NAG Toolbox

nag_mip_ilp_dense (h02bb)

1 Purpose

nag_mip_ilp_dense (h02bb) solves 'zero-one', 'general', 'mixed' or 'all' integer programming problems using a branch and bound method. The function may also be used to find either the first integer solution or the optimum integer solution. It is not intended for large sparse problems.

2 Syntax

```
[itmax, toliv, tolfes, bigbnd, x, objmip, iwork, rwork, ifail] =
nag_mip_ilp_dense(itmax, msglvl, a, bl, bu, intvar, cvec, maxnod, intfst,
toliv, tolfes, bigbnd, x, 'n', n, 'm', m, 'maxdpt', maxdpt)
[itmax, toliv, tolfes, bigbnd, x, objmip, iwork, rwork, ifail] = h02bb(itmax,
msglvl, a, bl, bu, intvar, cvec, maxnod, intfst, toliv, tolfes, bigbnd, x, 'n',
n, 'm', m, 'maxdpt', maxdpt)
```

3 Description

nag_mip_ilp_dense (h02bb) is capable of solving certain types of integer programming (IP) problems using a branch and bound (B&B) method, see Taha (1987). In order to describe these types of integer programs and to briefly state the B&B method, we define the following linear programming (LP) problem:

Minimize

$$F(x) = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

subject to

$$\sum_{j=1}^{n} a_{ij}x_j \left\{ \begin{array}{c} = \\ \leq \\ \geq \end{array} \right\} b_i, \quad i = 1,2,\ldots,m$$

$$l_j \leq x_j \leq u_j, \quad j = 1,2,\ldots,n \tag{1}$$

If, in (1), it is required that (some or) all the variables take integer values, then the integer program is of type (*mixed* or) *all* general IP problem. If additionally, the integer variables are restricted to take only 0–1 values (i.e., $l_j = 0$ and $u_j = 1$) then the integer program is of type (mixed or all) *zero-one* IP problem.

The B&B method applies directly to these integer programs. The general idea of B&B (for a full description see Dakin (1965) or Mitra (1973)) is to solve the problem without the integral restrictions as an LP problem (first *node*). If in the optimal solution an integer variable x_k takes a noninteger value x_k^*, two LP sub-problems are created by *branching*, imposing $x_k \leq \left[x_k^*\right]$ and $x_k \geq \left[x_k^*\right] + 1$ respectively, where $\left[x_k^*\right]$ denotes the integer part of x_k^*. This method of branching continues until the first integer solution (*bound*) is obtained. The hanging nodes are then solved and investigated in order to prove the optimality of the solution. At each node, an LP problem is solved using nag_opt_lp_solve (e04mf).

4 References

Dakin R J (1965) A tree search algorithm for mixed integer programming problems *Comput. J.* **8** 250–255

Mitra G (1973) Investigation of some branch and bound strategies for the solution of mixed integer linear programs *Math. Programming* **4** 155–170

Taha H A (1987) *Operations Research: An Introduction* Macmillan, New York

5 Parameters

5.1 Compulsory Input Parameters

1: **itmax** – INTEGER

An upper bound on the number of iterations for each LP problem.

2: **msglvl** – INTEGER

The amount of printout produced by nag_mip_ilp_dense (h02bb).

Value	Definition
0	No output.
1	The final IP solution only.
5	One line of output for each node investigated and the final IP solution.
10	The original LP solution (first node), one line of output for each node investigated and the final IP solution.

3: **a**$(lda,:)$ – REAL (KIND=nag_wp) array

The first dimension of the array **a** must be at least $\max(1,\mathbf{m})$.

The second dimension of the array **a** must be at least **n** if $\mathbf{m} > 0$ and at least 1 if $\mathbf{m} = 0$.

The ith row of **a** must contain the coefficients of the ith general constraint, for $i = 1,2,\ldots,m$.

If $\mathbf{m} = 0$ then the array **a** is not referenced.

