Introduction à CPLEX

1. Présentation

CPLEX est, à la base, un solveur de programmes linéaires. Il est commercialisé par la société ILOG depuis la version 6.0. La dernière version, à ce jour, est la version 11.0. Les composants de la suite d'optimisation ILOG sont illustrés dans la Figure 1.

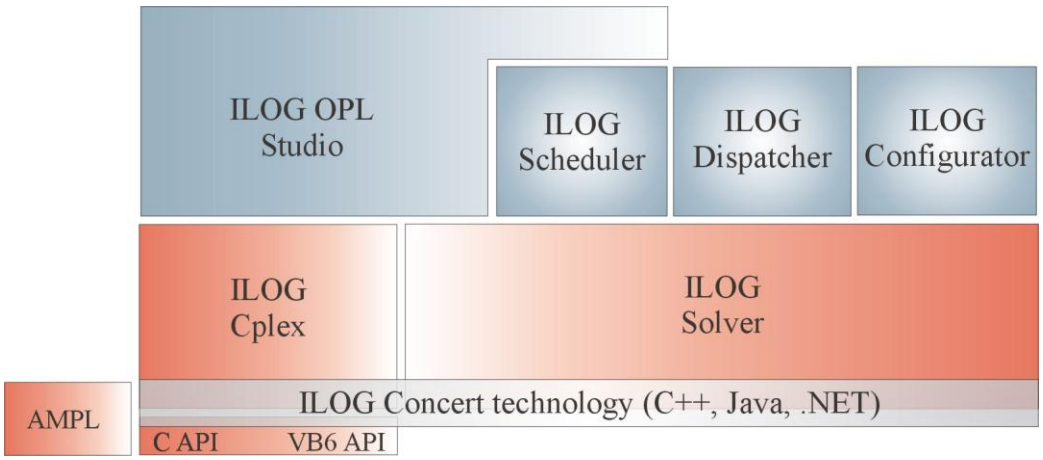


Figure 1: ILOG Optimization Suite

ILOG CPLEX – Le cœur du système résout des problèmes de programmation mathématique.

ILOG Solver – La partie principale du système résout des applications en utilisant la programmation par contraintes.

ILOG concert technology – Les bibliothèques contiennent les fonctionnalités du système. Elles sont disponibles pour les langages C++, Java et .NET.

ILOG Scheduler – Fournit des extensions pour résoudre des problèmes de planification.

ILOG Dispatcher – Fournit des extensions pour la résolution de problèmes de tournées de véhicules.

ILOG Configurator – Ce module contient des utilitaires pour l'optimisation des ventes en ligne (problèmes de e-commerce).

ILOG OPL Studio – OPL est un langage pour la modélisation des problèmes d'optimisation. Il interagit directement avec les modules ILOG Cplex, ILOG Solver et ILOG Dispacher.

AMPL – C'est un autre langage pour la modélisation, qui interagit avec le module ILOG CPLEX (AMPL a été développé par les laboratoires Bell).

C et VB6 APIs – Ce sont des bibliothèques pour des utilisateurs du langage C et de l'environnement VB6. Elles permettent de les interfacer avec le module ILOG CPLEX.

2. Historique

CPLEX a été initialement développé par l'équipe de Robert Bixby pour disposer d'un solveur performant pour résoudre des instances du problème de voyageur de commerce (TSP chez nos voisins d'outre-manche) de grande taille. Jusqu'à la version 6.0, il a été commercialisé par la société CPLEX. En 1996, cette société a été rachetée par ILOG pour étoffer son éventail de produits destinés à l'Aide à la Décision.

3. Capacités

Initialement, CPLEX est un solveur de programmes linéaires. A ce titre, il repose donc sur une implémentation performante du simplexe primal. Il dispose également du simplexe dual et du simplexe de réseau. Il peut aussi résoudre des programmes linéaires mixtes, en combinant le simplexe, le branch and bound et la génération de coupes. Depuis peu, il intègre également une technique à base de points intérieurs et peu traiter des problèmes quadratiques.
Actuellement, CPLEX est un des solveurs les plus performants disponibles, sinon le plus performant. Il peut ainsi traiter des problèmes contenant plusieurs dizaines de milliers de variables et plusieurs centaines de milliers de contraintes. Pour les problèmes mixtes, la limite est sensiblement plus basse, mais elle dépend grandement du type de problèmes et du modèle appliqué.

