cGOP: A Deterministic Global Optimization Package

User's Guide

V. Visweswaran and C. A. Floudas
Department of Chemical Engineering
Princeton University
Princeton, NJ 08544

VERSION 1.1

February 10, 1997

Preface

cGOP is a package for rigorously solving nonconvex optimization problems to global optimality. The package implements the GOP algorithm (Floudas and Visweswaran, 1993), which is applicable to general constrained biconvex problems, using a set of C subroutines to solve these problems using decomposition and branch-and-bound techniques. It also incorporates several improvements made to the original GOP algorithm to reduce the computational complexity (Visweswaran and Floudas, 1993), as well as new formulations that permit implicit solutions of some of the subproblems encountered during the algorithmic steps (Visweswaran and Floudas, 1995). The algorithms use local optimization solvers (currently MINOS (Murtagh and Saunders, 1988) and CPLEX (CPLEX, 1995)) to solve linear, mixed-integer linear and convex subproblems. This version of the **cGOP** package can be used to solve problems with linear constraints. The original algorithm and its variants can be accessed by calls to high-level subroutines as well as through a standalone mode for quadratic problems. Furthermore, it can also be accessed using a high-level interface that permits easy description of the problems. The package has been in use at the Computer-Aided Systems Laboratory in Princeton University for the last two years, and has been used to solve problems involving several hundred variables and constraints.

This version is intended for use only in **ACADEMIC RESEARCH**. Commercial sites wishing to use this product should contact one of the authors directly.

Contents

1. Introduction

cGOP is an efficient implementation of rigorous global optimization algorithms designed to solve a class of *nonconvex* nonlinear problems. The package is written in C, and can currently be used to solve problems of the following form:

$$\min_{x,y} \quad c_0^T x + d_0^T y + x^T Q y + F_1(x) + F_2(y) \tag{1}$$

$$\text{subject to} \quad c_i^T x + d_i^T y \leq e_i \qquad i = 1, \ldots, m_1 \tag{2}$$

$$c_i^T x + d_i^T y = e_i \qquad i = m_1 + 1, \ldots, m_2 \tag{3}$$

$$x^L \leq x \leq x^U \tag{4}$$

$$y^L \leq y \leq y^U \tag{5}$$

where x and y are n_1 and n_2 vectors of variables, c, d, l and u are constant vectors. Q is a constant matrix, and can be indefinite. The functions $F_1(x)$ and $F_2(y)$ are smooth *convex* scalar functions of x and y respectively, and are provided by the user through a callable Fortran or C subroutine. Ideally, these subroutines should also provide the first derivatives of the two functions; in the absence of such derivatives, the algorithms will estimate them using finite differences. Note that some (or all) of the y variables can be binary variables; in this case, the function $F_2(y)$ must be linear.

Note that the form described by (1)-(5) is quite general, covering indefinite quadratic problems and general biconvex problems, among others. In some cases, although the original problem may not fall into the category of (1)-(5), it can be transformed using additional variables and constraints into this form. For example, the general indefinite quadratic problem with linear constraints, given by

$$\min_{x} f(x) = c^T x + \frac{1}{2} x^T Q x$$

$$s.t. \qquad Ax \leq b$$

$$x \geq 0$$

can be transformed to the form (1)-(5) as shown below:

$$\min_{x,y} f(x, y) = c^T x + \frac{1}{2} x^T Q y$$

$$s.t. \qquad Ay \leq b$$

$$x - y = 0$$

$$x \geq 0$$

where a new set of variables y has been introduced along with a set of constraints describing the equivalence between x and y. It should be noted that there are usually several ways to achieve these kinds of transformations. The best transformations are usually those which keep the convex terms in the objective function unaltered, and only "bilinearize" the nonconvex terms.

cGOP is designed as a library of subroutines that can be called from any high level programming language. It has been written in portable ANSI C, and is currently available for the HP 9000, IBM RS6000, SUN4 and Silicon Graphics architectures. It requires a local optimization package for the solution of linear, convex and mixed-integer linear subproblems. Currently, the library can utilize the MINOS (Murtagh and Saunders, 1988) and CPLEX (CPLEX, 1995) packages for solving these subproblems; therefore, one or more of these packages is required for **cGOP** to run. There are no restrictions on the sizes of problems that can be solved; the algorithms are only limited by the available memory and CPU resources.

2. Description of the Algorithms

The primary methods utilized in **cGOP** are the GOP and GOP/MILP algorithms for optimizing nonconvex problems. These algorithms are described in Floudas and Visweswaran (1993) and Visweswaran and Floudas (1995), and are designed to solve problems of the form (1)-(5), among others. The GOP algorithm is a primal-relaxed dual decomposition method that solves the original problem through a series of linear/convex subproblems that can be solved by a conventional local optimization solver. The GOP/MILP algorithm is a variant of the GOP algorithm that solves these subproblems implicitly using a mixed-integer linear programming formulation.

cGOP implements the GOP and GOP/MILP algorithms using a branch-and-bound framework. At each node of the branch-and-bound tree, a *primal* problem is solved in the x variables to provide an upper bound on the global solution. The solution of this problem is used to construct a Lagrange function that underestimates the optimal solution of the original problem in the current domain of interest. The gradients of the Lagrange function are then used as a basis for partitioning this domain into subdomains. Subsequently, in each subdomain, a linear or convex lower bounding problem in y is solved. The solutions to these problems provide a set of "children" nodes for the incumbent node, and also provide a lower bound for the global solution. One of these children nodes is then selected as a candidate node for further exploration, and the procedure is repeated. Due to accumulation of cuts from previous iterations, the upper and lower bounds are successively tightened until the algorithm converges to the global solution.

The key features of the algorithm are described below. The full algorithm are described in Appendix A.

2.1. Root Node and Starting Region

At the beginning of the algorithm, there are no subdivisions in the y-space. Therefore, the root node in the branch and bound tree is simply the starting point for the algorithm, y^1. The region of application for this node (i.e., the *current* region) is the entire constraint region described by (2)-(5).

2.2. Primal Problem (Upper Bounding Problem)

The primal problem is simply the original problem (1)-(5) solved at the current value of y, say y^k. This problem is given by

$$\min_{x} \quad c_0^T x + d_0^T y^k + x^T Q y^k + F_1(x) + F_2(y^k) \tag{6}$$

$$\begin{aligned} \text{subject to} \quad & c_i^T x + d_i^T y^k \le e_i & i = 1, \ldots, m_1 \quad & (7) \\ & c_i^T x + d_i^T y^k = e_i & i = m_1 + 1, \ldots, m_2 \quad & (8) \\ & x^L \le x \le x^U & & (9) \end{aligned}$$

If the function $F_1(x)$ exists, this problem is convex; otherwise, it is a linear problem. In either case, a local solver (MINOS or CPLEX) can used to solve the problem. Because the y variables have been fixed in this problem, any feasible solution[1] to this problem provides an upper bound to the global solution of (1)-(5). Moreover, the solution provides a solution x^k for the x variables, and multipliers μ^k and λ^k for the constraints (7) and (8), and multipliers ν_L^k and ν_U^k for the active lower and upper bounds in (9).

[1]Note that the value of y^k that is used in this problem might be such that there is no feasible solution to the system of constraints (7)-(9). In such a case, a *relaxed* primal can be solved to minimize the sum of infeasibilities for these constraints. It will be shown later how it can be ensured that the values of y^k used for this problem always give feasible solutions.

2.3. Branching: The Lagrange Function

From the solution of the primal problem, the following Lagrange function can be constructed:

$$\begin{aligned} L^k(x,y,\mu^k,\lambda^k,\nu^k) &= c_0^T x + d_0^T y + x^T Q y + F_1(x) + F_2(y) \\ &+ \sum_{i=1}^{m_1} \mu_i^k \cdot (c_i^T x + d_i^T y - e_i) \\ &+ \sum_{i=m_1+1}^{m_2} \lambda_i^k \cdot (c_i^T x + d_i^T y - e_i) \\ &+ \sum_{i=1}^{n_1} \nu_{L_i}^k \cdot (x^L - x) + \sum_{i=1}^{n_1} \nu_{U_i}^k \cdot (x - x^U) \end{aligned}$$

This function can be linearized with respect to x, and the resulting function can be shown to be equal to

$$L^k(x,y,\mu^k,\lambda^k,\nu^k)\Big|_{x^k}^{lin} = \sum_{i=1}^{n_1} x_i g_i^k(y) + L_0^k(y) \tag{10}$$

where $g_i^k(y) = Q_i(y - y^k)$ (with Q_i being the i^th row of Q), and

$$L_0^k(y) = F_2(y) + P^k(y^k) + (d_0 + \sum_{i=1}^{m_1} \mu_i^k d_i + \sum_{i=m_1+1}^{m_2} \lambda_i^k d_i)^T y$$

The term $L_0^k(y)$ is convex or linear, and can be left untouched. The positivity or negativity of the gradients $g_i^k(y)$ define a set of equations called the *qualifying* constraints of $L^k(x,y,\mu^k,\lambda^k,\nu^k)$, and they partition the current region into a set of subregions. In each of these subregions, a lower bounding function is constructed, based upon the following inequality:

$$x_i g_i^k(y) \geq \begin{cases} x_i^L g_i^k(y) & \text{if } g_i^k(y) \geq 0 \\ x_i^U g_i^k(y) & \text{if } g_i^k(y) \leq 0 \end{cases}$$

where x_i^L and x_i^U are the lower and upper bounds respectively on x_i in the current region.

2.4. Reduction Tests

In some cases, it is possible to underestimate the terms $x_i^L g_i^k(y)$ without branching. To do this, a set of reduction tests are conducted as follows.