4: **bl**$(\mathbf{n}+\mathbf{m})$ – REAL (KIND=nag_wp) array

5: **bu**$(\mathbf{n}+\mathbf{m})$ – REAL (KIND=nag_wp) array

bl must contain the lower bounds and **bu** the upper bounds, for all the constraints in the following order. The first n elements of each array must contain the bounds on the variables, and the next m elements the bounds for the general linear constraints (if any). To specify a nonexistent lower bound (i.e., $l_j = -\infty$), set $\mathbf{bl}(j) \leq -\mathbf{bigbnd}$ and to specify a nonexistent upper bound (i.e., $u_j = +\infty$), set $\mathbf{bu}(j) \geq \mathbf{bigbnd}$. To specify the jth constraint as an equality, set $\mathbf{bl}(j) = \mathbf{bu}(j) = \beta$, say, where $|\beta| < \mathbf{bigbnd}$.

Constraints:

$\mathbf{bl}(j) \leq \mathbf{bu}(j)$, for $j = 1,2,\ldots,\mathbf{n}+\mathbf{m}$;
if $\mathbf{bl}(j) = \mathbf{bu}(j) = \beta$, $|\beta| < \mathbf{bigbnd}$.

6: **intvar**$(\mathbf{n})$ – INTEGER array

Indicates which are the integer variables in the problem. For example, if x_j is an integer variable then $\mathbf{intvar}(j)$ must be set to 1, and 0 otherwise.

Constraints:

$\mathbf{intvar}(j) = 0$ or 1, for $j = 1,2,\ldots,\mathbf{n}$;
$\mathbf{intvar}(j) = 1$ for at least one value of j.

7: **cvec**(**n**) – REAL (KIND=nag_wp) array

The coefficients c_j of the objective function $F(x) = c_1x_1 + c_2x_2 + \ldots + c_nx_n$. The function attempts to find a minimum of $F(x)$. If a maximum of $F(x)$ is desired, **cvec**(j) should be set to $-c_j$, for $j = 1, 2, \ldots, n$, so that the function will find a minimum of $-F(x)$.

8: **maxnod** – INTEGER

The maximum number of nodes that are to be searched in order to find a solution (optimum integer solution). If **maxnod** ≤ 0 and **intfst** ≤ 0, then the B&B tree search is continued until all the nodes have been investigated.

9: **intfst** – INTEGER

Specifies whether to terminate the B&B tree search after the first integer solution (if any) is obtained. If **intfst** > 0 then the B&B tree search is terminated at node k say, which contains the first integer solution. For **maxnod** > 0 this applies only if $k \leq$ **maxnod**.

10: **toliv** – REAL (KIND=nag_wp)

The integer feasibility tolerance; i.e., an integer variable is considered to take an integer value if its violation does not exceed **toliv**. For example, if the integer variable x_j is near unity then x_j is considered to be integer only if $(1 -$ **toliv**$) \leq x_j \leq (1 +$ **toliv**$)$.

11: **tolfes** – REAL (KIND=nag_wp)

The maximum acceptable absolute violation in each constraint at a 'feasible' point (feasibility tolerance); i.e., a constraint is considered satisfied if its violation does not exceed **tolfes**.

12: **bigbnd** – REAL (KIND=nag_wp)

The 'infinite' bound size in the definition of the problem constraints. More precisely, any upper bound greater than or equal to **bigbnd** will be regarded as $+\infty$ and any lower bound less than or equal to $-$**bigbnd** will be regarded as $-\infty$.

13: **x**(**n**) – REAL (KIND=nag_wp) array

An initial estimate of the original LP solution.

5.2 Optional Input Parameters

1: **n** – INTEGER

Default: the dimension of the arrays **cvec**, **x**, **intvar**. (An error is raised if these dimensions are not equal.)

n, the number of variables.

Constraint: **n** > 0.

2: **m** – INTEGER

Default: the first dimension of the array **a**.

m, the number of general linear constraints.

Constraint: **m** ≥ 0.

3: **maxdpt** – INTEGER

Suggested value: **maxdpt** $= 3 \times$ **n**$/2$.