Les problèmes traités par la suite d'optimisation ILOG sont : les programmes linéaires et linéaires mixtes, les programmes quadratiques et quadratiques mixtes, les programmes avec contraintes quadratiques et avec contraintes quadratiques mixtes.

4. Modes d'utilisation

Il existe deux manières d'utiliser CPLEX. La première consiste à travailler de manière interactive en invoquant un interpréteur de commande dédié. La seconde consiste à appeler

directement les fonctionnalités du moteur depuis son propre code, que ce soit du C, du C++ ou du Java.

Mode interactif

Le mode interactif est le plus simple lorsqu'on aborde CPLEX. De manière similaire à d'autres produits scientifiques comme Matlab ou Mapple, il consiste à se placer sous un interpréteur de commandes dédié pour ensuite appeler les fonctionnalités du produit. On peut ainsi charger un fichier contenant un programme linéaire ou bien construire le problème « à la main » ; afficher un problème ; lancer la résolution ; récupérer la solution optimale et les informations associées à la résolution ; sauvegarder les données, les résultats ; modifier le problème. Nous allons présenter rapidement chacune de ces fonctionnalités :

L'interpréteur de commandes

Les commandes sont saisies directement dans l'environnement. L'interpréteur se charge ensuite de les analyser. Il n'est pas nécessaire de saisir le nom de la commande en entier et dans la plupart des cas les deux premières lettres suffisent. Par la suite, les lettres nécessaires à l'interprétation des commandes par l'environnement sont indiquées en gras. Il n'y a pas d'historique ni de rappel de commande. Par contre, toutes les actions effectuées dans l'environnement ainsi que leur résultat sont stockés dans un fichier spécial, « cplex.log ». On a ainsi la possibilité de consulter ce fichier après avoir quitté l'environnement pour effectuer une analyse a posteriori sans avoir besoin de relancer l'environnement.

Le lancement de l'interpréteur s'effectue en saisissant « cplex » dans un terminal. Voici ce que vous devriez obtenir.

```
[duhamel@lxr ~] cplex

CPLEX 8.000, licensed to "isima-aubiere", options: e m b q

Welcome to CPLEX Interactive Optimizer 8.0.0
  with Simplex, Mixed Integer & Barrier Optimizers
Copyright (c) ILOG 1997-2002
CPLEX is a registered trademark of ILOG

Type 'help' for a list of available commands.
Type 'help' followed by a command name for more
information on commands.

CPLEX>
```

Pour quitter l'environnement, « **q**uit » vous replace dans le terminal.

Aide en ligne

« **h**elp » permet d'obtenir une aide générale. « **h**elp sujet » pour obtenir de l'aide sur un sujet défini. Noter que cette aide est progressive, c'est-à-dire que lorsque le sujet est trop général, l'intepréteur propose plusieurs rubriques plus spécifiques.

```
CPLEX> help

add             add constraints to the problem
baropt          solve using barrier algorithm
change          change the problem
display         display problem, solution, or parameter settings
enter           enter a new problem
help            provide information on CPLEX commands
mipopt          solve a mixed integer program
netopt          solve the problem using network method
optimize        solve the problem
primopt         solve using the primal method
quit            leave CPLEX
read            read problem or basis information from a file
set             set parameters
tranopt         solve using the dual method
write           write problem or solution info. to a file
xecute          execute a command from the operating system

Enter enough characters to uniquely identify commands & options.
Commands can be entered partially (CPLEX will prompt you for
further information) or as a whole.

CPLEX>
```

Chargement et sauvegarde d'un fichier de données

« **r**ead nom_fichier format ». Il existe plusieurs formats supportés, les principaux sont le format LP et le format MPS (voir plus loin pour une description). Il n'est pas nécessaire de spécifier le format (chaînes « LP » ou « MPS ») lorsque l'extension du fichier est explicite (« .lp » ou « .mps »). On ne peut avoir qu'un seul problème actif à la fois, ce qui fait que le chargement d'un fichier détruit le problème précédent, s'il y en avait un.

