PPRN 1.0 USER'S GUIDE

Jordi Castro
Statistics & Operations Research Dept.
UPC

DATE 9/94
DR 94/06

PPRN 1.0 USER'S GUIDE

Abstract: PPRN is a specialized code for solving multicommodity network problems considering additional linear side constraints on the flows on the arcs of the same or different commodities, with either linear or nonlinear objective functions. The package implements a specialized primal partitioning, which has proved to be really efficient in most tests performed. This document presents the main features of the package without paying attention to the underlying mathematical techniques. The document will describe mainly the way of linking the PPRN code with the user's applications, its input and output files and the different parameters that can be adjusted by the user depending on his particular problem.

Keywords: Multicommodity Network Flows, Network Simplex Methods, Nonlinear Optimization, Side Constraints, Numerical Software, Fortran Programming Language, C Programming Language.

1. Introduction.

PPRN is a new code for solving multicommodity network flow problems with linear side constraints. It is mainly written in Fortran-77, with the routines for the dynamic memory assignment coded in ANSI-C. These ANSI-C routines are the interface between the user application and the package. PPRN can be used as a stand-alone package (reading the data from an input file, and printing the result on an output file, with no other communication with the user), or as a subroutine, passing the definition of the problem and receiving the solution through parameters.

The problem dealt with by the code can be cast as:

$$\min_{X_1,X_2,\ldots,X_K} \quad f(X_1, X_2, \ldots, X_K) \tag{1}$$

$$\text{subj. to} \quad AX_k = R_k \qquad k = 1 \div K \tag{2}$$

$$\sum_{k=1}^{K} X_k \leq T \tag{3}$$

$$L \leq \sum_{k=1}^{K} L_k X_k \leq U \tag{4}$$

$$\underline{0} \leq X_k \leq \overline{X}_k \qquad k = 1 \div K \tag{5}$$

where $X_k \in \mathbb{R}^n$, (n: number of arcs) is the flow array for each commodity k ($k = 1 \div K$), K being the number of commodities of the problem, and f being a $\mathbb{R}^{K \times n} \rightarrow \mathbb{R}^1$ real valued function. $A \in \mathbb{R}^{m \times n}$ (m: number of nodes) is the arc–node incidence matrix. Equation (3) represents the mutual capacity constraints, where $T \in \mathbb{R}^n$. Linear side constraints are defined in (4) where $L_k \in \mathbb{R}^{p \times n}$, $k = 1 \div K$, and $L, U \in \mathbb{R}^p$ (p: number of side constraints). These side constraints can link flows on arcs of the same or different commodities. Constraints (5) are simple bounds on the flows, $\overline{X}_k \in \mathbb{R}^n, k = 1 \div K$ being the upper bounds.

The objective function (1) can be either linear or nonlinear. For the case of linear functions PPRN uses a specialized simplex code for multicommodity problems based on a primal partitioning algorithm [1,3,5,6]. When a nonlinear function has to be

minimized, it implements a multicommodity specialization of Murtagh and Saunders' strategy of dividing the set of variables into basic, superbasic and nonbasic [2,4,5,7], following some of the ideas of the Minos package developed by the same authors [8].

The structure of the network matrix A must be given by the user as a list of origen-destination nodes (O_a, D_a) for each arc a. This network is replicated for each commodity.

If the number of commodities is $K = 1$, PPRN will consider merely a single commodity network flow problem, and it will not take into account constraint (3).

The user may define as many linear side constraints (4) as necessary, even none ($p \geq 0$). For each side constraint, besides its structure, it must be given its lower and upper bounds. If the i-th side constraint is unbounded at its lower (upper) limit $l_i = -M$ ($u_i = +M$) will be set, M being an arbitrary large value defined by the user to be considered as infinity. For each nonzero element in the structure of the side constraints the coefficient value, the number of the side constraint, and the associated arc and commodity must be provided.

The current version of the PPRN code cannot deal with lower bounds other than zero in the variables (equation (5)). If some variable x_i is lower-bounded at a nonzero value ($\underline{x_i} \leq x_i \leq \overline{x_i}$) it suffices to just define a new variable $\hat{x}_i = x_i - \underline{x_i}$ and now $0 \leq \hat{x}_i \leq \overline{x_i} - \underline{x_i}$. Another, far less efficient, strategy would be to impose a side constraint like $\underline{x_i} \leq x_i$. If some arc has unlimited capacity then $\overline{x_i} = M$ must be set, M being the large value introduced before.

In the next sections we will present the main features of the package. First, the installation process of PPRN will be described. Secondly, the way of communicating with the code will be explained, followed by a description of the different parameters that can be tuned by the user. The next section will detail the files generated by PPRN when executing a model. The last two sections are devoted to the description of the different exit conditions of the package, and its possible extensions.

2. About installing PPRN.

PPRN was developed on a UNIX operating system, and the installation process will be described assuming its use. However, given its simplicity, it can be easily installed on any other system by merely translating the syntax of some commands.

The first thing to do is to copy all the source files to the directory where the package is to be installed. In alphabetical order, these files are:

`c_fortran.h`	`in_out.f`	`pprn.h`	`rutlexa.f`	`taxacio.f`
`constants.h`	`linsearch.f`	`pprn_c.c`	`rw_basis.f`	`trees.f`
`dirdescens.f`	`nolin.f`	`quasi_mi.f`	`sistemes.f`	`updates.f`
`etes.f`	`pprn.f`	`rtr_mi.f`	`strings.f`	`varis.f`

The "`.f`" files are coded in Fortran-77, the "`.c`" are written in ANSI-C, and the "`.h`" contain parameters, scalar data, and ANSI-C function prototype definitions which are included by the rest of files.

Besides of these files, there is a Makefile archive which should be used for installation when working on a UNIX system. The only thing to do would be to adjust some macros defined in the Makefile text, if necessary. These macros are:

- `CC`: ANSI-C compiler used.

- `F77`: Fortran-77 compiler used.
- `OPTSCLD_C`: compiling options for the ANSI-C compiler.
- `OPTSCLD_F`: compiling options for the Fortran-77 compiler.

Once these macros have been correctly defined, executing the command
will produce an object library called `libpprn.a`, ready for linking with the user's application.

If an operating system other than UNIX is being used, it will suffice to compile all the "`.f`" and "`.c`" source files, and, afterwards make the object library through the correct command(s) of the system.

IMPORTANT: In the UNIX operating system employed in the development of the code (Solaris SunOS) the Fortran-77 compiler adds two underscores (_) to all the routine and function names (one at the beginning and one at the end), while the ANSI-C compiler only adds one at the beginning. That is why at files `pprn_c.c` and `c_fortran.h` some name functions end with an "_" (to permit compatibility when linking C and Fortran modules). If your system does not have this discrepancy in the number of underscores, you should remove the last underscore of the name routines `pp_previ_alloc_`, `pp_get_dimensions1_`, `pp_get_dimensions2_`, `pp_get_exit_`, `pprn_cos_`, `pprn1_` and `pprn2_` of both files.

3. How to use PPRN.

Once the object library has been created, PPRN is ready to solve any multicommodity problem matching the formulation in (1–5). For the purpose of explaining the use of the code, the following easy example problem will be described:

EXAMPLE PROBLEM: Let us consider a 4-node, 5-arc and 2-commodity network as shown in Fig. 1. 18 units of commodity 1 and 20 units of commodity 2 must be transported from node N1 to node N4, through the arcs $x_{i,k},\ i = 1, \ldots, 5;\ k = 1, 2$. We will assume that the capacity of all the multicommodity arcs is 10 units ($x_{i,k} \leq 10\ i = 1, \ldots, 5;\ k = 1, 2$). The mutual capacity for all the arcs will be 15 units (thus the vector T of equation (3) will be $T^t = (15, 15, 15, 15, 15)$). And a side constraint will be considered: $x_{1,1} + x_{2,2} = 17$ (the flow of the first arc for the first commodity plus the flow of the second arc for the second commodity must be equal to 17 units).

Two different objective functions, the first linear and the second nonlinear,will be employed. The linear one is simply defined as

$$f_l(X_1, X_2) = C_1 X_1 + C_2 X_2 \tag{6}$$

where X_1 and X_2 are the vector flows for each commodity, and the vector linear costs are $C_1 = (1, 1, 1, 1, 1)^t$ and $C_2 = (2, 2, 2, 2, 2)^t$. The nonlinear objective function is the quadratic function:

$$f_n(X_1, X_2) = \sum_{k=1}^{2} \sum_{i=1}^{5} x_{i,k}^2 \tag{7}$$

Before starting to code the user application there are some important things that must be taken into account:

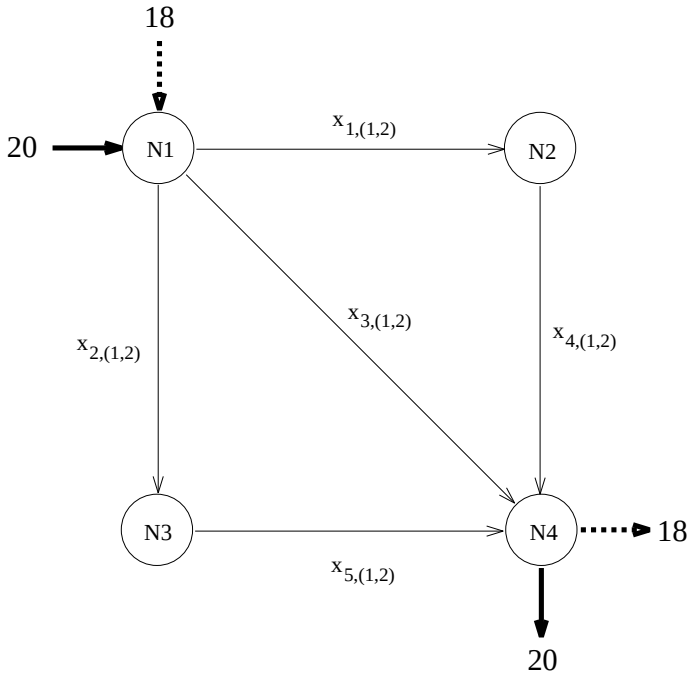

Fig.1. Network example problem.

