Anexo

Anexo 1. Programas para solucionar el problema "Cutting Stock" a través del método de generación de columnas.

A continuación se anexan los códigos de los diferentes programas utilizados para solucionar el problema "Cutting Stock" a través del método de generación de columnas.

El Anexo 1.1 es el programa con el algoritmo tradicional, en el Anexo 1.2 se incluye la heurística para solucionar el problema de la mochila, en el Anexo 1.3 se incluye la heurística para obtener un patrón de corte inicial, y llegar a la solución optima con menos iteraciones, y en el Anexo 1.4 se encuentra el programa que utiliza ambas heurísticas.

Anexo 1.1 Programa Algoritmo para solucionar el problema "*Cutting Stock*" a través del método de generación de columnas (Fair Isaac Corporation, 2008).

```
model Cuttingstock
uses "mmxprs"
uses "mmsystem"
        forward procedure generacion_columnas
        forward function knapsack(C:array(range) of real, A:array(range) of real,B:real,
mejor_x:array(range) of integer,pass: integer): real
        forward procedure nuevo_patron(columna: integer)
        forward procedure solucion
        declarations
                NO_FINALS: integer
                RAW: real
        end-declarations
        initializations from "cuttingstock.dat"
                NO_FINALS
                RAW
        end-initializations
        declarations
                FINALS = 1..NO_FINALS
                EPS = 1e-6
                RP: range
                ANCHO: array(FINALS) of real
                DEMANDA: array(FINALS) of integer
                BASICA: array(FINALS,FINALS) of integer
                rollos_usados: array(RP) of mpvar
                solusados: array(RP) of real
                Dem: array(FINALS) of linctr
                MinRollos: linctr
                KnapsackRestricciones, ObjetivoKnapsack: linctr
```

```
            x: array(FINALS) of mpvar
            npase: integer
            matriz: array(FINALS,RP) of integer
            starttime,endtime: real
        end-declarations
        initializations from "cuttingstock.dat"
            ANCHO
            DEMANDA
        end-initializations
        starttime:= gettime
        forall(j in FINALS) BASICA(j,j) := floor(RAW/ANCHO(j))
        writeln("La Matriz Basica es: ")
        writeln
        forall(i in FINALS)do
            write("Ancho= ",ANCHO(i), ": ")
            forall(j in FINALS )do
                write(BASICA(i,j),"      ")
            end-do
            writeln
        end-do
        forall(j in FINALS) do
            create(rollos_usados(j))
            rollos_usados(j) is_integer
            rollos_usados(j) <= integer(ceil(DEMANDA(j)/BASICA(j,j)))
            forall(i in FINALS)do
                create(matriz(j,i))
                matriz(j,i):=BASICA(j,i)
            end-do
        end-do
        MinRollos:= sum(j in FINALS) rollos_usados(j)
        forall(i in FINALS) do
            Dem(i):= sum(j in FINALS) BASICA(i,j) * rollos_usados(j) >=
DEMANDA(i)
        end-do
        generacion_columnas
        solucion
        endtime:=gettime-starttime
        writeln("El tiempo para solucionarlo es: ",endtime)
        procedure generacion_columnas
        declarations
            dualdem: array(FINALS) of real
            mejor_x: array(FINALS) of integer
            mejor_z,dw,valor_objetivo: real
            bas: basis
```

```
end-declarations
        defcut:=getparam("XPRS_CUTSTRATEGY")
        setparam("XPRS_CUTSTRATEGY", 0)
        setparam("XPRS_PRESOLVE", 0)
        setparam("zerotol", EPS)
        nColum:=NO_FINALS
        npase:=1
        while(true) do
                minimize(XPRS_LIN, MinRollos)
                savebasis(bas)
                valor_objetivo:= getobjval
                writeln
                writeln("Con un costo de: ",valor_objetivo," rollos")
                forall(j in FINALS)solusados(j):=getsol(rollos_usados(j))
                forall(i in FINALS)dualdem(i):=getdual(Dem(i))
                mejor_z:= knapsack(dualdem, ANCHO, RAW, mejor_x, npase) -
1.0
                writeln
                writeln
                writeln("En el pase ", npase, ": ")
                if mejor_z = 0 then
                        writeln("     No se encontro una mejor solución\n")
                        break
                else
                        nColum+=1
                        create(rollos_usados(nColum))
                        forall(i in FINALS)do
