expressions is not standard and should only be used to read and write MINLP problems for use by FICO Xpress Optimizer.

The `MPS` and `LP` formats are instead universally accepted for MINLPs with quadratic (both convex and

non-convex) objective and constraints, and the same holds for `NL` files.

Chapter 3

Problem Types

FICO Xpress is a powerful optimization tool for solving Mathematical Optimization problems. FICO Xpress Global allows users to formulate problems with nonlinear, potentially non-convex, constraints and objective over continuous and integer variables and solve them to proven global optimality.

Mathematical Optimization problems are usually classified according to the types of decision variables, constraints, and objective function. This section will briefly introduce some important types of problems, focusing on distinctions important to FICO Xpress Global.

3.1 Linear vs. Nonlinear Problems

The field of Mathematical Optimization has its roots in solving linear programming (LP) problems. In this type of problem, all constraints and the objective function are linear expressions of the decision variables. The FICO Xpress Optimizer implements very efficient algorithms for LP solving, namely the simplex, the barrier and the hybrid gradient methods. Computational theory confirms that we can solve LPs efficiently (in polynomial time) and precisely.

Nonlinearity makes the matter more complicated. An optimization problem is referred to as a nonlinear program (NLP) as soon as at least one constraint or the objective function contains a nonlinear expression (e.g., a polynomial or a trigonometric function). For computational complexity and the required solution methods, it does not, in general, make a difference how many of the constraints are nonlinear. However, many nonlinear problems in practice consist of a strong linear core with only a few additional nonlinear constraints. This can be favorable for the convergence speed of the solver. Concerning efficiency, on both the theoretical and the practical side, the most important distinction is between convex and non-convex nonlinearities; see the next section.

Finally, note that algorithms for nonlinear problems approximate the solution up to a certain precision. While small numerical violations may or may not occur in linear problems due to numerical issues, in nonlinear optimization, tiny violations both on the dual and the primal side are almost always to be expected.

3.2 Convex vs. Non-Convex Problems

A set (e.g., the feasible region of an optimization problem) is called convex when for any two points in the set, the line connecting them is entirely contained in the set itself. For example, a disc is convex, and a heart shape is non-convex, see Figure 3.1. This directly relates to a very crucial property for optimization. Every local optimum is a global optimum when optimizing a convex function over a convex set. This makes convex optimization much easier, both theoretically and practically. Local optima may occur when either the objective function or the feasible region described by the constraints are non-convex.

Even more challenging, a feasible set described by non-convex constraints might be disconnected. Consider the situation in Figure 3.2. We see the feasible region of a non-convex NLP, defined by some

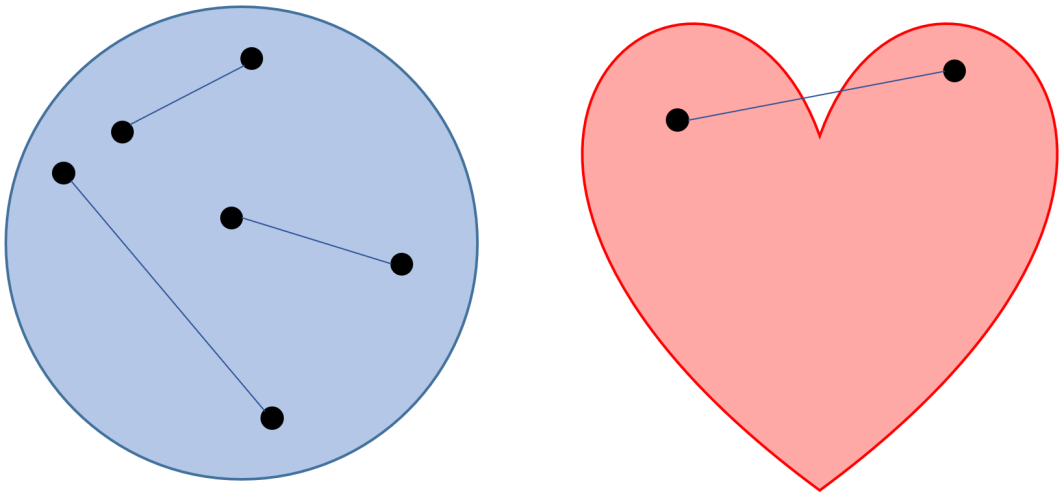


Figure 3.1: A convex and a non-convex set

variable bounds and two inequalities that contain a sine term. The feasible region, shaded in dark blue, consists of six distinct feasible components. When optimizing the red, linear objective function, there are nine local (including one global) optima: the red points.