(i) If $g_i^k(y) \geq 0$ for all y in the current region, then x_i is set to x_i^L in $L^k(x,y,\lambda^k,\mu^k)$. variables.

(ii) If $g_i^k(y) \leq 0$ for all y in the current region, then x_i is set to x_i^U in $L^k(x,y,\lambda^k,\mu^k)$. variables.

(iii) If $g_i^k(y) = 0$, then x_i will not appear in the Lagrange function.

In each of the three tests, the resulting gradient $g_i^k(y)$ can be removed from consideration as a candidate for branching in the current region. The remaining x variables are called the *connected* variables.

2.5. Relaxed Dual Problem (Lower Bounding Problem)

After the Lagrange function has been generated and the reduction tests have been applied to determine the current set of *connected* variables, a set of relaxed dual problems are solved so as to provide a new lower bound for the global solution and a set of candidate nodes for further investigation. Each of these problems has the following form:

$$
\begin{aligned}
& \min_{y \in Y, \mu_B} \quad \mu_B \\
\mu_B \quad \geq \quad & L^k(x^B, y, \mu^k, \lambda^k, \nu^k)\big|_{x^k}^{lin} \\
& g_i^k(y) \geq 0 \quad if\ x_i^B = x_i^L \\
& g_i^k(y) \leq 0 \quad if\ x_i^B = x_i^U \\
\mu_B \quad \geq \quad & L^p(x^p, y, \mu^p, \lambda^p, \nu^p)\big|_{x^p}^{lin} \qquad p = 1, \ldots, k-1 \\
& g_i^p(y) \geq 0 \quad if\ x_i^p = x_i^L \\
& g_i^p(y) \leq 0 \quad if\ x_i^p = x_i^U \\
& y^L \leq y \leq y^U
\end{aligned}
$$

The second set of constraints refers to cuts derived from previous iterations. Because of the successive partitioning scheme followed by the algorithm, all the Lagrange functions generated from the parent nodes are valid cuts to be included as underestimating functions for the current node. These cuts are obtained by starting from the current node and traversing the tree backwards until the root node is reached. Note that it is possible to append the constraints (2)-(4) to the relaxed dual problem. This ensures that the values of y found by these problems will always be feasible for the primal problem (6)-(9).

The relaxed dual problem is solved for all combinations of the *connected* variables x_i. The solutions of all these problems (the value of the objective function, μ_B, as well as the solution for y) are stored as nodes in the branch and bound tree. After these problems have been solved, all the nodes in the tree are checked, and the node corresponding to the lowest value for μ_B is selected for further branching. This μ_B becomes a new lower bound for the global solution, and the corresponding value for y provides the fixed value for the primal problem for the next iteration.

2.6. Convergence

The primary test for convergence is the distance between the lower and upper bounds, in absolute and relative terms. If either of these is within a specified tolerance, the algorithm converges. However, the algorithm can also be made to stop after a certain number of iterations have been reached.

2.7. The GOP/MILP Algorithm

The solution of the relaxed dual problems at each node is the most time-consuming step in the GOP algorithm. While the reduction tests can help to prune the branch-and-bound tree at each node, it is still necessary to solve a large number of subproblems at each iteration. It is very likely that the solution of most of these subproblems are useless as far as the succeeding iterations are concerned, i.e., most of the nodes will be fathomed as soon as they are spawned. The GOP/MILP algorithm gets around this problem by using a mixed-integer linear formulation that enables the implicit solution of all the relaxed dual problems as a single problem. This is done by introducing binary variables corresponding to the signs of the gradients. Currently, this method can only be used for problems where the function $F_2(y)$ is not present; otherwise, the resulting dual problem results in a mixed-integer nonlinear problem. The connection to an MINLP solver to solve these problems is currently being investigated.

3. Installation

This section describes how to obtain (if you have not already done so) and install the **cGOP** package at your site. In the following, we assume that you are issuing these commands from a UNIX workstation. In each case, the words shown in boldface are what you should type in at the appropriate point, and ”prompt>” is the command prompt that is normally part of your favorite shell.

3.1. Obtaining the software

The latest version of the **cGOP** software can be obtained using *anonymous ftp* from titan.princeton.edu. The software is available in a compressed, tarred file with the name *cGOP.tar.Z*, and contains versions for the following architectures:

Architecture	*Machine*	*Operating system*
hppa	HP 9000 series	HP-UX 9.0
iris4d	Silicon Graphics	IRIX 5.2
solaris	Sun SPARC	Solaris 2.5.1
rs6000	IBM RS6000	AIX 3.2

Note that the operating system version is the *minimum* required to run the code, i.e. if you have a HP workstation running HP-UX 10.0 then the software should run without any problems on this machine.

The following steps describe how to obtain the software.

prompt> **ftp titan.princeton.edu**
Connected to titan.princeton.edu.
220 titan.princeton.edu FTP server ready.
Name (titan.princeton.edu:viswswrn): **anonymous**
331 Guest login ok, send your complete e-mail address as password.
Password: **yourname@yoursite.yourdomain**
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> **cd pub/cGOP**
250 CWD command successful.
ftp> **get cGOP.tar.Z**
200 PORT command successful.
150 Opening BINARY mode data connection for cGOP.tar.Z(207 bytes).
226 Transfer complete.
207 bytes received in 0.01 seconds (30 Kbytes/s)
ftp> **quit**
221 Goodbye.
prompt>

If you do not have access to anonymous ftp, or if you have difficulty retrieving the package, please contact us via e-mail at cgop@titan.princeton.edu. We will be happy to send the package by electronic mail.

NOTE: If you retreive this package, please inform us via e-mail to cgop@titan.princeton.edu. This will let us keep you updated on bug fixes, new releases etc.

3.2. Installing cGOP

You must decide on a directory where the software is to be installed. Normally, this would be something like /usr/local/software or /usr/local/packages.

prompt> **cd /usr/local/software**
prompt> **uncompress cGOP.tar.Z**
prompt> **tar -xf cGOP.tar.Z**

The last step will create a directory called cGOP. This directory has the following files and subdirectories (indicated by a trailing "/"):

Copyright	Copyright notice
Makefile	Top level Makefile for the package
callable/	Examples of calling gop as a subroutine
config/	Configuration files for "make"
cparse/	Executable and example files for parser
doc/	Documentation (postscript files)
examples/	Sample data and solution output files
include/	C header files to include in your programs
lib/	The **cGOP** subroutine libraries
stand/	Source files to create a standalone version

NOTE: Please read the file "Copyright" first. This file details the conditions under which this package can be used.
The different subdirectories are discussed in detail below.

callable/

This directory contains examples of calling the GOP algorithm as a subroutine from a C program. The examples include a bilinear problem, as well as a problem with nonlinear terms in the objective function.

config/

This directory contains a common configuration file, *config.common*, that should included in any Makefile used while running **cGOP**. This file in turns includes an appropriate architecture dependent file (*config.hppa, config.rs6000* etc). This file describes the location of the various libraries needed for the package (MINOS, CPLEX) as well as the compilers to use, compile-time options etc. The common files should never need to be changed; any necessary alterations to the system configuration should be confined to the architecture specific file.

cparse/

This directory contains the executable for CPARSE, a parser that allows problems for **cGOP** to be entered in a very simple format. The manual for CPARSE (included with the distribution as cparsedoc.ps) contains a description of the data file format and some example files.

doc/

This directory contains the documentation for the package. Currently, the documentation is available in PostScript^{TM} format, in the file *cGOPdoc.ps*.

examples/

This directory contains data files, and listing (output) files for various examples. A list of these examples can be found in Appendix C.

include/

This directory contains C header files that need to be included in your source programs. Currently, the following files are present.

cgop.h	This is the main header file for the package. It defines all the data structures and the calling syntax for the various subroutines in the package, and is described in detail in Section 6.
config.h	This file sets problem sizes, algorithm limits etc. It is described in Section 3.4.
defines.h	Assorted parameters for cGOP. This file should not be changed.
options.h	Header file for cGOP options. This file should not be changed.

lib/

This directory contains a subirectory for each of the various architectures currently supported. Each subdirectory contains the architecture-dependent libraries for **cGOP** including the links to the MINOS 5.4 and CPLEX 4.0 packages. Note that the actual libraries for MINOS 5.4 and CPLEX 4.0 are not provided; you need to copy them here, or make links to the actual files where they exist. See the configuration section below for details.

gop.a	This is the main library for **cGOP**. It contains all the core subroutines for the various algorithms used in the package.
gopcplex.a	This library contains the subroutines that provide the link to CPLEX 4.0. It is needed if any of the y variables are binary, or if the GOP/MILP algorithm is to be used.
gopminos.a	This library contains the subroutines that provide the link to MINOS 5.4. It is needed if any of the subproblems is convex.
gopparse.a	This library contains subroutines that provide the link to a user-friendly interface for entering problems when using **cGOP**.

stand/

This directory contains C source files for creating a standalone version of **cGOP**. This version can be used to solve bilinear quadratic problems with linear constraints.

3.3. Setting the machine architecture and compile/link options

Before compiling a standalone executable or using the callable subroutine version of **cGOP**, you need to configure the software for the architecture type. To do this, edit the file config/config.common. This file should be included in all Makefiles that you use to build executables using **cGOP**. Make sure that the machine architecture (ARCH) is set to the correct one for your machine.

3.4. Selecting Available Solvers

This version of **cGOP** can use CPLEX (version 4.0) and/or MINOS (version 5.4) to solve the various subproblems. At least one of these solvers must be present on your system, as a callable library.