                                create(matriz(i,nColum))
                                matriz(i,nColum):=mejor_x(i)
                        end-do
                        nuevo_patron(nColum)
                rollos_usados(nColum) is_integer
                MinRollos+= rollos_usados(nColum)
                dw:=0
                forall(i in FINALS)
                if mejor_x(i) > 0 then
                Dem(i)+= mejor_x(i)*rollos_usados(nColum)
                dw:= maxlist(dw, ceil(DEMANDA(i)/mejor_x(i) ))
        end-if
                rollos_usados(nColum) <= dw
                loadprob(MinRollos)
                loadbasis(bas)
                end-if
                npase+=1
```

```
            end-do
            setparam("XPRS_CUTSTRATEGY", defcut)
            setparam("XPRS_PRESOLVE", 1)
        end-procedure
        function knapsack(C:array(range) of real, A:array(range) of real,
B:real,mejor_x:array(range) of integer, pass: integer):real
            forall(j in FINALS) sethidden(Dem(j), true)
            KnapsackRestricciones := sum(j in FINALS) A(j)*x(j) <= B
            ObjetivoKnapsack := sum(j in FINALS) C(j)*x(j)
            if(pass=1) then
                forall(j in FINALS) x(j) is_integer
            end-if
            maximize(ObjetivoKnapsack)
            returned:=getobjval
            forall(j in FINALS) do
                mejor_x(j):=round(getsol(x(j)))
            end-do
            KnapsackRestricciones := 0
            ObjetivoKnapsack := 0
            forall(j in FINALS) sethidden(Dem(j), false)
        end-function
        procedure nuevo_patron(columna: integer)
            declarations
            total: real
            end-declarations
            total:=0
            writeln
            writeln("El Nuevo Patron de corte es: ")
            writeln
            forall(i in FINALS)do
                writeln(ANCHO(i)," : ",matriz(i,columna))
                total+=ANCHO(i)*matriz(i,columna)
            end-do
            writeln("Cantidad de rollo usado: ",total)
        end-procedure
        procedure solucion
            minimize(MinRollos)
            writeln("La mejor solución entera es: ", getobjval, " rollos")
            writeln
            writeln(" Como se deben de cortar estos rollos es de la siguiente manera:
")
            writeln
            forall(i in RP|getsol(rollos_usados(i))<>0) do
                write(getsol(rollos_usados(i)))
```

```
                    if(i<NO_FINALS)then
                    write(", rollos con el patron de corte de la columna, ",i)
                    else
                        write(", rollos con el patron de corte del pase, ",i-
NO_FINALS)
                    end-if
                    writeln
                end-do
        writeln
        end-procedure
end-model
```

Anexo 1.2 Programa para solucionar el problema “Cutting Stock” a través del método de generación de columnas. Utilizando una heurística para solucionar el problema de la mochila.

```
model Cuttingstock
uses "mmxprs"
uses "mmsystem"
        forward procedure generacion_columnas
        forward function knapsack(C:array(range) of real, A:array(range) of real,B:real,
mejor_x:array(range) of integer,pass: integer): real
        forward function heuristicaknapsack(C:array(range) of real, A:array(range) of
real,B:real,mejor_x:array(range) of integer,pass: integer): real
        forward procedure nuevo_patron(columna:integer)
        forward procedure solucion
        declarations
            NO_FINALS: integer
            RAW: real
        end-declarations
        initializations from "cuttingstock.dat"
            NO_FINALS
            RAW
        end-initializations
        declarations
            FINALS = 1..NO_FINALS
            EPS = 1e-6
            RP: range
            ANCHO: array(FINALS) of real
            DEMANDA: array(FINALS) of integer
            BASICA: array(FINALS,FINALS) of integer
            rollos_usados: array(RP) of mpvar
            solusados: array(RP) of real
            Dem: array(FINALS) of linctr
```

```
            MinRollos: linctr
            KnapsackRestricciones, ObjetivoKnapsack: linctr
            x: array(FINALS) of mpvar
            matriz: array(FINALS,RP) of integer
            mejor_z,valor_objetivo:real
            npase,flag: integer
        end-declarations
        initializations from "cuttingstock.dat"
            ANCHO
            DEMANDA
        end-initializations
        starttime:= gettime
