The MINTO-OSL interface

M.W.P. Savenbergh
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven
The Netherlands

Abstract

MINTO is a software system that solves mixed-integer linear programs by a branch-and-bound algorithm with linear programming relaxations. It also provides automatic constraint classification, preprocessing, primal heuristics and constraint generation. Moreover, the user can enrich the basic algorithm by providing a variety of specialized application routines that can customize MINTO to achieve maximum efficiency for a problem class.

MINTO can be implemented on top of any LP-solver that provides capabilities to solve and modify linear programs and interpret their solutions. The current version can either be build on top of the CPLEX (TM) callable library, version 1.2 and up, or on top of the Optimization Subroutine Library (OSL), version 1.2.

This note describes the design of the MINTO-OSL interface.
The MINTO-OSL interface

M.W.P. Savelsbergh
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven
The Netherlands

Abstract

MINTO is a software system that solves mixed-integer linear programs by a branch-and-bound algorithm with linear programming relaxations. It also provides automatic constraint classification, preprocessing, primal heuristics and constraint generation. Moreover, the user can enrich the basic algorithm by providing a variety of specialized application routines that can customize MINTO to achieve maximum efficiency for a problem class.

MINTO can be implemented on top of any LP-solver that provides capabilities to solve and modify linear programs and interpret their solutions. The current version can either be build on top of the CPLEX (TM) callable library, version 1.2 and up, or on top of the Optimization Subroutine Library (OSL), version 1.2.

This note describes the design of the MINTO-OSL interface

1 Introduction

One of the goals set for MINTO was to develop a mixed integer optimizer that would be available on a number of hardware platforms and that could be build on top of a variety of LP-solvers.

To provide a portable code that would run on a number of hardware platforms MINTO has been written in the C programming language. To be able to build MINTO on top of a variety of LP-solvers it has been designed in such a way that it only requires and uses a very small subset of the functions usually provided by an LP-solver.

MINTO requires the following generic functions to be available in an LP-solver: read an MPS-file, load a formulation, solve an LP, add a column, delete a column, change a bound, add a row, and delete a row.

As can be seen from the above, MINTO doesn’t require any information retrieval functions besides those associated with interpreting the LP solution. The reason being that MINTO maintains its own copy of the formulation at all times. This is also the price that has to be paid, in terms of sacrificing some efficiency, for being as independent as possible from the underlying LP-solver; there are always two formulations that have
to be maintained: the formulation maintained by the LP-solver and the formulation
maintained by MINTO.

All the above mentioned generic functions are implemented as cover functions. In
the MINTO code it is always the cover function that is called. The real work is done
within the cover function. In the cover function the appropriate calls to the functions
provided by the specific LP-solver are made. Consequently, developing the MINTO-OSL
interface amounts to writing covers that use OSL functions calls.

2 C versus Fortran

MINTO is written in C, whereas OSL is written in Fortran. One important difference
between C and Fortran is that indexing in arrays is done differently. In C the elements
in an array of length \( n \) are indexed from 0 up to \( n - 1 \), whereas in Fortran they are
indexed from 1 up to \( n \). In order to be able to conform to C style indexing into dspace,
the following pointers, besides the pointer to dspace itself, have been set up:

```c
double *dspace1;
int *mspace, mspace1;
char *cspace, cspace1;

mspace = (int *) dspace;
cspace = (char *) dspace;

dspace1 = dspace - 1;
mspace1 = mspace - 1;
cspace1 = (char *) dspace1;
```

3 Row senses and right hand sides versus row bounds

MINTO has adopted the conventional view of rows, in which a row is specified by a sense
and a right-hand-side, contrary to the view that has been adopted in OSL, in which a
row is specified by a lower and an upper bound on its activity.

Consequently, whenever information concerning a row is exchanged between MINTO
and OSL a conversion has to take place. More important, OSL offers an easy way to
express ‘range’ rows, whereas MINTO requires two separate rows to do so. Therefore,
MINTO scans the original formulation for range rows and, if it encounters one, converts
it to two separate rows.
4 Reading an MPS file

MINTO uses the LP-solver to read the MPS file describing the problem that has to be solved (ekkmps).