A noter qu'un problème peut aussi être sauvegardé avec la commande « **w**rite nom_fichier format », selon le même principe.

Affichage d'un problème

La commande générale pour l'affichage est « **d**isplay ». Les arguments varient en fonction de ce qu'on souhaite afficher. Ainsi, si l'on veut afficher le problème en entier, la

commande est « **d**isplay **p**roblem **a**ll ». Si l'on ne veut afficher que la fonction objectif, la commande est « **d**isplay **p**roblem **c**onstraints 0 ».

Construction et modification d'un problème

A faire… change add enter

Optimisation

La commande générique pour lancer l'optimisation du problème courant est « **o**ptimize ». Selon le type du problème, le meilleur type d'algorithme sera choisi. On peut cependant imposer un algorithme, par exemple « **primopt** », « **dualopt** » ou « **netopt** » pour un programme linéaire continu, « **mipopt** » pour un programme linéaire mixte, etc…

Des informations concernant l'optimisation sont alors affichées. En principe, et sauf désactivation explicite, CPLEX commence par appliquer une phase de prétraitement destinée à réduire la taille du problème (élimination de contraintes et de variables redondantes, réduction de coefficients). Il applique ensuite l'algorithme choisi (affichage du nombre d'itérations et de la valeur courante entre autre) suivi éventuellement d'un branch and bound si le problème contient des variables entières (affichage du nombre de nœuds, des bornes sup et inf, etc…)

A la fin de l'algorithme, des informations générales concernant l'état final (solution optimale trouvée ou non, valeur, temps CPU, nombre d'itérations et nombre de nœuds) sont affichées.

```
CPLEX> opt
Tried aggregator 1 time.
No LP presolve or aggregator reductions.
Presolve time =    0.00 sec.

Iteration log . . .
Iteration:     1   Scaled dual infeas =             0.000000
Iteration:     2   Dual objective     =            46.444444

Dual simplex - Optimal:  Objective =    4.6444444444e+01
Solution time =    0.00 sec.  Iterations = 2 (1)

CPLEX>
```

```
CPLEX> opt
Tried aggregator 1 time.
Reduced MIP has 3 rows, 3 columns, and 9 nonzeros.
Presolve time =    0.00 sec.
MIP emphasis: balance optimality and feasibility
Root relaxation solution time =    0.00 sec.
```

```
        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer     Best Node    ItCnt
Gap

      0     0      -46.4444     1                   -46.4444        2
*     0+    0                   0      -45.3000      -46.4444        2
2.53%
                infeasible            -45.3000      Cuts:  3        2
0.00%

Mixed integer rounding cuts applied:  2

Integer optimal solution:  Objective =   -4.5300000000e+01
Solution time =    0.00 sec.  Iterations = 2  Nodes = 0

CPLEX>
```

Obtention de la solution et analyse du résultat

A l'issue de l'optimisation, l'affichage de la solution se fait avec la commande « **d**isplay **so**lution **v**ariables – » lorsqu'on souhaite afficher la valeur de chaque variable. Seules les variables non nulles sont affichées. L'affichage de la valeur de la solution se fait par « **d**isplay **so**lution **o**bjective». On peut également obtenir des informations sur les variables duales (« **d**isplay **so**lution **d**ual - », les variables d'écart (« **d**isplay **so**lution **s**lack - »), les coûts réduits (« **d**isplay **so**lution **r**educed – »), etc… On peut également obtenir une analyse de la sensibilité en demandant « **d**isplay **se**nsitivity » suivi de l'option (« **l**b », « **u**b », « **r**hs » ou « **o**bjective »).

Sauvegarde de la solution

De manière similaire à la sauvegarde d'un problème, la commande pour sauver une solution est « **w**rite fichier_solution format» où l'indication du format (« TXT » ou « BIN ») n'est nécessaire que si le fichier n'a pas l'extension « .txt » ou « .bin ». La solution sera stockée sous forme texte dans le premier cas et binaire dans le second cas.