        forall(j in FINALS) BASICA(j,j) := floor(RAW/ANCHO(j))
        writeln("La Matriz Basica es: ")
        writeln
        forall(i in FINALS)do
            write("Ancho= ",ANCHO(i), ": ")
            forall(j in FINALS )do
                write(BASICA(i,j),"      ")
            end-do
            writeln
        end-do
        forall(j in FINALS) do
            create(rollos_usados(j))
            rollos_usados(j) is_integer
            rollos_usados(j) <= integer(ceil(DEMANDA(j)/BASICA(j,j)))
        end-do
        MinRollos:= sum(j in FINALS) rollos_usados(j)
        forall(i in FINALS) do
            Dem(i):= sum(j in FINALS) BASICA(i,j) * rollos_usados(j) >=
DEMANDA(i)
        end-do
        generacion_columnas
        minimize(MinRollos)
        solucion
        writeln("El tiempo para solucionarlo es: ",gettime-starttime)
        procedure generacion_columnas
        declarations
            dualdem: array(FINALS) of real
            mejor_x: array(FINALS) of integer
            dw: real
            bas: basis
        end-declarations
            defcut:=getparam("XPRS_CUTSTRATEGY")
```

```
setparam("XPRS_CUTSTRATEGY", 0)
setparam("XPRS_PRESOLVE", 0)
setparam("zerotol", EPS)
nColum:=NO_FINALS
npase:=1
while(true) do
    minimize(XPRS_LIN, MinRollos)
    savebasis(bas)
    valor_objetivo:=getobjval
    writeln
    writeln("Con un costo de: ",valor_objetivo," rollos")
    forall(j in FINALS)solusados(j):=getsol(rollos_usados(j))
    forall(i in FINALS)dualdem(i):=getdual(Dem(i))
    mejor_z:= heuristicaknapsack(dualdem, ANCHO, RAW,
mejor_x, npase) - 1.0
    if(flag=1)then
        mejor_z:= knapsack(dualdem, ANCHO, RAW, mejor_x,
npase) - 1.0
    end-if
    writeln
    writeln
    writeln("En el pase ", npase, ": ")
    if mejor_z = 0 then
        writeln("     No se encontro una mejor solución\n")
        break
    else
        nColum+=1
        create(rollos_usados(nColum))
        forall(i in FINALS)do
            create(matriz(i,nColum))
            matriz(i,nColum):=mejor_x(i)
        end-do
        nuevo_patron(nColum)
    rollos_usados(nColum) is_integer
    MinRollos+= rollos_usados(nColum)
    dw:=0
    forall(i in FINALS)
    if mejor_x(i) > 0 then
    Dem(i)+= mejor_x(i)*rollos_usados(nColum)
    dw:= maxlist(dw, ceil(DEMANDA(i)/mejor_x(i) ))
end-if
    rollos_usados(nColum) <= dw
    loadprob(MinRollos)
    loadbasis(bas)
```

```
                end-if
                npase+=1
            end-do
            setparam("XPRS_CUTSTRATEGY", defcut)
            setparam("XPRS_PRESOLVE", 1)
        end-procedure
        function heuristicaknapsack(C:array(range) of real, A:array(range) of real,
B:real,mejor_x:array(range) of integer, pass: integer):real
        declarations
            anchobarra: real
            peso: array(FINALS,1..2) of real
            z_crit: real
            jasterisco: integer
            limite_x,guia: array(FINALS) of integer
        end-declarations
            forall(j in FINALS)do
                peso(j,1):=C(j)/A(j)
                peso(j,2):=j
                limite_x(j):=floor(B/A(j))
            end-do
            forall(i in 1..(NO_FINALS-1))do
                forall(j in i+1..(NO_FINALS)) do
                    if(peso(i,1)<peso(j,1))then
                        q:=peso(i,1)
                        e:=floor(peso(i,2))
                        peso(i,1):=peso(j,1)
                        peso(i,2):=peso(j,2)
                        peso(j,1):=q
                        peso(j,2):=e
                    end-if
                end-do
            end-do
            forall(j in FINALS)do
                guia(j):=floor(peso(j,2))
            end-do
            anchobarra:=B
            z_crit:=0
            flag:=0
            jasterisco:=1
            forall(j in FINALS) do
                if(floor(anchobarra/A(guia(j)))<limite_x(guia(j)))then
                    mejor_x(guia(j)):=floor(anchobarra/A(guia(j)))
                    anchobarra:=anchobarra-A(guia(j))*mejor_x(guia(j))
                    z_crit:=z_crit+C(guia(j))*mejor_x(guia(j))
```

```
                        else
                                mejor_x(guia(j)):=limite_x(guia(j))
                                anchobarra:=anchobarra-
A(guia(j))*mejor_x(guia(j))
                                z_crit:=z_crit+C(guia(j))*mejor_x(guia(j))
                        end-if

        if(limite_x(guia(j))*C(guia(j))>limite_x(guia(jasterisco))*C(guia(jasterisco)))the