While convex optimization is comparably easy (solvable in polynomial time, up to a certain precision), non-convex optimization is theoretically even harder than MIP. For non-convex problems, it is impossible to formulate guidelines for expected runtimes. In extreme cases, even non-convex NLPs with only a handful of variables can take an eternity to solve to global optimality. In contrast, others with thousands of constraints and variables might be solvable within seconds.

Note that FICO Xpress Global is different from FICO Xpress Nonlinear, whose documentation can be found here. FICO Xpress Nonlinear targets finding locally optimal solutions for non-convex problems. FICO Xpress Global finds globally optimal solutions for non-convex problems. For convex problems, both will find globally optimal solutions. It cannot be readily determined a priori which of the two solvers will be superior in performance since both tackle the same problem with somewhat different approaches. For a particular convex problem, it is worth checking both independently. These considerations hold both for the continuous and mixed-integer case.

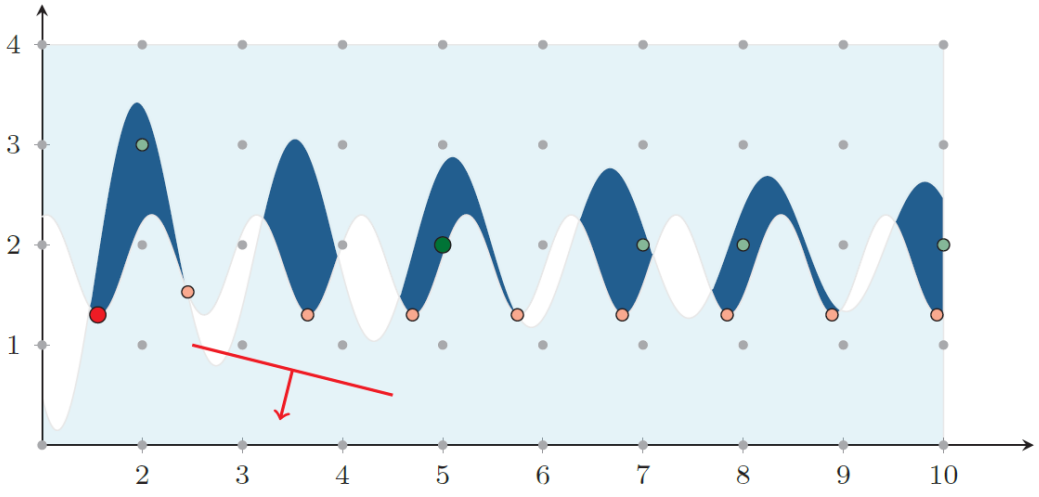


Figure 3.2: A non-convex (MI)NLP

3.3 Continuous vs. Mixed-Integer Problems

As in the linear case, an important distinction is between problems with purely continuous variables and those for which at least one variable is integer. Problems that are both nonlinear and have integer variables are called MINLPs, mixed-integer nonlinear programs. As in the LP/MIP case, discrete variables make convex nonlinear problems harder to solve but are often imperative for practical applications. Curiously, a non-convex MINLP can be easier to solve than its continuous counterpart; nevertheless, non-convex MINLPs are among the hardest optimization problems that occur in practice. As an example of a non-convex MINLP, consider again Figure 3.2, now assuming integrality conditions for both variables. The MINLP has five feasible solutions, indicated in green, with the globally optimal one being ”in the middle”, at (5,2).

Note that FICO Xpress Global will solve non-convex MINLPs to proven global optimality, whereas there are many solvers that can only guarantee local optimality for non-convex NLPs and have no optimality guarantees for non-convex MINLPs.

Even though we did not mention them explicitly yet, FICO Xpress Global supports other types of MIP entities, most importantly special-ordered sets (SOSs) and semi-continuous variables. Further, indicator constraints can be added to MINLPs.

Chapter 4

Solution Methods

This chapter provides a summary of the methods used within FICO Xpress Global to solve MINLP problems. While the solution method is based on the branch-and-bound paradigm, and most of its features are those described in the general manual, solving an MINLP problem requires an extra set of tools to provide a guaranteed optimal solution or, if the time limit is reached, a valid bound. Specifically, FICO Xpress Global implements the following specialized tools for solving MINLPs:

- reformulation of an MINLP;
- creation of an LP relaxation of an MINLP;
- branching on continuous variables, or *spatial* branching;
- bound reduction;
- heuristics.

