User's and programmer's manual of the network flows heuristic package for cell suppression in 1H2D tables

Jordi Castro
Dept. of Statistics and Operations Research
Universitat Politècnica de Catalunya
Pau Gargallo 5, 08028 Barcelona (Catalonia, Spain)
jcastro@eio.upc.es

Technical Report DR 2004-06
March 2004

Report available from http://www-eio.upc.es/~jcastro

User's and programmer's manual of the network flows heuristic package for cell suppression in 1H2D tables *

Jordi Castro
Dept. of Statistics and Operations Research
Universitat Politècnica de Catalunya
Pau Gargallo 5, 08028 Barcelona (Catalonia, Spain)
jcastro@eio.upc.es

Abstract

The network flows heuristic package for cell suppression was developed in the scope of the 5th European Union program IST-2000-25069 CASC project. It implements an efficient heuristic based on shortest-paths for secondary cell suppression in two-dimensional tables with one hierarchical dimension (1H2D). This document shows the main features of the package for 1H2D tables. It also describes the package interface and how to connect it with the user's application.

Key words: C/C++ programming languages, complementary cell suppression problem, linear programming, shortest-paths, minimum-cost network flows.

*Work supported by the IST-2000-25069 CASC project. This document is part of CASC deliverable 4.1-D6.

Contents

1 Introduction

The network flows package for cell suppression (NF_CSP) implements a fast heuristic for the protection of statistical data in two dimensional tables with one hierarchical dimension (1H2D tables). This new heuristic sensibly combines and improves ideas of previous approaches for the secondary cell suppression problem in two-dimensional general [2] and positive [7, 9] tables. Details about the heuristic can be found in [4, 5].

The heuristic is based on the solution of a sequence of shortest-path subproblems that guarantee a feasible pattern of suppressions (i.e., one that satisfies the protection levels of sensitive cells). Hopefully, this feasible pattern will be close to the optimal one.

The current package is linked with three solvers: CPLEX7.5/8.0 [8], PPRN [6], and an efficient implementation of the bidirectional Dijkstra's algorithm for shortest-paths (that will be denoted as "Dijkstra") [1]. Later releases of CPLEX will also work if the interface routines are the same than for version 8.0. The heuristic can use any of the three solvers for the solution of the shortest-path subproblems, although Dijkstra is recommended (and the default one) for efficiency reasons. CPLEX is needed if a lower bound of the optimal solution want to be computed. The auditing phase can be performed with either CPLEX or PPRN.

PPRN and Dijkstra were implemented at the Dept. of Statistics and Operations Research of the Universitat Politècnica de Catalunya, and are included in NF_CSP. PPRN was originally developed during 1992–1995, but it had to be significantly improved within the CASC project to work with NF_CSP. Dijkstra was completely developed in the scope of CASC. The third solver, CPLEX, is a commercial tool, and requires purchasing a license. However, PPRN is a fairly good replacement—although not so robust— for the network flows routines of CPLEX. Therefore, in principle, there is no need for an external commercial solver, unless lower bounds want to be computed.

Even though two of the three solvers are included in the distribution of NF_CSP, this document only describes the features of the heuristic, and from the user's point of view. A detailed description of PPRN and Dijkstra's solvers can be found in [3, 6] and [1], respectively.

The structure of the document is as follows. Section 2 presents the two versions of the package: standalone and callable library, including a simple program that shows how to use NF_CSP from the user's application. Section 3 describes the main options and features of the package. In Section 4 we present the set of routines to interface with NF_CSP, grouped by functional categories. A final Appendix lists all the files and routines of NF_CSP.

2 Two versions of the package

The package is provided as a standalone application and as a set of routines that can be called from the user's application (callable library). Before describing both versions we first show the common input format for 1H2D tables required by them.

2.1 Input format for 1H2D tables

The package assumes that the hierarchical variable is located in rows. If it is in columns, the user has to transpose the table before calling the package. The heuristic can deal with any number of levels in the hierarchical structure tree, and any number of rows for each level.

The input format requires a table for each row that has a hierarchical structure. Consider the

Table 1: Example of 1H2D table, with primaries in boldface

T_0

	C_0	C_1	C_2
R_0	**5**	6	11
R_1	10	15	25
R_2	15	21	36

T_1

	C_0	C_1	C_2
R_{10}	8	10	18
R_{11}	**2**	5	7
R_1	10	15	25

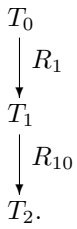
T_2

	C_0	C_1	C_2
R_{100}	6	6	12
R_{101}	2	4	6
R_{10}	8	10	18

simple example of Table 1 (primary cells in boldface):

In that example, row R_1 of table T_0 has a hierarchical structure: $R_1 = R_{10} + R_{11}$. The decomposition of R_1 is detailed in T_1. And row R_{10} of table T_1 is also hierarchical; T_2 shows its structure. Although in the example all the subtables have the same number of rows, this is not required in general. However, the number of columns of all the subtables must always be the same (otherwise, we would not preserve the hierarchical structure in only one dimension).

In that example, the hierarchical structure tree is



That tree has three levels, and one table per level. In general, the package can deal with hierarchical tables of any number of levels, and any number of subtables per level (i.e., any number of hierarchical rows for each table).

The information needed by the package is the following:

- Number of subtables of the tree.
  - 3 in the example.
- Number of columns of all the subtables (excluding marginal column).
  - 2 in the example.
- Number of rows for each table (excluding marginal rows).
  - 2 for T_0, 2 for T_1 and 2 for T_2 in the example.
- Hierarchical tree structure. For each hierarchical row of any table, table where it is decomposed. Since all the subtables, but the top one, show the decomposition of a hierarchical row, the number of entries is the number of subtables minus 1.
  - In the example, row R_1 of table T_0 is decomposed in table T_1; and row R_{10} of table T_1 is decomposed in table T_2.
- Number of primary cells.

  - In the example the 2 primary cells marked in boldface.

- Position, and lower and upper level protections for each primary cell.
  - In the example, for instance, the position of the first primary would be (T_0,R_0,C_0).
- Values and weights for the cells of all the subtables (including marginals).

Additional information —as, for instance, which cells must be considered *permanent* (i.e., they must be published)— can be provided through the set of interface routines to the package.

2.2 Standalone application

The input file for the standalone application is in AMPL format. The user can provide the main data for solving the problem through this format. For the rest of parameters, they can be set by using the appropriate interface routines of the callable library. For instance, the input file for the example of Subsection 2.1 is shown below. Comment lines (started with #) must be preserved, although they must be set empty, or without any comment. Lines with a ";" are also needed, since they mark the end of a list of values. For a table with M rows and N columns (without marginals), row and column indexes go from 0 to M, and from 0 to N; row M and column N are marginals. We assume the weight of each cell is 1 in that example, which means that the number of secondary cells will be minimized.

```
# The first two lines are for information about the problem
# (e.g., parameters used if generated through a random generator)
#T= number of 2D subtables
param T := 3;
#M= number of rows of each table (without marginal row)
param M :=
0               2
1               2
2               2
;
#N= number of columns common to all the subtables (without marginal column)
param N := 2;
#(T-1) lines with hierarchical info: row rh of table th decomposed in table tdh
param: rh         th           tdh :=
0      1          0            1
1      0          1            2
;
#P= number of primary suppression cells of each table
param P := 2;
#for each primary: table, row, column, lower prot., and upper prot.
param:  p_t        p_r           p_c             lpl             upl :=
0       0          0             0               2               2
1       1          1             0               2               2
;
#table values and weights (indexed by (table,row,column)), marginals included
param :  a    weight :=
0 0 0    5    1
```

```
0 0 1    6    1
0 0 2    11   1
0 1 0    10   1
0 1 1    15   1
0 1 2    25   1
0 2 0    15   1
0 2 1    21   1
0 2 2    36   1
1 0 0    8    1
1 0 1    10   1
1 0 2    18   1
1 1 0    2    1
1 1 1    5    1
1 1 2    7    1
1 2 0    10   1
1 2 1    15   1
1 2 2    25   1
2 0 0    5    1
2 0 1    6    1
2 0 2    11   1
2 1 0    3    1
2 1 1    4    1
2 1 2    7    1
2 2 0    8    1
2 2 1    10   1
2 2 2    18   1
;
```

The standalone application is called through:

```
main1H2D filename out_dir [-s s] [-c c] [-u u] [-m m] [-o o] [-a a]
```

The first two parameters are obligatory. The remaining ones are optional, and can be entered in any order. The meaning is:

- filename: name of the input file with the problem instance
- out_dir: directory (which must exist) for two output files with solution information. They are named `sec_1H2D.out` and `inf_1H2D.out`.
- `-s`: solver to be used. Possible values are:
  - p: PPRN
  - c: CPLEX
  - d: DIJKSTRA (default)
- `-c`: compute or not lower bound. Possible values are:
  - y: compute (needs CPLEX)
  - n: don't compute (default)

- `-u`: use the solution obtained by the lower bounding procedure to obtain an initial set of complementary removed cells (parameter -c above must be 'y' to use this option). Possible values are:
  - y: use solution
  - n: don't use (default)
- `-m`: type of merit order for processing primary cells. Possible values are:
  - n: normal order in which primaries were entered (default)
  - a: ascendent order by cell value
  - d: descendent order by cell value
- `-o`: type of costs for objective function. Possible values are:
  - f: fastest and possibly slightly worser solutions (default)
  - s: slower and possibly slightly better solutions
- `-a`: perform auditing. Possible values are:
  - n: don't perform (default)
  - y: perform

If `example.dat` is the name of the above input AMPL file corresponding to the example, we can protect the table calling:

```
main1H2D example.dat .
```

The two output files with the solution are written in the current directory ('.'). File `sec_1H2D.out` gives the number of secondary cells in the first line, and the location (table,row,cell) of each secondary cell in the remaining ones. In that example we get:

```
8
0,0,1
0,1,0
0,1,1
0,2,0
0,2,1
1,1,1
1,2,0
1,2,1
```

File `inf_1H2D.out` provides 12 values (in different lines) with a summary of the solution obtained. The 12 lines show, respectively,

1. the number of subtables;
2. overall number of different rows;
3. number of columns;
4. overall number of different cells;

Table 2: Protected table, with primaries in boldface and secondaries underlined

T_0

	C_0	C_1	C_2
R_0	**5**	6	11
R_1	10	15	25
R_2	15	21	36

T_1

	C_0	C_1	C_2
R_{10}	8	10	18
R_{11}	**2**	5	7
R_1	10	15	25

T_2

	C_0	C_1	C_2
R_{100}	6	6	12
R_{101}	2	4	6
R_{10}	8	10	18

5. lower bound (0 if not computed);
6. value suppressed for primary cells;
7. value suppressed for secondary cells;
8. weight suppressed for primary cells;
9. weight suppressed for secondary cells (the objective minimized);
10. number of primary cells;
11. number of secondary cells;
12. number of shortest-path subproblems solved.

In that example we get:

```
3
7
3
21
0
7
97
2
8
2
8
2
```

The suppression pattern is shown in Table 2, with primaries in boldface and secondaries underlined. If we call again `main1H2D` with `-c c` (i.e., compute a lower bound), we get a lower bound of 8 (total weight of secondary cells), which corresponds with the solution obtained, thus being optimal.

2.3 Callable library

The callable library provides a set of routines that can be embedded in a user's application. They provide full control over the heuristic implemented by the package.

The example program of pages 11–13 illustrates the main steps that need to be performed to protect Table 1 with NF_CSP. That example is a portion of the standalone version of the package.

The filename with the instance (in AMPL format) to be solved is provided as a command line parameter.

Any code that uses NF_CSP has to include the header file `csp_table1H2D.h`, as in line 7 of the example program. This file contains all the declarations (data structures and routines) needed to interface with NF_CSP.

We first need to declare a TABLE1H2D* variable (pointer to TABLE1H2D structure). In the code we named it `htab` (line 15). It will store all the required information for the table, both before and after its protection. After the declaration, we must create the real space for the table. This is done at line 18, calling `read_table_AMPL_format(htab,argv[1])`. That routine (lines 45–132) creates the table reading the input file provided as command line argument. We don't fully detail such routine; instead, we just describe the most interesting routines of the callable library that it uses:

- `create_table1H2D(ptab,ntables,nrows,ncols,p` (line 96): allocates and creates hierarchical `*ptab` table of `ntables` subtables, `nrows[i]` rows for subtable `i`, `ncols` and `p` primaries.
- `put_pcell1H2D(htab,i,pt,pi,pj,plpl,pupl)` (line 105): set information of the `i`-th primary cell (i.e., location (table,row,column)= `(pt,pi,pj)`, and lower and upper protection levels `plpl` and `pupl` respectively).
- `put_parent1H2D(htab,tdh[i],th[i])` and `put_rowinparent1H2D(htab,tdh[i],rh[i])` (lines 110,111): define the hierarchical structure tree of the subtables. In that case, subtable `th[i]` is the parent of subtable `tdh[i]`; the row of `th[i]` that corresponds to the marginal row of `tdh[i]` is `rh[i]`.
- `put_cellvalue1H2D(htab,tcell,icell,jcell,vcell)` and `put_cellweight1H2D(htab,tcell,icell,jcell,wcell)` (lines 123, 124) set the value `vcell` and weight `wcell` of cell (`tcell,icell,jcell`).
- `get_npcells1H2D(htab)`, `get_ntables1H2D(htab)`, `get_nrows1H2D(htab,t)` and `get_ncolumns1H2D(htab,t)` (lines 103,118–120), provide the number of primary cells and subtables of the hierarchical table (first two routines), and number of rows and columns of subtable `t` (last two routines).