n
                        jasterisco:=guia(j)
                end-if
        end-do
        if(limite_x(jasterisco)*C(jasterisco)>z_crit)then
                z_crit:=limite_x(jasterisco)*C(jasterisco)
                forall(i in FINALS) do
                        mejor_x(i):=0
                end-do
                mejor_x(jasterisco):=limite_x(jasterisco)
        end-if
        if(z_crit<=1)then
                forall(j in FINALS)mejor_x(j):=0
                flag:=1
                returned:=0
                else
                        returned:=z_crit
        end-if
end-function
function knapsack(C:array(range) of real, A:array(range) of real,
B:real,mejor_x:array(range) of integer, pass: integer):real
        forall(j in FINALS) sethidden(Dem(j), true)
        KnapsackRestricciones := sum(j in FINALS) A(j)*x(j) <= B
        ObjetivoKnapsack := sum(j in FINALS) C(j)*x(j)
        forall(j in FINALS) x(j) is_integer
        maximize(ObjetivoKnapsack)
        returned:=getobjval
        forall(j in FINALS) do
                mejor_x(j):=round(getsol(x(j)))
        end-do
        KnapsackRestricciones := 0
        ObjetivoKnapsack := 0
        forall(j in FINALS) sethidden(Dem(j), false)
end-function
procedure nuevo_patron(columna: integer)
        declarations
```

```
            total: real
            end-declarations
            total:=0
            writeln
            writeln("El Nuevo Patron de corte es: ")
            writeln
            forall(i in FINALS)do
                writeln(ANCHO(i)," : ",matriz(i,columna))
                total+=ANCHO(i)*matriz(i,columna)
            end-do
            writeln("Cantidad de rollo usado: ",total)
        end-procedure
        procedure solucion
        writeln("La mejor solución entera es: ", getobjval, " rollos")
        writeln
        writeln(" Como se deben de cortar estos rollos es de la siguiente manera: ")
        writeln
        forall(i in RP|getsol(rollos_usados(i))<>0) do
            write(getsol(rollos_usados(i)))
            if(i<NO_FINALS)then
            write(", rollos con el patron de corte de la columna, ",i)
            else
                write(", rollos con el patron de corte del pase, ",i-NO_FINALS)
            end-if
            writeln
        end-do
        writeln
        end-procedure
end-model
```

Anexo 1.3 Programa para solucionar el problema “Cutting Stock” a través del método de generación de columnas. Utilizando una heurística para obtener un patrón de corte inicial.

```
model Cuttingstock
uses "mmxprs"
uses "mmsystem"
        forward procedure generacion_columnas
        forward function knapsack(C:array(range) of real, A:array(range) of real,B:real,
mejor_x:array(range) of integer,pass: integer): real
        forward procedure nuevo_patron(columna: integer)
        forward procedure solucion
        forward procedure patron_inicial
        declarations
```

```
        NO_FINALS: integer
        RAW: real
end-declarations
initializations from "cuttingstock.dat"
        NO_FINALS
        RAW
end-initializations
declarations
        FINALS = 1..NO_FINALS
        EPS = 1e-6
        RP: range
        ANCHO,ANCHOf,x_fin: array(FINALS) of real
        DEMANDA: array(FINALS) of integer
        BASICA: array(FINALS,FINALS) of integer
        rollos_usados: array(RP) of mpvar
        solusados: array(RP) of real
        Dem: array(FINALS) of linctr
        MinRollos: linctr
        KnapsackRestricciones, ObjetivoKnapsack: linctr
        matriz: array(FINALS,RP) of integer
        x: array(FINALS) of mpvar
        mejor_z,valor_objetivo:real
        npase:integer
end-declarations
initializations from "cuttingstock.dat"
        ANCHO
        DEMANDA
end-initializations
starttime:= gettime
patron_inicial
forall(j in FINALS) do
        forall(i in FINALS)do
                create(matriz(j,i))
                matriz(j,i):=BASICA(j,i)
        end-do
end-do
generacion_columnas
minimize(MinRollos)
solucion
writeln("El tiempo para solucionarlo es: ",gettime-starttime)
procedure generacion_columnas
declarations
        dualdem: array(FINALS) of real
        mejor_x: array(FINALS) of integer
```

```
        dw: real
        bas: basis
    end-declarations
        defcut:=getparam("XPRS_CUTSTRATEGY")
        setparam("XPRS_CUTSTRATEGY", 0)
        setparam("XPRS_PRESOLVE", 0)
        setparam("zerotol", EPS)
        nColum:=NO_FINALS
        npase:=1
        while(true) do