This chapter describes these tools in detail.

4.1 Reformulation of an MINLP

We consider an optimization problem where a possibly nonlinear, non-convex function must be minimized either unconstrained or subject to a set of constraints, also possibly nonlinear and non-convex. We additionally consider constraints on the variables in the form of lower and upper bounds and integrality. The notation below will help us navigate all methods described in this chapter.

$$\begin{array}{lll} \min & f_0(\mathbf{x}) & \\ \text{s.t.} & f_i(\mathbf{x}) \leqslant 0 & \forall i = 1, 2, \ldots, m \\ & x_j \in [\ell_j, u_j] & \forall j = 1, 2, \ldots, n \\ & x_j \in \mathbb{Z} & \forall j \in J \end{array}$$

Here the set of all variables is represented by the vector $\mathbf{x}$. Both the objective function and the constraints are functions of $\mathbf{x}$. All variables have lower and upper bounds, and a subset J of them is also integral.

Similar to mixed-integer linear programming problems, how a problem is formulated can strongly impact how efficiently it is solved.

4.1.1 Factorable functions

We begin with an assumption on how the problem is formulated: all functions f_i, for $i = 0, 1, \ldots, m$, are *factorable*: this means that they result from the composition of a finite set of operators on variables and constants. In other words, the *closed form* of these functions is known; user functions computed by calling, e.g., a C/Java/Python function are currently not supported. The allowed set of operators is the

same we use to write nonlinear expressions, including but not restricted to the standard operators $\{+, -, \times, /, x^y\}$, univariate functions like $\{\sqrt{\cdot}, \sin, \log\}$ and many more. For an exhaustive list of the almost twenty supported nonlinear functions, see the section Internal Functions in the FICO Xpress Nonlinear manual. Examples of factorable functions, composed from those operators, include:

$$\sum_{j=1}^{n} e^{x_j} x_j^3 \qquad x_1 \sin(x_2) \qquad x \sin\left(\tfrac{1}{x}\right)$$
$$x_1^3 - 3x_1^2 + 3x_1 - 1 \qquad \sqrt{\sum_{j=1}^{n} x_j^2} \qquad \prod_{j=1}^{n} \log(x_j)$$

Functions that are not factorable are for example $x_1 \sin(x_2)\text{myfun}(x_3)$, where *myfun* is implemented as a *black-box* function using, for instance, our interface for user functions; Because *myfun* is not defined in closed form from the above set of operators, FICO Xpress Global will not be able to find a globally optimal solution, and instead, local optimization solvers in the FICO Xpress suite, such as SLP or Knitro, should be used.

From here on, we assume all constraints and the objective function are specified as factorable functions. We will use a small example to explain all steps taken by the Optimizer:

$$\begin{aligned} \min \quad & x_1^2 - 2x_2^4 \\ \text{s.t.} \quad & x_1 x_2 \geqslant 4 \\ & x_1 + \tfrac{1}{2} e^{x_2^2} \leqslant 4 \\ & 0 \leqslant x_1 \leqslant 4 \\ & -1 \leqslant x_2 \leqslant 3 \\ & x_2 \in \mathbb{Z} \end{aligned}$$

4.1.2 Reformulation and auxiliary variables

When solving an MINLP with the FICO Xpress Global solver, from the solver log one can notice that several new variables and constraints are introduced. Below we explain the process that leads to this possible increase in problem size.

Every factorable MINLP is solved by creating a set of new variables assigned to subexpressions appearing in the problem. These new variables are called *auxiliary variables*, or *auxiliaries* for short. Each forms a pair with a portion of an expression appearing in all functions of the problem, namely a subexpression that uses only one operator of the set mentioned above. For the example MINLP, we add five auxiliaries, which we call w_1, w_2, w_3, w_4, w_5 and which are assigned as follows:

$$w_1 = x_1^2; \qquad w_3 = x_1 x_2; \qquad w_5 = e^{w_4}$$
$$w_2 = x_2^4; \qquad w_4 = x_2^2$$