• Whenever possible routine names beginning with "`pp`" should be avoided since this is the starting prefix of all the routines of the PPRN package (the same applies for the Fortran-77 "common" data structures). Otherwise, some conflicts could happen when creating the executable file.

• PPRN manages some files from the Fortran routines. Since the Fortran logical units required to work with files are global variables (not local to the routine), the user should not use those already employed by PPRN in order to avoid problems. These logical units are: 11,12,13,14,15 and 16.

• All the integer variables used by PPRN are assumed to be 4-byte long, whereas the real ones are 8-byte. All the integer and real parameters passed by the user to PPRN must conform to these sizes (declaring "`integer*4`" and "`real*8`" in Fortran files, or "`int`" and "`double`" in C archives[1]).

The following subsections will describe the parameter list of the objective function to be coded by the user (if the problem is nonlinear), and the two possible ways of using the package.

3.1. The objective function.

If the problem to be solved is nonlinear, the user must provide a codification of the objective function. Any name can be used for the routine, and the parameter list is as follows:

$$\text{objective\_function}(n, K, x, f, g, nstate)$$

The meaning of the parameters is:

Input Parameters:

- n: (integer) number of arcs of the single-commodity network.

[1] In some systems and compilers "`int`" and "`double`" can have different sizes.

- K: (integer) number of commodities.
- $x(0{:}n, K)$: (real) current point to evaluate the objective function. In this vector we have the flows ordered by commodities (first all the flows of the first commodity, then of the second commodity. and so on). Also for each commodity an extra flow value $x(0, k)$ is considered. This is the flow on an artificial arc (root arc) which is added automatically by PPRN. This flow is not to be used (it is best to forget that it exists, even though it must be declared).
- *nstate*: (integer) this variable can take the values 0, 1 or 2. It is used to know what kind of call it is. The first time that PPRN calls to the routine *nstate* will be 0. The remaining calls during the execution of the problem will have *nstate*=1. The last call, made when the optimum point has been reached, will have *nstate*=2.

Output Parameters:

- f: (real) the value of the objective function at the current iterate.
- $g(0{:}n, K)$: (real) the gradient of the objective function at the current iterate, that is, $g(i, k) = \partial f / \partial x(i, k)$. As with vector x, the positions $g(0, k)$ must not be used, and must be never modified.

When coding the objective function in Fortran, the correct heading of the routine would be:

```
subroutine any_name(n,k,x,f,g,nstate)
integer*4 n,k,nstate
real*8 x(0:n,k),f,g(0:n,k)
```

When coding it in C we would write:

```
void any_name(n,k,x,f,g,nstate)
int *n,*k,*nstate;
double *x,*f,*g;
```

The next program shows how a possible Fortran codification of the objective function (7) of the example put forward could be.

```
subroutine funobj (n,k,x,f,g,nstate)
integer*4 n,k,nstate
real*8 x(0:n,k),f,g(0:n,k)
integer*4 i,j
real*8 raux

f= 0.0d0
do i=1,k
  do j= 1,n
    raux= x(j,i)
    f= f+raux*raux
    g(j,i)= 2.0d0*raux
  end do
end do
```

```
        end
```

The C code for this objective function would be similar.

3.2. Passing the model through a file.

PPRN accepts two different ways of receiving the problem to be solved: through an input file, or via parameters. The first method will be described here, and the second one will be postponed for the next subsection. The communication between the user application and the PPRN package is made through two interface routines, called `pprn1` and `pprn2`, depending of the way the problem is sent.

Passing the problem definition through an input file is the easiest option, though not the most useful. The user merely has to communicate the name of the input file to PPRN, by making a call in its source code to routine `pprn1` of the package, and it will obtain the desired solution in an output file. PPRN will perform all the work related to the memory assignment of the data structures required for solving the problem, simplifying the user task. However, once the optimal solution has been found and the control is returned to the user application from routine `pprn1`, the only way to know the optimum achieved will be to look at the output file, and no information will be available during the run of the user application. This makes this working mode very close to a stand-alone version of the package.

The routine `pprn1` is written in ANSI-C, and its heading is:

```
void pprn1(int *ln, char *name, int *exit,
           void (f)(int*,int*,double*,double*,double*,int*))
```

All the parameters have been declared as a pointer to permit calling this routine either from C or Fortran coded user applications. The meaning of the parameters is the following:

Input parameters:

- `ln`: (integer escalar) denotes if the problem to be solved is linear (in this case `ln`=0) or nonlinear (`ln`$\neq$0).
- `name`: (string or character vector) this is the name of the input file that contains the problem specification.
- `f`: (function) this is the nonlinear objective function to be minimized. When the problem is merely linear this parameter can be removed if the user application is coded in Fortran, or set to a NULL pointer when coded in C.

Output parameter:

- `exit`: (integer escalar) the exit condition variable communicates to the user application the state of the package and the status of the solution found (if any) once PPRN has returned the control. A comprehensive list of the exit values will be presented below.

Some comments have to be made about these parameters. By default, for the "`name`" parameter, if the user only provides a name without extension, it will be assumed that the extension is "`.dat`". If the user provides a name with a "`.`", the substring at the right of this dot will be considered as an extension (if there are two or more "`.`" the first one found starting from the left is taken into account). The name of the file without extension is then employed to work with three more

files: the output, the basis, and the specification file. The output file is associated to the input file by using the same name with the extension "`.lst`" ("`.lst`" comes from "listing file"). A detailed description of the structure of the output file will be presented in the following sections. The basis file will store at the end of the execution (even though it is saved periodically during the runtime) the status of the optimization process. PPRN makes it possible to restart the execution taking as the initial point that stored in the basis file. The basis file takes the name of the input file, adding the extension "`.bas`". The specification file can be employed by the user to modify some default values or features of the package. This specification file is directly searched for by the PPRN package. If it is found, the default values will be modified. If it is not, the default values will be used and a warning message will appear in the output file (the "`.lst`" file) advising that the specification file was not found. The specification file looked for has the same name as the input file but with the "`.spe`" extension. A more comprehensive description of the specification file will be made below. To underline the main points, if the user gives, for instance, `name= "example"`, the code will consider the input file "`example.dat`", it will write the output file "`example.lst`" and the basis file "`example.bas`", and it will search for the specification file "`example.spe`". On the other hand, if the user gives `name= "example.a.b"`, the code will read the input file "`example.a.b`", it will write the files "`example.lst`" and "`example.bas`", and it will look for the specification file "`example.spe`".

It must be pointed out that because routine `pprn1` is written in C, the string variable `name` is supposed to finish with the null character "`\0`". When the user application is written in C, this is directly performed by the compiler. However, when using Fortran, this cannot be ensured, and sometimes it will be task of the user to add `char(0)` to the end of the string (e.g, the Fortran variable of the user application "`character*32 name`" could be initialized as "`name= 'example'//char(0)`"). If trouble is encountered, the best thing to do is to read in the reference manuals of your own system how to pass character variables from Fortran to C routines.

Once the routine `pprn1` has been introduced, the user application code for solving the example problem put forward in previous sections could be as follows (considering that it is written in Fortran by the user):

```
      implicit none
      external funobj
      character*80 strdat
      integer*4 nolin,iexit

      strdat= 'example'
      nolin= 1
      call pprn1(nolin,strdat,iexit,funobj)
      write(*,*) iexit
      end
```

In this case it is assumed that the problem is nonlinear (the variable `nolin` is initiliazed to a nonzero value), and the objective function is coded in an external routine called `funobj`. If the problem is merely linear, we will replace