The above is the minimum information required to protect the table. We can now proceed with the protection, calling `csp_heur1H2D(htab)` (line 25). Since we did not modify them, the default settings will be used. The default values of main options are:

- Solver Dijkstra.
- Primary cells are processed in NORMAL order, i.e., the order provided by the user through `put_pcell1H2D()`.
- A lower bound of the optimal weight suppressed is not computed. Although this is useful to know how far the solution provided by the heuristic is from the optimal one, that procedure can be fairly time consuming for large tables (in fact, it can take more time than the heuristic).

If the table is successfully protected, `csp_heur1H2D(htab)` returns 0. Otherwise, a nonzero value will be returned. See Section 4 for the list of possible return status.

Once the protection is performed, we can retrieve the solution obtained through several routines. What to be done at this stage is specific of each application. In our example, we just print the location of secondary cells (lines 31–38). The list of secondary cells is obtained by calling for all the cells `get_cellstatus1H2D(htab,t,i,j)`. This routine returns the current status of cell (`t,i,j`) which can take the values PRIMARY, SECONDARY, PERMANENT or NONREMOVED (described in detail in Section 3. We could also have used routine `is_cell_secondary1H2D(htab,t,i,j)`, which is a shortcut for `get_cellstatus1H2D(htab,t,i,j)==SECONDARY`. Analogous routines `is_cell_primary1H2D`, `is_cell_permanent1H2D()` and `is_cell_nonremoved1H2D()` are provided.

Finally, the memory space of the table is freed at line 41, calling `delete_table(&htab)`.

We next display the full example program in C/C++.

Example program using the callable library

```
 1  /*************************************************/
 2  // Simple main program for the 1H2D callable library
 3  /*************************************************/
 4  #include <stdio.h>
 5  #include <stdlib.h>
 6  #include <string.h>
 7  #include "csp_table1H2D.h"
 8
 9  /* prototypes */
10  int read_table_AMPL_format(TABLE1H2D **tab, char *filename);
11
12  int main(int argc, char *argv[])
13  {
14      int ret_stat= 0;
15      TABLE1H2D *htab= NULL;
16
17      /* create and read table from filename argv[1] */
18      if (read_table_AMPL_format(&htab,argv[1])) {
19          fprintf(stderr,"error: reading table\n");
20          ret_stat= -1;
21          goto TERMINATE;
22      }
23
24      /* call heuristic */
25      if ((ret_stat= csp_heur1H2D(htab))) {
26          fprintf(stderr,"error %d reported by csp_heur1H2D\n",ret_stat);
27          ret_stat= -1;
28          goto TERMINATE;
29      };
30
31      // write secondary cells to standard output
32      for(int t=0;t<get_ntables1H2D(htab);t++) {
33          for(int i= 0;i<get_nrows1H2D(htab,t);i++)
34              for (int j=0;j<get_ncolumns1H2D(htab,t);j++) {
35                  if (get_cellstatus1H2D(htab,t,i,j)==SECONDARY)
```

```
36                      printf("%d,%d,%d\n",t,i,j);
37              }
38      }
39
40   TERMINATE:
41      delete_table1H2D(&htab);
42      return(ret_stat);
43  }
44
45  int read_table_AMPL_format(TABLE1H2D **ptab, char *filename)
46  // create and read table in AMPL format from filename
47  // returns 0 if everything goes fine
48  // returns -1 if problems creating table1H2D object
49  // returns -2 if file not found
50  {
51  #define MYALLOC(n,type) (type *)malloc((n)*sizeof(type))
52  #define MYFREE(p) if (p) {free(p); (p)= NULL;}
53      FILE *fp;
54      int ntables,*nrows,ncols,p,*rh,*th,*tdh;
55      const int MJUNK= 1000;
56      char junk[MJUNK];
57      int i,j,t;
58      double vcell,wcell;
59      int tcell, icell, jcell;
60
61      if (!(fp= fopen(filename,"r"))) return(-2);
62
63      // read table dimensions
64
65      fgets(junk,MJUNK,fp);
66      fgets(junk,MJUNK,fp);
67      fgets(junk,MJUNK,fp);
68      fscanf(fp,"%*s %*s %*s %d ",&ntables);fgets(junk,MJUNK,fp);
69      if (! (nrows= MYALLOC(ntables,int)) ) return(-1);
70      fgets(junk,MJUNK,fp);
71      fgets(junk,MJUNK,fp);
72      for(i=0;i<ntables;i++) {
73          fscanf(fp,"%*d %d\n",&nrows[i]);
74          nrows[i]++;
75      }
76      fgets(junk,MJUNK,fp);
77      fgets(junk,MJUNK,fp);
78      fscanf(fp,"%*s %*s %*s %d ",&ncols);fgets(junk,MJUNK,fp);
79      ncols++;
80
81      // read hierarchical structure
82      if (! (rh= MYALLOC(ntables-1,int)) ) return(-1);
83      if (! (th= MYALLOC(ntables-1,int)) ) return(-1);
84      if (! (tdh= MYALLOC(ntables-1,int)) ) return(-1);
85      fgets(junk,MJUNK,fp);
```

```
 86      fgets(junk,MJUNK,fp);
 87      for(i=0;i<ntables-1;i++) {
 88          fscanf(fp,"%*d %d %d %d\n",&rh[i],&th[i],&tdh[i]);
 89      }
 90
 91      // read primaries info
 92      fgets(junk,MJUNK,fp);
 93      fgets(junk,MJUNK,fp);
 94      fscanf(fp,"%*s %*s %*s %d ",&p);fgets(junk,MJUNK,fp);
 95
 96      if (create_table1H2D(ptab,ntables,nrows,ncols,p)) return(-1);
 97      TABLE1H2D *htab= *ptab;
 98
 99      fgets(junk,MJUNK,fp);
100      fgets(junk,MJUNK,fp);
101      int pt,pi,pj;
102      double plpl,pupl;
103      for(i= 0;i<get_npcells1H2D(htab);i++) {
104          fscanf(fp,"%*d %d %d %d %lf %lf\n",&pt,&pi,&pj,&plpl,&pupl);
105          put_pcell1H2D(htab,i,pt,pi,pj,plpl,pupl);
106      }
107
108      // set hierarchical structure
109      for(i=0;i<ntables-1;i++) {
110          put_parent1H2D(htab,tdh[i],th[i]);
111          put_rowinparent1H2D(htab,tdh[i],rh[i]);
112      }
113
114      //read cells info
115      fgets(junk,MJUNK,fp);
116      fgets(junk,MJUNK,fp);
117      fgets(junk,MJUNK,fp);
118      for(t=0;t<get_ntables1H2D(htab);t++)
119          for(i=0;i<get_nrows1H2D(htab,t);i++)
120              for (j=0;j<get_ncolumns1H2D(htab,t);j++) {
121                  fscanf(fp,"%d %d %d %lf %lf\n",&tcell,&icell,&jcell,
122                         &vcell,&wcell);
123                  put_cellvalue1H2D(htab,tcell,icell,jcell,vcell);
124                  put_cellweight1H2D(htab,tcell,icell,jcell,wcell);
125              }
126      fclose(fp);
127      MYFREE(nrows);
128      MYFREE(rh);
129      MYFREE(th);
130      MYFREE(tdh);
131      return(0);
132  }
```

The secondary cells obtained are those listed in Subsection 2.2, since we are using the same example instance.

If a lower bound for the optimal solution wants to be computed, we can call `put_comp_lowbound1H2D(htab,true)` before `csp_heur1H2D(htab)`. Computing a lower bound means solving a linear program, and a CPLEX license is needed for this.

NF_CSP also includes an auditing phase for computing the lower and upper bounds that an external attacker could derive for the primary cells after the publication of the table. These values are computed in the package by solving two minimum-cost network flows problems. This is done in NF_CSP either by CPLEX or PPRN. NF_CSP first attempts to use CPLEX; if no license is available then it switches to PPRN. Note that the auditing phase of NF_CSP was just developed for testing purposes, and that it is not the most efficient procedure. It may be significantly improved by computing the two bounds through specialized maximum-flows algorithms. In our example, to compute the lower and upper protection provided by the suppression pattern, we could insert at line 39 the following code:

```
{
    // perform auditing phase--just for testing purposes
    if ((ret_stat= csp_auditing1H2D(htab,0,get_npcells1H2D(htab)-1))) {
        fprintf(stderr,"error %d reported by csp_auditing1H2D\n",ret_stat);
        ret_stat= -1;
        goto TERMINATE;
    };
    int i,pt,pi,pj= 0;
    double lpl,upl,plpl,pupl;
    for(i=0;i<get_npcells1H2D(htab);i++) {
        get_pcell1H2D(htab,i,&pt,&pi,&pj,&lpl,&upl);
        plpl= get_pcelllpl1H2D(htab,i);
        pupl= get_pcellupl1H2D(htab,i);
        printf("primary (%d,%d,%d):\t lower %f > %f \t upper %f > %f\n",
               pt,pi,pj,plpl,lpl,pupl,upl);
    }
}
```

`csp_auditing1H2D(htab,first,last)` performs the auditing of primary cells between `first` and `last`. In our code, we perform the auditing for the 2 primary cells (i.e., `first` is 0 and `last` is `get_npcells1H2D(htab)-1` $= 2-1 = 1$). After that, we retrieve the required lower and upper protection for each primary calling `get_pcell1H2D(htab,i,&pt,&pi,&pj,&lpl,&upl)`. This routine returns the information of primary cell `i`, providing its position (`pt`,`pi`,`pj`) in the table, and the required lower and upper protection `lpl` and `upl`. Next we get the lower and upper protection level obtained for primary `i` calling `get_pcelllpl1H2D(htab,i)` and `get_pcellupl1H2D(htab,i)`. Finally we print the lower and upper protection levels obtained, showing they are greater than those required. The output of the above piece of code in our example is:

```
primary (0,0,0):    lower 5.000000 > 2.000000       upper 6.000000 > 2.000000
primary (1,1,0):    lower 2.000000 > 2.000000       upper 5.000000 > 2.000000
```

3 Package options

3.1 Conditional compilation

The package has been successfully compiled an tested in both Linux (using gcc 2.95) and MS-Windows XP (using MS-Visual C++ 6.0, MSVC6 for short). It should also work in any other Unix or MS-Windows system.

Three symbols are available for conditional compilation depending on the environment. This is done through /D*symbol_name* in MSVC6 and -D*symbol_name* in gcc. Note that two of these symbols are only required for compiling the package, whereas the first one needs also to be defined for compiling the user's application, as explained below. The three symbols are:

- `WIN32`. This symbol must be defined for compiling NF_CSP with MSVC6 in a MS-Windows system. It is also needed for the user's routines that interface with NF_CSP, again only in MS-Windows systems. When `WIN32` is defined, three more symbols are required for compiling the heuristic, PPRN and Dijkstra: `CSP_NF_1H2D_EXPORTS`, `PPRN_EXPORTS` and `DIJKSTRA_EXPORTS`. They allow exporting the interface functions in the .dll libraries. The distribution of the package already includes those symbols, and the user/programmer does not have to care about them. These three export symbols **DON'T** have to be defined for compiling the user's application, otherwise it will fail to interface with NF_CSP.
- `CPLEXN`. This symbol is required if one has a CPLEX license and plans to use it. It is not needed for compiling the user's application. If the symbol is defined, NF_CSP will consider either CPLEX7.5 or CPLEX8.0 as one of the available solvers. The right version is chosen defining one of the symbols `CPLEX75` or `CPLEX80`. If `CPLEXN` is not defined and NF_CSP is asked to use CPLEX it will return an error.
- `LOGINFO`. If this symbol is defined NF_CSP will display information about the evolution of the protection procedure. It is only intended for debugging purposes. It is not needed for compiling the user's application.

3.2 Solvers

The package can work with three network flows solvers: PPRN, CPLEX and an implementation of the bidirectional Dijkstra algorithm. PPRN and Dijkstra are included in the package. CPLEX needs a commercial license. By default the package uses the Dijkstra solver, which is by far the fastest of the three available. This default is set by routine `create_table()`.

A particular solver can be set through

```
put_solver1H2D(htab,solver),
```

where `solver` can be `DIJKSTRA`, `PPRN` or `CPLEX`. Routine

```
get_solver1H2D(htab)
```

returns the current solver.

Once the solver was selected, the protection is performed through

```
csp_heur1H2D(htab).
```

This routine returns 0 if the table was successfully protected. Otherwise it returns a nonzero code. Section 4 shows the return codes of all the interface routines.