            minimize(XPRS_LIN, MinRollos)
            savebasis(bas)
            valor_objetivo:= getobjval
            writeln
            writeln("Con un costo de: ",valor_objetivo," rollos")
            forall(j in FINALS)solusados(j):=getsol(rollos_usados(j))
            forall(i in FINALS)dualdem(i):=getdual(Dem(i))
            mejor_z:= knapsack(dualdem, ANCHO, RAW, mejor_x, npase) -
1.0
            writeln
            writeln
            writeln("En el pase ", npase, ": ")
            if mejor_z = 0 then
                writeln("     No se encontro una mejor solución\n")
                break
            else
                nColum+=1
                create(rollos_usados(nColum))
                forall(i in FINALS)do
                    create(matriz(i,nColum))
                    matriz(i,nColum):=mejor_x(i)
                end-do
                nuevo_patron(nColum)
            rollos_usados(nColum) is_integer
            MinRollos+= rollos_usados(nColum)
            dw:=0
            forall(i in FINALS)
            if mejor_x(i) > 0 then
            Dem(i)+= mejor_x(i)*rollos_usados(nColum)
            dw:= maxlist(dw, ceil(DEMANDA(i)/mejor_x(i) ))
        end-if
            rollos_usados(nColum) <= dw
            loadprob(MinRollos)
            loadbasis(bas)
```

```
                end-if
                npase+=1
            end-do
            setparam("XPRS_CUTSTRATEGY", defcut)
            setparam("XPRS_PRESOLVE", 1)
        end-procedure
        function knapsack(C:array(range) of real, A:array(range) of real,
B:real,mejor_x:array(range) of integer, pass: integer):real
            forall(j in FINALS) sethidden(Dem(j), true)
            KnapsackRestricciones := sum(j in FINALS) A(j)*x(j) <= B
            ObjetivoKnapsack := sum(j in FINALS) C(j)*x(j)
            if(pass=1) then
                forall(j in FINALS) x(j) is_integer
            end-if
            maximize(ObjetivoKnapsack)
            returned:=getobjval
            forall(j in FINALS) do
                mejor_x(j):=round(getsol(x(j)))
            end-do
            KnapsackRestricciones := 0
            ObjetivoKnapsack := 0
            forall(j in FINALS) sethidden(Dem(j), false)
        end-function
        procedure nuevo_patron(columna: integer)
            declarations
            total: real
            end-declarations
            total:=0
            writeln
            writeln("El Nuevo Patron de corte es: ")
            writeln
            forall(i in FINALS)do
                writeln(ANCHO(i)," : ",matriz(i,columna))
                total+=ANCHO(i)*matriz(i,columna)
            end-do
            writeln("Cantidad de rollo usado: ",total)
        end-procedure
        procedure solucion
        writeln("La mejor solucón entera es: ", getobjval, " rollos")
        writeln
        writeln(" Como se deben de cortar estos rollos es de la siguiente manera: ")
        writeln
        forall(i in RP|getsol(rollos_usados(i))<>0) do
            write(getsol(rollos_usados(i)))
```

```
        if(i<NO_FINALS)then
        write(", rollos con el patron de corte de la columna, ",i)
        else
                write(", rollos con el patron de corte del pase, ",i-NO_FINALS)
        end-if
        writeln
end-do
writeln
end-procedure
procedure patron_inicial
        declarations
                No_Columnas: array(FINALS) of integer
                numer: array(FINALS) of integer
                a_inicial: array(FINALS) of integer
                posx: real
                posy: integer
                b_prima:array(FINALS) of integer
                x_prima,demprima:array(FINALS) of real
                crudo: real
                posxmin,c:integer
        end-declarations
        forall(i in 1..(NO_FINALS-1))do
                forall(j in i+1..(NO_FINALS)) do
                        if(ANCHO(i)<ANCHO(j))then
                        posx:=ANCHO(i)
                        posy:=DEMANDA(i)
                        ANCHO(i):=ANCHO(j)
                        DEMANDA(i):=DEMANDA(j)
                        ANCHO(j):=posx
                        DEMANDA(j):=posy
                        end-if
                end-do
        end-do
        forall(i in FINALS)do
                No_Columnas(i):=i
                demprima(i):=DEMANDA(i)
        end-do
        c:=1
        while(c<=NO_FINALS) do
        crudo:=RAW
        forall(i in FINALS)numer(i):=i
        forall(i in FINALS|No_Columnas(i)>0)do