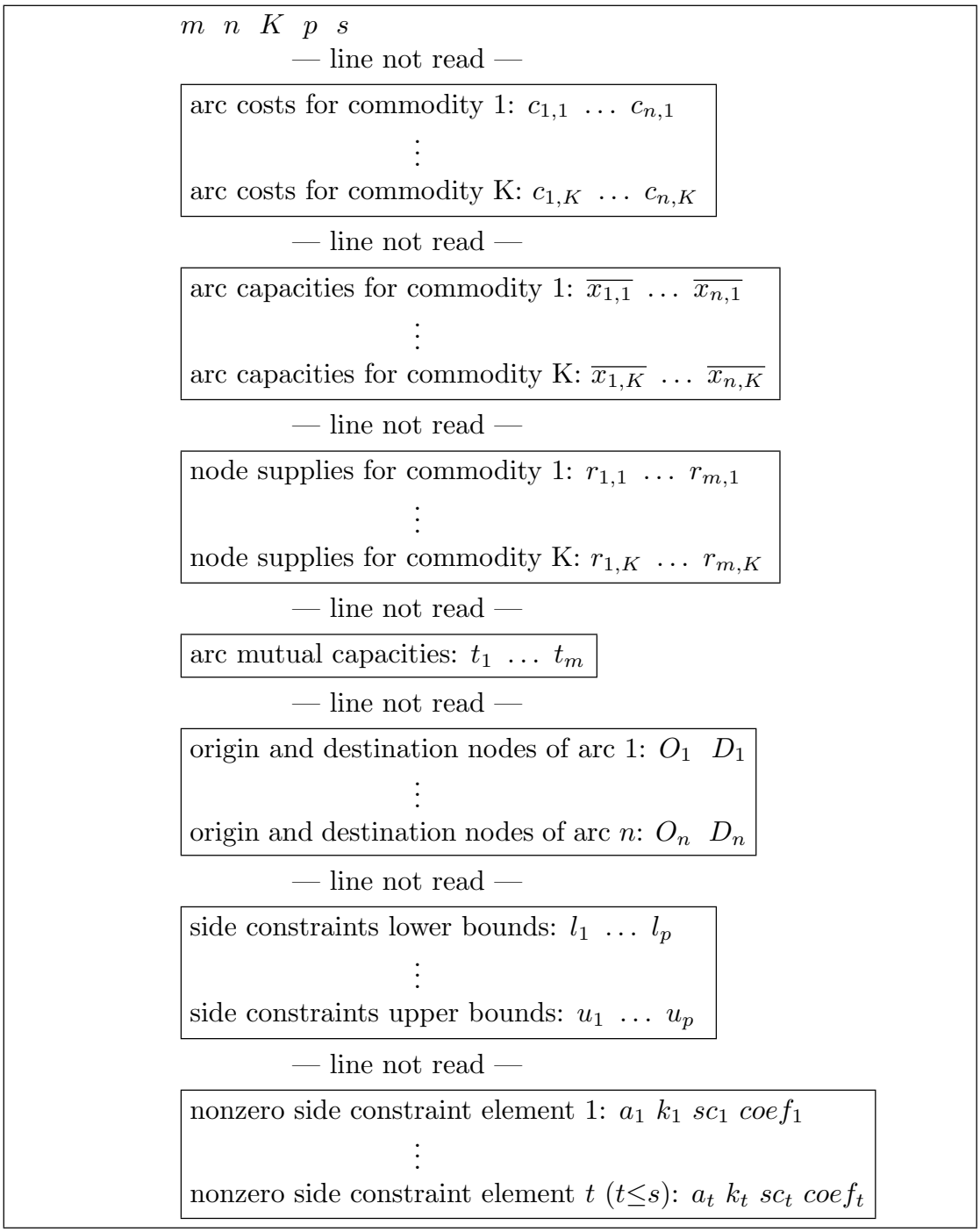

Fig.2. Structure of the input file.

"`nolin= 1`" by "`nolin= 0`", and the call to `pprn1` could be written as "`call pprn1(nolin,strdat,iexit)`". Once the control is returned to the user application, the value of the variable "`iexit`" is written. As already mentioned, this variable shows the status of the package and the solution found at the end of the execution. This exit status is also presented in the "`.lst`" file (output file) generated by PPRN. The different exit conditions will be described in more detail below.

The structure of the input file is still to be defined. In this file the user must provide first the dimensions of the problem and the different vectors and matrices required (arc costs, arc capacities, node supplies —or demands—, arc mutual capacities, network topology, and side constraints structure). The structure of this file is shown in Fig.2.

Looking at Fig.2, it can be seen that the first line of the file has the dimensions of the problem (m is the number of nodes, n the number of arcs, K the number of commodities, p the number of side constraints and s an upper bound to the number of nonzero elements in the side constraints). Like others in the rest of the file, the next line is not read by the program. These lines separate different sections of the input file, and the user can write any comment (without exceeding one line length).

Once the dimensions of the problem are known, PPRN will read the arc costs for the different commodities. For each commodity the user must provide a list of n costs separated by blanks (they can be in one or more lines). Costs for the next commodity have to start in a new line. This rule (for each commodity datum separated by blanks in one or more lines, and starting in a new line datum for the next commodity) will be followed in the rest of the file (for capacities, supplies etc.). The costs read will determine the objective function to be minimized if the problem to be solved is merely linear. If the problem is nonlinear the costs must be provided too, and will be used to find a "good" initial feasible point. This means that when the model allows it, the user can give some costs which are a linear approximation to the nonlinear function. Thus PPRN can exploit this information when finding the initial feasible point. More details can be found in [3,4].

The next section of the file has the arc capacities (in the same way as detailed above: first for the first commodity separated by blanks, and then for the next commodities, each one starting in a new line). These values correspond to the bounds in equation (5) of the formulation of the problem presented at the beginning of this document. The node supplies (or demands if the value is negative) for each commodity will be read in the following section (that is, the right hand side of equations (2) of the formulation of the problem). The vector T of equation (3) with the arc mutual capacities is the next vector found in the input file. The topology of the network matrix (of equation (2)) is given as n couples of origin-destination nodes. Finally, the information related to the side constraints (if $p > 0$) is given. First the lower and upper limits of the side constraints (vectors L and U of equation (4)) are read. Then, the structure of the side constraints must be fed. For each nonzero element (let us suppose there are t nonzero elements, where $t \leq s$) in matrices L_k of (4), a line of information will appear in the input file, indicating:

- a_i and k_i: arc and commodity associated with the i-th nonzero element (equivalent to giving the column in the matrix constraints).
- sc_i: side constraint associated with the i-th nonzero element (equivalent to giving the row in the matrix constraints).
- $coef_i$: coefficient associated with this nonzero element.

For instance, the input file associated with the example problem that is being considered in this document would be as follows:

```
4 5 2 1 2
ARC COSTS BY COMMODITIES
1.0 1.0 1.0 1.0 1.0
2.0 2.0 2.0 2.0 2.0
ARC CAPACITIES BY COMMODITIES
10.0 10.0 10.0 10.0 10.0
10.0 10.0 10.0 10.0 10.0
NODE INJECTIONS BY COMMODITIES
18.0 0.0 0.0 -18.0
20.0 0.0 0.0 -20.0
ARC MUTUAL CAPACITIES
15.0 15.0 15.0 15.0 15.0
NETWORK TOPOLOGY
1 2
1 3
1 4
2 4
3 4
LOWER AND UPPER BOUNDS FOR SIDE CONSTRAINTS
17.0
17.0
SIDE CONSTRAINTS STRUCTURE
1 1 1 1.0
2 2 1 1.0
```

As can be seen in this example file, the only side constraint of the problem is an equality one (the lower and upper limit are the same, 17.0). When one of these side constraints has no limit (lower or upper limit) then the user must use a big value that could be considered as infinity. By default this infinity value is 10^{32}. However, this value can be changed by the user in the specification file. Another important remark is that side constraints whose structure is $l_i \leq c^t x \leq +\infty$ are automatically transformed by PPRN to the equivalent one $-\infty \leq -c^t x \leq -l_i$, and they will be presented in the output file in this new form. This change is simply a question of implementation details.

3.3. Passing the model through parameters.

In the last section the way of using PPRN as a stand-alone package was presented. This was done by passing the problem definition and receiving the result through files. For this purpose the routine `pprn1` of the package was described. There is one more way of using the code, which requires a little extra work by the user but permits much more flexibility. The main difference between this way of working and the previous one is the transmission of the model to PPRN. In this case the model is sent through parameters, calling the second interface routine `pprn2` of the package, instead of through a file. This implies that the user must declare a list of vectors with enough dimension to store the model. PPRN will use these vectors (and others that it will assign dynamically) as workspace, and once the execution has been performed, it will store the optimal point in the solution flow vector, returning the control to

the user application. In this case the user can know the optimal solution without needing to look at the output file (the output file is also created in this case).

As `pprn1`, routine `pprn2` is coded in C. The heading of this routine is as follows:

```
void pprn2(int *ln, char *name, int *exit,
           int *n, int *k, int *m, int *p, int *s,
           int *oa, int *da, double *cap,
           double *gra, double *mcap, double *supp,
           double *lowcb, double *upcb, int *acb,
           int *kcb, int *ncb, double *coefcb, double *flow,
           void (f)(int*,int*,double*,double*,double*,int*))
```

In this case all the parameters have been declared as a pointer too (as in the routine `pprn1`), to make it possible calling the routine from Fortran or C user applications. In fact, this long list of parameters is equivalent to the input file presented in the previous section. Let us describe the meaning of each parameter:

Input parameters:

- `ln`: (integer escalar) denotes whether the problem to be solved is linear (in this case `ln`=0) or nonlinear (`ln`$\neq$0).
- `name`: (string or character vector) this is the name of the input file that contains the problem formulation.
- `n`: (integer escalar) number of arcs of the problem.
- `k`: (integer escalar) number of commodities of the problem.
- `m`: (integer escalar) number of nodes of the problem.
- `p`: (integer escalar) number of side constraints of the problem.
- `s`: (integer escalar) number of nonzero elements in the side contraints structure.
- `oa(n)`: (integer vector) vector with the origin nodes for all the arcs. The dimension of this vector must be at least of n positions.
- `da(n)`: (integer vector) vector with the destination nodes for all the arcs. The dimension of this vector must be at least of n positions.
- `cap(0:n,k)`: (real vector) vector with the capacities of all the arcs. The dimension of this vector must be at least of $(n+1) \cdot k$ positions. PPRN expects to receive first the capacities for the first commodity, and so on. Also, the user must reserve for each commodity extra space for an artificial (or root) arc (if this vector is viewed as a matrix `cap(0:n,k)`, these artificial arcs will occupy the positions `cap(0,i)`). This same order will be assumed in the rest of vectors passed as parameters.
- `gra(0:n,k)`: (real vector) vector with the costs of the arcs (first for those of the first commodity, reserving an initial extra space for one artificial arc, and so on). The dimension of this vector must be at least of $(n+1) \cdot k$ positions. If the problem is linear these costs determine the objective function and this vector will not be modified. If the problem is nonlinear these costs are used to find a "good" initial feasible point, and afterwards this vector will store the gradient evaluations of the objective function (thus modifying the initial values of the vector).