3.3 Type of arc costs

The heuristic solves several shortest-paths problems on a network whose topology is defined by the table. The costs of arcs in this network are dynamically created for each primary cell by the heuristic. The purpose of these costs is to guide the protection procedure, making unsuppressed cells with low weights better candidates for suppression than those with larger weights. Computing these costs can be fairly expensive, and NF_CSP offers two alternatives. The first one, called `FASTER_WORSER`, computes a set of costs efficiently; however these costs may provide worser solutions than the second set of costs. This second set is the `SLOWER_BETTER`. As its name shows, the heuristic is significantly slower if these costs are computed, although the solution provided may be slightly better. Several tests showed that, in average, the `FASTER_WORSER` option reduces in a 20% the execution times of the `SLOWER_BETTER` one. The difference in value suppressed is highly dependent on the particular table. However in most tests performed, it was not significant (even 0 in several tests). The default option of NF_CSP is `FASTER_WORSER`. It can be changed with

```
put_cost_type1H2D(htab,cost_type),
```

where `cost_type` can be `FASTER_WORSER` or `SLOWER_BETTER`. The current type of cost is returned by function

```
get_cost_type1H2D(htab).
```

3.4 Cell status and zero cells

By default, cells with zero values are considered `PERMANENT`. A cell is permanent if it can not be removed by the heuristic. The other possible status of a cell are `PRIMARY`, `SECONDARY` and `NONREMOVED`. When creating a table, by default all the cells are `NONREMOVED` (candidates for suppression). The cells that the user define as primaries (through `put_pcell1H2D()`), are marked as `PRIMARY`. The heuristic will remove additional cells, marking them as `SECONDARY`. The user can retrieve the current status of cell (`subtable`,`row`,`col`) calling

```
get_cellstatus1H2D(htab,subtable,row,col),
```

which returns one of the four possible status. It is also possible to set the status of cell (`subtable`,`row`,`col`) through

```
put_cellstatus1H2D(htab,subtable,row,col,status).
```

This routine is intended for setting some cells as permanent. It should not be used for other status, because it could make the heuristic fail.

The only cells set as permanent by default by the heuristic are those with zero values. The user is allowed to change this default behaviour (therefore zero cells will be candidates for suppression, unless explicitly stated in the code with `put_cellstatus1H2D()`) through

```
put_cells_0_permanent1H2D(htab,false).
```

The default behaviour can be recovered calling the above routine with `true` as second argument. The current situation for zero cells can be retrieved using

```
get_cells_0_permanent1H2D(htab).
```

It returns `true` if zero cells will be set permanent; otherwise returns `false`.

3.5 Merit order for primary cells

The heuristic of NF_CSP is an iterative process that sequentially protects each primary cell. The order primary cells are selected (named merit order in NF_CSP) may modify the final solution. The user can choose between three merit orders: `NORMAL`, `ASCENDENT` and `DESCENDENT`. `NORMAL` is the order defined by the user when setting the primary cells through `put_pcell1H2D()`. If the `ASCENDENT` order is selected, cells will be protected according to theirs cell values sorted in ascendent order (i.e., the first cell protected will be that with the lowest cell value, and so on). The order is the opposite if `DESCENDENT` is chosen. The default merit order is `NORMAL`. This default can be modified with

```
put_typemorder1H2D(htab,merit),
```

where the second parameter can be any of the three merit orders. The current merit order is returned by

```
get_typemorder1H2D(htab).
```

To get information about the primary cells, NF_CSP offers two sets of functions, one for "unsorted" and other for "sorted" primary cells. All of them have as one of theirs parameters the position `i` of the primary cell. The unsorted-version functions provide the information for the `i`-th primary cell entered by the user. The sorted-version functions return the information for the `i`-th primary cell according to the current merit order. Clearly, if the current merit order is `NORMAL` both the sorted and unsorted-version functions return the same information. The sorted-version functions are

```
get_pcell1H2D(htab,i,&subtable,&row,&col,&lpl,&upl),
              get_pcelllpl1H2D(htab,i),
              get_pcellupl1H2D(htab,i).
```

The unsorted-version ones are

```
get_sorted_pcell1H2D(htab,i,&subtable,&row,&col,&lpl,&upl),
              get_sorted_pcelllpl1H2D(htab,i),
              get_sorted_pcellupl1H2D(htab,i).
```

The first function of each set provides the position (`subtable`, `row`,`col`) and required lower (`lpl`) and upper (`upl`) protection of primary `i`. The last two functions of each set return respectively the obtained lower and upper protection for primary `i`, either after calling the heuristic, or after performing the auditing. After the auditing, they return the real protection. After the heuristic, they provide a lower bound of the real protection.

3.6 Lower bounding procedure

The heuristic provide an approximate solution to the cell suppression problem. To know how far that solution is from the optimal one, we should get some lower bound to the optimal objective function (i.e., minimum value or minimum weight suppressed). NF_CSP includes a procedure for computing such lower bound. It is computed inside the heuristic before starting the protection procedure. The lower bounding procedure is an improvement (i.e., provides better lower bounds) of that originally suggested in [9]. Computing the lower bound means solving a linear programming problem, and a CPLEX license is needed for that. Moreover, for large tables, it can be an expensive computation, even more costly than the protection procedure. For these reasons, by default the lower bound is not computed. If we want to compute it, we can call

```
put_comp_lowbound1H2D(htab,true).
```

Setting the second parameter of the above function to `false` we can deactivate the computation of the lower bound. To know if the lower bounding procedure is activated or not, NF_CSP provides

```
get_comp_lowbound1H2D(htab).
```

If returns `true` if activated, otherwise it returns `false`. The lower bound obtained is returned by function

```
get_comp_lowbound1H2D(htab).
```

If the lower bounding procedure was not activated, the above function returns 0.

The solution provided by the lower bounding procedure can also be used as a starting point for the heuristic. By default, that solution is not considered, and the heuristic starts from scratch. This behaviour can be modified with

```
put_use_lowbound1H2D(htab,uselb).
```

The lower bound solution will either be used or not in the heuristic if the second parameter of the above function is respectively `true` or `false`. To know the current situation about the usage of the lower bound solution, we can call

```
get_use_lowbound1H2D(htab).
```

It returns `true` if the lower bound solution is going to be used, otherwise it returns `false`. Clearly, if the lower bound solution is asked to be used, and the lower bound procedure was not activated, the heuristic will return an error code.

3.7 Auditing

NF_CSP implements an auditing phase for computing the lower and upper bounds than an external attacker could derive for the primary cells after the publication of the table. For two dimensional tables with at maximum one hierarchical dimension, these values can be computed by solving two maximum-flows problems. This is done in NF_CSP either by CPLEX or PPRN, solving two minimum-cost network flows problems. NF_CSP first attempts to use CPLEX; if no license is available then it switches to PPRN. The auditing phase of NF_CSP was just developed for testing

purposes, and it is not the most efficient procedure. It may be significantly improved by computing the two bounds through specialized maximum-flows algorithms.

The function that performs the auditing is

```
csp_auditing1H2D(htab,beg,end).
```

Arguments `beg` and `end` provide the range of primary cells considered (from `beg` to `end`). If one just wants to audit the primary cell `i`, we'll set `beg` and `end` to `i`.

After the auditing is performed, the lower and upper protection levels for primary `i` can be retrieved respectively by functions

```
get_pcelllpl1H2D(htab,i),
get_pcellupl1H2D(htab,i),
```

and

```
get_sorted_pcelllpl1H2D(htab,i),
get_sorted_pcellupl1H2D(htab,i)
```

(see Subsection 3.5 for an explanation of the differences between the plain and sorted variants). If x is the cell value of primary `i`, and l and u are the values returned respectively by the `get_pcelllpl1H2D(htab,i)` and `get_pcellupl1H2D(htab,i)` functions, the attacker knows that the real value of this primary is in the range $[x - l, x + u]$.

4 Interface routines

This section describes the user's interface routines to NF_CSP. They are grouped by the type of manipulation performed to a table.

4.1 Creating and removing tables

- **Function:** `int create_table(TABLE1H2D **htab, int t, int m, int n, int p)`

  Purpose: It creates and initializes a table of `t` subtables, `m[i]` rows for each subtable i, `n` columns and `p` primaries.

  Returns: 0 if the table was successfully created; otherwise (e.g., if not enough memory for allocating the table) it returns -1.

  Input arguments: `t` is the number of subtables; `m` is a vector that gives at position `m[i]` the number of rows of table i ; `n` is the number of columns; `p` is the number of primary cells.

  Output arguments: `*htab` is a pointer to the newly created table.

  Input/Output arguments: None

  Example:

```
TABLE1H2D *htab;
int ret;
const t= 5; //5 subtables
```

```
int m[t];
...// initialize m[0]..m[4] with some values
ret= create_table(&htab,t,m,15,20); // 5 subtables, 15 columns, 20 primaries
```

- **Function:** `void delete_table(TABLE1H2D **htab)`

  Purpose: Deletes a table, freeing its memory space.

  Returns: Nothing.

  Input arguments: None

  Output arguments: None

  Input/Output arguments: `htab` on input is a table (possibly empty); on output, is an empty table.

  Example:

```
TABLE1H2D *htab;
...
delete_table(&htab);
```

4.2 Entering table information

- **Function:** `void put_cellvalue1H2D(TABLE1H2D *htab, int tab, int row, int column, double value)`

  Purpose: It fills cell (`tab`,`row`,`column`) with `value` without checking that `tab`, `row` and `column` are within bounds.

  Returns: Nothing.

  Input arguments: `tab` is the subtable; `row` is the cell row; `column` is the cell column; `value` is the cell value.

  Output arguments: None.

  Input/Output arguments: `htab` is the table to be updated.

  Example:

```
TABLE1H2D *htab;
...
put_cellvalue1H2D(htab,2,1,1,10.0); // cell (2,1,1) is 10.0
```

- **Function:** `void put_cellweight1H2D(TABLE1H2D *htab, int tab, int row, int column, double weight)`

  Purpose: It sets the weight of cell (`tab`,`row`,`column`) without checking that `tab`, `row` and `column` are within bounds.

  Returns: Nothing.

  Input arguments: `tab` is the subtable; `row` is the cell row; `column` is the cell column; `weight` is the cell weight.

Output arguments: None.

Input/Output arguments: `htab` is the table to be updated.

Example:

```
TABLE1H2D *htab;
...
put_cellweight1H2D(htab,2,1,1,2.5); // weight of cell (2,1,1) is 2.5
```

- **Function:** `void put_cellstatus1H2D(TABLE1H2D *htab, int tab, int row, int column, STATUS_CELL status)`

  Purpose: It sets the status of cell (`tab`, `row`,`column`) without checking that `tab`, `row` and `column` are within bounds. See Subsection 3.4 for a detailed explanation of the available status.

  Returns: Nothing.

  Input arguments: `tab` is the subtable; `row` is the cell row; `column` is the cell column; `status` is the cell status, which can be `PRIMARY`, `SECONDARY`, `PERMANENT` or `NONREMOVED`. The user should only call this routine to set `PERMANENT` cells.

  Output arguments: None.

  Input/Output arguments: `htab` is the table to be updated.

  Example:

```
TABLE1H2D *htab;
...
put_cellstatus1H2D(htab,2,1,1,PERMANENT); // cell (2,1,1) is set PERMANENT
```

- **Function:** `void put_cells_0_permanent1H2D(TABLE1H2D *htab, bool setperm)`

  Purpose: It sets if cells with a zero value must be automatically set permanent by the heuristic. If not changed through this function, the default is to consider zero cells permanent. See Subsection 3.4 for a detailed explanation.

  Returns: Nothing.

  Input arguments: `setperm` is a boolean; if `true`, zero cells will be set permanent, otherwise they will preserve their current status.

  Output arguments: None.

  Input/Output arguments: `htab` is the table to be updated.

  Example:

```
TABLE1H2D *htab;
...
put_cells_0_permanent1H2D(htab,false);
```

- **Function:** `void put_pcell1H2D(TABLE1H2D *htab, int pcell, int pt, int pi, int pj, double plpl, double pupl)`

Purpose: It sets the information for the primary cell number `pcell`, without checking that `pcell` is within bounds.

Returns: Nothing

Input arguments: `pcell` is the primary cell number to be considered; `pt`, `pi` and `pj` are the subtable, row and column position of this primary cell; `plpl` and `pupl` are the lower and upper protection levels required for this primary.

Output arguments: None.

Input/Output arguments: `htab` is the table to be updated.

Example:

```
TABLE1H2D *htab;
...
put_pcell1H2D(htab,3,2,7,9,10.5,20.1);
// cell (2,7,9) is the third primary, with required lower protection
// level 10.5 and required upper protection level 20.1
```

- **Function:** `void put_comp_lowbound1H2D(TABLE1H2D *htab, bool comp_lb)`

  Purpose: It sets if the lower bound must or not be computed for table `htab`. See Subsection 3.6 for details.

  Returns: Nothing.

  Input arguments: `comp_lb` is a boolean; if `true`, the lower bound will be computed, otherwise it will not.

  Output arguments: None.

  Input/Output arguments: `htab` is the table to be updated.

  Example:

```
TABLE1H2D *htab;
...
put_comp_lowbound1H2D(htab,true);
```

- **Function:** `void put_use_lowbound1H2D(TABLE1H2D *htab, bool use_lb)`

  Purpose: It sets if the solution obtained by the lower bounding procedure must or not be used in the heuristic for table `tab`. See Subsection 3.6 for details.

  Returns: Nothing.

  Input arguments: `use_lb` is a boolean; if `true`, the solution provided by the lower bounding procedure will be used in the heuristic, otherwise it will not.

  Output arguments: None.

  Input/Output arguments: `htab` is the table to be updated.

  Example:

```
TABLE1H2D *htab;
...
put_use_lowbound1H2D(htab,true);
```

- **Function:** `void put_typemorder1H2D(TABLE1H2D *htab, TYPE_MERIT_ORDER morder)`

  Purpose: It sets the type of merit order for the primary cells (i.e., the order they will be processed by the heuristic). If not changed through this function, the default order is `NORMAL`.

  Returns: Nothing.

  Input arguments: `morder` is the type of merit order. It can be `NORMAL`, `ASCENDENT` or `DESCENDENT`. See Subsection 3.5 for an explanation of each type of merit order.

  Output arguments: None.

  Input/Output arguments: `htab` is the table to be updated.