                a_inicial(i):=floor(crudo/ANCHO(i))
                crudo:=crudo-a_inicial(i)*ANCHO(i)
```

```
                    if(a_inicial(i)<>0)then
                    x_prima(i):=demprima(i)/a_inicial(i)
                    else
                        x_prima(i):=0
                    end-if
                end-do
                forall(i in 1..(NO_FINALS-1))do
                    forall(j in i+1..(NO_FINALS)) do
                        if(x_prima(i)>x_prima(j))then
                        posx:=x_prima(i)
                        posy:=numer(i)
                        x_prima(i):=x_prima(j)
                        numer(i):=numer(j)
                        x_prima(j):=posx
                        numer(j):=posy
                        end-if
                    end-do
                end-do
                forall(i in FINALS)do
                    if(x_prima(i)<>0)then
                        posxmin:=numer(i)
                        posx:=x_prima(i)
                        break
                    end-if
                end-do
                x_fin(c):=posx
                forall(i in FINALS)do
                    if(x_prima(i)>0 and x_prima(i)<>x_fin(c))then
                        demprima(numer(i)):=demprima(numer(i))-
x_fin(c)*a_inicial(numer(i))
                    end-if
                end-do
                ANCHOf(c):=ANCHO(posxmin)
                forall(i in FINALS)BASICA(i,c):=a_inicial(i)
                forall(i in FINALS)do
                a_inicial(i):=0
                x_prima(i):=0
                end-do
                c:=c+1
                No_Columnas(posxmin):=0
                end-do
                writeln("La Matriz Basica es: ")
                writeln
                forall(i in FINALS)do
```

```
                write("Ancho= ",ANCHO(i), ": ")
                        forall(j in FINALS )do
                        write(BASICA(i,j),"      ")
                        end-do
                writeln
        end-do
        forall(j in FINALS) do
                create(rollos_usados(j))
                rollos_usados(j) is_integer
                rollos_usados(j) <= ceil(x_fin(j))
        end-do
        MinRollos:= sum(j in FINALS) rollos_usados(j)
        forall(i in FINALS) do
                Dem(i):= sum(j in FINALS) BASICA(i,j) * rollos_usados(j) >=
DEMANDA(i)
        end-do
    end-procedure
end-model
```

Anexo 1.4 Programa para solucionar el problema “Cutting Stock” a través del método de generación de columnas. Utilizando ambas heurísticas; para obtener un patrón de corte inicial, y para resolver el problema de la mochila.

```
model Cuttingstock
uses "mmxprs"
uses "mmsystem"
    forward procedure generacion_columnas
    forward function knapsack(C:array(range) of real, A:array(range) of real,B:real,
mejor_x:array(range) of integer,pass: integer): real
    forward function heuristicaknapsack(C:array(range) of real, A:array(range) of
real,B:real,mejor_x:array(range) of integer,pass: integer): real
    forward procedure nuevo_patron(columna:integer)
    forward procedure solucion
    forward procedure patron_inicial
    declarations
        NO_FINALS: integer
        RAW: real
    end-declarations
    initializations from "cuttingstock.dat"
        NO_FINALS
        RAW
    end-initializations
    declarations
        FINALS = 1..NO_FINALS
```

```
        EPS = 1e-6
        RP: range
        ANCHO,ANCHOf,x_fin: array(FINALS) of real
        DEMANDA: array(FINALS) of integer
        BASICA: array(FINALS,FINALS) of integer
        rollos_usados: array(RP) of mpvar
        solusados: array(RP) of real
        Dem: array(FINALS) of linctr
        MinRollos: linctr
        KnapsackRestricciones, ObjetivoKnapsack: linctr
        matriz: array(FINALS,RP) of integer
        x: array(FINALS) of mpvar
        mejor_z,valor_objetivo:real
        npase,flag:integer
        starttime,endtime: real
end-declarations
initializations from "cuttingstock.dat"
        ANCHO
        DEMANDA
end-initializations
starttime:= gettime
patron_inicial
forall(j in FINALS) do
        forall(i in FINALS)do
                create(matriz(j,i))
                matriz(j,i):=BASICA(j,i)
        end-do
end-do
generacion_columnas
minimize(MinRollos)
solucion
endtime:=gettime-starttime
writeln("El tiempo para solucionarlo es: ",endtime)
procedure generacion_columnas
declarations
        dualdem: array(FINALS) of real
        mejor_x: array(FINALS) of integer
        dw: real