- `mcap(n)`: (real vector) mutual capacities for all the arcs. The dimension of this vector must be at least of n positions. Here the artificial arc is not required.
- `supp(m+1,k)`: (real vector) supplies (demands) for all the nodes and commodities. The dimension of this vector must be at least of $(m+1) \cdot k$ positions. First there will be the supplies for the first commodity, and finally for the last commodity. For each commodity the supply of node 1 until node m will be given in this order, and an extra position will be left after node m, as if there were an artificial (or root) node $m+1$.
- `lowcb(p)`: (real vector) lower bounds of the side constraints. If there are no side constraints, this vector will not be used. However, it must be sent as a parameter anyway. The dimension of this vector must be at least of p positions.
- `upcb(p)`: (real vector) upper bounds of the side constraints. If there are no side constraints, this vector will not be used. However, it must be sent as a parameter anyway. The dimension of this vector must be at least of p positions.
- `acb(s)`: (integer vector) for each nonzero element (from 1 to s) in the side constraints structure, this provides the arc associated with it. The dimension of this vector must be at least of s positions.
- `kcb(s)`: (integer vector) for each nonzero element (from 1 to s) in the side constraints structure, this provides the commodity associated with it. The dimension of this vector must be at least of s positions.
- `ncb(s)`: (integer vector) for each nonzero element (from 1 to s) in the side constraints structure, this gives the number of side constraint associated with it. The dimension of this vector must be at least of s positions.
- `coefcb(s)`: (real vector) this vector gives the value of each nonzero element (from 1 to s) in the side constraints structure. The dimension of this vector must be at least of s positions. The vectors associated with the side constraints structure (`lowcb`, `upcb`, `acb`, `kcb`, `ncb` and `coefcb`) can be modified by PPRN. However, the new side constraints structure stored in them will be equivalent to the original one.
- `f`: (function) this is the nonlinear objective function to be minimized. When the problem is merely linear this parameter can be removed if the user application is coded in Fortran, or set to a NULL pointer if it is coded in C.

Output parameters:

- `exit`: (integer escalar) the exit condition variable communicates to the user application the state of the package and the status of the solution found (if any) once PPRN has returned the control. A comprehensive list of the exit values will be presented below.
- `flow(0:n,k)`: (real vector) variable where the code PPRN will load the optimal solution found. Its dimension must be of at least $(n+1) \cdot k$ positions. PPRN will return the optimal point starting with the flows of the first commodity (with an initial arc that must not be considered), then for the second one, and so on. For each commodity PPRN will return $n+1$ values, the first one associated with an artificial arc that must not be taken into account (this arc will have a small absolute value).

In this case the variable `name` is only used for searching the specification file and writing the output and basis files (in routine `pprn1` this variable also gave the name of the input file from which the problem should be read).

The user application code for solving the example problem that we are considering using the routine `pprn2` could be as follows (considering that it is coded in Fortran):

```
      implicit none
      external funobj

      integer*4 m,k,n,p,s
      parameter (m=4,k=2,n=5,p=1,s=2)

      character*80 strdat
      integer*4 i,nolin,iexit
      integer*4 oa(n),da(n),acb(s),kcb(s),ncb(s)
      real*8 cap(0:n,k),gra(0:n,k),mcap(n),supp(m+1,k)
      real*8 lowcb(p),upcb(p),coefcb(s),flow(0:n,k)

c     loading the problem in the vectors
      data oa /1,1,1,2,3/, da /2,3,4,4,4/
      data acb /1,2/, kcb /1,2/, ncb /1,1/, coefcb /1.0,1.0/
      do i= 1,n
          gra(i,1)= 1.0
          gra(i,2)= 2.0
          cap(i,1)= 10.0
          cap(i,2)= 10.0
          mcap(i)= 15.0
      end do
      supp(1,1)= 18.0
      supp(4,1)= -18.0
      supp(1,2)= 20.0
      supp(4,2)= -20.0
      lowcb(1)= 17.0
      upcb(1)= 17.0

      strdat= 'example'
      nolin= 1
      call pprn2(nolin,strdat,iexit,n,k,m,p,s,oa,da,cap,gra,
     +     mcap,supp,lowcb,upcb,acb,kcb,ncb,coefcb,flow,funobj)
      write(*,*) iexit
      end
```

4. The specification file.

The way to use PPRN has been shown in the previous sections . In most cases the information presented so far suffices to obtain a desirable result. However, for some

problems it will be necessary to modify some default values or features of the package in order to improve its efficiency (or even to permit the running of some models). Such changes can be easily introduced by the user through a specification file which PPRN always tries to read at the beginning of the execution. If the specification file is not found, PPRN simply gives a warning and continues the execution. If it is found, PPRN will read all the keywords and their values, modifying the default ones.

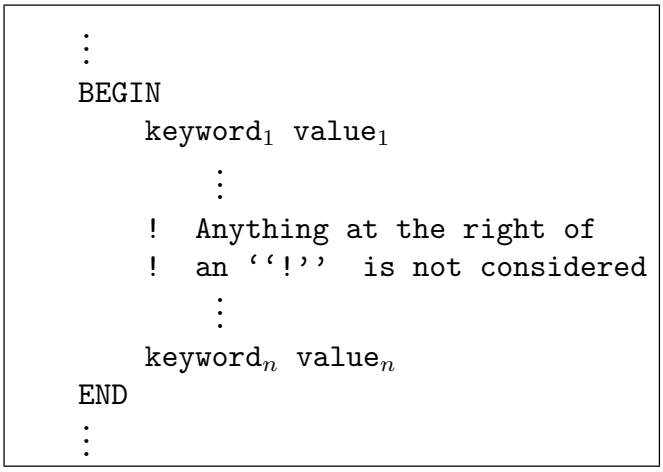

Fig.3. Structure of the specification file.

The structure of the specification file is very simple, and can be seen in Fig.3. PPRN will search for a "`BEGIN`" keyword (or "`begin`": upper and lower case letters can be used indistinctly) . Everything above the "`BEGIN`" is not considered. Then it will start to read lines with the same pattern: "`KEYWORD VALUE`". Comments can be introduced using the character "`!`". Everything at the right of an "`!`" will automatically not be considered. PPRN will stop reading keywords when an "`END`" is found (everything below the "`END`' keyword is discarded).

The rest of this section will be devoted to the description of all the keywords and the values that can be assigned to them. Excluding the "`BEGIN`" and "`END`" keywords, there are 30 different ones. These are presented in alphabetical order. The default value for each one is also given. It must be noted that some explanations will require a little knowledge of some internal details of the package [1,3,4].

1. ALPHA_TOLRG α: (default: α= 0.5) When optimizing a nonlinear function, PPRN tries to reduce the projected gradient of the current subspace before changing the active set constraints. The parameter α ($0 \leq \alpha \leq 1$) controls how much the current projected gradient must be reduced. A value $\alpha \approx 0$ will imply a great reduction (which is expensive in number of iterations and not appropiate for highly nonlinear functions), whereas a value close to 1 will imply fast changes in the active constraint set.

2. BLACK_LIST s: (default: s= "NO") During the phase 0 iterations PPRN can control the degenerate steps, and avoid them by maintaining a "black list" of arcs that produce such degeneracy. If s= "NO" (default value) this control will be inactive, whereas degeneracy will be avoided when s= "YES". In general the default value has shown to be more efficient.

3. CONNECTIVITY s: (default: s= "NO") To solve a problem, PPRN requires that all the nodes of the network must be connected (i.e., no disjoint graphs are permitted). By default PPRN does not check the connectivity of the network, assuming that the problem verifies it. The user can active the check by just making s= "YES".

4. DESCENT_DIR i: (default: i= 3) This keyword determines the descent direction that will be used by PPRN during the phase 2 when optimizing a nonlinear function. There are three different possibilities. The value $i = 1$ means using a simple reduced gradient descent direction; if $i = 2$ the descent direction will be computed using a truncated-Newton algorithm; finally, when $i = 3$ (the default), PPRN will use a quasi-Newton update. In general the last alternative seems to be the most efficient, if the number of superbasic variables is not really high. The truncated-Newton algorithm requires extra objective function evaluations, and in general this can considerably increase the execution time. However, for problems with very many superbasic variables, it can be the best choice. The reduced gradient direction (i= 1) has a very slow convergence, and its use can become prohibitive in most cases.