Limites du mode interactif

Le mode interactif est adapté à une utilisation limitée, par exemple la résolution d'un problème isolé et l'obtention de la solution optimale. En revanche, il est totalement inadapté à la génération de gros problèmes, à la résolution itérative (génération de colonnes, de coupes, relaxation lagrangienne, décomposition de Dantzig Wolfe par exemple), au traitement

automatique ou à l'intégration dans un progiciel. Dans ces cas-ci, il est préférable d'appeler les fonctionnalités de CPLEX à l'intérieur de son propre code.

Interfaçage avec du C++ : concerto library

Consulter le manuel disponible dans le répertoire du cours.

5. Représentation interne des données

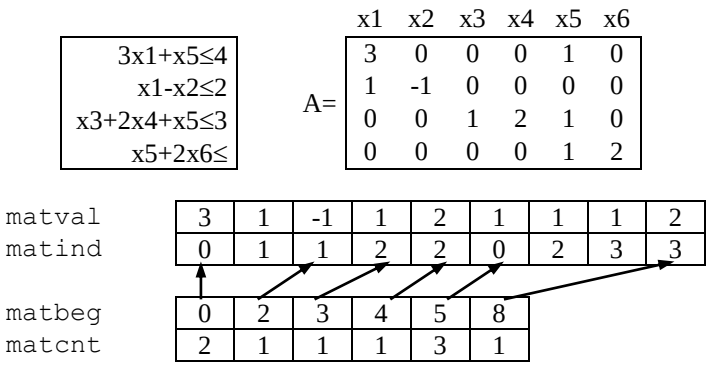
CPLEX utilise en interne une représentation en matrice creuse de la matrice des contraintes. Cette représentation est orientée colonnes et nécessite plusieurs vecteurs :

- double matval[]
- int matind[]
- int matbeg[]
- int matcnt[]

Les vecteurs matval[] et matind[] vont stocker les coefficients des variables dans chaque contrainte. Leur dimension est donc au moins le nombre de coefficients non nuls dans la matrice. Le vecteur matval[] va conserver les coefficients tandis que le vecteur matind[] va conserver les indices des contraintes associées. Les vecteurs matbeg[] et matind[] permettent de définir les blocs dans matval[] et matind[] qui correspondent à chaque colonne. Leur dimension est au moins le nombre de variables du problème. Le vecteur matbeg[] conserve l'indice du premier coefficient dans matval[] et matind[] pour chaque colonne. Le vecteur matcnt[] contient le nombre de coefficients dans matval[] et matind[] pour chaque colonne. Ainsi, matcnt[] est redondant puisqu'on a

$$matcnt[i] = matbeg[i+1] - matbeg[i]$$

L'exemple suivant permet de se faire une idée de la manière de stocker les coefficients :



Notons qu'il existe aussi une représentation en matrice creuse orientée ligne. Le principe est le même. Le nom des vecteurs change, on ajoute un 'r' devant (`rmatval[]`, `rmatind[]`, `rmatbeg[]` et `rmatcnt[]`).

6. Format de fichiers LP

Ce format de fichiers repose une approche orientée lignes, radicalement différente de l'approche orientée colonnes adoptée dans le format MPS. Ici, le programme linéaire est présenté ligne par ligne, c'est-à-dire contrainte par contrainte. Ce format est donc nettement plus intuitif puisque sa manière de présenter les données se rapproche de la forme canonique et de la forme standard.

En principe, le format LP est reconnu maintenant par la plupart des solveurs, même si le format MPS reste prépondérant par le fait qu'il est davantage reconnu comme le standard. Ainsi, CPLEX reconnaît les deux formats, mais il consomme davantage de mémoire pour lire un fichier LP parce qu'il doit ensuite le convertir en orienté colonnes. Cette conversion s'accompagne d'un effet de bord sur l'ordre de stockage des variables, elles sont classées dans l'ordre de lecture dans le fichier.