  Example:

```
TABLE1H2D *htab;
...
put_typemorder1H2D(htab,DESCENDENT);
```

- **Function:** `void put_solver1H2D(TABLE1H2D *htab, SOLVER solver)`

  Purpose: It sets which of the three available solvers will be used for the subproblems generated by the heuristic. If not changed through this function, the default solver is Dijkstra. See Subsection 3.2 for more details.

  Returns: Nothing.

  Input arguments: `solver` is the solver to be used. It can take values `DIJKSTRA`, `CPLEX` or `PPRN`.

  Output arguments: None.

  Input/Output arguments: `htab` is the table to be updated.

  Example:

```
TABLE1H2D *htab;
...
put_solver1H2D(htab,PPRN);
```

- **Function:** `void put_cost_type1H2D(TABLE1H2D *htab, COST_TYPE cost_type)`

  Purpose: It sets the type of arc costs for the shortest-path subproblems. If not changed through this function, the default type is `FASTER_WORSER`.

  Returns: Nothing.

  Input arguments: `cost_type` is the type of costs. It can be `FASTER_WORSER` or `SLOWER_BETTER`. See Subsection 3.3 for an explanation of each type of costs.

  Output arguments: None.

  Input/Output arguments: `htab` is the table to be updated.

  Example:

```
TABLE1H2D *htab;
...
put_cost_type1H2D(htab,SLOWER_BETTER);
```

4.3 Retrieving table information

- **Function:** `int get_nrows1H2D(TABLE1H2D *htab, int tab)`

  Purpose: It provides the number of rows of subtable `tab` of table `htab`.

  Returns: The number of rows of subtable `tab`.

  Input arguments: `htab` is the table; `tab` is the subtable.

  Output arguments: None.

  Input/Output arguments: None.

  Example:

```
TABLE1H2D * htab;
...
int nrows= get_nrows1H2D(htab,2); //number of rows of subtable 2
```

- **Function:** `int get_ncolumns1H2D(TABLE1H2D *htab, int tab)`

  Purpose: It provides the number of columns of subtable `tab` of table `htab`. Although all subtables have the same number of columns, the subtable is included as a parameter for consistence with routine `get_nrows1H2D`, and for generality.

  Returns: The number of columns of subtable `tab`.

  Input arguments: `htab` is the table; `tab` is the subtable.

  Output arguments: None.

  Input/Output arguments: None.

  Example:

```
TABLE1H2D * htab;
...
int ncols= get_ncolumns1H2D(htab,2); //number of columns of subtable 2
```

- **Function:** `int get_npcells1H2D(TABLE1H2D *htab)`

  Purpose: It provides the number of primary cells of table `htab`.

  Returns: The number of primary cells.

  Input arguments: `htab` is the table.

  Output arguments: None.

  Input/Output arguments: None.

  Example:

```
TABLE1H2D * htab;
...
int p= get_npcells1H2D(htab);
```

- **Function:** `int get_nseccells1H2D(TABLE1H2D *htab)`

Purpose: It provides the number of secondary cells of table `htab`.

Returns: The number of secondary cells.

Input arguments: `htab` is the table.

Output arguments: None.

Input/Output arguments: None.

Example:

```
TABLE1H2D * htab;
...
int s= get_nseccells1H2D(htab);
```

- **Function:** `double get_cellvalue1H2D(TABLE1H2D *htab, int tab, int row, int column)`

  Purpose: It provides the value of cell (`tab`,`row`,`column`) without checking that `tab`, `row` and `column` are within bounds.

  Returns: The value of cell (`tab`,`row`,`column`).

  Input arguments: `htab` is the table; `tab` is the subtable; `row` is the cell row; `column` is the cell column.

  Output arguments: None.

  Input/Output arguments: None.

  Example:

```
TABLE1H2D *htab;
...
double v= get_cellvalue1H2D(htab,2,3,4); // value of cell (2,3,4)
```

- **Function:** `double get_cellweight1H2D(TABLE1H2D *htab, int tab, int row, int column)`

  Purpose: It provides the weight of cell (`tab`,`row`,`column`) without checking that `tab`, `row` and `column` are within bounds.

  Returns: The weight of cell (`tab`,`row`,`column`).

  Input arguments: `htab` is the table; `tab` is the subtable; `row` is the cell row; `column` is the cell column.

  Output arguments: None.

  Input/Output arguments: None.

  Example:

```
TABLE1H2D *htab;
...
double v= get_cellweight1H2D(htab,2,3,4); // weight of cell (2,3,4)
```

- **Function:** `STATUS_CELL get_cellstatus1H2D(TABLE1H2D *htab, int tab, int row, int column)`

Purpose: It provides the status of cell (tab,row,column) without checking that tab, row and column are within bounds.

Returns: The status of cell (tab,row,column) (which can be PRIMARY, SECONDARY, PERMANENT or NONREMOVED). See Subsection 3.4 for an explanation of each status.

Input arguments: htab is the table; tab is the subtable; row is the cell row; column is the cell column.

Output arguments: None.

Input/Output arguments: None.

Example:

```
TABLE1H2D *htab;
...
STATUS_CELL status= get_cellstatus1H2D(htab,2,3,4); // status of cell (2,3,4)
```

- **Function:**

```
bool is_cell_primary1H2D(TABLE1H2D *htab, int tab, int row, int column)
bool is_cell_secondary1H2D(TABLE1H2D *htab, int tab, int row, int column)
bool is_cell_permanent1H2D(TABLE1H2D *htab, int tab, int row, int column)
bool is_cell_nonremoved1H2D(TABLE1H2D *htab, int tab, int row, int column)
```

Purpose: Auxiliary functions to know if cell (tab,row,column) has a particular status.

Returns: true if the cell has the particular status; otherwise, false.

Input arguments: htab is the table; tab is the subtable; row is the cell row; column is the cell column.

Output arguments: None.

Input/Output arguments: None.

Example:

```
TABLE1H2D *tab;
...
if (is_cell_secondary1H2D(htab,2,3,4)) {
...  // treatment for cell (2,3,4) if secondary
}
```

- **Function:** `bool get_cells_0_permanent1H2D(TABLE1H2D *htab)`

Purpose: To know if cells with a zero value must be automatically set permanent by the heuristic. See Subsection 3.4 for a detailed explanation.

Returns: true if zero cells will be set permanent; otherwise, false.

Input arguments: htab is the table.

Output arguments: None.

Input/Output arguments: None.

Example:

```
TABLE1H2D *htab;
...
if (get_cells_0_permanent1H2D(htab)) {
...
}
```

- **Function:**
  `void get_pcell1H2D(TABLE1H2D *htab, int pcell, int * pt, int * pi, int* pj, double *lpl, double *upl)`
  `void get_sorted_pcell1H2D(TABLE1H2D *htab, int pcell, int * pt, int * pi, int* pj, double *lpl, double *upl)`

  Purpose: These functions provide the location and required protection levels of the primary cell number `pcell`. The first function considers primary cells are in the order provided by the user. The second function considers the current merit order. See Subsection 3.5 for details.

  Returns: Nothing.

  Input arguments: `htab` is the table; `pcell` is the primary cell number.

  Output arguments: `pt`, `pi` and `pj` are respectively the subtable, row and column of the primary cell; `lpl` and `upl` are respectively the required lower and upper protection levels.

  Input/Output arguments: None.

  Example:

```
TABLE1H2D *htab;
int pt,pi,pj;
double lpl,upl;
...
// location and protection required for the third primary cell
// entered by the user
get_pcell1H2D(htab,3,&pt,&pi,&pj,&lpl,&upl);
...
// location and protection required for the third primary cell
// according to the merit order
get_sorted_pcell1H2D(htab,3,&pt,&pi,&pj,&lpl,&upl);
```

- **Function:**
  `double get_pcelllpl1H2D(TABLE1H2D *htab, int pcell)`
  `double get_sorted_pcelllpl1H2D(TABLE1H2D *htab, int pcell)`

  Purpose: These functions provide the lower protection level currently obtained by the primary cell number `pcell`. The first function considers primary cells are in the order provided by the user. The second function considers the current merit order. See Subsection 3.5 for details.

  Returns: Current lower protection level of the primary cell.

Input arguments: `htab` is the table; `pcell` is the primary cell number.

Output arguments: None

Input/Output arguments: None.

Example:

```
TABLE1H2D *htab;
...
// current lower protection obtained for the third primary cell
// entered by the user
double lpl= get_pcelllpl1H2D(htab,3);
...
// current lower protection obtained for the third primary cell
// according to the merit order
double lpl= get_sorted_pcelllpl1H2D(htab,3);
```

- **Function:**
`double get_pcellupl1H2D(TABLE1H2D *htab, int pcell)`
`double get_sorted_pcellupl1H2D(TABLE1H2D *htab, int pcell)`

  Purpose: These functions provide the upper protection level currently obtained by the primary cell number `pcell`. The first function considers primary cells are in the order provided by the user. The second function considers the current merit order. See Subsection 3.5 for details.

  Returns: Current upper protection level of the primary cell.

  Input arguments: `htab` is the table; `pcell` is the primary cell number.

  Output arguments: None

  Input/Output arguments: None.

  Example:

```
TABLE1H2D *htab;
...
// current upper protection obtained for the third primary cell
// entered by the user
double upl= get_pcellupl1H2D(htab,3);
...
// current upper protection obtained for the third primary cell
// according to the merit order
double upl= get_sorted_pcellupl1H2D(htab,3);
```

- **Function:** `bool get_comp_lowbound1H2D(TABLE1H2D *htab)`

  Purpose: To know if the lower bound will or not be computed by the heuristic. See Subsection 3.6 for details.

  Returns: `true` if the lower bound has to be computed; otherwise, `false`.

  Input arguments: `htab` is the table.

  Output arguments: None.

Input/Output arguments: None.

Example:

```
TABLE1H2D *htab;
...
if (get_comp_lowbound1H2D(htab)) {
...
}
```

- **Function:** `bool get_use_lowbound1H2D(TABLE1H2D *htab)`

  Purpose: To know if the solution computed by the lower bounding procedure will or not be used by the heuristic. See Subsection 3.6 for details.

  Returns: `true` if the lower bound solution will be used; otherwise, `false`.

  Input arguments: `htab` is the table.

  Output arguments: None.

  Input/Output arguments: None.

  Example:

```
TABLE1H2D *htab;
...
if (get_use_lowbound1H2D(htab)) {
...
}
```

- **Function:** `double get_lowerbound1H2D(TABLE1H2D *htab)`

  Purpose: It provides the lower bound computed by the lower bounding procedure. See Subsection 3.6 for details.

  Returns: The lower bound.

  Input arguments: `tab` is the table.

  Output arguments: None.

  Input/Output arguments: None.

  Example:

```
TABLE1H2D *htab;
...
double lb= get_lowerbound1H2D(htab);
```

- **Function:** `TYPE_MERIT_ORDER get_typemorder1H2D(TABLE1H2D *htab)`

  Purpose: It provides the type of merit order for the primary cells (i.e., the order they will be processed by the heuristic).

  Returns: `NORMAL`, `ASCENDENT` or `DESCENDENT`. See Subsection 3.5 for an explanation of each type of merit order.

  Input arguments: `htab` is the table.

Output arguments: None.

Input/Output arguments: None.

Example:

```
TABLE1H2D *htab;
...
TYPE_MERIT_ORDER tmorder= get_typemorder1H2D(htab);
```

- **Function:** `SOLVER get_solver1H2D(TABLE1H2D *htab)`

  Purpose: It provides which of the three available solvers will be used for the subproblems generated by the heuristic.

  Returns: `DIJKSTRA`, `CPLEX` or `PPRN`. See Subsection 3.2 for details on each solver.

  Input arguments: `htab` is the table.

  Output arguments: None.

  Input/Output arguments: None.

  Example:

```
TABLE1H2D *htab;
...
SOLVER solver= get_solver1H2D(htab);
```

- **Function:** `COST_TYPE get_cost_type1H2D(TABLE1H2D *htab)`

  Purpose: It provides the type of arc costs for the shortest-path subproblems of the heuristic.

  Returns: `FASTER_WORSER` or `SLOWER_BETTER`. See Subsection 3.3 for an explanation of each type of costs.

  Input arguments: `htab` is the table.

  Output arguments: None.

  Input/Output arguments: None.

  Example:

```
TABLE1H2D *htab;
...
COST_TYPE tcost= get_cost_type1H2D(htab);
```

- **Function:** `int get_numnfproblems1H2D(TABLE1H2D *htab)`

  Purpose: It provides the number of shortest-path subproblems solved by the heuristic for the protection of the table.

  Returns: The number of shortest-path subproblems.

  Input arguments: `htab` is the table.

  Output arguments: None.

  Input/Output arguments: None.

  Example:

```
TABLE1H2D *htab;
...
int nf= get_numnfproblemes1H2D(htab);
```

4.4 Executing heuristic and other procedures

- **Function:** `int csp_heur1H2D(TABLE1H2D *htab);`

  Purpose: It protects a table. The type of solver, lower bounding information, etc., must have been previously set by the user; otherwise, the default ones will be used.