5. INFINITY r: (default: r= 10^{32}) This is the infinity value considered by PPRN. This value should be used in the input file as infinity (e.g, when defining uncapacited arcs, or unbounded side constraints).

6. INVERTBASIS i: (default: i= 50) The primal partitioning method implemented by PPRN implies the use of a working basis (or working matrix) instead of the whole basis. This working basis is updated at each iteration. However this updating process can introduce some round-off errors, so it is advisable to recompute and reinvert it once every specified number of iterations. The keyword INVERTBASIS controls the number of iterations between two consecutive reinversions of the working matrix. A smaller value of i means reducing the possible round-off error but increasing the execution time. On the other hand, increasing i does not always mean reducing the execution time (for a really high i value the update process can become very slow) and poorer solutions will be found when solving systems of linear equations with the working matrix.

7. LINSEARCH_TOL r: (default: r=0.1) Precision required at the point obtained by the line search routine. The value r must be between 0 and 1. The closer to 0 r is, the more accurate the line search performed (increasing the number of objective function evaluations) will be.

8. LISTVAR s: (default: s="NO") By default PPRN does not list the value of the variables and constraints at the optimal solution (for big sized problems that would mean having to write a lot of information). If the user wants this list, the value of the keyword must be activated (s="YES").

9. LOGFILE s: (default: s="NO") PPRN has the capability of writing a log file. By default (s="NO") this option is disabled. When it is actived by the user (s="YES"), PPRN will produce a "`.log''` file, which, by lines, presents the next information at each iteration: number of iteration, number of seconds (real time, not CPU time) spent since the beginning of the execution, objective function value, number of objective function evaluations and number of superbasic variables. The first lines will provide information for the phase 1 iterations (obtaining a feasible point), whereas

the remaining lines are related to phase 2. Since the number of seconds is computed through calls to the routine "secnds()" (which does not provide CPU time) "strange" execution times can be obtained when the execution is running together with other processes in the same CPU.

10. MAX_QUASI i: (default: i=500) PPRN, by default, uses a quasi-Newton update to compute the descent direction. When the number of superbasic variables is not really high this is the best option. However, when the number of superbasics increases considerably the number of operations required to maintain the quasi-Newton update become prohibitive, and thus the truncated-Newton algorithm can be a better choice. This threshold number of superbasic variables (where the switching from the quasi-Newton to the truncated-Newton method is made) is determined by the parameter MAX_QUASI (by default this number is i=500). This parameter is meaningless if the descent direction is not being computed with the quasi-Newton update method (DESCENT_DIR= 1 or 2). In general, very high values of i should be avoided so as not to increase the execution time excessively.

11. MAX_T5 i: (default: $i = -1$) PPRN has four different logical conditions that are tested at each iteration, and depending on its values it decides whether or not the optimum of the current subspace has been achieved. (See [4] for a comprehensive description of these test conditions). For non-smooth functions the user can activate a fifth condition, which will consider that we are at the optimum of the current subspace when the value of the first three logical conditions has been the same during i consecutive iterations. By default $i = -1$, which means that the test is inactive. It suffices to give a positive integer value to i to activate this fifth test.

12. MAXITER i: (default: i=2147483647=2^{31}-1) Maximum number of iterations allowed (phase 1 and phase 2 iterations). The default value is the maximum signed integer number that can be stored in four bytes, almost equivalent to imposing no limit on the number of iterations.

13. MAXITER_NEWT_TR i: (default: i=20) Maximum number of iterations allowed to the truncated-Newton algorithm when computing the descent direction using this option. Increasing this value excessively can mean a prohibitive increase in the number of function evaluations (degrading the performance of the algorithm).

14. NBLOCSNB i : (default: i=10) This keyword identifies the number of blocks in which the nonbasic variables are supposed to be divided. During the pricing operation, instead of all the nonbasic variables being priced, they are priced by blocks. If a candidate has been found within the current block, the rest are not priced until the next iteration. By changing the default value of i the behaviour of the execution can be modified. This keyword is directly related to the NOBASPRICE one (see below).

15. NEWBASIS $file$: (default: no $file$) PPRN can save the current status periodically (according to the value of the keyword SAVEBASIS) and at the end of the execution. This can be done by simply writing the name of the basis file in the NEWBASIS keyword. By default no basis information will be saved.

16. NOBASPRICE i : (default: i=max(20,k*(n-m+1)/NBLOCSNB)) This keyword gives the exact number of nonbasic variables that will be priced consecutively when

a candidate is sought. PPRN will price the first i nonbasic variables. If there is a candidate it will stop. If not, it will continue with the next i nonbasic variables. At the next iteration it will continue pricing where it stopped in the previous pricing operation. The default value of the keyword NOBASPRICE is computed using the NBLOCSNB number that was already introduced.

17. OLDBASIS $file$: (default: no $file$) PPRN can restart executions from a point loaded from a basis file created in a previous run. This can be done by simply writing the name of the basis file after the OLDBASIS keyword. By default no basis file is loaded and the initial point will be chosen by the code.

18. ONLY_FEASIBLE s: (default: s="NO") When this parameter is set to the other available value (s="YES"), PPRN will only try to find a feasible point (phases 0 and 1) and will then stop the execution. With the default value (s="NO") PPRN will continue the optimization process until it reaches the optimal point.

19. OUTPUT_FREQ i: (default: i=0, if problem sent through parameters, or i=100, if problem read from file) This parameter controls the frequency of lines of information to appear in the output file. A value of 0 means that no output file will be generated. By default, a line of information will be writen each 100 iterations if the problem is read from a file. If the problem is sent through parameters, no output file will be created by default.

20. PCT_ACTIVES r: (default: r=0.25) PPRN uses as workspace some vectors which are allocated dynamically in run-time. Its dimension is approximated through a coefficient that should be understood as the "percentage of sparsity" of the vector (since it would be impossible to allocate the complete dimension of the vector due to the amount of memory required). This keyword and the next five below (all of them starting with the prefix "PCT_") present the "percentages of sparsity" to be considered in the different vectors that are dynamically assigned. All these "percentages" must take a value between 0 and 1 (per unit values). In some problems the default value will not be sufficient to solve the problem. Thus PPRN will stop the execution and the exit condition will notify that some "percentage" parameter must be increased. It is the task of the user to modify such a parameter in the specification file (taking care not to increase it too much in order to maintain the memory requirements within reasonable limits). The keyword PCT_ACTIVES gives the percentage of mutual capacity and side constraints that can be considered as active (equivalent to giving the maximum dimension of the working matrix). By default the code assumes that at most 50% of the mutual and side constraints can become active.

21. PCT_ETA r: (default: r=0.2) This parameter is used to dimension the eta matrices that will be used when updating at each iteration the working matrix (via eta matrices). See the explanation of keyword PCT_ACTIVES for more details.

22. PCT_PATHS r: (default: r=0.1) PPRN stores for each complementary arc its path into the basic tree associated with its commodity. In the worst case, the total amount of space required would be: maximum_number_of_complementary_arcs $\times$ nodes_of_the_network. However, this number is extremely large, and in fact only a percentage of this value will be allocated. This percentage is fixed by the parameter

PCT_PATHS (and by default, 10% of the previous computed number is reserved). See the explanation of keyword PCT_ACTIVES for more details.

23. PCT_Q r: (default: r=0.02) The dimension of the working matrix (also called matrix Q) is computed through the keyword PCT_ACTIVES. If we denote this dimension by $|Q|$ it follows that if we store the matrix Q in dense form we will need $|Q| \times |Q|$ real positions. However, this working matrix is stored in sparse form, and the number of positions reserved is computed as: PCT_Q $\times |Q| \times |Q|$, where the parameter PCT_Q can be suitably modified by the user. See the explanation of keyword PCT_ACTIVES for more details.

24. PCT_SUPER r: (default: r=0.3) This parameter gives the percentage of the total amount of variables that can become superbasic. See the explanation of keyword PCT_ACTIVES for more details.

25. PIVOT_LU ϵ: (default: $\epsilon = 10^{-11}$) Minimum value allowed as a pivot element when performing a LU decomposition of a matrix.

26. SAVEBASIS i: (default: i= 500) PPRN can save the current status of the execution process periodically . The keyword SAVEBASIS allows us to change the frequency of this saving operation by modifying the value i (which is the number of iterations between two consecutive savings).

27. PHASE_0 s: (default: s= "FEASIBLE"). The phase 0 of the algorithm implemented by PPRN tries to solve merely K linear network problems, one for each of the K commodities of the original problem. The user can decide, when solving this linear network problems, if PPRN has to obtain just a feasible point (s= "FEASIBLE", which is the default value) or the optimal solution (s= "OPTIMAL"). If PPRN is being used to solve a single commodity linear problem, the best option will of course be s="OPTIMAL". This option can also be the best one when, in the optimal solution of the multicommodity linear problem, there is a small number of active mutual capacity constraints; thus, solving K single commodity problems, and joining their solutions can be a good approximation of the optimal point of the multicommodity problem. For the remaining cases the default option (just finding a feasible point) suffices in general.