Voyons maintenant les principaux blocs syntaxiques :

- Description du problème (facultatif) : il n'existe pas de mot clé pour définir cette section, mais on peut la simuler à l'aide d'un bloc de commentaires. Les commentaires débutent par le caractère '\'. Tout ce qui suit sur la ligne sera alors traité comme un commentaire. Pour avoir un commentaire sur plusieurs lignes, il faut placer '\' au début de chaque ligne. L'usage veut qu'on indique au moins le nom du problème ainsi qu'éventuellement diverses informations utiles telles que sa valeur optimale, une borne supérieure ou inférieure, ou encore son origine.

```
\problem: transport1.lp
\comment: premiere version
\creation: 18/09/2001
\upper bound: 2113
\lower bound: 1954
```

- Fonction objectif (obligatoire) : le bloc décrivant la fonction objectif est composé de deux éléments. Le premier définit le type d'optimisation que l'on recherche, à savoir une

maximisation (`Maximize` ou `Max`) ou une minimisation (`Minimize` ou `Min`), sans distinction de majuscule ou de minuscule. Vient ensuite la description de la fonction objectif. Elle débute par un premier élément optionnel qui est le nom de la fonction objectif. Elle se poursuit par l'expression algébrique de la fonction objectif. Le nom doit être d'un seul bloc et doit suivre les conventions de nommage. Il doit se terminer par ' :'. Si l'expression de la fonction objectif nécessite plusieurs lignes, on peut couper n'importe où, du moment que l'on ne brise pas les lexèmes. Aucun caractère de coupe n'est nécessaire. La fonction objectif peut comporter des termes constants. Le nom de chaque variable est libre est suit la convention de nommage des variables du C. La seule différence notable est qu'il ne peut débuter par la lettre 'e' ou 'E', réservée pour l'expression des nombres sous forme exponentielle (« `e-10` » ou « `E3` » par exemple).

```
Maximize
cout: 3x1 + x3 - 6.5f1_2
```

- Bloc des contraintes (obligatoire) : il débute par la balise « `subject To` » (ou « `Such That` » ou « `st` » ou « `s.t.` » ou « `st.` ») sans distinction majuscule ou minuscule. Elle est suivie des contraintes, avec au plus une contrainte par ligne. Chaque contrainte est exprimée sous forme algébrique en plaçant les variables dans le terme de gauche et la constante dans le terme de droite. Le sens des contraintes peut être « = », « <= », « >= », à la rigueur « < » ou « > ». Une contrainte ne peut être de type diségalité « `!=` ». Chaque contrainte peut avoir un nom, auquel cas celui-ci est placé au tout début de la ligne, comme pour la fonction objectif. Une contrainte peut être exprimée sur plusieurs lignes, du moment que la césure ne coupe pas de lexème.

```
Subject to
conserv_flot_1: f2_1 + f3_1 + f4_1 - f1_2 - f1_3 - f1_4 = -10
conserv_flot_2: f1_2 + f3_2 + f4_2 - f2_1 - f2_3 - f2_4 = 0
conserv_flot_3: f1_3 + f2_3 + f4_3 - f3_1 - f3_2 - f3_4 = 10
conserv_flot_4: f1_4 + f2-4 + f3_4 - f4_1 - f4_2 - f4_3 = 0
capa_1:  f2_1 + f3_1 + f4_1 <= 5
```

- Bornes sur les variables (facultatif) : ce bloc débute par la balise « `Bounds` » ou « `bound` » sans distinction de majuscule ou minuscule. Chaque variable peut être bornée inférieurement et supérieurement. Les bornes peuvent être décrites en une seule partie « `a <= x <= b` » ou bien sur deux lignes « `a <= x` » et « `x <= b` ». Le terme pour l'infini est « `infinity` » ou « `inf` ». Par défaut, les variables sont supposées être

positives « `0 <= x <= inf` ». Pour définir une variable libre, on la définit donc explicitement avec une borne inférieure infinie « `-inf <= x` » ou alors on dit qu'elle est libre « `x free` ».

```
Bounds
x1 <= 10
4 <= x2 <= 6
-10 <= x3
x4 free
```