        bas: basis
end-declarations
        defcut:=getparam("XPRS_CUTSTRATEGY")
        setparam("XPRS_CUTSTRATEGY", 0)
        setparam("XPRS_PRESOLVE", 0)
        setparam("zerotol", EPS)
```

```
nColum:=NO_FINALS
npase:=1
while(true) do
    minimize(XPRS_LIN, MinRollos)
    savebasis(bas)
    valor_objetivo:=getobjval
    writeln
    writeln("Con un costo de: ",valor_objetivo," rollos")
    forall(j in FINALS)solusados(j):=getsol(rollos_usados(j))
    forall(i in FINALS)dualdem(i):=getdual(Dem(i))
    mejor_z:= heuristicaknapsack(dualdem, ANCHO, RAW,
mejor_x, npase) - 1.0
    if(flag=1)then
        mejor_z:= knapsack(dualdem, ANCHO, RAW, mejor_x,
npase) - 1.0
    end-if
    writeln
    writeln
    writeln("En el pase ", npase, ": ")
    if mejor_z = 0 then
        writeln("     No se encontro una mejor solución\n")
        break
    else
        nColum+=1
        create(rollos_usados(nColum))
        forall(i in FINALS)do
            create(matriz(i,nColum))
            matriz(i,nColum):=mejor_x(i)
        end-do
        nuevo_patron(nColum)
    rollos_usados(nColum) is_integer
    MinRollos+= rollos_usados(nColum)
    dw:=0
    forall(i in FINALS)
    if mejor_x(i) > 0 then
    Dem(i)+= mejor_x(i)*rollos_usados(nColum)
    dw:= maxlist(dw, ceil(DEMANDA(i)/mejor_x(i) ))
end-if
    rollos_usados(nColum) <= dw
    loadprob(MinRollos)
    loadbasis(bas)
    end-if
    npase+=1
end-do
```

```
        setparam("XPRS_CUTSTRATEGY", defcut)
        setparam("XPRS_PRESOLVE", 1)
    end-procedure
    function heuristicaknapsack(C:array(range) of real, A:array(range) of real,
B:real,mejor_x:array(range) of integer, pass: integer):real
    declarations
        anchobarra: real
        peso: array(FINALS,1..2) of real
        z_crit: real
        jasterisco: integer
        limite_x,guia: array(FINALS) of integer
    end-declarations
        forall(j in FINALS)do
            peso(j,1):=C(j)/A(j)
            peso(j,2):=j
            limite_x(j):=floor(B/A(j))
        end-do
        forall(i in 1..(NO_FINALS-1))do
            forall(j in i+1..(NO_FINALS)) do
                if(peso(i,1)<peso(j,1))then
                    q:=peso(i,1)
                    e:=floor(peso(i,2))
                    peso(i,1):=peso(j,1)
                    peso(i,2):=peso(j,2)
                    peso(j,1):=q
                    peso(j,2):=e
                end-if
            end-do
        end-do
        forall(j in FINALS)do
            guia(j):=floor(peso(j,2))
        end-do
        anchobarra:=B
        z_crit:=0
        flag:=0
        jasterisco:=1
        forall(j in FINALS) do
            if(floor(anchobarra/A(guia(j)))<limite_x(guia(j)))then
                mejor_x(guia(j)):=floor(anchobarra/A(guia(j)))
                anchobarra:=anchobarra-A(guia(j))*mejor_x(guia(j))
                z_crit:=z_crit+C(guia(j))*mejor_x(guia(j))
                else
                    mejor_x(guia(j)):=limite_x(guia(j))
```

```
                                        anchobarra:=anchobarra-
A(guia(j))*mejor_x(guia(j))
                                        z_crit:=z_crit+C(guia(j))*mejor_x(guia(j))
                        end-if

        if(limite_x(guia(j))*C(guia(j))>limite_x(guia(jasterisco))*C(guia(jasterisco)))the
n
                                jasterisco:=guia(j)
                        end-if
                end-do
                if(limite_x(jasterisco)*C(jasterisco)>z_crit)then
                        z_crit:=limite_x(jasterisco)*C(jasterisco)
                        forall(i in FINALS) do
                                mejor_x(i):=0
                        end-do
                        mejor_x(jasterisco):=limite_x(jasterisco)
                end-if
                if(z_crit<=1)then
                        forall(j in FINALS)mejor_x(j):=0
                        flag:=1
                        returned:=0
                        else
                                returned:=z_crit
                end-if
        end-function
        function knapsack(C:array(range) of real, A:array(range) of real,
B:real,mejor_x:array(range) of integer, pass: integer):real
                forall(j in FINALS) sethidden(Dem(j), true)
                KnapsackRestricciones := sum(j in FINALS) A(j)*x(j) <= B
                ObjetivoKnapsack := sum(j in FINALS) C(j)*x(j)
                forall(j in FINALS) x(j) is_integer
                maximize(ObjetivoKnapsack)
                returned:=getobjval