28. PRICE_PH1 i: (default: i= 1) This parameter denotes the type of ordering of the variables considered during the pricing operation in phase 1. There are 4 different orderings (thus i can take only the values 1, 2, 3 and 4). If one of the first two (types 1 and 2) is chosen, PPRN will initially always price the nonbasic slack variables. If no candidate is found, then it will start pricing nonbasic arcs by blocks of NOBASPRICE elements (see keyword NOBASPRICE explained above). However, in types 3 and 4, nonbasic slacks are priced together with the nonbasic arcs (and not in first place). The difference between types 1 and 2 is that in the first PPRN considers (type 1) K circular lists, one for each commodity (and K arc-pointers, one per list, to the next arc to be priced at next iteration, and one more commodity-pointer to the next commodity to be priced), whereas in the second it only maintains one big circular list with all the arcs joined (and a single pointer to the next arc to be priced). Thus, when the pricing operation is applied with the ordering type number 1, it will start with the commodity stored in the commodity-pointer, and the arc of

the associated arc-pointer. And the pricing will continue with this commodity by blocks of dimension NOBASPRICE until the initial arc (the one stored in the arc-pointer) is reached again. Then the next commodity list will be started, and so on. In the second ordering type, however, if we start at a given arc of a certain commodity, we will change to the next commodity when the last arc of the current commodity has been visited, without restarting with the first arc. The same difference exists between types 3 and 4. In most tests performed, types 1 and 2 seemed to be more efficient than 3 and 4. On the other hand, no significant difference was found in the performance of the code between types 1 and 2 (and the same with 3 and 4).

29. PRICE_PH2 i: (default: i= 1) This parameter has the same meaning as keyword PRICE_PH1 explained above, but applied to the pricing operation during phase 2. See keyword PRICE_PH1 for more details.

30. TOL_NT r: (default: r= 0.5) This tolerance r ($0 \leq r \leq 1$) is used to control the accuracy of the solution found when computing the descent direction through the truncated-Newton algorithm. The closer r is to 0, the more precision will be required in the solution found by the truncated-Newton algorithm, and the number of objective function evaluations will increase considerably (and the execution time spent by PPRN).

31. QUASI_ACTIVES s: (default: s= "NO") When the descent direction for the superbasic variables is computed, in some cases a degenerate step can be obtained. This can happen if, for instance, a superbasic varible is near its lower or upper bound (it will be said that this is a quasi-active superbasic variable), and its component in the descent direction vector drives it to violate the active bound. In such cases a special treatment can be applied to these variables in order to avoid this situation whenever possible. By default (s= "NO") this check is inactive. The user can activate it during the whole execution by setting s= "YES". (It must be noted that this check is sometimes activated automatically by PPRN, even if s= "NO", to avoid some cycling problems associated with degenerate steps due to quasi-active superbasic variables).

32. TOLOPTIM ϵ_{opt}: (default: ϵ_{opt}= 10^{-6}) Optimality tolerance required at a point to be considered as the solution of the problem. This situation is detected when, at the pricing operation, all the multipliers λ satisfy $\lambda < \epsilon_{opt} \cdot \epsilon(\|\pi\|_1)$, ($\epsilon(\|\pi\|_1)$ being a function of the 1-norm of the π vector computed during the phase 2 algorithm). See [4] for more details.

5. The files generated by PPRN.

PPRN will always write two files when executing a model: the output file (with extension "`.lst`") and the basis file ("`.bas`"). A third file, the log file, is created depending on the value of parameter LOGFILE in the specification file (see previous section for a brief description of the structure of the log file). By default, this log file is not created.

As was said in previous sections, the basis file stores the status of the optimization process at the end of the execution. In this file all the data structures required to restart the execution process from a given point are saved. No explanation

is given about the contents of the basis file, since this would require knowing the internal data structures of the code, which is outside the scope of this guide.

We will focus mainly on the description of the output file generated by PPRN. Depending on whether the objective function to be minimized is linear or nonlinear, the structure of the output file is slightly different. In fact, phases 0 and 1 have the same kind of output for both types of function, whereas the information disclosed in phase 2 is directly related to whether a linear or nonlinear objective. The following pages present the output obtained with the example problem, first with the linear function, and then with the nonlinear one:

Output file for the example problem (LINEAR objective function)

```
****************** PARAMETERS ******************
OBJECTIVE FUNCTION TYPE       : linear
MAXIMUM NUMBER OF ITERATIONS  :  2147483647
SAVE BASIS FREQUENCY          :  500
REINVERSION BASIS FREQUENCY   :  50
NUMBER NON BASICS FOR PRICING :  20
PRICING TYPE AT PHASE 1       :  1
PRICING TYPE AT PHASE 2       :  1
INFINITY VALUE CONSIDERED     :    1.0000000000000D+32
OUTPUT FREQUENCY              :  1
SOLUTION OBTAINED AT PHASE 0  : feasible
BLACK LIST AT PHASE 0         : inactive
CONNECTIVITY CHECK            : inactive
L-U PIVOT TOLERANCE           :    1.0000000000000D-11
PERCENTAGE ACTIVES            :   0.50000000000000
SPARSITY ETA-MATRICES         :   0.20000000000000
SPARSITY Q (WORKING MATRIX)   :    2.0000000000000D-02
SPARSITY PATHS                :   0.90000000000000
************************************************

 STARTING PHASE 0.

 INITIAL FEASIBLE TREE FOUND FOR COMMODITY  1  ITERATIONS:  2
 INITIAL FEASIBLE TREE FOUND FOR COMMODITY  2  ITERATIONS:  2

 STARTING PHASE 1. INFEASIBILITIES=    20.000000000000

Ph.....It........Nb->Bs........Bs->Nb.......Fobj/Sinf...Ncmp............Rg.......Alph
 1      1       5     1       1     0  0.17000000D+02      1  -0.10000D+01  0.300D+01
 1      2       3     1       3     0  0.12000000D+02      2   0.10000D+01 -0.500D+01
 1      3       5     2       1     1  0.20000000D+01      2  -0.20000D+01  0.500D+01
 1      4       1     0       4     0  0.20000000D+01      2  -0.20000D+01  0.000D+00
 1      5       4     0      -1     0  0.00000000D+00      2  -0.10000D+01  0.200D+01

 STARTING PHASE 2. OBJECTIVE FUNCTION=    91.000000000000

 ooooooooooooooooooooooooo
 o                        o
 o       E  X  I  T       o
 o                        o
 ooooooooooooooooooooooooo
 STATUS: OPTIMAL SOLUTION FOUND
```

```
 ITERATION:  5
 PHASE:  2
 PHASE 0 ITERATIONS:  4
 PHASE 1 ITERATIONS:  5
 PHASE 2 ITERATIONS:  0
 DIMENSION OF WORKING MATRIX:  2
 OBJECTIVE FUNCTION VALUE:    91.000000000000

   1. VARIABLES.

....arc..commodity...status_arc.............flow...........capacity
      1          1        UPPER          10.000000          10.000000
      1          2         TREE           3.000000          10.000000
      2          1         TREE           3.000000          10.000000
      2          2         TREE           7.000000          10.000000
      3          1         TREE           5.000000          10.000000
      3          2        UPPER          10.000000          10.000000
      4          1         TREE          10.000000          10.000000
      4          2         TREE           3.000000          10.000000
      5          1        CMPLT           3.000000          10.000000
      5          2        CMPLT           7.000000          10.000000

   2. MUTUAL CAPACITY CONSTRAINTS.

....arc...status_slack.............flow.............slack.....mutual_capacity
      1          BASIC          13.000000           2.000000          15.000000
      2          BASIC          10.000000           5.000000          15.000000
      3          LOWER          15.000000           0.000000          15.000000
      4          BASIC          13.000000           2.000000          15.000000
      5          BASIC          10.000000           5.000000          15.000000

   3. SIDE CONSTRAINTS.

....nsc...status_slack.........lower_bound.............value.........upper_bound
      1          UPPER          17.000000          17.000000          17.000000
```