– Type des variables (facultatif) : après les bornes, on peut spécifier la nature des variables. Par défaut, elles sont considérées comme continues et n'ont donc pas besoin de voir leur type défini. Lorsqu'elles sont entières, on distingue deux cas : les mot clé « `binary` » ou « `binaries` » ou « `bin` » pour les variables de décision, à valeur dans {0,1} et les mots-clés « `general` », « `generals` » ou encore « `gen` » pour les autres variables entières. Il n'y a pas de distinction majuscule ou minuscule pour les mots clé. Les variables de décision reçoivent automatiquement des bornes « `0 <= x <= 1` ». Il n'est donc pas nécessaire de les indiquer de manière explicite. Dans chacun des deux types de bloc, les variables sont indiquées les unes à la suite des autres, séparées par des espaces.

```
Binaries
x1 x2 x3

Generals
x4
```

– Fin de fichier (recommandé) : le fichier doit en principe se terminer par le mot clé « `End` » dans distinction majuscule ou minuscule. Il doit être placé sur une ligne à part.

Notons que la distinction majuscule/minuscule existe pour le nom des variables, de sorte que « `x2` » et « `X2` » sont deux variables différentes. Enfin, voici un exemple de programme linéaire et de format LP associé.

Maximiser	$4x1 + 5x2 + 3x3$
s.c.	
	$x1 + x2 + 2x3 \leq 20$
	$15x1 + 6x2 - 5x3 \leq 50$
	$x1 + 3x2 + 5x3 \leq 30$
	$x3 \leq 5.5$
	$x1, x2 \geq 0$
	$x3 \in N$

```
\problem: exemple

Maximize
4x1 + 5x2 + 3x3

Subject To
x1 + x2 + 2x3 ≤ 20
15x1 + 6x2 −5x3 ≤ 50
x1 + 3x2 + 5x3 ≤ 30

Bounds
x3 ≤ 5.5

Generals
x3

End
```

7. Format de fichiers MPS

Ce format de fichiers a été développé par IBM dans les années 60/70. Il constitue le format standard pour stocker les programmes linéaires. Sa particularité est de présenter le problème par rapport à ses variables (il est donc orienté colonnes). Pour chaque variable, MPS décrit son type, sa contribution dans la fonction objectif ainsi que dans chaque variable. Même si cette approche n'est pas très intuitive, cette approche présente certains avantages dans le stockage des données, en particulier pour les solveurs dont la représentation interne est orientée colonnes. En effet, on peut montrer un gain dans les calculs en stockant en colonne plutôt qu'en ligne.

Dans les anciennes versions de MPS, de manière analogue au FORTRAN, les données devaient de trouver à des emplacements fixés dans les lignes. Ce n'est en principe plus le cas désormais et l'on peut séparer les différentes données sur la ligne par autant de caractères de séparation qu'on le souhaite. Le caractère '*' en début de ligne indique que le reste de la ligne sera un commentaire.

Tout comme pour le format LP, le format MPS range les données dans différentes sections. Chacune débute par une balise spécifique :

– Balise « NAME » (facultative) : elle débute le fichier et elle est suivie du nom du problème.

- Balise « OBJSENSE » (facultative) : elle définit le sens de l'optimisation. La ligne suivante contient alors « MAX » ou « MIN », en fonction de ce qu'on recherche. Par défaut, le sens est la minimisation.

- Balise « OBJNAME » (facultative) : elle indique le nom de la ligne (dans la section ROWS) qui jouera le rôle de fonction objectif. La ligne suivante contient le nom. Par défaut, c'est la première ligne de la section ROWS qui sera la fonction objectif.

- Balise « ROWS » (obligatoire) : elle décrit le nom et le sens de chaque contrainte du problème, à raison d'une par ligne. Chaque ligne contient deux champs, le type de contrainte et le nom. Le type peut être 'N' pour une contrainte libre (c'est-à-dire la fonction objectif en principe), 'G' pour une contrainte '≥', 'L' pour une contrainte '≤' et 'E' pour une contrainte '='. Le nom de la contrainte suit les conventions classiques de nommage. La fonction objectif est généralement nommée « obj ».