                forall(j in FINALS) do
                        mejor_x(j):=round(getsol(x(j)))
                end-do
                KnapsackRestricciones := 0
                ObjetivoKnapsack := 0
                forall(j in FINALS) sethidden(Dem(j), false)
        end-function
        procedure nuevo_patron(columna: integer)
                declarations
                total: real
                end-declarations
```

```
        total:=0
        writeln
        writeln("El Nuevo Patron de corte es: ")
        writeln
        forall(i in FINALS)do
                writeln(ANCHO(i)," : ",matriz(i,columna))
                total+=ANCHO(i)*matriz(i,columna)
        end-do
        writeln("Cantidad de rollo usado: ",total)
end-procedure
procedure solucion
writeln("La mejor solución entera es: ", getobjval, " rollos")
writeln
writeln(" Como se deben de cortar estos rollos es de la siguiente manera: ")
writeln
forall(i in RP|getsol(rollos_usados(i))<>0) do
        write(getsol(rollos_usados(i)))
        if(i<NO_FINALS)then
        write(", rollos con el patron de corte de la columna, ",i)
        else
                write(", rollos con el patron de corte del pase, ",i-NO_FINALS)
        end-if
        writeln
end-do
writeln
end-procedure
procedure patron_inicial
        declarations
                No_Columnas: array(FINALS) of integer
                numer: array(FINALS) of integer
                a_inicial: array(FINALS) of integer
                posx: real
                posy: integer
                b_prima:array(FINALS) of integer
                x_prima,demprima:array(FINALS) of real
                crudo: real
                posxmin,c:integer
        end-declarations
        forall(i in 1..(NO_FINALS-1))do
                forall(j in i+1..(NO_FINALS)) do
                        if(ANCHO(i)<ANCHO(j))then
                        posx:=ANCHO(i)
                        posy:=DEMANDA(i)
                        ANCHO(i):=ANCHO(j)
```

```
                DEMANDA(i):=DEMANDA(j)
                ANCHO(j):=posx
                DEMANDA(j):=posy
                end-if
        end-do
end-do
forall(i in FINALS)do
        No_Columnas(i):=i
        demprima(i):=DEMANDA(i)
end-do
c:=1
while(c<=NO_FINALS) do
crudo:=RAW
forall(i in FINALS)numer(i):=i
forall(i in FINALS|No_Columnas(i)>0)do
        a_inicial(i):=floor(crudo/ANCHO(i))
        crudo:=crudo-a_inicial(i)*ANCHO(i)
        if(a_inicial(i)<>0)then
        x_prima(i):=demprima(i)/a_inicial(i)
        else
                x_prima(i):=0
        end-if
end-do
forall(i in 1..(NO_FINALS-1))do
        forall(j in i+1..(NO_FINALS)) do
                if(x_prima(i)>x_prima(j))then
                posx:=x_prima(i)
                posy:=numer(i)
                x_prima(i):=x_prima(j)
                numer(i):=numer(j)
                x_prima(j):=posx
                numer(j):=posy
                end-if
        end-do
end-do
forall(i in FINALS)do
        if(x_prima(i)<>0)then
                posxmin:=numer(i)
                posx:=x_prima(i)
                break
        end-if
end-do
x_fin(c):=posx
forall(i in FINALS)do
```

```
                    if(x_prima(i)>0 and x_prima(i)<>x_fin(c))then
                        demprima(numer(i)):=demprima(numer(i))-
x_fin(c)*a_inicial(numer(i))
                    end-if
                end-do
                ANCHOf(c):=ANCHO(posxmin)
                forall(i in FINALS)BASICA(i,c):=a_inicial(i)
                forall(i in FINALS)do
                a_inicial(i):=0
                x_prima(i):=0
                end-do
                c:=c+1
                No_Columnas(posxmin):=0
                end-do
                writeln("La Matriz Basica es: ")
                writeln
                forall(i in FINALS)do
                    write("Ancho= ",ANCHO(i), ": ")
                        forall(j in FINALS )do
                        write(BASICA(i,j),"      ")
                        end-do
                    writeln
                end-do
                forall(j in FINALS) do
                    create(rollos_usados(j))
                    rollos_usados(j) is_integer
                    rollos_usados(j) <= ceil(x_fin(j))
                end-do
                MinRollos:= sum(j in FINALS) rollos_usados(j)
                forall(i in FINALS) do
                    Dem(i):= sum(j in FINALS) BASICA(i,j) * rollos_usados(j) >=
DEMANDA(i)
                end-do
        end-procedure
end-model
```