1. Edit config/config.common. Go to the section for solver libraries. If you do not have CPLEX, then comment out the CPLEXLIB line with a "#" in the first column. If you do not have MINOS, then comment out the MINOSLIB line with a "#" in the first column.
2. Go to lib/*arch*. The software expects the libraries to be called *cplex.a* and *minos.a* respectively.
    1. If you have CPLEX, then either copy libcplex.a or make a link to it. For example, if the libraries are in /usr/local/lib, then
       prompt> cd Gop/lib
       prompt> ln -s /usr/local/lib/libcplex.a cplex.a
    2. If you have MINOS, then either copy minos.a or make a link to it. For example, if the libraries are in /usr/local/lib, then
       prompt> cd Gop/lib
       prompt> ln -s /usr/local/lib/minos.a minos.a
3. Edit include/config.h. Go to the solver section, and select the appropriate choices for the solvers as explained there.

3.5. Creating A Standalone Version

The easiest way to use **cGOP** is to create a standalone executable. The files needed to create this executable are contained in the subdirectory *stand/*. The UNIX™ *make* command is used with the *Makefile* provided in this directory. This file includes the appropriate configuration files from the *config* directory, setting the correct options for the compilation and linking stages.

The standalone version *cgop* is created by the command

prompt> make cgop

Note: For certain Silicon Graphics machines running the IRIX operating system, you may get the message

```
ld:
Warning: /usr/lib/libisam.a(isread.o): isstart: multiply defined
         previous (used) definition from '../lib/iris4d/cplex.a';
         new (ignored) definition from '/usr/lib/libisam.a'
```

This is normal, and can be ignored.

3.6. Testing the standalone version

There are several problems in the examples directory. The executable created above can be used to solve these problems. For example, to solve the problem in bilin1.gop, do

prompt> cd examples
prompt> ../stand/cgop [options] bilin1[.gop]

where the ".gop" extension can be omitted, and [options] refers to a variety of run-time options available. These options are described in Section 5.

The results for the problems in the examples directory can be found at the end of the README file in the *examples* directory. You should obain similar results with the executable, except for the CPU times, which will vary with the processor speed, and available memory and swap space on your machine.

3.7. Expiration of License

This version of the **cGOP** package is provided without any charge to the end user. We are constantly in the process of improving the software, adding new versions, features, removing bugs etc. To ensure that you'll always be using the most up to date version, this version (Version 1.1) expires after December 31, 1997. You can obtain the most recent version of the package by using anonymous ftp to titan.princeton.edu and following the instructions given above. If this does not work, please send us e-mail at cgop@titan.princeton.edu and we will send you a copy of the package.

Control section
...
Description of constraints
...
Nonzeros in objective and constraints
...
Bounds on variables

Figure 1: Structure of the data file

4. Format Of The *cgop* Data File

cGOP expects to find the data for quadratic terms in an file using an MPS-like format. This file specifies, among other things, the number of variables and constraints and various tolerances for convergence, as well as the data for the linear and quadratic terms. In addition, it provides a pointer to the appropriate subroutine to be used for the nonlinear terms $F_1(x)$ and $F_2(x)$. The basic structure of the file is shown in Figure 1.

Control Section
: The first section in the datafile contains information related to the overall size of the problem, number of variables and constraints, type of objective function and constraints, a starting point for the algorithm and tolerances for convergence.

Description of Constraints
: A general description of each constraint in the problem must be provided in this section. The description consists of one line per constraint, describing the type of constraint, the relational type ($\leq, =$), the RHS, and number of terms in the constraint. The type of constraint is used to specify where it should be used, i.e. whether the constraint goes to the primal problem only, the relaxed dual only, or both.

Nonzeros in Objective and Constraints
: This section contains an entry for each nonzero in the objective function and constraints. Only linear and bilinear entries can be provided this way. Nonlinear entries need to be given via the user subroutines.

Bounds on Variables
: If the variables have lower and upper bounds that are different from the default ones (0 for lower bounds, $+\infty$ for upper bounds) then they must be specified in this section.

Each line in the datafile consists of *two* or more fields separated by whitespace. Fields can be separated by any number of blanks or tab characters. Comments in the data file are allowed in the following forms:

1. Any line starting with a "#" or "!" is assumed to be a comment line, and is ignored.

2. If the character "#" or "!" appears in a line, then the rest of the line is assumed to be an end-of-line comment and is ignored.

3. All blank lines are ignored.

We now discuss each of these sections in more detail below

Control Section

The control section specifies the problem sizes and various tolerances. Each line in this section has the form

keyword *value*

where *value* can be either a character string or an integer or floating point number. The section *must* have the following keywords, (in any order):

BEGIN *string*
: The string (if present) indicates the name of the problem.

OBJTYPE *string*
: This line specifies the type of objective function. *string* can be one of *linear*, *bilinear*, *quadratic* or *nonlinear*.

CONTYPE *string*
: This line specifies the highest degree of nonlinearity of the constraints. *string* can be one of *linear*, *bilinear*, *quadratic* or *nonlinear*.

NXVAR *integer*
: This line specifies the number of x variables in the problem.

NYVAR *integer*
: This line specifies the number of y variables in the problem.

NCON *integer*
: This line specifies the *total* number of constraints in the problem.

EPSA *float*
: This line specifies the absolute tolerance desired for convergence. It is best to give the number in exponential form.

EPSR *float*
: This line specifies the relative tolerance desired for convergence. It is best to give the number in exponential form.

YSTART *integer*
: This line specifies the kind of starting point desired for the algorithm. The values are interpreted as follows:

1. If YSTART = -1, start from $y = y^L$ (lower bounds).
2. If YSTART = 1, start from $y = y^U$ (upper bounds).
3. If YSTART = 0, start from $y = 0$.
4. If YSTART = 2, start from $y = \frac{y^L + y^U}{2}$.
5. Otherwise, the program will prompt you for a starting point.

MAXITER *integer*
: This line gives the maximum number of iterations for the algorithm.

-1 Start from the lower bound for the y variables
1 Start from the upper bound for the y variables
0 Use **0** as the starting point.

Any other value causes the algorithm to prompt for a starting point.

OBJCONST *float*

Any constant terms in the objective function can be specified here.

USERFUNC *string*

This line should indicate the name of the function used to describe the nonlinear terms in the objective function. This line can be omitted (or set to "default") if there are no nonlinear terms.

Note: EPSA, EPSR, YSTART, OBJCONST and USERFUNC do not need to be specified.

Description of Constraints

This section gives the overall details of the constraints in the problem. The section must begin with the line

ROWS *value*

where *value* is the total number of constraints in the problem, and must match the value given in the NCON line in the control section.

Following the ROWS line, this section must contain one line for each constraint in the problem. Each line has the form

Row_Num *Row_Type* *Rel_Type* *RHS* *Num_Terms*

where

- *Row_Num* is the number of the constraint, and must be a non-zero number less than the total number of constraints specified in the NCON line in the control section.
- *Row_Type* is the type of constraint, and is interpreted as
  1 The constraint is to be included in both primal and relaxed dual problems.
  2 The constraint is to be included only in the primal problem.
  3 The constraint is to be included only in the relaxed dual problem.
- *Rel_Type* is the type of relational equality for the constraint, and can be one of
  -1 The constraint is of $\leq$ type.
  0 The constraint is an equality constraint.
- *RHS* is a floating point number indicating the RHS of the constraint.
- *Num_Terms* indicates the number of linear and bilinear terms in the constraint.

Nonzeros in Objective and constraints

This section specifies the nonzero linear and bilinear entries in the problem. It begins with the line

COLUMNS *value*

where *value* is the total number of nonzeros being specified, and is followed by lines of the form

Row_Num *X–Index* *Y–Index* *Value*

where

- *Row_Num* is the number of the constraint (as specified in the ROWS section) to which the nonzero belongs. The objective function is indicated by the value 0.
- *X–Index* and *Y–Index* indicate the index of the x and y variables in the term. If the entry indicates a linear term in x, then *Y–Index* must be 0. If the entry indicates a linear term in y, then *X–Index* must be 0.
- *Value* gives the coefficient of the term.

Note that the lines in this section must be in increasing order of the constraints. Moreover, only the objective function terms (with $Row\_Num = 0$) are allowed to be bilinear.

Bounds on Variables

By default, each variable is assumed to have a lower bound of 0 and upper bound of ∞. If the bounds on any of the variables is otherwise, they are specified in this section. The section begins with the line

BOUNDS *value*

where *value* is the number of bounds being specified here. It is followed by lines of the form

Bound_Type *X–Index* *Y–Index* *Value*

where

- *Bound_Type* indicates the type of bound being specified, and is either 'L' or 'U'.
- *X–Index* and *Y–Index* indicate the index of the x or y variable whose bound is being specified. Note that only one of these two values can be non-zero in each entry.
- *Value* gives the actual value of the bound.

END statement

The word **END** on a separate line by itself is used to signify the end of the data file.

5. Command line options

There are several run-time command line options that are available for controlling the behavior of the program. These options typically determine the levels of debugging and solution output, but can also be used to specify the type of problem being solved, the solvers to use for the various subproblems, and the name of output and debugging files.

Options are specified when the executable (**cgop**) is run by including them on the command line:

cgop [options] problem

The options should be in the standard UNIX™ getopt format (i.e. each option consists of a – followed by a single character and an optional value). Currently, the executables created with the GOP library can be called with the following options:

–b	*level*	Compute tight bounds for the variables (*level* = 0-2).
–c		Show the default configuration.
–d		Produce debugging information.
–g		Use the Generalized Benders' Decomposition Algorithm.
–h		Print out a help message.
–m		Use the GOP/MILP Algorithm (requires CPLEX for the mixed-integer problem).
–n		Solve the full NLP problem at each iteration.
–p	*level*	Set the print level for solution output (*level* = 0-4).
–r	*N*	Print real numbers with N decimals (N = 0-9).
–P	*solver*	Solver for the primal problem (minos or cplex).
–R	*solver*	Solver for the relaxed dual problem (minos or cplex).