Output file for the example problem (NONLINEAR objective function)

```
 ******************** PARAMETERS ********************
 OBJECTIVE FUNCTION TYPE       : non linear
 MAXIMUM NUMBER OF ITERATIONS  :  2147483647
 SAVE BASIS FREQUENCY          :  500
 REINVERSION BASIS FREQUENCY   :  50
 NUMBER NON BASICS FOR PRICING :  20
 PRICING TYPE AT PHASE 1       :  1
 PRICING TYPE AT PHASE 2       :  1
 INFINITY VALUE CONSIDERED     :    1.0000000000000D+32
 OUTPUT FREQUENCY              :  1
 SOLUTION OBTAINED AT PHASE 0  : feasible
 BLACK LIST AT PHASE 0         : inactive
 CONNECTIVITY CHECK            : inactive
 L-U PIVOT TOLERANCE           :    1.0000000000000D-11
```

```
 PERCENTAGE ACTIVES            :   0.50000000000000
 SPARSITY ETA-MATRICES         :   0.20000000000000
 SPARSITY Q (WORKING MATRIX)   :    2.00000000000000D-02
 SPARSITY PATHS                :   0.90000000000000
 DESCENT DIRECTION             : quasi Newton
 MAXIMUM DIMENSION OF R'R      :  4
 QUASI ACTIVES DIRECTION       : inactive
 REDUCTION SUBSPACE GRADIENT   :   0.50000000000000
 OPTIMALITY TOLERANCE          :    1.00000000000000D-06
 LINESEARCH TOLERANCE          :    1.00000000000000D-01
 T5 CONTROL                    : inactive
 ************************************************

 STARTING PHASE 0.

 INITIAL FEASIBLE TREE FOUND FOR COMMODITY  1  ITERATIONS:  2
 INITIAL FEASIBLE TREE FOUND FOR COMMODITY  2  ITERATIONS:  2

 STARTING PHASE 1. INFEASIBILITIES=    20.000000000000

Ph.....It........Nb->Bs........Bs->Nb.......Fobj/Sinf...Ncmp............Rg.......Alph
 1      1       5     1       1     0  0.17000000D+02      1  -0.10000D+01  0.300D+01
 1      2       3     1       3     0  0.12000000D+02      2   0.10000D+01 -0.500D+01
 1      3       5     2       1     1  0.20000000D+01      2  -0.20000D+01  0.500D+01
 1      4       1     0       4     0  0.20000000D+01      2  -0.20000D+01  0.000D+00
 1      5       4     0      -1     0  0.00000000D+00      2  -0.10000D+01  0.200D+01

 STARTING PHASE 2. OBJECTIVE FUNCTION=    459.00000000000

Ph.It.Nb->Sup........Rg..Bs->Nb.Sup->Nb/Bs.......Fobj..Ncmp..Nsup.....Alph......Nrg...Cv.
 2  6   1    1  0.12D+02                   0.4545D+03     2     1 0.75D+00 0.00D+00 TFTT
 2  7   3    2  0.16D+02                   0.4438D+03     2     2 0.13D+01 0.10D+02 TTTT
 2  8                                      0.4385D+03     2     2 0.12D+01 0.35D-14 TFFF
 2  9   3    0 -0.30D+01                   0.4377D+03     2     3 0.50D+00 0.30D+01 TFTT
 2 10                                      0.4370D+03     2     3 0.70D+00 0.17D-13 TTFF

 ooooooooooooooooooooooooo
 o                        o
 o       E  X  I  T       o
 o                        o
 ooooooooooooooooooooooooo
 STATUS: OPTIMAL SOLUTION FOUND

 ITERATION:  10
 PHASE:  2
 PHASE 0 ITERATIONS:  4
 PHASE 1 ITERATIONS:  5
 PHASE 2 ITERATIONS:  5
 DIMENSION OF WORKING MATRIX:  2
 NUMBER OF SUPERBASICS:  3
 OBJECTIVE FUNCTION AVALUATIONS:  12
 RG NORM:    1.7763568394003D-14
 PI NORM:    18.173764450045
 RG NORM/PI NORM:    9.7742921907182D-16
 OBJECTIVE FUNCTION VALUE:    437.00000000000
```

```
   1. VARIABLES.

....arc..commodity...status_arc.............flow...........capacity
      1          1        SUPER           8.250000          10.000000
      1          2         TREE           3.750000          10.000000
      2          1         TREE           3.250000          10.000000
      2          2         TREE           8.750000          10.000000
      3          1         TREE           6.500000          10.000000
      3          2        SUPER           7.500000          10.000000
      4          1         TREE           8.250000          10.000000
      4          2         TREE           3.750000          10.000000
      5          1        CMPLT           3.250000          10.000000
      5          2        CMPLT           8.750000          10.000000

   2. MUTUAL CAPACITY CONSTRAINTS.

....arc...status_slack.............flow.............slack.....mutual_capacity
      1          BASIC          12.000000            3.000000          15.000000
      2          BASIC          12.000000            3.000000          15.000000
      3          SUPER          14.000000            1.000000          15.000000
      4          BASIC          12.000000            3.000000          15.000000
      5          BASIC          12.000000            3.000000          15.000000

   3. SIDE CONSTRAINTS.

....nsc...status_slack.........lower_bound.............value.........upper_bound
      1          UPPER          17.000000           17.000000          17.000000
```

As can be seen in the output files shown, PPRN presents first the values of some parameters (default values or those introduced by the user in the specification file) whose meaning was specified in previous sections. It must be stressed that when the objective function is nonlinear there are far more parameters involved, in particular, those related to the phase 2 algorithm. The remaining parameters are common to linear and nonlinear functions.

The executions presented were made using a specification file with only two keywords. The first one, "OUTPUT_FREQ 1", was used to present a line of information per iteration. The second one, "PCT_PATHS 0.9", was required since, with the default value of this parameter, there was not enough room to store the paths of the complementary arcs (see the description of keyword PCT_PATHS in the previous section). This is due to the very small size of this problem (the default values were adjusted considering larger models). Since a specification file has been provided, PPRN did not show any warning message at the beginning of the output file. Otherwise, the first line of the output file would have been: "`WARNING: CANNOT OPEN SPECIFICATION FILE`".

When the parameters used in the execution have been presented, the rest of the file is divided into two main parts: first, a summary of the optimization process (detailing phase 0, phase 1 and phase 2), and then the point achieved together with some information about the exit status of the package, the objective function value, etc.

Let us start with the first part, concerning the evolution of the optimization process. It can be seen in the output files that the information for phases 0, 1 and 2 is introduced by the comment lines "`STARTING PHASE 0.`", "`STARTING PHASE 1.`" and "`STARTING PHASE 2.`". As stated above, phase 0 solves a linear network problem for each commodity. In the output file a line of information will appear for each of these linear network problems, indicating whether the tree achieved was only a feasible one or the optimal solution of the single-commodity problem (depending on the value of the parameter PHASE_0 of the specification file), and also showing the number of iterations required (of the specialized network simplex algorithm implemented). The appearance of one such line of information is as follows:

```
INITIAL FEASIBLE TREE FOUND FOR COMMODITY  1  ITERATIONS 2
```

At the beginning of the output of phase 1, the code shows initially the value of the sum of infeasibilities (with the message "`INFEASIBILITIES= xx`"). These infeasibilities are related only to the mutual capacity and side constraints (the network constraints have already been satisfied in phase 0). Now the code will write periodically (with a frequency depending on the parameter OUTPUT_FREQ of the specification file) a line in the output file disclosing the following information:

```
Ph....It....Nb->Bs....Bs->Nb....Fobj/Sinf....Ncmp....Rg....Alph
```

The first field ("`Ph`") indicates the current phase of the algorithm. "`It`", gives the number of iteration. "`Nb->Bs`" shows which variable has left the nonbasic set to become basic, whereas "`Bs->Nb`" indicates the basic variable that will be a nonbasic one. In the last two fields, variables are coded by two integer values. If the second value is 0, it means than the variable associated is a slack. In this case, if the first number is positive, the slack refers to a mutual capacity constraint, and if negative to a side constraint. On the other hand, if the second number is positive with a value k, the variable is associated with an arc of the k-th commodity. In this case the first number will denote the number of the arc. For instance, the pair of numbers "$1, 0$" would refer to the slack of the first mutual capacity constraint, "$-2, 0$" would be the slack of the second side constraint, and "$1, 2$" would denote the first arc of the second commodity.

The following field, "`Fobj/Sinf`", gives the current value of the sum of infeasibilites (or the current value of the objective function if we are solving phase 2 of a linear problem). "`Ncmp`" indicates the number of complementary arcs, which is equivalent to the dimension of the working matrix at the current iteration. The field "`Rg`" shows the value of the multiplier of the variable that has been chosen to become a basic one. The last field, "`Alph`", denotes the maximum step that can be performed to maintain the feasibility with respect to the bounds of the basic variables when performing the pivot operation.

When optimizing a linear objective function, the information presented in phase 2 is the same as that of phase 1, since in both cases a specialized multicommodity simplex method is being applied. In the output file listed above, for the linear function no iteration was required in phase 2, since the feasible point achieved was the optimum one. It can be seen, however, that at the beginning of phase 2, besides the message announcing its starting, the code gives the value of the objective function at the initial feasible point (with the comment "`OBJECTIVE FUNCTION= xx`").

On the other hand, in the nonlinear case (whose output file is listed above) the initial feasible point was not the optimum one, so there are phase 2 iterations. The information disclosed in phase 2 nonlinear iterations is the following:

```
Ph..It..Nb->Sup..Rg..Bs->Nb..Sup->Nb/Bs..Fobj..Ncmp..Nsup..Alph..Nrg..Cv
```

The first two fields ("`Ph`" and "`It `") have the same meaning as in phase 1 iterations. "`Nb->Sup`" shows the nonbasic variable that becomes a superbasic one (the variables are coded by two integer values, in the same way as was put forward when we detailed the phase 1 output information). The field "`Rg`" gives the value of the multiplier associated with the nonbasic variable chosen to become superbasic. The two last fields can be empty if in the current iteration PPRN decided not to price any variable and to keep on optimizing in the current subspace. Field "`Bs->Nb`" shows the basic variable that will become nonbasic (this will only happen when the basic variable reaches one of its bounds when moving along the descent direction found; otherwise this field will be empty). The next field, "`Sup->Nb/Bs`", denotes the superbasic variable that will be made nonbasic (if, when moving along the descent direction, the superbasic reaches a bound) or basic (when, as stated above, a basic variable becomes nonbasic, thus having to be replaced by a superbasic one); this field can also be empty if neither a basic nor a superbasic variable reaches its bound. Fields "`Fobj`" and "`Ncmp`" have the same meaning as in phase 1 iterations. "`Nsup`" gives the number of superbasic variables in the current iteration. "`Alph`" shows the step performed in the descent direction, found by the line search routine. "`Nrg`" is the infinity norm of the reduced gradient of the current subspace. Finally, the field "`Cv`" shows the boolean values (`T`= true, `F`= false) of the different convergence tests (by default, only 4 tests, but the user can activate the fifth one using the keyword MAX_T5 in the specification file). It is considered that the code has achieved the optimum in the current subspace if either the first three values are true (a small step has been performed, the objective function has not changed considerably, and the projected gradient has been reduced sufficiently), or the fourth test is active (the reduced gradient has such a small value that the optimum point can be assumed achieved). We refer the reader to [4] for more details on the convergence tests performed.