  Returns:
  0 if the table was successfully protected;
  -1 if not enough memory for protecting the table;
  -2 if the table is wrong because some cell appears with different values in two subtables;
  -3 if the table is wrong because some cell appears with different values in two subtables;
  -4 if the table is wrong because some cell appears with different protection levels in two subtables;
  -5 if the shortest-path subproblem was detected as infeasible;
  -6 if some network flows subproblem produced some error during its solution;
  -7 if there are problems with the CPLEX license;
  -8 if some error happened creating the CPLEX network object;
  -9 if some error was found in the lower bounding procedure;
  -51 if some error happened when ordering the primary cells by merit order;
  -53 if the package was compiled without CPLEX support and CPLEX is the solver to be used;
  -54 if the lower bound wants to be computed and the package was compiled without CPLEX support;
  -55 if the lower bound solution wants to be used without being computed.

  Return codes -2 to -4 are due to a wrong input table. Return codes -5 to -9 are mainly associated to the solvers. Return codes -50 to -55 are mainly due to bad user settings.

  Input arguments: None.

  Output arguments: None.

  Input/Output arguments: On input, `htab` is the table to be protected. On output, `htab` is the protected table.

  Example:

```
TABLE1H2D *tab;
...
int retstat= csp_heur1H2D(htab);
```

- **Function:** `int csp_auditing1H2D(TABLE1H2D *htab, int beg, int end)`

  Purpose: It performs the auditing (i.e., lower and upper protection levels obtained) for the primary cells in the range `beg`...`end`, once the table was successfully protected. This function considers the order provided by the user for primary cells (not the merit order). This auditing was designed just for testing purposes, and it does not implement the most efficient algorithm. By default it attempts to use CPLEX; if a license is not available it switches to PPRN.

  Returns:
  0 if the primaries were successfully audited;
  -1 if not enough memory for auditing the primaries;

-2 if `beg` or `end` are out of bounds;
-5 if some error was found solving the minimum-cost network flows subproblem for the lower or upper protection level of some primary.

Input arguments: `beg` is the first primary to be protected; `end` is the last primary to be protected.

Output arguments: None.

Input/Output arguments: On input, `htab` is a protected table. On output, `htab` is a protected and audited (only for cells in range `beg`...`end`) table.

Example:

```
TABLE1H2D *htab;
...
//auditing for all the primaries
int retstat= csp_auditing1H2D(htab,0,get_npcells1H2D(htab)-1);
```

References

[1] Ahuja, R.K, Magnanti, T.L., Orlin, J.B., *Network Flows*, Prentice Hall (1993).

[2] Carvalho, F.D., Dellaert, N.P., Osório, M.D., Statistical disclosure in two-dimensional tables: general tables. *J. Am. Stat. Assoc.* 89, (1994) 1547–1557

[3] Castro, J., PPRN 1.0, User's Guide, Technical report DR 94/06 Dept. of Statistics and Operations Research, Universitat Politècnica de Catalunya, Barcelona, Spain, 1994.

[4] Castro, J., Network flows heuristics for complementary cell suppression: an empirical evaluation and extensions. *Lecture Notes in Computer Scinence* 2316, (2002) 59–73. Volume *Inference Control in Statistical Databases*, J. Domingo-Ferrer (Ed).

[5] Castro, J., A fast network flows heuristic for cell suppression in positive tables. *Lecture Notes in Computer Scinence*, (2004), in press.

[6] Castro, J., Nabona, N. An implementation of linear and nonlinear multicommodity network flows. *European Journal of Operational Research* 92, (1996) 37–53.

[7] Cox, L.H., Network models for complementary cell suppression. *J. Am. Stat. Assoc.* 90, (1995) 1453–1462.

[8] ILOG CPLEX, *ILOG CPLEX 7.5 Reference Manual Library*, ILOG, (2000).

[9] Kelly, J.P., Golden, B.L, Assad, A.A., Cell Suppression: disclosure protection for sensitive tabular data, *Networks* 22, (1992) 28–55.

APPENDIX

The information of this appendix was generated from the code, which can be object of future revisions. It can then present some inaccuracies or be out of date. Look at the code for full details. The location of files and routines corresponds to the MS-Windows distribution of the package.

A Global information

Package Name	Network Flows heuristics for Tau-Argus (1H2D tables)
Package Owner	Dept. of Statistics and Operations Research Universitat Politècnica de Catalunya Barcelona
Contact Person	Jordi Castro, `jcastro@eio.upc.es`
Starting Date	January 2001
Ending Date	December 2003
Programming Environment	Linux and gcc, ported to Windows and MS-Visual C++

B List of files (alphabetical order)

	File Name	Location	Lines	Bytes
1	csp_auditing1H2D.cpp	Tau-Argus-UPC\csp_nf\src\1H2Dtables\	190	5460
2	csp_heur1H2D.cpp	Tau-Argus-UPC\csp_nf\src\1H2Dtables\	195	5974
3	csp_heur1H2D.h	Tau-Argus-UPC\csp_nf\src\1H2Dtables\	201	6702
4	csp_heur1H2D_01flows.cpp	Tau-Argus-UPC\csp_nf\src\1H2Dtables\	1275	42244
5	csp_lower_bound.cpp	Tau-Argus-UPC\csp_nf\src\1H2Dtables\	545	13989
6	csp_solve_cplex.cpp	Tau-Argus-UPC\csp_nf\src\1H2Dtables\	181	4079
7	csp_solve_dijkstra.cpp	Tau-Argus-UPC\csp_nf\src\1H2Dtables\	128	3538
8	csp_solve_pprn.cpp	Tau-Argus-UPC\csp_nf\src\1H2Dtables\	152	4109
9	csp_table1h2d.cpp	Tau-Argus-UPC\csp_nf\src\1H2Dtables\	186	4744
10	csp_table1H2D.h	Tau-Argus-UPC\csp_nf\src\1H2Dtables\	490	15325
11	c_qsort_comp.c	Tau-Argus-UPC\csp_nf\src\1H2Dtables\	16	317
12	c_qsort_comp.h	Tau-Argus-UPC\csp_nf\src\1H2Dtables\	18	306
13	**TOTAL**		3577	106787

C List of routines

1. csp_auditing1H2D(TABLE1H2D *htab, int ini, int fin)
2. csp_heur1H2D(TABLE1H2D *htab)
3. csp_allocateNF(int nnu, int nar, NFPro **pnf)
4. csp_freeNF(NFPro **pnf)
5. objcostNF(NFPro *nf)
6. show_infoarcs(NFPro *nf)
7. csp_create_array2d(ARRAY2D **a, int n1, int *n2, int initval)
8. csp_delete_array2d(ARRAY2D **a)
9. csp_allocateNF(int nnu, int nar, NFPro **pnfpro)
10. csp_freeNF(NFPro **nfpro)
11. csp_SolveNF_PPRN_0lb(NFPro *nfpro)
12. csp_SolveNF_CPLEXN(NFPro *nfpro, CPXENVptr env, CPXNETptr net, TYPE_CPLEX_PROBLEM type)
13. csp_SolveNF_Dijkstra(DSP *sppro)
14. objcostNF(NFPro *nfpro)
15. show_infoarcs(NFPro *nfpro)
16. csp_delete_cycle(CYCLE **cycle)
17. csp_create_cycle(CYCLE **cycle, int ini_max_card)
18. csp_get_card_cycle(CYCLE *cycle)
19. csp_add_arc_cycle(CYCLE *cycle, CYCLE_ELEM arc)
20. csp_empty_cycle(CYCLE *cycle)
21. csp_check_add_cycle(CYCLE *cycle, int narcs)
22. csp_put_array2d(ARRAY2D *a, int i1, int i2, int v)
23. csp_get_array2d(ARRAY2D *a, int i1, int i2)
24. csp_lower_bound(TABLE1H2D *htab)
25. csp_heur1H2D_01flows(TABLE1H2D* tab)
26. csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro)
27. csp_compute_new_status_of_arc(STATUS_CELL parent, STATUS_CELL current, STATUS_CELL *newstat)
28. csp_ini_dijkstra(TABLE1H2D *htab, NFPro *nfpro, DSP **psppro)

29. csp_set_primary_NF(TABLE1H2D *htab, NFPro *nfpro, DSP *sppro, int prim_arc)
30. csp_set_costsNF_v1(TABLE1H2D *htab, int prim_arc, double prot_req, CYCLE *cycles_primary_arc, NFPro *nfpro, DSP *sppro)
31. csp_set_costsNF_v2(TABLE1H2D *htab, int prim_arc, double prot_req, CYCLE *cycles_primary_arc, NFPro *nfpro, DSP *sppro)
32. csp_resetNF(TABLE1H2D *htab, NFPro *nfpro, CYCLE *cycles_primary_arc, int prim_arc)
33. csp_get_cycleNF(TABLE1H2D *htab, NFPro *nfpro, DSP *sppro, int prim_arc, CYCLE *cycle, double *pos_min_cycle, double *neg_min_cycle)
34. csp_heur1H2D_01flows(TABLE1H2D *htab)
35. csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro)
36. csp_get_cycleNF(TABLE1H2D *htab, NFPro *nfpro, DSP *sppro, int prim_arc, CYCLE *cycle, double *pos_min_cycle, double *neg_min_cycle)
37. csp_resetNF(TABLE1H2D *htab, NFPro *nfpro, CYCLE *cycles_primary_arc, int prim_arc)
38. csp_set_costsNF_v1(TABLE1H2D *htab, int prim_arc, double prot_req, CYCLE *cycles_primary_arc, NFPro *nfpro , DSP *sppro)
39. csp_set_costsNF_v2(TABLE1H2D *htab, int prim_arc, double prot_req, CYCLE *cycles_primary_arc, NFPro *nfpro , DSP *sppro)
40. csp_SolveNF_CPLEXN(NFPro *Ncp, CPXENVptr env, CPXNETptr net, TYPE_CPLEX_PROBLEM type)
41. csp_ini_dijkstra(TABLE1H2D *tab, NFPro *nf, DSP **psp)
42. csp_SolveNF_Dijkstra(DSP *sp)
43. csp_SolveNF_PPRN_0lb(NFPro *Ncp)
44. create_table1H2D(TABLE1H2D **ptab, int t, int* m, int n, int p)
45. delete_table1H2D(TABLE1H2D **ptab)
46. set_0_value_cells_permanent1H2D(TABLE1H2D *htab)
47. check_protection_levels_for_lbp1H2D(TABLE1H2D *htab)
48. sort_primarycells1H2D(TABLE1H2D *htab)
49. get_ntables1H2D(TABLE1H2D *htab)
50. get_npcells1H2D(TABLE1H2D *htab)
51. get_nseccells1H2D(TABLE1H2D *htab)
52. get_nrows1H2D(TABLE1H2D *htab, int tab)
53. get_ncolumns1H2D(TABLE1H2D *htab, int tab)
54. get_cellvalue1H2D(TABLE1H2D *htab, int tab, int row, int column)

55. put_cellvalue1H2D(TABLE1H2D *htab, int tab, int row, int column, double value)
56. get_cellweight1H2D(TABLE1H2D *htab, int tab, int row, int column)
57. put_cellweight1H2D(TABLE1H2D *htab, int tab, int row, int column, double weight)
58. put_cellstatus1H2D(TABLE1H2D *htab, int tab, int row, int column, STATUS_CELL status)
59. get_cellarc1H2D(TABLE1H2D *htab, int tab, int row, int column)
60. put_cellarc1H2D(TABLE1H2D *htab, int tab, int row, int column, int pos_arc)
61. get_parent1H2D(TABLE1H2D *htab, int tab)
62. put_parent1H2D(TABLE1H2D *htab, int tab, int parent)
63. get_rowinparent1H2D(TABLE1H2D *htab, int tab)
64. put_rowinparent1H2D(TABLE1H2D *htab, int tab, int row)
65. put_index_of_primary1H2D(TABLE1H2D *htab, int tab, int row, int col, int pcell)
66. get_index_of_primary1H2D(TABLE1H2D *htab, int tab, int row, int col)
67. get_sorted_pcell1H2D(TABLE1H2D *htab, int pcell, int *pt, int *pi, int*pj, double *lpl, double *upl)
68. get_pcell1H2D(TABLE1H2D *htab, int pcell, int *pt, int *pi, int*pj, double *lpl, double *upl)
69. put_pcell1H2D(TABLE1H2D *htab, int pcell, int pt, int pi, int pj, double plpl, double pupl)
70. put_protlev_le_cellvalue1H2D(TABLE1H2D *htab, bool boolean)
71. qsort(void *base, size_t nmemb, size_t size, C_CMP cmp)
72. get_sorted_pcelllpl1H2D(TABLE1H2D *htab, int pcell)
73. put_sorted_pcelllpl1H2D(TABLE1H2D *htab, int pcell, double lpl)
74. get_pcelllpl1H2D(TABLE1H2D *htab, int pcell)
75. put_pcelllpl1H2D(TABLE1H2D *htab, int pcell, double lpl)
76. get_sorted_pcellupl1H2D(TABLE1H2D *htab, int pcell)
77. put_sorted_pcellupl1H2D(TABLE1H2D *htab, int pcell, double upl)
78. get_pcellupl1H2D(TABLE1H2D *htab, int pcell)
79. put_pcellupl1H2D(TABLE1H2D *htab, int pcell, double upl)
80. put_comp_lowbound1H2D(TABLE1H2D *htab, bool comp_lb)
81. put_use_lowbound1H2D(TABLE1H2D *htab, bool use_lb)
82. put_typemorder1H2D(TABLE1H2D *htab, TYPE_MERIT_ORDER morder)
83. put_solver1H2D(TABLE1H2D *htab, SOLVER solver)