Introducing auxiliaries does *not* change the problem substantially, but it helps solve it. Now we can rewrite the problem as

$$\begin{aligned} \min \quad & w_1 - 2w_2 \\ \text{s.t.} \quad & w_3 \geqslant 4 \\ & x_1 + \tfrac{1}{2} w_5 \leqslant 4 \\ & w_1 = x_1^2 \\ & w_2 = x_2^4 \\ & w_3 = x_1 x_2 \\ & w_4 = x_2^2 \\ & w_5 = e^{w_4} \\ & 0 \leqslant x_1 \leqslant 4 \\ & -1 \leqslant x_2 \leqslant 3 \\ & 0 \leqslant w_1 \leqslant 16 \\ & 1 \leqslant w_2 \leqslant 81 \\ & -4 \leqslant w_3 \leqslant 12 \\ & 0 \leqslant w_4 \leqslant 9 \\ & 1 \leqslant w_5 \leqslant e^9 \\ & w_2, x_2 \in \mathbb{Z} \end{aligned}$$

The following three observations shall help to understand why the reformulation is performed this way. First, note that the only nonlinear part of the problem is now in the constraints that define the auxiliaries – the objective function and the original constraints have become linear by replacement with some auxiliaries. Second, these auxiliaries also have lower and upper bounds because the expression associated with each of them is bounded. Finally, because x_2 is integer, so is w_2.

4.2 Creation of an LP relaxation of an MINLP

The second step in solving an MINLP is creating a relaxation of the nonlinear constraints that consists only of linear constraints. A valid relaxation provides a lower bound on the optimal objective value and can be exploited in a branch-and-bound algorithm that is similar to what the FICO Xpress Optimizer uses for mixed-integer linear programming problems.

To this purpose, all constraints that determine the relationship between an auxiliary and a nonlinear function are relaxed by one or more linear constraints. The term *relaxation* here refers to the idea that all feasible solutions to the original problem are also feasible for the problem with the amended linear inequalities.

For example, the constraint $w_1 = x_1^2$, which is nonlinear and non-convex, can be relaxed by four linear constraints as shown below (note that any (x_1, w_1) in the bound intervals specified above and satisfying $w_1 = x_1^2$ also satisfy all of these linear constraints), the first three of which are so-called McCormick inequalities, while the fourth one is an Outer Approximation cut:

$$w_1 \leqslant 4x_1$$
$$w_1 \geqslant 0$$
$$w_1 \geqslant 8x_1 - 16$$
$$w_1 \geqslant 4x_1 - 4$$

Similar sets of linear constraints can be created to replace the other nonlinear constraints $w_2 = x_2^4$, $w_3 = x_1x_2$, $w_4 = x_2^2$ and $w_5 = e^{w_4}$. We call such linear inequalities *convexification cuts* because they allow us to create a convex (LP) relaxation of a non-convex nonlinear problem. The result is a problem with only linear constraints and a linear objective function, and solving it yields a valid lower bound to the original MINLP.

Such a problem is handled by FICO Xpress Global's branch-and-cut solver, which applies the same machinery as the classic MILP solver and extended features that are suited to MINLPs.

4.2.1 Refining the relaxation

The MILP solver in the FICO Xpress Optimizer is a branch-and-cut algorithm: a branch-and-bound solver that also adds linear inequalities at, possibly, every branch-and-bound subproblem to speed up computation and obtain a tight lower bound (in case of a minimization problem) of the optimal objective function value.

These linear inequalities are added after the first (root) relaxation in an iterative fashion, and also during tree search.

Such linear inequalities added in the MILP framework exploit the structure and features of the MILP problem. Common names for these classes of inequalities are *Gomory cuts*, *Mixed-Integer Rounding cuts*, etc.

FICO Xpress Global uses a similar framework to amend the initial LP relaxation with extra linear inequalities for each auxiliary variable multiple times in the *root-cutting* phase and also during the branch-and-bound. This helps *refine* the relaxation shown above in much the same way as an MILP solver.

4.3 Running branch-and-bound on an MINLP

The above section has pointed out the role of convexification cuts in solving an MINLP. Another key factor in an efficient MINLP solver is branching, which subdivides a single branch-and-bound subproblem into two or more subproblems by employing extra constraints on each subproblem.