These options are explained in more detail below.

Improving variable bounds (–b)

It is possible to improve the bounds on the X and Y variables based upon the current qualifying constraints. This feature can be turned on by using the –b option. There are currently three levels for computing the bounds –

- –b 0
  The algorithm does not try to compute tight bounds for the variables.
- –b 1
  This computes the bounds for the variables once at the beginning of the algorithm.
- –b 2
  This computes the bounds for the variables once at the beginning of each iteration.

The default level is 2.

Setting the print and debug levels (–p, –d)

By default, the program will print out a short summary of each iteration of the algorithm, giving the current upper and lower bounds, and the fixed value being used for the Y variables. However, more (or less) details of the solutions can be obtained by using the –p option. The print level can be set to 0, 1, 2, 3 or 4. Level 0 produces the least output, with just one line per iteration. Print level 1 is the default level. At level

2, more details of the primal and dual problems and the lagrange function are printed out. Levels 3 and 4 produce large amounts of output (including detailed listings of the subproblems being solved), and should be avoided unless you are trying to debug the program. The –d option turns on debugging output in the debug file (problem.dbg). The information produced is not likely to be very useful in normal instances.

See section 7 for sample solution files for various print levels, and a description of their contents.

Selecting alternate algorithms (–g, –m)

In addition to the main GOP algorithm, the program can also be made to use two other algorithms.

The –m option will cause the GOP/MILP algorithm to be used instead of the GOP algorithm. This algorithm produces identical results to the GOP algorithm on a macro level, i.e. at the end of each iteration, the bounds obtained by either algorithm will be identical. However, the GOP/MILP algorithm solves the dual problems implicitly using a mixed-integer formulation. As such, this requires the availability of CPLEX. This can be a very useful option when the number of *connected* variables (i.e. number of branching variables) is large.

The –g option will cause the program to solve the problem using the Generalized Benders' Decomposition method. This algorithm is not rigorous, i.e. it does not have a guarantee of convergence to a global solution. However, it can be extremely useful in finding good solutions very quickly.

Solving the full NLP problem (–n)

When the –n option is used, the algorithm will solve the full nonconvex nonlinear problem in the current region as defined by the current node in the branch and bound tree. This option can be useful for finding local solutions during the progress of the algorithm. Note that because a nonlinear problem is being solved, this option requires MINOS to be available.

Selecting the solver for various problems (–P, –R)

This version of the **cGOP** package can use either MINOS or CPLEX to solve the primal and relaxed dual problems. The –P option (with either "minos" or "cplex" as an argument) selects the solver to be used for the primal problems, while the –R option selects the solver for the relaxed dual problems. Note that these options should be used appropriately depending upon the type of problem being solved. For instance, if the dual problem is a convex nonlinear problem, then obviously CPLEX would not be of any use there. Moreover, these options depend upon the availability of the solvers.

Controlling the output precision (–r)

All output from the program, by default, will be printed to 5 decimal places for double precision numbers. To get better precision, specify the –r option followed by the number of decimal places to which the precision is desired.

Example

The following command

cgop –b 2 –n –P cplex –R minos file1.gop

Solves the problem specified in file1.gop with the variable bounds calculated at each iteration (–b 2), the full NLP solved at each iteration (–n), the primal problems solved by CPLEX and the dual problems solved by MINOS.

6. Using cGOP as a subroutine from a C or Fortran Program

The **cGOP** package can also be called as a subroutine, avoiding the need for a data file. In this case, the data needs to be provided explicitly through arguments to the driver programs. In this section, we describe how this can be done from either C or Fortran programs.

6.1. Calling cGOP from a C Program

The routine *gop* is the main driver for the GOP algorithm. This routine takes the data for the problem as a C data structure, and solves it using the appropriate algorithm specified via the options. The syntax for calling this routine is as follows:

```
#include     "cgop.h"

int          gop (p)
struct gop   *p;
```

The only argument to gop is a pointer to a structure containing the data for the problem. This structure (*struct gop*) is defined in the header file *include/cgop.h*. The elements of this structure are shown in Figure 2, and are described in detail below.

probname A character string of length up to 255 characters. This contains the name for the problem.

objtype This is a character that describes the type of the objective function. It can be set to 'L', 'B', 'Q' or 'N', to denote linear, bilinear, quadratic or nonlinear respectively.

contype This describes the type of the constraints. Currently, the only allowable value for this is 'L' for linear constraints.

primaltype This is a character that describes the type of the primal problems solved. It can be set to 'L', 'Q' or 'N', to denote linear, quadratic or nonlinear respectively.

rdualtype This is a character that describes the type of the relaxed dual problems solved. It can be set to 'L', 'Q', 'N', or 'M', to denote linear, quadratic, nonlinear or mixed-integer linear problems respectively.

nxvar An integer variable denoting the number of X variables in the problem.

nyvar An integer variable denoting the number of Y variables in the problem.

ncon An integer variable denoting the total number of constraints in the problem.

npcon An integer variable denoting the number of constraints that are to be included in the primal problems.

nrcon An integer variable denoting the number of constraints that are to be included in the relaxed dual problems.

nzcnt An integer variable denoting the total number of nonzeros in the objective function and constraints. This only includes the linear and bilinear terms, since all higher order nonlinear (but convex) terms are specified using a user-defined function, which is described in more detail later.

```
struct gop {
    char        probname[255]; /* Name of original problem */
    char        objtype;       /* Type of objective function */
    char        contype;       /* Type of constraints */
    char        primaltype;    /* Type of primal problems */
    char        rdualtype;     /* Type of relaxed dual problems */
    int         nxvar;         /* Number of x variables */
    int         nyvar;         /* Number of y variables */
    int         ncon;          /* Number of constraints */
    int         nzcnt;         /* Total number of non-zeros */
    int         *sense;        /* Sense of row:  <=, ==, >= */
    double      *rhs;          /* Right hand sides of the rows */
    int         *count;        /* Number of entries in each row */
    int         *begin;        /* Start of entries for each row */
    int         npcon;         /* Number of constraints to primal */
    int         nrcon;         /* Number of constraints to relaxed dual */
    int         *prows;        /* Set of rows going to the primal problem */
    int         *rrows;        /* Set of rows going to the relaxed dual */
    int         *cindex;       /* Index for the constraint */
    int         *xindex;       /* Index for X variable, if any */
    int         *yindex;       /* Index for Y variable, if any */
    double      *value;        /* Value of the coefficient */
    double      *xlbd;         /* Lower bounds on X variables */
    double      *xubd;         /* Upper bounds on X variables */
    double      *ylbd;         /* Lower bounds on Y variables */
    double      *yubd;         /* Upper bounds on Y variables */
    char        *xtype;        /* Type of X variables (continuous,binary) */
    char        *ytype;        /* Type of Y variables (continuous,binary) */
    double      objconst;      /* Constants in the objective function */
    double      epsa;          /* Absolute tolerance specified */
    double      epsr;          /* Relative tolerance specified */
    int         maxiter;       /* Maximum number of iterations */
    int         niter;         /* Number of iterations so far */
    double      primalubd;     /* Current upper bound from primals */
    double      rdlbd;         /* Current lower bound from duals */
    double      *x;            /* Starting point and optimal solution for X */
    double      *y;            /* Starting point and optimal solution for Y */
    double      abserror;      /* Current absolute error between bounds */
    double      relerror;      /* Current relative error between bounds */
    int         nrduals;       /* Total number of relaxed duals used up */
    int         max_depth;     /* Maximum depth of branch-and-bound tree */
    double      cpu_total;     /* Total CPU Time taken for the algorithm */
    double      cpu_primal;    /* CPU Time taken for primal problems */
    double      cpu_rdual;     /* CPU Time taken for relaxed dual problems */
    double      cpu_bounds;    /* CPU Time taken for bounds problems */
    double      cpu_nlp;       /* CPU Time taken for NLP problems */
    FILE        *solfile;      /* Solutions file */
    FILE        *dbgfile;      /* Debugging file */
    USEROBJ     userobj;       /* Nonlinear terms in objective function */
};
```

Figure 2: GOP Data structure

sense This is an integer array of length *ncon+1* that describes the sense of the constraints. The first entry in this array refers to the objective function, and should always be equal to 0. The remaining entries are for the constraints, and can take the values of -1, 0 or 1 (for $\leq$, $=$ and $\geq$ type rows respectively).

rhs This is double precision array of length *ncon+1* that contains the right hand sides for the constraints. The first entry refers to the objective function, and must be 0.

prows This integer array of length *npcon* that gives the indices for the constraints that are to be included in the primal problem.

rrows This integer array of length *nrcon* that gives the indices for the constraints that are to be included in the relaxed dual problem.

count This integer array of length *ncon+1* contains the number of non-zero entries in the objective function and in each constraint.

begin This integer array of length *ncon+1* contains the index of the beginning entry for the objective function and each constraint in the arrays *cindex, xindex, yindex* and *value*.

cindex
xindex
yindex
value These four arrays (each of length *nzcnt*) describe the non-zeros corresponding to the linear and bilinear terms in the objective function and constraints. The non-zero elements must be grouped by rows, with the the objective function terms appearing first, followed by the non-zero elements for the first constraint, the non-zeros for the second constraint and so on. Within each constraint, there is no restriction on the specification of the non-zeros.
For the i^{th} non-zero element,
cindex[i] gives the index of the constraint in which the non-zero occurs (with 0 indicating the objective function),
xindex[i] gives the index of the X variable in the linear or bilinear term (with -1 indicating no X variable in that term)
yindex[i] gives the index of the Y variable in the linear or bilinear term (with -1 indicating no Y variable in that term)
value[i] gives the coefficient for that non-zero.