84. put_cost_type1H2D(TABLE1H2D *htab, COST_TYPE cost_type)
85. get_lowerbound1H2D(TABLE1H2D *htab)
86. put_lowerbound1H2D(TABLE1H2D *htab, double lb)
87. get_numnfproblems1H2D(TABLE1H2D *htab)
88. put_numnfproblems1H2D(TABLE1H2D *htab, int nnf)
89. put_cells_0_permanent1H2D(TABLE1H2D *htab, bool setperm)
90. comp_ascendent(const void *a, const void *b)
91. comp_descendent(const void *a, const void *b)
92. comp_ascendent(const void *, const void *)
93. comp_descendent(const void *, const void *)

D Routines description

Routine Name	csp_auditing1H2D(TABLE1H2D *htab, int ini, int fin)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_auditing1H2D.cpp
Routine Size	182 Line(s)
Routine Comment	
	Auditing for 1H2D CSP. NOTE: it is not an efficient procedure, and just used for testing the solution reported by the protection heuristic Input parameters: ini: initial primary cell to audit (according to non-sorted order) fin: final primary cell to audit (according to non-sorted order) Input/output parameters: tab: table to be audited, previously protected by a call to csp_heur1H2D Returns: 0: if everything was fine -1: if not enough memory -2: if ini or fin out of bounds -3: problems with cplex license (not applicable) -4: error creating cplex network object (not applicable) -5: error solving problem
Parent Routines	
	• put_cells_0_permanent1H2D(TABLE1H2D *htab, bool setperm)
Child Routines	
	• csp_freenf(nfpro **pnf) • csp_freenf(nfpro **nfpro) • csp_solvenf_pprn_0lb(nfpro *nfpro) • csp_solvenf_cplexn(nfpro *nfpro, cpxenvptr env, cpxnetptr net, type_cplex_problem type) • csp_ininfproblem(table1h2d *htab, nfpro **pnfpro) • csp_ininfproblem(table1h2d *htab, nfpro **pnfpro) • csp_solvenf_cplexn(nfpro *ncp, cpxenvptr env, cpxnetptr net, type_cplex_problem type) • csp_solvenf_pprn_0lb(nfpro *ncp) • get_npcells1h2d(table1h2d *htab) • get_cellvalue1h2d(table1h2d *htab, int tab, int row, int column) • get_cellarc1h2d(table1h2d *htab, int tab, int row, int column) • get_pcell1h2d(table1h2d *htab, int pcell, int *pt, int *pi, int*pj, double *lpl, double *upl) • get_pcelllpl1h2d(table1h2d *htab, int pcell) • put_pcelllpl1h2d(table1h2d *htab, int pcell, double lpl) • get_pcellupl1h2d(table1h2d *htab, int pcell) • put_pcellupl1h2d(table1h2d *htab, int pcell, double upl)

Routine Name	csp_heur1H2D(TABLE1H2D *htab)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.cpp
Routine Size	41 Line(s)
Routine Comment	
	Heuristic for 1H2D CSP Input/output parameters: htab: table to be protected Returns: -51: error sorting primary cells by merit order -53: code compiled without CPLEX library; you can not use CPLEX -54: code compiled without CPLEX library; lower bounding procedure not available -55: lower bound solution can not be used if not computed other: error code returned by the heuristic (0 if everything goes fine)
Parent Routines	
	• put_cells_0_permanent1H2D(TABLE1H2D *htab, bool setperm)
Child Routines	
	• csp_heur1h2d_01flows(table1h2d* tab) • csp_heur1h2d_01flows(table1h2d *htab) • set_0_value_cells_permanent1h2d(table1h2d *htab) • check_protection_levels_for_lbp1h2d(table1h2d *htab) • sort_primarycells1h2d(table1h2d *htab)

Routine Name	csp_allocateNF(int nnu, int nar, NFPro **pnf)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.cpp
Routine Size	42 Line(s)
Routine Comment	
	Allocates network flow problem Input/Output parameters: nnu: number of nodes nar: number of arcs NFPro **pnf : NF structure Returns: 0: if everything was fine -1: if not enough memory
Parent Routines	
	• csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro)
Child Routines	
	• csp_freenf(nfpro **pnf) • csp_freenf(nfpro **nfpro)

Routine Name	csp_freeNF(NFPro **pnf)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.cpp
Routine Size	27 Line(s)
Routine Comment	
	Frees the allocated memory for the network flow problem Input/Output parameters: NFPro pnf : ** to the data structure of problem
Parent Routines	
	• csp_auditing1H2D(TABLE1H2D *htab, int ini, int fin) • csp_allocateNF(int nnu, int nar, NFPro **pnf) • csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	objcostNF(NFPro *nf)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.cpp
Routine Size	10 Line(s)
Routine Comment	
	print objective function cost of flows in nf
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	show_infoarcs(NFPro *nf)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.cpp
Routine Size	14 Line(s)
Routine Comment	
	show some information about arcs of a NF problem
Parent Routines	
Child Routines	

Routine Name	csp_create_array2d(ARRAY2D **a, int n1, int *n2, int initval)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.cpp
Routine Size	34 Line(s)
Routine Comment	
	Create array2d with variable length in secod dimension Input/Output parameters: a: ** to array2d structure n1: first dimension n2: vector of values of second dimension for each first dimension initval: initialization value Returns: 0: if everything was fine -1: if not enough memory
Parent Routines	
	• csp_check_add_cycle(CYCLE *cycle, int narcs) • csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro)
Child Routines	
	• csp_delete_array2d(array2d **a)

Routine Name	csp_delete_array2d(ARRAY2D **a)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.cpp
Routine Size	15 Line(s)
Routine Comment	
	Deletes an array2d Input/Output parameters: a: ** to the array2d
Parent Routines	
	• csp_create_array2d(ARRAY2D **a, int n1, int *n2, int initval) • csp_check_add_cycle(CYCLE *cycle, int narcs) • csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro)
Child Routines	

Routine Name	csp_allocateNF(int nnu, int nar, NFPro **pnfpro)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	3 Line(s)
Routine Comment	
Parent Routines	
	• csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro)
Child Routines	

Routine Name	csp_freeNF(NFPro **nfpro)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	4 Line(s)
Routine Comment	
Parent Routines	
	• csp_auditing1H2D(TABLE1H2D *htab, int ini, int fin) • csp_allocateNF(int nnu, int nar, NFPro **pnf) • csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	csp_SolveNF_PPRN_0lb(NFPro *nfpro)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	5 Line(s)
Routine Comment	
Parent Routines	
	• csp_auditing1H2D(TABLE1H2D *htab, int ini, int fin) • csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	csp_SolveNF_CPLEXN(NFPro *nfpro, CPXENVptr env, CPXNETptr net, TYPE_CPLEX_PROBLEM type)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	5 Line(s)
Routine Comment	
Parent Routines	
	• csp_auditing1H2D(TABLE1H2D *htab, int ini, int fin) • csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	csp_SolveNF_Dijkstra(DSP *sppro)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	3 Line(s)
Routine Comment	
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	objcostNF(NFPro *nfpro)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	1 Line(s)
Routine Comment	
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	show_infoarcs(NFPro *nfpro)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	18 Line(s)
Routine Comment	
Parent Routines	
Child Routines	

Routine Name	csp_delete_cycle(CYCLE **cycle)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	8 Line(s)
Routine Comment	
Parent Routines	
	• csp_create_cycle(CYCLE **cycle, int ini_max_card) • csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	csp_create_cycle(CYCLE **cycle, int ini_max_card)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	13 Line(s)
Routine Comment	
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	
	• csp_delete_cycle(cycle **cycle)

Routine Name	csp_get_card_cycle(CYCLE *cycle)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	12 Line(s)
Routine Comment	
	get cardinality of cycle
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab) • csp_resetNF(TABLE1H2D *htab, NFPro *nfpro, CYCLE *cycles_primary_arc, int prim_arc) • csp_set_costsNF_v1(TABLE1H2D *htab, int prim_arc, double prot_req, CYCLE *cycles_primary_arc, NFPro *nfpro , DSP *sppro) • csp_set_costsNF_v2(TABLE1H2D *htab, int prim_arc, double prot_req, CYCLE *cycles_primary_arc, NFPro *nfpro , DSP *sppro)
Child Routines	

Routine Name	csp_add_arc_cycle(CYCLE *cycle, CYCLE_ELEM arc)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	8 Line(s)
Routine Comment	
	add arc to cycle without checking bounds; a prior call to csp_check_add_cycle() is assumed
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab) • csp_get_cycleNF(TABLE1H2D *htab, NFPro *nfpro, DSP *sp-pro, int prim_arc, CYCLE *cycle, double *pos_min_cycle, double *neg_min_cycle)
Child Routines	
	• csp_check_add_cycle(cycle *cycle, int narcs)

Routine Name	csp_empty_cycle(CYCLE *cycle)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	6 Line(s)
Routine Comment	
	empty cycle
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab) • csp_get_cycleNF(TABLE1H2D *htab, NFPro *nfpro, DSP *sp-pro, int prim_arc, CYCLE *cycle, double *pos_min_cycle, double *neg_min_cycle)
Child Routines	

Routine Name	csp_check_add_cycle(CYCLE *cycle, int narcs)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	33 Line(s)
Routine Comment	
	check if narcs can be added to cycle without exceeding its current storage; if not, it reallocates the cycle. returns 0 if everything goes fine returns -1 if not enough memory for reallocation
Parent Routines	
	• csp_add_arc_cycle(CYCLE *cycle, CYCLE_ELEM arc) • csp_heur1H2D_01flows(TABLE1H2D *htab) • csp_get_cycleNF(TABLE1H2D *htab, NFPro *nfpro, DSP *sp-pro, int prim_arc, CYCLE *cycle, double *pos_min_cycle, double *neg_min_cycle)
Child Routines	
	• csp_create_array2d(array2d **a, int n1, int *n2, int initval) • csp_delete_array2d(array2d **a)

Routine Name	csp_put_array2d(ARRAY2D *a, int i1, int i2, int v)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	7 Line(s)
Routine Comment	
	sets v in position (i1,i2)
Parent Routines	
	• csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro)
Child Routines	

Routine Name	csp_get_array2d(ARRAY2D *a, int i1, int i2)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	9 Line(s)
Routine Comment	
	returns value of position (i1,i2)
Parent Routines	
	• csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro)
Child Routines	

Routine Name	csp_lower_bound(TABLE1H2D *htab)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	6 Line(s)
Routine Comment	
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	csp_heur1H2D_01flows(TABLE1H2D* tab)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	2 Line(s)
Routine Comment	
Parent Routines	
	• csp_heur1H2D(TABLE1H2D *htab)
Child Routines	

Routine Name	csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	1 Line(s)
Routine Comment	
Parent Routines	
	• csp_auditing1H2D(TABLE1H2D *htab, int ini, int fin) • csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	csp_compute_new_status_of_arc(STATUS_CELL parent, STATUS_CELL current, STATUS_CELL *newstat)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	3 Line(s)
Routine Comment	
Parent Routines	
	• csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro)
Child Routines	

Routine Name	csp_ini_dijkstra(TABLE1H2D *htab, NFPro *nfpro, DSP **psppro)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	1 Line(s)
Routine Comment	
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	csp_set_primary_NF(TABLE1H2D *htab, NFPro *nfpro, DSP *sppro, int prim_arc)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	1 Line(s)
Routine Comment	
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab) • csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro)
Child Routines	

Routine Name	csp_set_costsNF_v1(TABLE1H2D *htab, int prim_arc, double prot_req, CYCLE *cycles_primary_arc, NFPro *nfpro, DSP *sppro)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	3 Line(s)
Routine Comment	
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	csp_set_costsNF_v2(TABLE1H2D *htab, int prim_arc, double prot_req, CYCLE *cycles_primary_arc, NFPro *nfpro, DSP *sppro)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	3 Line(s)
Routine Comment	
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	csp_resetNF(TABLE1H2D *htab, NFPro *nfpro, CYCLE *cycles_primary_arc, int prim_arc)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	2 Line(s)
Routine Comment	
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	csp_get_cycleNF(TABLE1H2D *htab, NFPro *nfpro, DSP *sppro, int prim_arc, CYCLE *cycle, double *pos_min_cycle, double *neg_min_cycle)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D.h
Routine Size	4 Line(s)
Routine Comment	
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	csp_heur1H2D_01flows(TABLE1H2D *htab)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D_01flows.cpp
Routine Size	430 Line(s)
Routine Comment	
	Shortest-paths (01-flows) based heuristic for 1H2D CSP Input/output parameter: htab: table to be protected (input); protected table (output) Returns: 0: if everything was fine -1: if not enough memory -2: wrong table (same cell with different values in two subtables) -3: wrong table (same cell with different status in two subtables) -4: wrong table (same cell with different protection levels in two subtables) -5: if problem detected as infeasible (no solution was found) -6: error solving NF problem -7: problems with cplex license -8: error creating cplex network object -9: error in lower bounding procedure
Parent Routines	• csp_heur1H2D(TABLE1H2D *htab)