For solving an MINLP, it is necessary to generalize the MILP paradigm: in the latter, an integer (or binary) variable x_i that takes a fractional value x_i^* in a subproblem can be branched upon, thus creating two new subproblems: the first has the additional constraint $x_i \leqslant \lfloor x_i^* \rfloor$, while $x_i \geqslant \lceil x_i^* \rceil$ is added to the second.

Because of the nonlinear, non-convex constraints that appear in MINLPs, it can be necessary to branch on *continuous* variables as well as on integer variables.

4.3.1 Spatial Branching

Consider again the auxiliary w_1 and its associated expression $w_1 = x_1^2$, which is nonlinear and non-convex. Variable x_1 is continuous and bounded between 0 and 4. Figure 4.1 shows the graph of this function and, in the shaded area, the feasible region created by the following four convexification cuts:

$w_1 \leqslant 4x_1$ $\qquad$ $w_1 \geqslant 0$
$w_1 \geqslant 8x_1 - 16$ $\qquad$ $w_1 \geqslant 4x_1 - 4$.

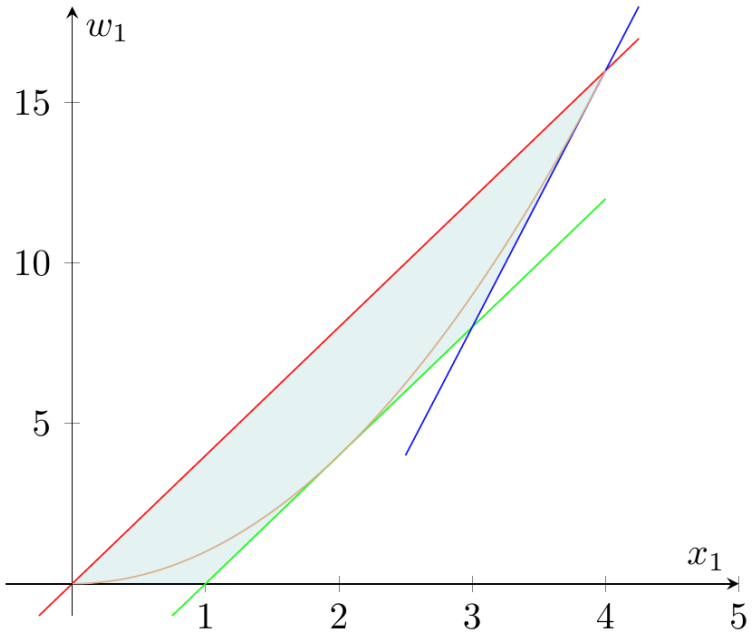


Figure 4.1: A linear relaxation of the square function

As mentioned above, these linear inequalities are satisfied by all points (x_1, w_1) which satisfy $w_1 = x_1^2$. While more convexification cuts can be added to strengthen the relaxation, one can easily see that no such cut exists for the point $(x_1, w_1) = (2, 5)$. The only way to eliminate this solution is a branch operation, even though x_1 is a continuous variable.

Branching on continuous variables is called *spatial branching* for historical reasons, but the effect is similar to that of branching on integer variables: two new subproblems are created, and the tree search continues.

The FICO Xpress Global solver extends the branch-and-bound algorithm that works for the FICO

Xpress Optimizer to continuous variables in MINLP problems. While the structure of the two algorithms is essentially the same, the FICO Xpress Global solver tackles MINLP problems by equipping the MILP solver with extra features for reducing the process of solving an MINLP to that of solving an MILP.

4.4 Bound reduction

Convexification cuts, mentioned above, are one of the crucial MINLP solving capabilities of the FICO Xpress Global solver. The quality of such cuts, measured by how tight the resulting lower bound is, depends strongly on the lower and upper bounds of all problem variables, including the auxiliaries.

For this reason, one of the main presolving components used by the Global solver is *bound reduction*, sometimes also referred to as propagation or node presolving. Bound reduction is a class of algorithms that, given the lower and upper bounds of all variables, can obtain tighter bounds on one or more variables that any feasible and optimal solution has to satisfy. Bound reduction techniques might exclude some feasible solutions but ensure that always at least one optimal solution remains, thereby not changing to optimal objective value for the reduced problem.

Given an MINLP in the general form $\min\{f(\mathbf{x}) : \mathbf{x} \in X\}$, the purpose of a bound reduction algorithm is to tighten the bounds on all variables x_i of the problem.

One common technique is a straight generalization of what is used in MILP.