xlbd Double precision array of length *nxvar* containing lower bounds for the X variables.

xubd Double precision array of length *nxvar* containing upper bounds for the X variables.

ylbd Double precision array of length *nyvar* containing lower bounds for the Y variables.

yubd Double precision array of length *nyvar* containing upper bounds for the Y variables.

xtype Character array of length *nxvar* that indicates the type of each X variable. Allowable values are 'C' or 'B' for continuous or binary variables respectively.

ytype Character array of length *nyvar* that indicates the type of each Y variable. Allowable values are 'C' or 'B' for continuous or binary variables respectively.

objconst Double precision variable that contains any constants in the objective function (it can be set equal to 0 otherwise).

epsa Double precision variable that indicates the desired absolute tolerance convergence. The algorithm will terminate when $(UBD - LBD) < \textit{epsa}$.

epsa Double precision variable that indicates the desired absolute tolerance convergence. The algorithm will terminate when $\frac{UBD-LBD}{LBD} < \textit{epsr}$.

maxiter Integer variable that indicates the maximum number of iterations for the algorithm.

niter Integer variable that indicates the actual number of iterations required by the algorithm.

primalubd Double precision variable that contains the best upper bound on the global optimum identified by the algorithm.

rdlbd Double precision variable that contains the best lower bound on the global optimum identified by the algorithm.

x Double precision array of length $nxvar$. On input, this array should contain a starting point for the X variables. On output, the array will contain the best solution found for the X variables.

y Double precision array of length $nyvar$. On input, this array should contain a starting point for the Y variables. On output, the array will contain the best solution found for the Y variables.

abserror Double precision variable that indicates the absolute error between the best upper and lower bounds found by the algorithm.

relerror Double precision variable that indicates the relative error between the best upper and lower bounds found by the algorithm.

nrduals Integer variable that shows the number of relaxed dual subproblems solved by the algorithm.

max_depth Integer variable that shows the maximum depth of the branch and bound tree explored by the algorithm.

cpu_total Double precision variable giving the total cpu time required by the algorithm.

cpu_primal Double precision variable giving the cpu time required to solve the primal problems.

cpu_rdual Double precision variable giving the cpu time required to solve the relaxed dual problems.

cpu_bounds Double precision variable giving the cpu time required to solve any bounds problems (problems that tighten bounds on X and Y variables).

cpu_nlp Double precision variable giving the total cpu time required to solve the full nonlinear problem at each iteration.

solfile FILE pointer (obtained by an *fopen()* call) that gives the file to which the solution output should be written. If it is set to NULL (i.e. zero) there will be no solution output.

dbgfile FILE pointer (obtained by an *fopen()* call) that gives the file to which any debug output should be written. If it is set to NULL (i.e. zero), then there will be no debug output.

userobj Pointer to a function that provides any nonlinear terms in the objective function. This function is described in greater detail in Section 6.2.

6.2. Nonlinear Terms in the Objective Function

The nonlinear terms in the objective function are specified through a single function for both the primal and relaxed dual problems. The function is required to provide both the values of these terms, and their gradients with respect to the x and y variables. The format of this function is given below:

void userobj (int mode, int nxvar, int nyvar, double *x, double *y, double *fx, double *fy, double *gx, double *gy)

Parameters:

mode (Input) Selection parameter. This variable is used to indicate which gradients are being requested, as follows:
If mode = 1, then only provide **fx**.
If mode = 2, then only provide **fy**.
If mode = 3, then provide **fx** and **fy**.
If mode = 5, then provide **fx** and **gx**.
If mode = 10, then provide **fy** and **gy**.
If mode = 15, then provide **fx**, **fy**, **gx** and **gy**.

nxvar (Input) The number of x variables.

nyvar (Input) The number of y variables.

x (Input) An array of dimension **nxvar** containing the current values of the x variables.

y (Input) An array of dimension **nyvar** containing the current values of the y variables.

fx (Output) Contribution of terms in x to the objective function.

fy (Output) Contribution of terms in y to the objective function.

gx (Output) An array of dimension **nxvar** that contains the computed components of the gradients of the objective function with respect to the x variables.

gy (Output) An array of dimension **nyvar** that contains the computed components of the gradients of the objective function with respect to the y variables.

7. Example 1: Solving a Bilinear Problem with cGOP

Consider the following problem (Al-Khayyal and Falk, 1983):

$$\begin{aligned} \min_{x,y} \quad & -x + xy - y \\ s.t. \quad & -6x + 8y \leq 3 \\ & 3x - y \leq 3 \\ & 0 \leq x, y \leq 5 \end{aligned}$$

This problem has the optimal solution of -1.0833 at $x = 1.1667, y = 0.5$. The problem can be solved using either the parser, the data file or through a direct call to the *gop()* subroutine. In this section, we discuss in detail all these methods for solving this problem.

7.1. Using the Data File

This data can be found in the file *bilin1.gop* in the *examples* directory, and is shown in Figure 3. The following points should be noted:

- Because all the nonlinearities in the problem are of a bilinear form, all the data for this problem is provided through the data file. Therefore, the USERFUNC is set to "default".
- No special function is being provided to compute bounds on the variables. Therefore, the BOUNDS-FUNC is set to "default".
- By default, all variables are assumed to be bounded between 0 and ∞. Therefore, there is no need to provide the lower bounds for the variables in the BOUNDS section.

The problem can then be solved by invoking the **cgop** executable with the bilin1 file as the argument, and any additonal desired options (as described in Section 5. At this point, the solver will prompt for a starting point for y and the maximum number of iterations desired. Thereafter, it will solve the problem and put the solution results in the file bilin1.sol.

7.1.1. Description of the Solution Output

Depending on the chosen print level, the algorithm produces varying amounts of printed output in the solution file.

Print level 0

The command "**cgop** -p 0 bilin1" sets the solution print level set to 0, the lowest. The resulting solution file is shown in Figure 4. The top section of this file contains a summary of the options used during this solution. These include the algorithm being used, the print and bounds problem levels, the solvers used for the subproblems, as well as the branching strategy used by the algorithm. Then, there is a summary of each iteration of the algorithm, giving the best bounds found so far, the absolute error between the bounds, the number of nodes explored in that iteration, and the cumulative cpu time in seconds. Finally, the algorithm prints a summary of the best solution found. This summary includes the following:

- Best solution found for the X and Y variable(s).

- Best upper and lower bounds on the global solution, and the absolute and relative errors between the bounds. Note that the upper bounds are obtained by solving the primal problem; therefore, the best upper bound reported here corresponds to the solution of the original problem at the reported best values for X and Y.
- Total number of iterations required for convergence. If the algorithm terminated before convergence, then this should be equal to the maximum number of iterations specified.
- Maximum depth of the branch and bound tree and the total number of nodes explored in the tree.
- CPU time for the bounds and relaxed dual problems as well as the total time required by the algorithm.

Print Level 1

Specifying a print level greater than 0 increases the amount of output produced at each iteration. For print level 1, the solution output for this example is shown in Figure 5. The only additional information provided at this print level is the current value for the y variable(s), and the number of relaxed dual problems solved at each iteration.

Print Level 2

For print level 2, the solution output for this example is shown in Figure 6. In this case, the output is significantly increased. The algorithm will now print a summary of the primal and dual problems as well as the lagrange function formulated from the primal. For the primal problem, the output includes the current fixed value for the Y variable(s), the solution for the X variable(s), the objective value, and the values for the constraint and variable bound marginals (multipliers). The Lagrange function formulated from the primal problem contains linear and bilinear terms, and the coefficients for these terms are displayed in the next section. For the relaxed dual problem, the solution output includes an overview of the size of the problem as well as the solution for each subproblem in terms of both the objective value (lower bound value) as well as the solution for the Y variable(s).

Print Levels 3 and 4

For those with a masochistic bent, copious amounts of additional information can be obtained using print levels 3 and 4. At these levels, the details of each subproblem being solved are printed out, as well as the solution lists, statistics from the solvers etc. This is not recommended for the faint of heart.

7.2. Solving the Example Using the Callable Library

The above example can also be solved without the use of a data file. This can be done as explained in Section 6, by writing a C routine that calls the *gop* subroutine with the data in the appropriate structure. The source code for such a subroutine is shown in Figure 7.