Child Routines	
	• csp_freenf(nfpro **pnf) • objcostnf(nfpro *nf) • csp_freenf(nfpro **nfpro) • csp_solvenf_pprn_0lb(nfpro *nfpro) • csp_solvenf_cplexn(nfpro *nfpro, cpxenvptr env, cpxnetptr net, type_cplex_problem type) • csp_solvenf_dijkstra(dsp *sppro) • objcostnf(nfpro *nfpro) • csp_delete_cycle(cycle **cycle) • csp_create_cycle(cycle **cycle, int ini_max_card) • csp_get_card_cycle(cycle *cycle) • csp_add_arc_cycle(cycle *cycle, cycle_elem arc) • csp_empty_cycle(cycle *cycle) • csp_check_add_cycle(cycle *cycle, int narcs) • csp_lower_bound(table1h2d *htab) • csp_ininfproblem(table1h2d *htab, nfpro **pnfpro) • csp_ini_dijkstra(table1h2d *htab, nfpro *nfpro, dsp **psppro) • csp_set_primary_nf(table1h2d *htab, nfpro *nfpro, dsp *sppro, int prim_arc) • csp_set_costsnf_v1(table1h2d *htab, int prim_arc, double prot_req, cycle *cycles_primary_arc, nfpro *nfpro, dsp *sppro) • csp_set_costsnf_v2(table1h2d *htab, int prim_arc, double prot_req, cycle *cycles_primary_arc, nfpro *nfpro, dsp *sppro) • csp_resetnf(table1h2d *htab, nfpro *nfpro, cycle *cycles_primary_arc, int prim_arc) • csp_get_cyclenf(table1h2d *htab, nfpro *nfpro, dsp *sppro, int prim_arc, cycle *cycle, double *pos_min_cycle, double *neg_min_cycle) • csp_ininfproblem(table1h2d *htab, nfpro **pnfpro) • csp_get_cyclenf(table1h2d *htab, nfpro *nfpro, dsp *sppro, int prim_arc, cycle *cycle, double *pos_min_cycle, double *neg_min_cycle) • csp_resetnf(table1h2d *htab, nfpro *nfpro, cycle *cycles_primary_arc, int prim_arc) • csp_set_costsnf_v1(table1h2d *htab, int prim_arc, double prot_req, cycle *cycles_primary_arc, nfpro *nfpro , dsp *sppro) • csp_set_costsnf_v2(table1h2d *htab, int prim_arc, double prot_req, cycle *cycles_primary_arc, nfpro *nfpro , dsp *sppro) • csp_solvenf_cplexn(nfpro *ncp, cpxenvptr env, cpxnetptr net, type_cplex_problem type) • csp_ini_dijkstra(table1h2d *tab, nfpro *nf, dsp **psp) • csp_solvenf_dijkstra(dsp *sp) • csp_solvenf_pprn_0lb(nfpro *ncp) • get_npcells1h2d(table1h2d *htab) • put_cellstatus1h2d(table1h2d *htab, int tab, int row, int column, status_cell status)

- get_cellarc1h2d(table1h2d *htab, int tab, int row, int column)
- get_index_of_primary1h2d(table1h2d *htab, int tab, int row, int col)
- get_sorted_pcell1h2d(table1h2d *htab, int pcell, int *pt, int *pi, int*pj, double *lpl, double *upl)
- get_sorted_pcelllpl1h2d(table1h2d *htab, int pcell)
- put_sorted_pcelllpl1h2d(table1h2d *htab, int pcell, double lpl)
- get_pcelllpl1h2d(table1h2d *htab, int pcell)
- put_pcelllpl1h2d(table1h2d *htab, int pcell, double lpl)
- get_sorted_pcellupl1h2d(table1h2d *htab, int pcell)
- put_sorted_pcellupl1h2d(table1h2d *htab, int pcell, double upl)
- get_pcellupl1h2d(table1h2d *htab, int pcell)
- put_pcellupl1h2d(table1h2d *htab, int pcell, double upl)
- get_lowerbound1h2d(table1h2d *htab)
- get_numnfproblems1h2d(table1h2d *htab)
- put_numnfproblems1h2d(table1h2d *htab, int nnf)

Routine Name	csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D_01flows.cpp
Routine Size	521 Line(s)
Routine Comment	
	Defines the topology of the network, default lower and upper bounds of flows, and initializes injections to 0. Also sets information of the interrelations cell-arcs: values, weights, status, list of cells associated to arcs, arc associated to cell. Input parameters: TABLE1H2D *htab : Data structure of the CSP problem Output parameters: NFPro **nfpro : current NF problem Returns: 0: if everything was fine -1: if not enough memory to create network -2: wrong table (same cell with different values in two subtables) -3: wrong table (same cell with different status in two subtables) -4: wrong table (same cell with different protection levels in two subtables)
Parent Routines	
	• csp_auditing1H2D(TABLE1H2D *htab, int ini, int fin) • csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	
	• csp_allocatenf(int nnu, int nar, nfpro **pnf) • csp_create_array2d(array2d **a, int n1, int *n2, int initval) • csp_delete_array2d(array2d **a) • csp_allocatenf(int nnu, int nar, nfpro **pnfpro) • csp_put_array2d(array2d *a, int i1, int i2, int v) • csp_get_array2d(array2d *a, int i1, int i2) • csp_compute_new_status_of_arc(status_cell parent, status_cell current, status_cell *newstat) • csp_set_primary_nf(table1h2d *htab, nfpro *nfpro, dsp *sppro, int prim_arc) • get_ntables1h2d(table1h2d *htab) • get_nrows1h2d(table1h2d *htab, int tab) • get_ncolumns1h2d(table1h2d *htab, int tab) • get_cellvalue1h2d(table1h2d *htab, int tab, int row, int column) • get_cellweight1h2d(table1h2d *htab, int tab, int row, int column) • get_cellarc1h2d(table1h2d *htab, int tab, int row, int column) • put_cellarc1h2d(table1h2d *htab, int tab, int row, int column, int pos_arc) • get_parent1h2d(table1h2d *htab, int tab) • get_rowinparent1h2d(table1h2d *htab, int tab) • get_index_of_primary1h2d(table1h2d *htab, int tab, int row, int col) • get_pcell1h2d(table1h2d *htab, int pcell, int *pt, int *pi, int*pj, double *lpl, double *upl)

Routine Name	csp_get_cycleNF(TABLE1H2D *htab, NFPro *nfpro, DSP *sppro, int prim_arc, CYCLE *cycle, double *pos_min_cycle, double *neg_min_cycle)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D_01flows.cpp
Routine Size	62 Line(s)
Routine Comment	
	Finds the cycle in the current optimal solution of NF and the minimum capacities (protection) of the arcs in either positive and negative sense Input parameters: htab : hierarchical table nfpro : NF problem sppro: shortest path problem prim_arc:primary arc in network Output parameters: cycle: arcs in cycle. double *pos_min_cycle : minimum a_ij of sense POS in the cycle (including primary arc) double *neg_min_cycle : minimum a_ij of sense NEG in the cycle (including primary arc) Returns: 0: if everything was fine -1: if not enough memory
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	
	• csp_add_arc_cycle(cycle *cycle, cycle_elem arc) • csp_empty_cycle(cycle *cycle) • csp_check_add_cycle(cycle *cycle, int narcs)

Routine Name	csp_resetNF(TABLE1H2D *htab, NFPro *nfpro, CYCLE *cycles_primary_arc, int prim_arc)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D_01flows.cpp
Routine Size	35 Line(s)
Routine Comment	
	reset values of primary arc in nfpro, preparing network for next primary only done if dijkstra not used (since it doesn't makes use of injections and capacities) Input parameters: htab: hierarchical table cycles_primary_arc: arcs in cycles of this primary arc prim_arc: primary arc Input/Output parameters: nfpro: NF problem
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	
	• csp_get_card_cycle(cycle *cycle)

Routine Name	csp_set_costsNF_v1(TABLE1H2D *htab, int prim_arc, double prot_req, CYCLE *cycles_primary_arc, NFPro *nfpro , DSP *sppro)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D_01flows.cpp
Routine Size	124 Line(s)
Routine Comment	
	Set costs of NF problem, using a stratification similar to that suggested by Cox. It is an accurate stratification, but expensive to compute Input parameters: htab: hierarchical table prim_arc: primary arc prot_req : Protection required in current N.F. prob. cycles_primary_arc : arcs in cycles needed to protect prim_arc
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	
	• csp_get_card_cycle(cycle *cycle)

Routine Name	csp_set_costsNF_v2(TABLE1H2D *htab, int prim_arc, double prot_req, CYCLE *cycles_primary_arc, NFPro *nfpro , DSP *sppro)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_heur1H2D_01flows.cpp
Routine Size	88 Line(s)
Routine Comment	
	Set costs of NF problem, using an alternative stratification where a value BIGCOST ¿ maxsumS2 is used. We consider BIGCOST= sumfor all t (4*a_m[t]1,n[t]), which always satisfies the above property. This cost stratification is not not so accurate as that of csp_set_costsNF_v1, but cheaper to compute. Input parameters: htab: hierarchical table prim_arc: primary arc prot_req : Protection required in current N.F. prob. cycles_primary_arc : arcs in cycles needed to protect prim_arc
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	
	• csp_get_card_cycle(cycle *cycle)

Routine Name	csp_SolveNF_CPLEXN(NFPro *Ncp, CPXENVptr env, CPXNETptr net, TYPE_CPLEX_PROBLEM type)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_solve_cplex.cpp
Routine Size	175 Line(s)
Routine Comment	
	Solves the network flow problem with CPLEX Input parameters: NFPro Ncp: NF problem env: cplex environment net: cplex network problem type: solve the problem as NET or LP (LP means extra work to copy the network model to a LP, but seems to be required to avoid problems during auditing). The LP model is created and destroyed within the routine. Returns: 0: if everything was fine -1: if not enough memory -2: error managing cplex network or lp object (returned by cplex) -3: error solving or recovering the solution of cplex network or lpobject (returned by cplex) -4: infeasible problem -5: unbounded problem -6: infeasible or unbounded problem -7: error creating lp object
Parent Routines	
	• csp_auditing1H2D(TABLE1H2D *htab, int ini, int fin) • csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	csp_ini_dijkstra(TABLE1H2D *tab, NFPro *nf, DSP **psp)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_solve_dijkstra.cpp
Routine Size	92 Line(s)
Routine Comment	
	Allocates space for the shortest-path structure and sets Dsp-¿mnk, mnl, car of permanent cells and node adjacencies for the Dijkstra algorithm Input parameters: TABLE1H2D *tab : 1H2D table NFPro *nf : NF problem Input/Output parameters: DSP **psp : pointer to shortest path structure Returns: 0: if everything was fine -1: if not enough memory to create dijkstra structure
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	csp_SolveNF_Dijkstra(DSP *sp)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_solve_dijkstra.cpp
Routine Size	25 Line(s)
Routine Comment	
	Solves the network flow (shorthest path) problem Input parameters: DSP *sp : Dijkstra structure, already updated Returns: 0: if everything was fine -5: if error in solving shortest path problem
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	csp_SolveNF_PPRN_0lb(NFPro *Ncp)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_solve_pprn.cpp
Routine Size	141 Line(s)
Routine Comment	
	Solves the network flow problem through a change of variables with zero lower bounds. Input parameters: NFPro Ncp : NF problem Returns: 0: if everything was fine -1: if not enough memory -2: if infeasible problem -3: if return status in pprn2 other than optimal or infeasible or unbounded -4: if unbounded problem
Parent Routines	
	• csp_auditing1H2D(TABLE1H2D *htab, int ini, int fin) • csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	create_table1H2D(TABLE1H2D **ptab, int t, int* m, int n, int p)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1h2d.cpp
Routine Size	84 Line(s)
Routine Comment	
	creates and initializes 1H2D table of t 2Dtables, m[1..t] rows, n columns and p primaries returns 0 if everything goes fine return -1 if not enough memory
Parent Routines	
	• put_protlev_le_cellvalue1H2D(TABLE1H2D *htab, bool boolean)
Child Routines	
	• delete_table1h2d(table1h2d **ptab) • sort_primarycells1h2d(table1h2d *htab)

Routine Name	delete_table1H2D(TABLE1H2D **ptab)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1h2d.cpp
Routine Size	15 Line(s)
Routine Comment	
	deletes a non-empty table
Parent Routines	
	• create_table1H2D(TABLE1H2D **ptab, int t, int* m, int n, int p) • put_protlev_le_cellvalue1H2D(TABLE1H2D *htab, bool boolean)
Child Routines	
	• get_ntables1h2d(table1h2d *htab)

Routine Name	set_0_value_cells_permanent1H2D(TABLE1H2D *htab)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1h2d.cpp
Routine Size	13 Line(s)
Routine Comment	
	mark cells with 0 value as permanent
Parent Routines	
	• csp_heur1H2D(TABLE1H2D *htab) • put_protlev_le_cellvalue1H2D(TABLE1H2D *htab, bool boolean)
Child Routines	
	• get_ntables1h2d(table1h2d *htab) • get_nrows1h2d(table1h2d *htab, int tab) • get_ncolumns1h2d(table1h2d *htab, int tab) • get_cellvalue1h2d(table1h2d *htab, int tab, int row, int column) • put_cellstatus1h2d(table1h2d *htab, int tab, int row, int column, status_cell status)

Routine Name	check_protection_levels_for_lbp1H2D(TABLE1H2D *htab)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1h2d.cpp
Routine Size	22 Line(s)
Routine Comment	
	check if some (upper) protection level is higher than cell value; this is required to know if the improved lower bounding procedure can or not be used
Parent Routines	
	• csp_heur1H2D(TABLE1H2D *htab) • qsort(void *base, size_t nmemb, size_t size, C_CMP cmp)
Child Routines	
	• get_npcells1h2d(table1h2d *htab) • get_cellvalue1h2d(table1h2d *htab, int tab, int row, int column) • get_pcell1h2d(table1h2d *htab, int pcell, int *pt, int *pi, int*pj, double *lpl, double *upl) • put_protlev_le_cellvalue1h2d(table1h2d *htab, bool boolean)