4.4.1 Feasibility-based bound reduction

Perhaps the best-known form of bound reduction uses the original constraints of the MINLP to infer better bounds on both original and auxiliary variables.

Consider the part of our MINLP example that stems from the second constraint, $x_1 + \frac{1}{2}w_5 \leqslant 4$ and consider the variables involved in it, i.e., x_1, x_2, and w_4, w_5, with initial bound intervals equal to $[0, 4]$, $[-1, 3]$, $[1, e^9]$, and $[0, 9]$ respectively. We copy that portion here for convenience:

$$
\begin{aligned}
& x_1 + \tfrac{1}{2}w_5 \leqslant 4 \\
& w_5 = e^{w_4} \\
& w_4 = x_2^2 \\
& 0 \leqslant x_1 \leqslant 4 \\
& -1 \leqslant x_2 \leqslant 3 \\
& 0 \leqslant w_4 \leqslant 9 \\
& 1 \leqslant w_5 \leqslant e^9 \\
& x_2 \in \mathbb{Z}
\end{aligned}
$$

The constraint $x_1 + \frac{1}{2}w_5 \leqslant 4$ implies, given that $x_1 \geqslant 0$, that $w_5 \leqslant 8 < e^3$, which in turn leads to new upper bounds on w_4 and x_2, i.e., $w_4 \leqslant \ln(8)$ and $x_2 \leqslant \sqrt{\ln(8)}$. Integrality of x_2 further tightens its upper bound to $\lfloor \sqrt{\ln(8)} \rfloor = 1$.

Bound reductions like the ones above usually have a knock-on effect on other elements of the problem. Tighter bounds on some variables often lead to tighter bounds of other variables and so on. Moreover, tighter variable bounds frequently lead to tighter LP relaxations, see the beginning of this chapter, and therefore to a better bound on the optimal solution value and potentially even better feasible solutions being found more quickly.

The above bound reduction technique is often denoted as *feasibility-based bound reduction* and uses the nonlinear information from the problem. Other techniques are known that further restrict the variable bounds, but some of them are computationally expensive, and their use is normally limited to a small number of calls.

4.4.2 Optimization-Based Bound Tightening

Optimization-Based Bound Tightening (OBBT) uses the LP relaxation of a MINLP problem to infer tighter bounds on its variables, both original and auxiliary.

Given an LP relaxation to a MINLP described as $\min\{c^T x : Ax \geqslant b, x \in [\ell, u]\}$ and subset of variables $x_1, x_2, \ldots, x_k$, OBBT seeks a better lower bound on each of them by solving $\min\{x_i : Ax \geqslant b, x \in [\ell, u]\}$ for each $i \in 1, 2, \ldots, k$.

If the result of the i-th optimization problem is $x_i^\star > \ell_i$, then $x_i^\star$ is a tighter lower bound on x_i. Similarly, x_i can be maximized in order to obtain a (possibly tighter) upper bound.

OBBT is generally understood to provide tight bounds on the problem's variables. However, given that it entails solving a possibly large number of LP problems, this type of bound reduction can be computationally taxing. Therefore an effort limit control is available for this class of bound reduction algorithms.

4.5 Heuristics

The FICO Xpress Global solver uses heuristics to find a feasible solution to an MINLP. Similar to an MILP solver, heuristics can be called both at the beginning of the solution process or during branch-and-bound.

To find feasible solutions, enforcing nonlinear constraints while working with an LP relaxation is not easy. For this reason, heuristics use a nonlinear optimization solver such as SLP or Knitro. Both are nonlinear solvers that can find a locally optimal solution, hence of objective value possibly inferior to that of the global optima. The effectiveness of such an approach stems from the efficiency of the local solvers, which can find a local solution in a fraction of the time it would require an MINLP solver to find a global optimum.

One technique to find feasible solutions is to fix all the integer variables to an integral value and then solve the resulting continuous nonlinear problem. If the resulting solution is feasible for the restricted problem, it is also feasible for the original problem and can be used as a cutoff. This procedure is regularly applied to integer solutions of the LP relaxation (which are typically not feasible for the nonlinear constraints), as well as solutions found by the MILP heuristics.