However, since MINTO applies various preprocessing and probing techniques to this original formulation, the initial formulation that will be loaded into the LP-solver to start the solution process may differ considerably. In fact, the original formulation is only needed to initialize MINTO’s internal administration, because MINTO will do all preprocessing and probing using its own copy of the formulation.

OSL’s storage management functions provide the capability to create temporary storage within dspace. This is accomplished by first saving the current status of dspace with ekkpshs and later restoring that status of dspace with ekkpops. Every piece of information that has been stored in dspace between calls to these two functions can be considered temporary.

MINTO uses this mechanism to temporarily store the original matrix; it first saves the current status of dspace (ekkpshs), it then reads the MPS file (ekkmps) and sets up and initializes its internal data structures, and it then restores the status of dspace (ekkpops).

5 Solving LPs

MINTO solves the initial LP using the primal simplex method (ekksslv) with a crash basis (ekkcrsh). It uses the default settings suggested in the OSL guide and reference. MINTO solves all subsequent LPs using the dual simplex method (ekkevnu).

This is not the standard dual simplex method, but the function that is used by OSL’s mixed integer optimizer to evaluate nodes. MINTO uses it to be able to cut off the dual simplex when the value drops below the best primal value.

Interpreting of the solution is done directly by copying the necessary information from dspace.

6 Modifying the active formulation

OSL divides the active formulation over the low end and high end of dspace: the coefficient matrix at the low end of dspace and all other information at the high end of dspace. Modifications at the high end of dspace, such as for instance modifying bounds, are always trivial to accomplish and will not be described.

Adding rows and columns

Adding rows and columns
Adding rows and columns from MINTO’s internal administration to the coefficient matrix currently loaded in dspace is done in two steps. First, a new block is created that contains the rows or columns in storage-by-triplets form (ekkdscb). Second, the matrix is compressed (ekknwmt).

Deleting rows and columns
Deleting rows and columns is a little more difficult because MINTO assumes that when some rows or columns are deleted the remaining rows and columns will be immediately renumbered, so that at any time during the solution process, the columns are numbered from 1 up to the number of columns and the rows are numbered from 1 up to the number of rows.

OSL does not offer the functionality to handle this form of deleting rows and columns through any of the available functions, so MINTO has to do it himself. (Note that when you delete a row or column from the matrix in OSL by changing all the coefficients to zero and compressing the matrix, the row or column still exists but is empty, i.e., the number of rows or columns is not reduced.)

Since deleting rows or columns may require a complete reorganization of the matrix, MINTO just deletes the current matrix and loads the modified matrix.

For the deletion of the current matrix, MINTO relies on the storage management routines provided by OSL. Whenever a new matrix has to be loaded at the low end of dspace the current matrix is deleted by a pop operation (ekkpops) and the new matrix loaded by a push operation (ekkpshs). The modified matrix can be constructed from MINTO’s internal administration very easily and is passed on to OSL as a new block in storage-by-columns form (ekkdscb). Finally, the new matrix is compressed (ekknwmt).

7 Using in OSL in application functions

As mentioned in the abstract, MINTO allows a user to enrich the basic algorithm by providing a variety of specialized application routines that can customize MINTO to achieve maximum efficiency for a problem class. To be able to use OSL within one of these applications routines, MINTO has to make sure that when it hands over control to the application all the information pertaining to MINTO’s dspace is stored safely (ekkptmi) and that it is reset when the application hands back control (ekkgltmi).

8 Modified OSL routines

To build an executable of MINTO with OSL as LP-solver some modified OSL routines have to be linked. These modifications either fix small bugs and should therefore dis-
appear in future updates of the OSL library or provide additional functionality required by MINTO:

- ekkpopsf.o: to be able to change IMAXINT and to get rid of warning messages from ekknwmt;
- ekkpshsf.o: to get rid of warning messages from ekknwmt;
- ekkevnu.o: to be able to cut off the dual simplex based;
- ekkbcd2.o: to be able to write mps-files containing the binary/integer information;
- ekkbcdof.o: to be able to write mps-files containing the binary/integer information;
- ekkimd2.o: fixes a bug.