Default: $3 \times$ **n**$/2$

The maximum depth of the B&B tree used for branch and bound.

Constraint: **maxdpt** ≥ 2.

5.3 Output Parameters

1: **itmax** – INTEGER

Unchanged if on entry **itmax** > 0, else **itmax** $= \max(50, 5 \times (\mathbf{n} + \mathbf{m}))$.

2: **toliv** – REAL (KIND=nag_wp)

Unchanged if on entry **toliv** > 0.0, else **toliv** $= 10^{-5}$.

3: **tolfes** – REAL (KIND=nag_wp)

Unchanged if on entry **tolfes** > 0.0, else **tolfes** $= \sqrt{\epsilon}$ (where ϵ is the ***machine precision***).

4: **bigbnd** – REAL (KIND=nag_wp)

Unchanged if on entry **bigbnd** > 0.0, else **bigbnd** $= 10^{20}$.

5: **x**(**n**) – REAL (KIND=nag_wp) array

With **ifail** $= 0$, **x** contains a solution which will be an estimate of either the optimum integer solution or the first integer solution, depending on the value of **intfst**. If **ifail** $= 9$, then **x** contains a solution which will be an estimate of the best integer solution that was obtained by searching **maxnod** nodes.

6: **objmip** – REAL (KIND=nag_wp)

With **ifail** $= 0$ or 9, **objmip** contains the value of the objective function for the IP solution.

7: **iwork**($liwork$) – INTEGER array

8: **rwork**($lrwork$) – REAL (KIND=nag_wp) array

9: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Note: nag_mip_ilp_dense (h02bb) may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the function:

ifail $= 1$

No feasible integer point was found, i.e., it was not possible to satisfy all the integer variables to within the integer feasibility tolerance (determined by **toliv**). Increase **toliv** and rerun nag_mip_ilp_dense (h02bb).

ifail $= 2$

The original LP solution appears to be unbounded. This value of **ifail** implies that a step as large as **bigbnd** would have to be taken in order to continue the algorithm (see Section 9).

ifail $= 3$

No feasible point was found for the original LP problem, i.e., it was not possible to satisfy all the constraints to within the feasibility tolerance (determined by **tolfes**). If the data for the constraints

are accurate only to the absolute precision σ, you should ensure that the value of the feasibility tolerance is greater than σ. For example, if all elements of A are of order unity and are accurate only to three decimal places, the feasibility tolerance should be at least 10^{-3} (see Section 9).

ifail $= 4$

The maximum number of iterations (determined by **itmax**) was reached before normal termination occurred for the original LP problem (see Section 9).

ifail $= 5$

Not used by this function.

ifail $= 6$

An input argument is invalid.

ifail $= 7$ (*warning*)

The IP solution reported is not the optimum IP solution. In other words, the B&B tree search for at least one of the branches had to be terminated since an LP sub-problem in the branch did not have a solution (see Section 9).

ifail $= 8$

The maximum depth of the B&B tree used for branch and bound (determined by **maxdpt**) is too small. Increase **maxdpt** and rerun nag_mip_ilp_dense (h02bb).

ifail $= 9$ (*warning*)

The IP solution reported is the best IP solution for the number of nodes (determined by **maxnod**) investigated in the B&B tree.

ifail $= 10$

No feasible integer point was found for the number of nodes (determined by **maxnod**) investigated in the B&B tree.

ifail $= 11$

The maximum depth of the B&B tree used for branch and bound (determined by **maxdpt**) is too small. Increase **maxdpt** and rerun nag_mip_ilp_dense (h02bb).

Overflow

It may be possible to avoid the difficulty by increasing the magnitude of the feasibility tolerance (**tolfes**) and rerunning the program. If the message recurs even after this change, see Section 9.

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

Your licence key may have expired or may not have been installed correctly.

ifail $= -999$

Dynamic memory allocation failed.

7 Accuracy

nag_mip_ilp_dense (h02bb) implements a numerically stable active set strategy and returns solutions that are as accurate as the condition of the problem warrants on the machine.