CHAPTER 5

References

For documentation of controls, attributes and callable functions, we refer to the reference manuals of the FICO Xpress Optimizer and FICO Xpress Nonlinear. The Nonlinear manual gives references for NLP-specific controls, attributes, and functions (e.g., setting of the SLP algorithm), while the Optimizer manual refers to general optimization and branch-and-cut controls, attributes and functions.

Appendix A

Contacting FICO

FICO provides clients with support and services for all our products.

FICO Customer Support

FICO Customer Support offers technical support and services ranging from self-help tools to direct assistance with a FICO technical support engineer. Support is available to all clients who have an active maintenance contract.

The FICO Customer Self-Service Portal (support.fico.com) is a secure web portal that allows users to open, review, and update their support cases; manage their organization's portal users; find solutions to common problems in the FICO Knowledge Base; and view the availability of their cloud applications 24 hours a day, 7 days a week.

You can find support contact information and a link to the FICO Customer Self-Service Portal (online support) on the Product Support home page (www.fico.com/en/product-support).

Please include *'Xpress'* in the subject line of your support queries.

Documentation

FICO continually looks for new ways to improve and enhance the value of the products and services we provide.

If you have comments or suggestions regarding how we can improve this documentation, let us know by sending your suggestions to techpubs@fico.com. Please include your contact information (name, company, email address, and optionally, your phone number) so we may reach you if we have questions.

FICO Learning

FICO Learning is the principal provider of product training for our clients and partners. FICO Learning offers instructor-led classroom courses, web-based training, seminars, and training tools for both new user enablement and ongoing performance support.

For additional information, visit the FICO Learning home page at www.fico.com/en/product-training or email producteducation@fico.com.

Sales and maintenance

If you need information on other Xpress Optimization products, or you need to discuss maintenance contracts or other sales-related items, contact FICO by:

- Phone: +1 (408) 535-1500 or +44 207 940 8718
- Web: www.fico.com/optimization and use the available contact forms

About FICO

FICO (NYSE:FICO) is a leading analytics software company, helping businesses in 90+ countries make better decisions that drive higher levels of growth, profitability, and customer satisfaction. Learn more at www.fico.com or contact us at www.fico.com/en/contact-us.

Index

A
Artificial bounds, 10
Attributes, 5
Auxiliary variable, 16

B
Black-box optimization, 16
Bound reduction, 19
    Feasibility based, 19
Bounding box, 10
Branch-and-cut, 17
Branching, 18

C
Callable library, 4
Console Optimizer, 3
Controls, 5
Convex optimization, 12
Convex set, 12
Convexification cut, 17

D
Decomposition, 16
Disconnected, 12

E
Exponential and logarithmic functions, 16
Expression strings, 8

F
Factorable function, 15
File formats, 10

G
`GLOBALBOUNDINGBOX`, 10

H
`HELP`, 3
Heuristics, 20

I
Indicator constraints, 14
Infeasibility, 9

K
Kissing Number Problem, 8

L
License, 4
Local vs global optimality, 4, 12
LP files, 10
LP relaxation, 17

M
McCormick, 17
MINLP, 14
MIP entities, 14
MIQP, 6
Mosel, 8
MPS files, 10

N
NL files, 10
`NLPOPTIMIZE`, 3
`NLPSOLVER`, 3
Node presolving, 19
Non-convex optimization, 12
Non-convex quadratic, 8
Nonlinear elements, 7
Nonlinear program, 12

O
`OPTIMIZE`, 3
Outer Approximation, 17

P
Polynomials, 16
Problem
    attributes, 5
    basic solving, 3
    callable library, 4
    controls, 5
    infeasible, 9
    pointers, 5
    release, 4
    solution, 5
    unbounded, 9
Propagation, 19
Python, 8

Q
Quadratic optimization, 6

R
`READPROB`, 3, 10
Reformulation, 15, 16

S
Security system, 4
Semi-continuous variables, 14
Spatial branching, 18
Special-ordered sets (SOS), 14

T
Trigonometric functions, 16

U
Unboundedness, 9
User functions, 15

X
XPRSaddnames, 8
XPRScreateprob, 5
XPRSdestroyprob, 5
XPRSfree, 4
XPRSgetsolution, 5
XPRSinit, 4
XPRSloadmiqcp, 6
XPRSloadmiqp, 6
XPRSloadqcp, 6
XPRSloadqp, 6
XPRSnlpchgformulastr, 8
XPRSoptimize, 5, 8
XPRSreadprob, 10
XPRSwriteprob, 10