This source file must be compiled and linked to the *gop.a* library, and one or more of *gopcplex.a* and *gopminos.a*, depending on which solvers you have available. Once compiled, the executable can be used much the same as **cgop** is, i.e. the same options are available as before.

```
#
# Example 1 from the cGOP manual.
#
# This is a two-variable bilinear problem from Al-Khayyal and Falk (1982).
#
# Optimal Solution:
#
#      x = 1.16667, y = 0.5, f = -1.08333
#
# If the "#" or "!" appears on a line, then the rest of the line is ignored.
# Fields can be separated by any non-zero number of spaces or tabs.
#
BEGIN       Bilin-1                                 # Problem name
OBJTYPE     Bilinear                                # Type of Objective Function
CONTYPE     Linear                                  # Type of constraints
NXVAR       1                                       # Number of X variables
NYVAR       1                                       # Number of Y variables
NCON        2                                       # Total number of constraints
EPSA        1.0E-03                                 # Absolute convergence tolerance
EPSR        1.0E-03                                 # Relative convergence tolerance
YSTART      21                                      # Type of starting point desired
OBJCONST    0                                       # Constants in the objective
USERFUNC    default                                 # Name of the user function
BOUNDSFUNC  default                                 # Name of the bounds function
ROWS    2                                           # No. of actual constraints
        0       0       0       0.0       3         # Objective function row
        1       1      -1       3.0       2         # Linear constraints, they go to
        2       1      -1       3.0       2         # both primal and dual problems.
COLUMNS 7                                           # Total no. of column entries
        0       1       0      -1.0                 # Linear X term in objective
        0       1       1       1.0                 # Bilinear term in objective
        0       0       1      -1.0                 # Linear Y term in objective
        1       1       0      -6.0                 # Linear X term in constraint 1
        1       0       1       8.0                 # Linear Y term in constraint 1
        2       1       0       3.0                 # Linear X term in constraint 2
        2       0       1      -1.0                 # Linear Y term in constraint 2
BOUNDS  2                                           # Number of bounds specified
        U       1       0       1.5                 # Upper bound on X
        U       0       1       1.5                 # Upper bound on Y
END
```

Figure 3: Data file for the bilinear example in Section 7

```
CURRENT OPTIONS
---------------
 Algorithm                        GOP   Primal Solver             MINOS 5.4
 Bounds Level               Iteration   Relaxed Dual Solver       CPLEX 2.1
 Full NLP Solved                   No   Print Level                       0
 Branching Method             Min X/Y   Debug Level                       0


  Iteration  Upper Bound  Lower Bound    Abs Error    No. Nodes    CPU (sec)
  ---------  -----------  -----------    ---------    ---------    ---------
       1       -1.00000     -1.50000      0.50000         2          0.0100
       2       -1.00000     -1.50000      0.50000         1          0.0200
       3       -1.00000     -1.26667      0.26667         1          0.0300
       4       -1.05333     -1.22222      0.16889         2          0.0300
       5       -1.07407     -1.19048      0.11640         2          0.0500
       6       -1.08163     -1.16667      0.08503         2          0.0700
       7       -1.08333     -1.14722      0.06389         2          0.0900
       8       -1.08333     -1.13073      0.04739         2          0.1100
       9       -1.08333     -1.11671      0.03338         2          0.1300
      10       -1.08333     -1.10476      0.02143         2          0.1500
      11       -1.08333     -1.09452      0.01119         2          0.1700
      12       -1.08333     -1.09347      0.01014         2          0.2000
      13       -1.08333     -1.09302      0.00969         2          0.2200
      14       -1.08333     -1.09153      0.00820         2          0.2400
      15       -1.08333     -1.08815      0.00482         2          0.2600
      16       -1.08333     -1.08602      0.00269         2          0.2900
      17       -1.08333     -1.08570      0.00237         2          0.3100
      18       -1.08333     -1.08412      0.00078         2          0.3300


                  ****************************************
                  BEST SOLUTION FOUND BY THE GOP ALGORITHM
                  ****************************************


 Best Solution for X   (       1.16667 )
 Best Solution for Y   (       0.50000 )
 Upper Bound                  -1.08333   Absolute Error              0.00078
 Lower Bound                  -1.08412   Relative Error              0.00072
 Number of iterations               18   CPU for bounds problems     0.18000
 Maximum Depth of Tree              11   CPU for dual problems       0.13000
 Number of nodes explored           34   Total CPU time (sec)        0.47000
```

Figure 4: Solution output with print level 0 for the bilinear example in Section 7

```
CURRENT OPTIONS
---------------
 Algorithm                      GOP   Primal Solver             MINOS 5.4
 Bounds Level             Iteration   Relaxed Dual Solver       CPLEX 2.1
 Full NLP Solved                 No   Print Level                       1
 Branching Method           Min X/Y   Debug Level                       0

                         I T E R A T I O N     1
ITERATION SUMMARY

 Fixed Values of Y     (      1.00000 )
 Upper Bound                -1.00000   Absolute Error              0.50000
 Lower Bound                -1.50000   Relative Error              0.33333
 Relaxed Dual Problems             2   Cumulative CPU (sec)        0.13000

                         I T E R A T I O N     2
ITERATION SUMMARY

 Fixed Values of Y     (      0.00000 )
 Upper Bound                -1.00000   Absolute Error              0.50000
 Lower Bound                -1.50000   Relative Error              0.33333
 Relaxed Dual Problems             1   Cumulative CPU (sec)        0.15000

                         I T E R A T I O N     3
ITERATION SUMMARY

 Fixed Values of Y     (      1.50000 )
 Upper Bound                -1.00000   Absolute Error              0.26667
 Lower Bound                -1.26667   Relative Error              0.21053
 Relaxed Dual Problems             1   Cumulative CPU (sec)        0.16000

(SOLUTION OUTPUT FROM 15 MORE ITERATIONS HAS BEEN REMOVED FOR REASONS OF SPACE.)

                  ****************************************
                  BEST SOLUTION FOUND BY THE GOP ALGORITHM
                  ****************************************

 Best Solution for X   (      1.16667 )
 Best Solution for Y   (      0.50000 )
 Upper Bound                -1.08333   Absolute Error              0.00078
 Lower Bound                -1.08412   Relative Error              0.00072
 Number of iterations             18   CPU for bounds problems     0.13000
 Maximum Depth of Tree            11   CPU for dual problems       0.16000
 Number of nodes explored         34   Total CPU time (sec)        0.49000
```

Figure 5: Solution output with print level 1 for the bilinear example in Section 7

```
                         I T E R A T I O N      1
PRIMAL PROBLEM

 Number of variables               1   Number of rows                   2
 Name of solver            MINOS 5.4   Solution Status           Feasible
 Objective value            -1.00000   Best upper bound          -1.00000
 Fixed Values of Y     (     1.00000 )
 Solution for X        (     0.83333 )
 Constraint Marginals  (     0.00000      0.00000 )
 Bound Marginals       (     0.00000 )

LAGRANGE FUNCTION

 Connected variables               1   Y Variables                      1
 Constant Term               0.00000
 Lower bounds on X     (     0.00000 )
 Upper bounds on X     (     1.50000 )
 Lower bounds on G     (    -1.00000 )
 Upper bounds on G     (     0.50000 )
 Gradient constants    (    -1.00000 )
 Gradient terms in Y
                       (     1.00000 )
 Linear Terms in Y     (    -1.00000 )

RELAXED DUAL PROBLEM

 Name of Solver            CPLEX 2.1   Number of Problems               2
 Number of Variables               3   Number of Constraints            4

 Node    1:  Objective:     -1.50000
             Y:        (     1.50000 )
 Node    2:  Objective:     -1.50000
             Y:        (     0.00000 )

ITERATION SUMMARY

 Fixed Values of Y     (     1.00000 )
 Upper Bound                -1.00000   Absolute Error             0.50000
 Lower Bound                -1.50000   Relative Error             0.33333
 Relaxed Dual Problems             2   Cumulative CPU (sec)       0.11000
```

Figure 6: Solution output with print level 2 for the bilinear example in Section 7

```
#include "cgop.h"

int main (argc, argv)
int        argc;
char     **argv;
{
    /* Data */
    GOP         p;                   /* Data for current problem */

    /* Assorted variables */
    int         status;              /* Results of various commands */


    /* Register default user functions */
    defuser_register();

    /* Get space for the data */
    p             = gop_data_setup(1, 1, 2, 7);

    /* Name and type of problem */
    strcpy(p->probname, "Bilin-1");
    p->objtype    = 'B'; p->contype       = 'L';
    p->primaltype = 'L'; p->rdualtype     = 'L';

    /* Number and type of variables, constraints, nonzeros */
    p->nxvar      =  1;  p->nyvar         =  1;
    p->ncon       =  2;  p->nzcnt         =  7;
    p->xtype[0]   = 'C'; p->ytype[0]      = 'C';

    /* Constraint data */
    p->sense[0]   =  0; p->sense[1]   = -1; p->sense[2]   = -1;
    p->rhs[0]     =  0; p->rhs[1]     =  3; p->rhs[2]     =  3;
    p->npcon      =  2; p->nrcon      =  2;
    p->prows[0]   =  1; p->prows[1]   =  2;
    p->rrows[0]   =  1; p->rrows[1]   =  2;

    /* Nonzero data */
    p->count[0]   =  3; p->count[1]   =  2; p->count[2]   =  2;
    p->begin[0]   =  0; p->begin[1]   =  3; p->begin[2]   =  5;
```

Figure 7: C subroutine for solving Example 1 using the callable *gop* subroutine

```
    /* Terms in the objective function */
    p->cindex[0]  = -1; p->xindex[0]  =  0; p->yindex[0]  = -1; p->value[0]   = -1;
    p->cindex[1]  = -1; p->xindex[1]  =  0; p->yindex[1]  =  0; p->value[1]   =  1;
    p->cindex[2]  = -1; p->xindex[2]  = -1; p->yindex[2]  =  0; p->value[2]   = -1;

    /* Terms in the first constraint */
    p->cindex[3]  =  0; p->xindex[3]  =  0; p->yindex[3]  = -1; p->value[3]   = -6;
    p->cindex[4]  =  0; p->xindex[4]  = -1; p->yindex[4]  =  0; p->value[4]   =  8;

    /* Terms in the second constraint */
    p->cindex[5]  =  1; p->xindex[5]  =  0; p->yindex[5]  = -1; p->value[5]   =  3;
    p->cindex[6]  =  1; p->xindex[6]  = -1; p->yindex[6]  =  0; p->value[6]   = -1;

    /* Bounds for variables */
    p->xlbd[0]    =   0; p->xubd[0]    = 1.5; p->ylbd[0]    =   0; p->yubd[0]    = 1.5;

    /* Starting point */
    p->x[0]       = 1.0; p->y[0]       = 1.0;

    /* Functions for nonlinear objective terms and bounds */
    p->userobj    = getuserobj("default");
    p->bimprove   = getboundsfunc("default");

    /* Tolerances, problem bounds, iteration limits etc */
    p->objconst   = 0.0;      p->epsa       = 0.001;
    p->epsr       = 0.001;    p->maxiter    =  100;
    p->primalubd  =  1.0E+09; p->rdlbd      = -1.0E+09;

    /* File pointers */
    p->solfile    = fopen("bilin1.sol", "w"); p->dbgfile    = NULL;

    /* Extract any options */
    option(argc, argv);


    /* Call the main algorithm */
    status        =    gop(p);

    /* Garbage collection */
    gop_data_free(p);

}
```

Figure 7 (continued).