8 Further Comments

The original LP problem may not have an optimum solution, i.e., nag_mip_ilp_dense (h02bb) terminates with **ifail** $= 2$, 3 or 4 or overflow may occur. In this case, you are recommended to relax the integer restrictions of the problem and try to find the optimum LP solution by using nag_opt_lp_solve (e04mf) instead.

In the B&B method, it is possible for an LP sub-problem to terminate without finding a solution. This may occur due to the number of iterations exceeding the maximum allowed. Therefore the B&B tree search for that particular branch cannot be continued. Thus the returned solution may not be optimal. (**ifail** $= 7$). For the second and unlikely case, a solution could not be found despite a second attempt at an LP solution.

9 Example

This example solves the integer programming problem:

maximize

$$F(x) = 3x_1 + 4x_2$$

subject to the bounds

$$\begin{aligned} x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

and to the general constraints

$$\begin{aligned} 2x_1 + 5x_2 &\leq 15 \\ 2x_1 - 2x_2 &\leq 5 \\ 3x_1 + 2x_2 &\geq 5 \end{aligned}$$

where x_1 and x_2 are integer variables.

The initial point, which is feasible, is

$$x_0 = (1,1)^{\mathrm{T}},$$

and $F(x_0) = 7$.

The optimal solution is

$$x^* = (2,2)^{\mathrm{T}},$$

and $F(x^*) = 14$.

Note that maximizing $F(x)$ is equivalent to minimizing $-F(x)$.

9.1 Program Text

```
function h02bb_example

fprintf('h02bb example results\n\n');

% Maximize (3,4).x; negate and minimize
cvec = [-3; -4];
% subject to constraints bl <= Ax <= bu
a = [ 2,  5;
      2, -2;
      3,  2];
big = 1e+20;
% first 2 elements are bounds on x, the remainder are bounds on Ax
bl = [  0;    0; -big; -big;    5];
bu = [big;  big;   15;    5;  big];

% both x variables are integers
intvar = nag_int([1;1]);
```

```
itmax = nag_int(0);
msglvl = nag_int(1);
maxnod = nag_int(0);
intfst = nag_int(0);
toliv  = 0;
tolfes = 0;
bigbnd = big;
% Initial guess for x
x = [1;  1];
[itmax, toliv, tolfes, bigbnd, x, objmip, iwork, rwork, ifail] = ...
    h02bb(...
          itmax, msglvl, a, bl, bu, intvar, cvec, maxnod, intfst, toliv, ...
          tolfes, bigbnd, x, 'maxdpt', nag_int(4));
```

9.2 Program Results

```
     h02bb example results

 *** IP solver

 Parameters
 ----------

 Linear constraints......         3        First integer solution..       OFF
 Variables...............         2        Max depth of the tree...         4

 Feasibility tolerance...  1.05E-08        Print level.............         1
 Infinite bound size.....  1.00E+20        EPS (machine precision).  1.11E-16

 Integer feasibility tol.  1.00E-05        Iteration limit.........        50
 Max number of nodes.....      NONE

 ** Workspace provided with MAXDPT =     4: LRWORK =    84  LIWORK =   137
 ** Workspace required with MAXDPT =     4: LRWORK =    84  LIWORK =   137

 Total of     9 nodes investigated.

 Exit IP solver - Optimum IP solution found.

 Final IP objective value =   -14.00000

 Varbl State     Value     Lower Bound   Upper Bound    Lagr Mult   Residual

 V  1    UL    2.00000       0.00000       2.00000      -3.000       0.000
 V  2    EQ    2.00000       2.00000       2.00000      -4.000       0.000

 L Con State     Value     Lower Bound   Upper Bound    Lagr Mult   Residual

 L  1    FR    14.0000         None        15.0000       0.000       1.000
 L  2    FR    0.00000         None        5.00000       0.000       5.000
 L  3    FR    10.0000       5.00000         None        0.000       5.000
```