- Balise « COLUMNS » (obligatoire) : elle permet d'indiquer la contribution de chaque variable dans les contraintes et la fonction objectif. Chaque ligne débute avec un nom de variable, suivie d'une ou deux contributions. Une contribution consiste en le nom de la contrainte ou de la fonction objectif suivi du coefficient lié à la variable. On ne peut avoir plus de deux contributions par ligne. Il est inutile d'indiquer les contributions nulles.

- Balise « RHS » (obligatoire) : elle permet de définir le second membre des contraintes. Chaque ligne débute par « rhs » suivie d'un ou deux seconds membres. Un second membre est constitué par le nom de la contrainte puis la valeur du second membre. On ne peut avoir plus de deux seconds membres par ligne.

- Balise « RANGES » (facultative) : elle permet de spécifier les intervalles pour les contraintes bornées supérieurement et inférieurement. Comme la contrainte possède déjà une des bornes (dans la section RHS), nous n'avons besoin que de spécifier la seconde borne. Elle se définit de manière relative par rapport à la première borne. Pour chaque contrainte bornée, on indique ainsi le nom de la contrainte suivi de la largeur de l'intervalle (négative si on avait défini une borne supérieure et positive si on avait défini la borne inférieure). On ne peut mettre plus de deux bornes par ligne.

– Balise « BOUNDS » (facultative) : elle définit les bornes pour les variables, à raison d'une borne par ligne. Chaque ligne commence par le type de borne, « UP » pour une borne supérieure, « LO » pour une borne inférieure, « FX » pour une variable dont la valeur est fixée, « FR » pour une variable libre. On spécifie ensuite « bnd » suivi du nom de la variable et de la valeur de la borne. Par défaut, une variable est supposée positive. Les variables entières ont un type de borne spécifique, « BV » pour les variables binaires, « LI » pour la borne inférieure d'un variable entière et « UI » pour la borne supérieure d'une variable entière Par défaut, une variable entière n'a pas de borne supérieure, mais pour indiquer qu'elle est entière, il faut spécifier « LI » suivi de la valeur 0. Notons qu'il existe une autre manière de spécifier les variables entière, à l'aide de marqueurs.

– Balise « ENDATA » (obligatoire) : elle indique la fin des données. En principe, elle constitue la dernière ligne du fichier.

Si l'on reprend l'exemple vu dans la section précédente, le format MPS est le suivant :

Maximiser	$4x1 + 5x2 + 3x3$
s.c.	
	$x1 + x2 + 2x3 \leq 20$
	$15x1 + 6x2 - 5x3 \leq 50$
	$x1 + 3x2 + 5x3 \leq 30$
	$x3 \leq 5.5$
	$x1, x2 \geq 0$
	$x3 \in N$

format LP	format MPS

`\problem: exemple`
 	
`Maximize`
`4x1 + 5x2 + 3x3`
 	
`Subject To`
`x1 + x2 + 2x3 <= 20`
`15x1 + 6x2 -5x3 <= 50`
`x1 + 3x2 + 5x3 <= 30`
 	
`Bounds`
`x3 <= 5.5`
 	
`Generals`
`x3`
 	
`End`
	`NAME exemple`
	`ROWS`
	` N obj`
	` L c1`
	` L c2`
	` L c3`
	`COLUMNS`
	` x1 obj -4 c1 1`
	` x1 c2 15 c3 1`
	` x2 obj -5 c1 1`
	` x2 c2 6 c3 3`
	` x3 obj -3 c1 2`
	` x3 c2 -5 c3 5`
	`RHS`
	` rhs c1 20 c2 50`
	` rhs c3 30`
	`BOUNDS`
	` LI bnd x2 0`
	` UP bnd x3 5.5`
	`ENDATA`

Notons que les termes « rhs » et « bnd » ne sont pas imposés. Ils définissent seulement le nom du vecteur second membre et de borne respectivement.