Routine Name	sort_primarycells1H2D(TABLE1H2D *htab)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1h2d.cpp
Routine Size	42 Line(s)
Routine Comment	
	sort primary cells by value according to the type of merit order returns 0 if everything goes fine return -1 if not enough memory for auxiliar workspace
Parent Routines	
	• csp_heur1H2D(TABLE1H2D *htab) • create_table1H2D(TABLE1H2D **ptab, int t, int* m, int n, int p) • qsort(void *base, size_t nmemb, size_t size, C_CMP cmp)
Child Routines	
	• get_npcells1h2d(table1h2d *htab) • get_cellvalue1h2d(table1h2d *htab, int tab, int row, int column) • get_pcell1h2d(table1h2d *htab, int pcell, int *pt, int *pi, int*pj, double *lpl, double *upl) • qsort(void *base, size_t nmemb, size_t size, c_cmp cmp)

Routine Name	get_ntables1H2D(TABLE1H2D *htab)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	6 Line(s)
Routine Comment	
	get number of 2D tables
Parent Routines	
	• csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro) • delete_table1H2D(TABLE1H2D **ptab) • set_0_value_cells_permanent1H2D(TABLE1H2D *htab)
Child Routines	

Routine Name	get_npcells1H2D(TABLE1H2D *htab)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	6 Line(s)
Routine Comment	
	get number of primary cells
Parent Routines	
	• csp_auditing1H2D(TABLE1H2D *htab, int ini, int fin) • csp_heur1H2D_01flows(TABLE1H2D *htab) • check_protection_levels_for_lbp1H2D(TABLE1H2D *htab) • sort_primarycells1H2D(TABLE1H2D *htab)
Child Routines	

Routine Name	get_nseccells1H2D(TABLE1H2D *htab)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	6 Line(s)
Routine Comment	
	get number of sencondary cells
Parent Routines	
Child Routines	

Routine Name	get_nrows1H2D(TABLE1H2D *htab, int tab)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	6 Line(s)
Routine Comment	
	get number of rows of 2Dtable tab
Parent Routines	
	• csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro) • set_0_value_cells_permanent1H2D(TABLE1H2D *htab)
Child Routines	

Routine Name	get_ncolumns1H2D(TABLE1H2D *htab, int tab)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	6 Line(s)
Routine Comment	
	get number of columns of 2Dtable tab
Parent Routines	
	• csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro) • set_0_value_cells_permanent1H2D(TABLE1H2D *htab)
Child Routines	

Routine Name	get_cellvalue1H2D(TABLE1H2D *htab, int tab, int row, int column)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	7 Line(s)
Routine Comment	
	get cell value without checking bounds
Parent Routines	
	• csp_auditing1H2D(TABLE1H2D *htab, int ini, int fin) • csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro) • set_0_value_cells_permanent1H2D(TABLE1H2D *htab) • check_protection_levels_for_lbp1H2D(TABLE1H2D *htab) • sort_primarycells1H2D(TABLE1H2D *htab)
Child Routines	

Routine Name	put_cellvalue1H2D(TABLE1H2D *htab, int tab, int row, int column, double value)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	7 Line(s)
Routine Comment	
	put cell value without checking bounds
Parent Routines	
Child Routines	

Routine Name	get_cellweight1H2D(TABLE1H2D *htab, int tab, int row, int column)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	7 Line(s)
Routine Comment	
	get cell weight without checking bounds
Parent Routines	
	• csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro)
Child Routines	

Routine Name	put_cellweight1H2D(TABLE1H2D *htab, int tab, int row, int column, double weight)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	14 Line(s)
Routine Comment	
	put cell weight without checking bounds
Parent Routines	
Child Routines	

Routine Name	put_cellstatus1H2D(TABLE1H2D *htab, int tab, int row, int column, STATUS_CELL status)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	9 Line(s)
Routine Comment	
	put cell status without checking bounds and updates number of secondary cells
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab) • set_0_value_cells_permanent1H2D(TABLE1H2D *htab) • put_pcell1H2D(TABLE1H2D *htab, int pcell, int pt, int pi, int pj, double plpl, double pupl)
Child Routines	

Routine Name	get_cellarc1H2D(TABLE1H2D *htab, int tab, int row, int column)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	8 Line(s)
Routine Comment	
	get position of positive arc in network associated to this cell without checking bounds
Parent Routines	
	• csp_auditing1H2D(TABLE1H2D *htab, int ini, int fin) • csp_heur1H2D_01flows(TABLE1H2D *htab) • csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro)
Child Routines	

Routine Name	put_cellarc1H2D(TABLE1H2D *htab, int tab, int row, int column, int pos_arc)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	9 Line(s)
Routine Comment	
	put position of positive arc in network associated to this cell without checking bounds
Parent Routines	
	• csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro)
Child Routines	

Routine Name	get_parent1H2D(TABLE1H2D *htab, int tab)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	6 Line(s)
Routine Comment	
	get table parent without checking bounds
Parent Routines	
	• csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro)
Child Routines	

Routine Name	put_parent1H2D(TABLE1H2D *htab, int tab, int parent)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	6 Line(s)
Routine Comment	
	put table parent without checking bounds
Parent Routines	
Child Routines	

Routine Name	get_rowinparent1H2D(TABLE1H2D *htab, int tab)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	6 Line(s)
Routine Comment	
	get row of parent decomposed in table tab without checking bounds
Parent Routines	
	• csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro)
Child Routines	

Routine Name	put_rowinparent1H2D(TABLE1H2D *htab, int tab, int row)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	6 Line(s)
Routine Comment	
	put row of parent table decomposed in table tab without checking bounds
Parent Routines	
Child Routines	

Routine Name	put_index_of_primary1H2D(TABLE1H2D *htab, int tab, int row, int col, int pcell)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	7 Line(s)
Routine Comment	
	put index of primary cell in array of primaries, without checking status
Parent Routines	
	• put_pcell1H2D(TABLE1H2D *htab, int pcell, int pt, int pi, int pj, double plpl, double pupl)
Child Routines	

Routine Name	get_index_of_primary1H2D(TABLE1H2D *htab, int tab, int row, int col)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	6 Line(s)
Routine Comment	
	get index of primary cell in array of primaries, without checking status
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab) • csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro)
Child Routines	

Routine Name	get_sorted_pcell1H2D(TABLE1H2D *htab, int pcell, int *pt, int *pi, int*pj, double *lpl, double *upl)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	14 Line(s)
Routine Comment	
	returns basic info of sorted primary cell pcell without checking bounds according to current merit order sort
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	get_pcell1H2D(TABLE1H2D *htab, int pcell, int *pt, int *pi, int*pj, double *lpl, double *upl)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	12 Line(s)
Routine Comment	
	returns basic info of primary cell pcell without checking bounds
Parent Routines	
	• csp_auditing1H2D(TABLE1H2D *htab, int ini, int fin) • csp_iniNFproblem(TABLE1H2D *htab, NFPro **pnfpro) • check_protection_levels_for_lbp1H2D(TABLE1H2D *htab) • sort_primarycells1H2D(TABLE1H2D *htab)
Child Routines	

Routine Name	put_pcell1H2D(TABLE1H2D *htab, int pcell, int pt, int pi, int pj, double plpl, double pupl)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	21 Line(s)
Routine Comment	
	put basic info of primary cell pcell without checking bounds status of cell is also updated
Parent Routines	
Child Routines	
	• put_cellstatus1h2d(table1h2d *htab, int tab, int row, int column, status_cell status) • put_index_of_primary1h2d(table1h2d *htab, int tab, int row, int col, int pcell)

Routine Name	put_protlev_le_cellvalue1H2D(TABLE1H2D *htab, bool boolean)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	17 Line(s)
Routine Comment	
	put if protection levels are less or equal than cell values
Parent Routines	
	• check_protection_levels_for_lbp1H2D(TABLE1H2D *htab)
Child Routines	
	• create_table1h2d(table1h2d **ptab, int t, int* m, int n, int p) • delete_table1h2d(table1h2d **ptab) • set_0_value_cells_permanent1h2d(table1h2d *htab)

Routine Name	qsort(void *base, size_t nmemb, size_t size, C_CMP cmp)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	7 Line(s)
Routine Comment	
	detect if (upper) protection levels are greater than cell value; this is useful for using or not the improved lower bounding procedure
Parent Routines	
	• sort_primarycells1H2D(TABLE1H2D *htab)
Child Routines	
	• check_protection_levels_for_lbp1h2d(table1h2d *htab) • sort_primarycells1h2d(table1h2d *htab)

Routine Name	get_sorted_pcelllpl1H2D(TABLE1H2D *htab, int pcell)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	7 Line(s)
Routine Comment	
	returns current provided lower protection level of primary cell pcell without checking bounds according to current merit order sort
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	put_sorted_pcelllpl1H2D(TABLE1H2D *htab, int pcell, double lpl)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	7 Line(s)
Routine Comment	
	put current provided lower protection level of primary cell pcell without checking bounds according to current merit order sort
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	get_pcelllpl1H2D(TABLE1H2D *htab, int pcell)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	7 Line(s)
Routine Comment	
	returns current provided lower protection level of primary cell pcell without checking bounds
Parent Routines	
	• csp_auditing1H2D(TABLE1H2D *htab, int ini, int fin) • csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	put_pcelllpl1H2D(TABLE1H2D *htab, int pcell, double lpl)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	7 Line(s)
Routine Comment	
	put current provided lower protection level of primary cell pcell without checking bounds
Parent Routines	
	• csp_auditing1H2D(TABLE1H2D *htab, int ini, int fin) • csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	get_sorted_pcellupl1H2D(TABLE1H2D *htab, int pcell)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	7 Line(s)
Routine Comment	
	returns current provided upper protection level of primary cell pcell without checking bounds according to current merit order sort
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	put_sorted_pcellupl1H2D(TABLE1H2D *htab, int pcell, double upl)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	7 Line(s)
Routine Comment	
	put current provided upper protection level of primary cell pcell without checking bounds according to current merit order sort
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	get_pcellupl1H2D(TABLE1H2D *htab, int pcell)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	7 Line(s)
Routine Comment	
	returns current provided upper protection level of primary cell pcell without checking bounds
Parent Routines	
	• csp_auditing1H2D(TABLE1H2D *htab, int ini, int fin) • csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	put_pcellupl1H2D(TABLE1H2D *htab, int pcell, double upl)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	13 Line(s)
Routine Comment	
	put current provided upper protection level of primary cell pcell without checking bounds
Parent Routines	
	• csp_auditing1H2D(TABLE1H2D *htab, int ini, int fin) • csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	put_comp_lowbound1H2D(TABLE1H2D *htab, bool comp_lb)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	12 Line(s)
Routine Comment	
	put if lower bound must be computed
Parent Routines	
Child Routines	

Routine Name	put_use_lowbound1H2D(TABLE1H2D *htab, bool use_lb)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	12 Line(s)
Routine Comment	
	put if lower bound solution must be used
Parent Routines	
Child Routines	

Routine Name	put_typemorder1H2D(TABLE1H2D *htab, TYPE_MERIT_ORDER morder)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	12 Line(s)
Routine Comment	
	put type of merit order
Parent Routines	
Child Routines	

Routine Name	put_solver1H2D(TABLE1H2D *htab, SOLVER solver)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	12 Line(s)
Routine Comment	
	put solver
Parent Routines	
Child Routines	

Routine Name	put_cost_type1H2D(TABLE1H2D *htab, COST_TYPE cost_type)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	6 Line(s)
Routine Comment	
	put type of cost stratification
Parent Routines	
Child Routines	

Routine Name	get_lowerbound1H2D(TABLE1H2D *htab)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	6 Line(s)
Routine Comment	
	get lower bound
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	put_lowerbound1H2D(TABLE1H2D *htab, double lb)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	6 Line(s)
Routine Comment	
	put lower bound
Parent Routines	
Child Routines	

Routine Name	get_numnfproblems1H2D(TABLE1H2D *htab)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	6 Line(s)
Routine Comment	
	get number of nf problems
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	put_numnfproblems1H2D(TABLE1H2D *htab, int nnf)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	12 Line(s)
Routine Comment	
	put number of nf problems
Parent Routines	
	• csp_heur1H2D_01flows(TABLE1H2D *htab)
Child Routines	

Routine Name	put_cells_0_permanent1H2D(TABLE1H2D *htab, bool setperm)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\csp_table1H2D.h
Routine Size	40 Line(s)
Routine Comment	
	put if 0 value cells must be set permanent
Parent Routines	
Child Routines	
	• csp_auditing1h2d(table1h2d *htab, int ini, int fin) • csp_heur1h2d(table1h2d *htab)

Routine Name	comp_ascendent(const void *a, const void *b)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\c_qsort_comp.c
Routine Size	5 Line(s)
Routine Comment	
Parent Routines	
Child Routines	

Routine Name	comp_descendent(const void *a, const void *b)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\c_qsort_comp.c
Routine Size	8 Line(s)
Routine Comment	
Parent Routines	
Child Routines	

Routine Name	comp_ascendent(const void *, const void *)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\c_qsort_comp.h
Routine Size	1 Line(s)
Routine Comment	
Parent Routines	
Child Routines	

Routine Name	comp_descendent(const void *, const void *)
Routine Location	Tau-Argus-UPC\csp_nf\src\1H2Dtables\c_qsort_comp.h
Routine Size	8 Line(s)
Routine Comment	
Parent Routines	
Child Routines	