Once the information of the optimization process is presented, the rest of the output file is devoted to the description of the point achieved. The first thing to appear is a message giving the status of the point found by PPRN, or an error message if something was wrong (the next section will describe at length the different exit and error conditions). The following lines give a short description of some data of interest for linear problems, such as the number of iterations performed (total and in each phase), the final dimension of the working matrix, and the objective function value. For nonlinear objective functions, this is extended with the number of superbasics, the number of objective function evaluations, the norm of the reduced gradient, the norm of the multiplier vector (pi vector), and the ratio between the last two values.

Finally, PPRN lists the values and status of all the variables at the point found, first for the arcs, then for the mutual capacity constraints slacks, and finally for the side constraint slacks. For the arcs, (besides of the number of arc and commodity) the code gives its flow, its capacity, and the status of the arc. The different values of the status field (with their meaning) are the following:

• UPPER: the arc is nonbasic at its upper bound.
• LOWER: the arc is nonbasic at its lower bound.
• TREE: the arc is basic and belongs to the tree associated with its commodity.
• CMPLT: the arc is basic, and is a complementary arc.
• SUPER: the arc is superbasic.

For the slacks, the information given is the number of slack, its status, its lower and upper bounds, and its value. The status field can take the values UPPER (nonbasic slack at its upper bound), LOWER (nonbasic slack at its lower bound), BASIC (basic slack), and SUPER (superbasic slack).

6. Exit conditions.

We have postponed until this section the comprehensive list of exit conditions of the PPRN package. The exit conditions were already introduced when we described the two interface routines between the user and the package, pprn1 and pprn2. One of the parameters returned by these routines was the exit value, through which PPRN communicated the status of the solution found. This same information also appeared in the output file written by the package.

The exit conditions can be divided into two main types: those that can be considered as a normal exit of the problem (which will be strictly called "exit conditions"), and those corresponding to abnormal exits or failures of the program (referred to as "error conditions"). In some cases, however, it is not clear what kind a given stop of the program belongs to. Thus, the classification could have been made in another way and may be subject to future changes. In the first case ("exit conditions"), the parameter exit of routines pprn1 and pprn2 will have a positive value. In the second case ("error conditions") this parameter will have a negative value. The following is a comprehensive list with all the possible values (and their meanings) that the exit parameter can take at the end of the execution.

EXIT CONDITIONS (normal exit).
• exit=0: Optimal solution found.
• exit=1: Infeasible problem at phase 0.
• exit=2: Infeasible problem at phase 1.
• exit=3: Working matrix is singular (the basis is thus singular). If the problem is well defined, this error should not occur.
• exit=4: A complementary arc could not be found to replace a basic arc. This message should not occur if the problem is well defined.
• exit=5: Too many complementary arcs. The user should increase the parameter PCT_ACTIVES in the specification file to solve this problem.
• exit=6: PPRN cannot pivot in a basic tree. This message should not occur in a well defined problem.
• exit=7: Too many iterations. The user should increase the value of keyword MAXITER in the specification file.
• exit=8: The problem is unbounded. If the problem is linear, it means that the objective value can be decreased as much as desired (the problem is a minimization one). If the problem is nonlinear, this may be because it has an extremely large step, found in the line search routine.

• `exit`=9: A superbasic variable to replace a basic one could not be found. This message should not occur if the problem is well defined.
• `exit`=10: Too many superbasic variables. The user should increase the value of the keyword PCT_SUPER in the specification file.
• `exit`=11: Initial feasible point found. This message will appear when the keyword ONLY_FEASIBLE is set to the value "YES" in the specification file (the user only wants to find an initial feasible point).

ERROR CONDITIONS (abnormal exit).
• `exit`=−1: The input file could not be opened (when the user passed the problem through a file, calling the routine `pprn1`).
• `exit`=−2: Some wrong data were found when the input file was read.
• `exit`=−3: Currently not used.
• `exit`=−4: Currently not used.
• `exit`=−5: Currently not used.
• `exit`=−6: Currently not used.
• `exit`=−7: Wrong command in the specification file.
• `exit`=−8: The old basis file introduced by the user with the keyword OLDBASIS of the specification file, could not be found.
• `exit`=−9: The old basis file does not match the current problem.
• `exit`=−10: There is not enough room for storing the working matrix, or performing its LU decomposition. The parameter PCT_Q in the specification file should be increased.
• `exit`=−11: There is not enough room for updating the working matrix through eta vectors. The parameter PCT_ETA in the specification file should be increased.
• `exit`=−12: Currently not used.
• `exit`=−13: There is not enough room for storing the paths of the complementary arcs in the basic trees. The parameter PCT_PATHS in the specification file should be increased.
• `exit`=−14: Self-loops (arcs pointing to their source nodes) are not supported by PPRN.
• `exit`=−15: The lower bound is greater than the upper bound in some constraints.
• `exit`=−16: Too many coefficients in the side constraints structure.
• `exit`=−17: A wrong node was found when the network structure was read.
• `exit`=−18: A wrong arc was found when the side constraints structure was read.
• `exit`=−19: A wrong commodity was found when the side constraints structure was read.
• `exit`=−20: A wrong side constraint was found when the side constraints structure was read.
• `exit`=−21: The network has two or more disjoint graphs. This message will only appear if the keyword CONNECTIVITY has been activated in the specification file.
• `exit`=−50: There is not enough memory in the system to solve the problem at this moment.

7. Extensions and induced stops.

The current version of the PPRN package can solve a wide range of linear and nonlinear multicommodity problems. However, some improvements and extensions

can clearly be made. The following is a list of possible extensions, some of which could be considered in successive versions of the package:

• Allowing problems with lower bounds in the arcs (arcs with a minimum capacity) to be considered.

• Adding a module for checking (through finite differences) the gradient vector computed by the user.

• Implementing a "full dynamic assignment" of the memory, by reallocating a vector (which has reached its maximum dimension) automatically with more room in another part of the memory.

• Allowing the communication of some parameters or keywords through routines, avoiding in this case the use of a specification file.

• Allowing some models to be saved in the memory, and to be modified (e.g, bounds of the variables) for a later reexecution of the problem.

• In the case of passing the model through parameters, increasing the information returned by PPRN (now it only gives the optimal flows achieved) with the status of the arcs and slacks, and the multipliers associated with the slacks of the constraints (mutual capacity and side constraints).

There are also in the code some abnormal conditions where a Fortran "STOP" command aborts the execution. Theoretically, they should never occur but if they come up it would be helpful to analyze the code with the problem and data where they appear. Messages are welcome at the e-mail address:

jcastro@eio.upc.es

Any comments and suggestions will also be welcome.

REFERENCES

[1] Castro, J. 1993. *Efficient computing and updating of the working matrix of the multicommodity network flow problem with side constraints through primal partitioning.* DR 93/03. Statistics and Operations Research Dept., Universitat Politècnica de Catalunya, Barcelona. (written in Catalan).

[2] Castro, J. and N. Nabona. 1994. *Nonlinear multicommodity network flows through primal partitioning and comparison with alternative methods.* System Modelling and Optimization. Proceedings of the 16th IFIP Conference. J. Henry and J.-P. Yvon editors. pp. 875-884. Springer-Verlag.

[3] Castro, J. and N. Nabona. 1994. *Computational tests of a linear multicommodity network flow code with linear side constraints through primal partitioning.* DR 94/02. Statistics and Operations Research Dept., Universitat Politècnica de Catalunya, Barcelona.

[4] Castro, J. and N. Nabona. 1994. *Computational tests of a nonlinear multicommodity network flow code with linear side constraints through primal partitioning.* DR 94/05. Statistics and Operations Research Dept., Universitat Politècnica de Catalunya, Barcelona.

[5] Castro, J. and N. Nabona. 1996. *An implementation of linear and nonlinear multicommodity network flows*, European Journal of Operational Research, 1996, 92, pp. 37-53.

[6] Kennington, J.L. and R.V. Helgason. 1980. *Algorithms for network programming.* John Wiley & Sons, New York.

[7] Murtagh, B.A. and M.A. Saunders. 1978. *Large-scale linearly constrained optimization.* Mathematical Programming, v. 14, pp. 41–72

[8] Murtagh, B.A. and M.A. Saunders. 1983. *MINOS 5.0. User's guide.* Dept. of Operations Research, Stanford University, CA 9430, USA.