8. Example 2: Solving an Indefinite Quadratic Problem with cGOP

This example is a large indefinite quadratic programming problem, and is taken from Floudas and Pardalos (1990).

$$\min \quad f(x,y) = f_c - \sum_{i=1}^{10} (\frac{1}{2} Q_i x_i y_i + c_i y_i) + f_1(y)$$

$$\begin{aligned} s.t. \quad Ay &\leq b \\ x_i - y_i &= 0 \quad i = 1, 2, \ldots, 10 \\ x, y &\geq 0 \end{aligned}$$

where

$$\begin{aligned} f_c &= -363762.0, \\ Q &= \text{Diag}\,(63, 15, 44, 91, 45, 50, 89, 58, 86, 82) \\ c &= (1197, 405, 1012, 4823, 1890, 1300, 2937, 1334, -3526, -1558) \\ b &= (380, 415, 385, 405, 470, 415, 400, 460, 400, 200) \\ f_1(y) &= \sum_{i=11}^{20} d_i (y_i - \overline{y_i})^2 \\ d &= (21, 49, 24, 45.5, 5.5, 31.5, 30.5, 30.5, 19, 13) \\ \overline{y} &= (-52, -3, 81, 30, -85, 68, 27, -81, 97, -73) \end{aligned}$$

and

$$A = \begin{bmatrix} 3 & 5 & 5 & 6 & 4 & 4 & 5 & 6 & 4 & 4 & 8 & 2 & 4 & 1 & 1 & 1 & 2 & 1 & 7 & 3 \\ 5 & 4 & 5 & 4 & 1 & 4 & 4 & 2 & 5 & 2 & 3 & 6 & 1 & 7 & 7 & 5 & 8 & 7 & 2 & 1 \\ 1 & 5 & 2 & 4 & 7 & 3 & 1 & 5 & 7 & 6 & 1 & 7 & 2 & 4 & 7 & 5 & 3 & 4 & 1 & 2 \\ 3 & 2 & 6 & 3 & 2 & 1 & 6 & 1 & 7 & 3 & 7 & 7 & 8 & 2 & 3 & 4 & 5 & 8 & 1 & 2 \\ 6 & 6 & 6 & 4 & 5 & 2 & 2 & 4 & 3 & 2 & 7 & 5 & 3 & 6 & 7 & 5 & 8 & 4 & 6 & 3 \\ 5 & 5 & 2 & 1 & 3 & 5 & 5 & 7 & 4 & 3 & 4 & 1 & 7 & 3 & 8 & 3 & 1 & 6 & 2 & 8 \\ 3 & 6 & 6 & 3 & 1 & 6 & 1 & 6 & 7 & 1 & 4 & 3 & 1 & 4 & 3 & 6 & 4 & 6 & 5 & 4 \\ 1 & 2 & 1 & 7 & 8 & 7 & 6 & 5 & 8 & 7 & 2 & 3 & 5 & 5 & 4 & 5 & 4 & 2 & 2 & 8 \\ 8 & 5 & 2 & 5 & 3 & 8 & 1 & 3 & 3 & 5 & 4 & 5 & 5 & 6 & 1 & 7 & 1 & 2 & 2 & 4 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The global solution of this problem occurs at

$$\begin{aligned} \overline{x} &= (0, 0, 0, 62.609, 0, 0, 0, 0, 0, 0) \\ \overline{y} &= (0, 0, 0, 0, 0, 4.348, 0, 0, 0, 0) \end{aligned}$$

This problem is provided in the *callable* subdirectory as *indef1*. The main routine for the problem is in *indef1.c*, which also contains the subroutine for computing $f_1(y)$. The data for Q, c, b and A can be found in *indef1.gop*. The file *indef1.c* can be compiled with the Makefile provided in the same directory. The resulting executable, when run, will produce the results in the file *indef1.sol*. Figure 8 provides the listing in this file for print level 1.

```
                         I T E R A T I O N     1
ITERATION SUMMARY

 Fixed Values of Y     (       0.00000      0.00000      0.00000     62.60870
                               0.00000      0.00000      0.00000      0.00000
                               0.00000      0.00000      0.00000      0.00000
                               0.00000      0.00000      0.00000      4.34783
                               0.00000      0.00000      0.00000      0.00000
                         )
 Upper Bound             49318.01796   Absolute Error            3787.30531
 Lower Bound             45530.71265   Relative Error               0.08318
 Relaxed Dual Problems             2   Cumulative CPU (sec)         0.18000

                         I T E R A T I O N     2
ITERATION SUMMARY

 Fixed Values of Y     (       5.93607      0.00000      0.00000     58.99543
                               0.00000      0.00000      0.00000      0.00000
                               0.00000      0.00000      0.00000      0.00000
                               0.00000      0.00000      0.00000      8.21918
                               0.00000      0.00000      0.00000      0.00000
                         )
 Upper Bound             49318.01796   Absolute Error               0.00006
 Lower Bound             49318.01790   Relative Error               0.00000
 Relaxed Dual Problems             4   Cumulative CPU (sec)         0.36000

                  ****************************************
                  BEST SOLUTION FOUND BY THE GOP ALGORITHM
                  ****************************************

 Best Solution for X   (       0.00000      0.00000      0.00000     62.60870
                               0.00000      0.00000      0.00000      0.00000
                               0.00000      0.00000 )
 Best Solution for Y   (       0.00000      0.00000      0.00000     62.60870
                               0.00000      0.00000      0.00000      0.00000
                               0.00000      0.00000      0.00000      0.00000
                               0.00000      0.00000      0.00000      4.34783
                               0.00000      0.00000      0.00000      0.00000
                         )
 Upper Bound             49318.01796   Absolute Error               0.00006
 Lower Bound             49318.01790   Relative Error               0.00000
 Number of iterations              2   CPU for bounds problems      0.24000
 Maximum Depth of Tree             2   CPU for dual problems        0.07000
 Number of nodes explored          6   Total CPU time (sec)         0.36000
```

Figure 8: Solution output with print level 1 for Example 3

```
/* indef1.c    -- Example of solving an indefinite quadratic problem using cGOP. */

#include "cgop.h"

/*
 * Main program for Example 2.  We first register the function for computing nonlinear
 * terms in the objective function (in this case, convex quadratic terms) with the
 * algorithm, reads in the remaining data, and then calls gop().
 */
int main (int argc, char **argv)
{
    /* Variables */
    GOP        p;                   /* Data for current problem */
    int        status;              /* Results of various commands */
    void       indef1_obj();        /* User function, described below */

    /* Extract any options */
    gop_option(argc, argv);

    /* Read in the data */
    p    = gop_data_read("indef1.gop");
    if (p == NULL) gop_error("Read of data file failed.");

    /* Make sure the right function is used for nonlinear objective terms */
    p->userobj = indef1_obj;

    /* Call the main algorithm */
    status    =    gop(p);

    /* Print out the results */
    gop_print_results(stdout, p);
    printf("\nThe file indef1.sol contains the full solution output.\n\n");

    /* Garbage collection */
    if (p->solfile) fclose(p->solfile);
    gop_data_free(p);

    /* We are done */
    return(0);
}
```

Figure 9: Main program for Example 2

```
/*
 * Function to compute terms in the objective for Example 2.
 */

void indef1_obj (int mode, int nxvar, int nyvar, double *x,
    double *y, double *fx, double *fy, double *gx, double *gy)
{

    int     i, j;
    double  tempvar, tempvar2;

    /* Some constant data */
    static  double cbar[] = {-52, -3, 81, 30, -85, 68, 27, -81, 97, -73 };
    static  double cmat[] = { 21, 49, 24, 45.5, 5.5, 31.5, 30.5, 30.5, 19, 13};


    /*
     * Contribution from Y terms to the objective function and its gradient.
     * Note the use of the "&" (AND) operator to check for the various modes.
     */
    if (mode & 2) {                      /* True if mode=2, 3, 10 or 15 */
        tempvar = 0;
        for (i = 0; i < 10; i++) {
            j             = 10 + i;
            tempvar2     = y[j]-cbar[i];

            /* Contribution from Y terms to objective gradient */
            if (mode & 8) {              /* True if mode is 10 or 15 */
                gy[j]    = 2*cmat[i]*tempvar2;
            }
            tempvar     += cmat[i]*tempvar2*tempvar2;
        }
        *fy     = tempvar;
    }
}
```

Figure 10: Objective function subroutine for Example 2

9. Error Messages

The program usually catches errors in the data file pretty well, and prints out a detailed message explaining what the error was. In most instances, these errors arise from wrong input (or order of input) and can be easily rectified by checking the proper format for the data file. For example, a typical error message might be

```
****************************************************************************
Error: File bilin1.gop, line 31.
       ROWS section must be defined before COLUMNS.
       Please see the cGOP manual for more details.
****************************************************************************
```

This indicates that the data file has a COLUMNS line before the definition of the rows. On the other hand, you might sometimes receive a message such as

```
******************************************************************************
Error: Signal handler caught signal 4 (Illegal instruction)...Exiting.
       Please see the cGOP manual for more details.
******************************************************************************
```

This indicates a more serious error, usually caused during the actual execution of the algorithm. If you encounter such an error, please send us an e-mail (at cgop@titan.princeton.edu) with a copy of the data file, the command line options that you used during the run, and a brief description of your machine configuration (architecture, memory size etc). This will help us to pinpoint the exact source of the error.

References

F. A. Al-Khayyal and J. E. Falk. Jointly constrained biconvex programming. *Math. of Oper. Res.*, 8(2):273, 1983.

CPLEX. *Using the CPLEX callable library*. CPLEX Optimization Inc., Incline Village, NV, 1995.

C. A. Floudas and V. Visweswaran. A primal-relaxed dual global optimization approach. *Journal of Optimization Theory and Applications*, 78(2):187, 1993.

B. A. Murtagh and M. A. Saunders. *MINOS5.0 Users Guide*. Systems Optimization Laboratory, Dept. of Operations Research, Stanford University, CA., 1988. Appendix A: MINOS5.2, Technical Report SOL 83-20.

V. Visweswaran and C. A. Floudas. New properties and computational improvement of the GOP algorithm for problems with quadratic objective function and onstraints. *J. Global Optim.*, 3(3):439, 1993.

V. Visweswaran and C. A. Floudas. New Formulations and Branching Strategies for the GOP Algorithm. *Submitted for Publication*, 1995.

Appendix A. Description of the GOP Algorithm and its Variants

In this section, we describe in detail the steps in the GOP and GOP/MILP algorithms. The terminology used is as follows. Given a node j in the branch and bound tree, P_j is its parent node, and I_j is the iteration at which node j is created. R_j is the set of constraints defining the region corresponding to node j. At any point, N_j denotes the total number of nodes in the tree, and C denotes the current node. F denotes the set of iterations with a feasible primal problem, while I denotes the set of iterations when the primal problem was infeasible.

The GOP Algorithm

The major steps of the GOP algorithm are as follows.

STEP 0: Initialization

(a) Read in the data for the problem including tolerance for convergence, ϵ.

(b) Define initial upper and lower bounds (f^U, f^L) on the global optimum.

(c) Generate initial bounds for the x variables, x^L and x^U.

(d) Choose a starting point y^1 for the algorithm.

(e) Set $K = 1$, $C = P_C = 1$, $N_j = 1$.

STEP 1: Selection of Current Region

(a) If $K = 1$, set $R_C = \emptyset$ and goto Step 2.

(b) If $K \geq 2$, set $R_C = \emptyset$, $m = C$. Then:

 (i) Add the Lagrange function and *qualifying* constraints for node m to R_C.

 (ii) Set $m = P_m$. If $m = 1$, then goto Step 2.

 (iii) Repeat steps (i) and (ii).

STEP 2: Primal problem

(a) Solve the primal problem (6)-(9) to give $P^K(y^K)$.

 (i) If feasible, set $F = F \cup K$ and update $f^U = MIN[f^U, P^K(y^K)]$.

 (ii) If infeasible, solve a *relaxed* primal problem. Set $I = I \cup K$.

(b) Store y^K, λ^K and μ^K.

STEP 3: Determination of Current Partitions

(a) Generate the current Lagrange function $L^K(x, y, \lambda^K, \mu^K)$.

(b) Determine the set of *connected* variables I_c^K and the corresponding partial derivatives $g_i^K(y)$ $(i = 1, \ldots, I_c^K)$ of the current Lagrange function.

(c) For each *connected* variable, determine (if possible) tight lower and upper bounds x_i^L and x_i^U in the current region $y \in R_C$. Otherwise, use the original bounds.

(d) Evaluate lower and upper bounds on $g_i^K(y)$ in the region $y \in R_C$.

  (i) If $g_i^K(y) \geq 0 \ \ \forall y \in R_C$, set $x_i^B = x_i^L$ in the current Lagrange function, and remove i from the set I_c^K.

  (ii) If $g_i^K(y) \leq 0 \ \ \forall y \in R_C$, set $x_i^B = x_i^U$ in the current Lagrange function, and remove i from the set I_c^K.

STEP 4: Relaxed Dual Problem

(a) Select a combination of the bounds B_l of the *connected* variables, say $B_l = B_1$.

(b) Find the solution $(\mu_B^\star, y^\star)$ to the following *relaxed dual* subproblem:

$$\min_{y \in Y, \mu_B} \ \mu_B$$

$$\begin{array}{rcl} \mu_B & \geq & L^K(x^{B_l}, y, \lambda^K, \mu^K)\big|_{x^K}^{lin} \\ & & \nabla_{x_i} L^K(x, y, \lambda^K, \mu^K)\big|_{x^K} \geq 0 \quad if\ x_i^{B_l} = x_i^L \\ & & \nabla_{x_i} L^K(x, y, \lambda^K, \mu^K)\big|_{x^K} \leq 0 \quad if\ x_i^{B_l} = x_i^U \\ (y, \mu_B) & \in & R_C \end{array}$$

  (i) If $\mu_B^\star < f^U - \epsilon$, set $j = N_j + 1, P(j) = C, N_j = N_j + 1$, and store the solution in μ_B^j, y^j.

  (ii) If $\mu_B^\star \geq f^U - \epsilon$, fathom the solution.

(c) Select a new combination of bounds, say $B_l = B_2$, for the *connected* variables.

(d) Repeat steps (b) and (c) until all the combinations of bounds for the *connected* variables have been considered.

STEP 5: Selecting a new lower bound
Select the infimum of all μ_B^j, say μ_B^p. Set $C = p, y^{K+1} = y^p$, $f^L = \mu_B^p$.

STEP 6: Check for convergence
If $\left[\frac{f^U - f^L}{f^U}\right] < \epsilon$, STOP; otherwise, set $K = K + 1$ and return to step 1.

9.1. The GOP/MILP Algorithm

As before, given a node j in the branch and bound tree, P_j is its parent node, and I_j is the iteration at which node j is created. R_j is the set of constraints defining the region corresponding to node j. At any point, N_j denotes the total number of nodes in the tree, and C denotes the current node. F denotes the set of iterations with a feasible primal problem, while I denotes the set of iterations when the primal problem was infeasible. A_j denotes the set of integer cuts to be used when solving the MILP problem for the node j.

STEP 0: Initialization

This step is the same as for the GOP algorithm, with the addition of setting $A_1 = \emptyset$.

STEP 1 – Step 3:
Same as in the GOP algorithm.

STEP 4: Current Relaxed Dual Problem
Solve the following MILP problem:

$$\begin{array}{rcl} & & \min\limits_{\substack{y\in Y,\mu_B \\ t,\alpha}} \mu_B \\ \mu_B & \geq & \sum\limits_{i=1}^{NI_c^K} t_i^K + \sum\limits_{i=1}^{NI_c^K} x_i^L g_i^K(y) + L_0^K(y,\lambda^K,\mu^K) \\ t_i^K & \geq & \alpha_i^K(x_i^U - x_i^L)\underline{g_i^K} \\ t_i^K & \geq & (x_i^U - x_i^L)(g_i^K(y) - (1-\alpha_i^K)\overline{g_i^K}) \\ \alpha_i^K \underline{g_i^K} & \leq & g_i^K(y) \leq (1-\alpha_i^K)\overline{g_i^K} \\ (y,\mu_B) & \in & R_C \\ \alpha & \in & A_C \end{array}$$

(i) If $\mu_B^\star < f^U - \epsilon$, set $j = N_j + 1, P(j) = C, N_j = N_j + 1$, and store the solution in μ_B^j, y^j.

(ii) If $\mu_B^\star \geq f^U - \epsilon$, fathom the solution.

Let the solution for the binary variables in this problem be $\alpha = \alpha^C$. Let J_C be the set of variables which are 1 in this solution, and let NJ_C the number of such binary variables.

STEP 5: Selecting a new lower bound
Same as in the GOP algorithm.

STEP 6: Regenerating Solutions From Partitioned Nodes
Suppose that the solution selected in Step 5 corresponds to node C, and that this node was originally partitioned at iteration k. Then, add the cut

$$\sum_{i\in J_C} \alpha_i - \sum_{i\notin J_C} \alpha_i \leq NJ_C - 1$$

to the set of binary cuts A_C. Solve the following MILP:

$$\begin{array}{rcl} & & \min\limits_{\substack{y\in Y,\mu_B \\ t,\alpha}} \mu_B \\ \mu_B & \geq & \sum\limits_{i=1}^{NI_c^L} t_i^k + \sum\limits_{i=1}^{NI_c^L} x_i^L g_i^k(y) + L_0^k(y,\lambda^k,\mu^k) \\ t_i^L & \geq & \alpha_i^L(x_i^U - x_i^L)\underline{g_i^K} \\ t_i^L & \geq & (x_i^U - x_i^L)(g_i^k(y) - (1-\alpha_i^k)\overline{g_i^K}) \\ \alpha_i^k \underline{g_i^K} & \leq & g_i^k(y) \leq (1-\alpha_i^k)\overline{g_i^K} \\ (y,\mu_B) & \in & R_C \\ \alpha & \in & A_C \end{array}$$

Suppose the solution of this problem is $\mu_B^\star$.

(i) If $\mu_B^\star < f^U - \epsilon$, then set $j = N_j + 1, P(j) = C, N_j = N_j + 1$, and store the solution in μ_B^j, y^j. Also set α^C to be the solution of the binary variables in this formulation.

(ii) If $\mu_B^\star \geq f^U - \epsilon$, fathom the node C.

STEP 7: Check for convergence
Same as in GOP algorithm.