Treball de curs
Q1 2022-23

Integrants:
Miquel Muñoz García-Ramos
Cristian Sánchez Estapé

Índex

1. Codi del model i descripció de les variables

En aquest punt es descriuen els models d'algorisme que indica l'enunciat. També es mostra el codi utilitzat amb la pertinent explicació d'aquest.

1.1 Model sense generació de columnes

1.1.1 Descripció del model

Aquest primer model resol el *Cutting-Stock Problem* (d'ara endavant *CSP*) sense generació de columnes, és a dir, amb tota la informació referent a la instància particular del problema que resolem. Pel cas present, posat que treballem amb 3 tipus de tall i una longitud de rotllo ja fixa, hem de generar nosaltres mateixos la informació referent als patrons de tall, és a dir, la quantitat d'aparicions de cada tipus de tall per cada patró[1], juntament amb la longitud (en polzades, en el nostre cas) que un patró de tall fa servir d'un rotllo.

D'altra banda, pel que fa a les qüestions d'implementació, fem servir dos conjunts diferents, corresponents tant als tipus de tall (**talls**) com als patrons de tall (**patrons**). Pel que fa a paràmetres, enregistrem dades tals com la longitud màxima d'un rotllo (**amplada_total_rotllo**); la longitud de cada patró de tall (**longitud**), definida sobre *patrons*[2]; el nombre d'aparicions de cada tipus de tall per cada patró de tall (**D**[3]), definit sobre *talls* i *patrons*; i el nombre de comandes que tenim de cada tipus de tall (**comandes**), definits també sobre *patrons*. Finalment, respecte a les variables, tenim tant la quantitat de patrons de tall utilitzats (**X_D**) com les unitats sobrants de cada tipus de tall (**unitats**), ambdós definits sobre *patrons*. Aquesta darrera variable emmagatzema, en format fraccionari, la quantitat de rotllo que s'ha perdut per un determinat patró de tall.

Referent a les restriccions, hem trobat adient definir-ne dues: per una banda, aquella referent a l'enregistrament de les unitats sobrants de cada patró de tall (**ActualitzarUnitats**), que ens permetrà consultar, posteriorment, la suma total d'excedents per cada patró de tall; d'altra banda, aquella referida a satisfer la demanda de cada tipus de tall (**Demanda**), i que garanteix que totes les comandes siguin satisfetes.

Finalment, la funció objectiu, que minimitza la quantitat de rotllos consumits segons cada patró de tall. Amb aquesta, i mitjançant la restricció *ActualitzarUnitats*, entenem que es minimitza tant el sobrant com la quantitat de rotllos consumits, posat que minimitzar aquesta darrera mètrica és equivalent a cercar els talls que impliquin la menor quantitat de rotllo sobrant.

En definitiva, aquest model requeriria, d'entrada, de tota la informació referent al problema. Pel nostre cas, posat que és de dimensió reduïda, és factible atacar-lo d'aquesta

---

[1] Com a clarificació, entenem, per tipus de tall, els talls corresponents a les longituds en si mateixes (si mirem la figura 2, aquests correspondrien a 24, 32 i 40), mentre que un patró de tall seria una manera de tallar el rotllo (per exemple, si observem la matriu D de la mateixa figura 2, veiem que, pel patró 1, fem 1 tall de 24', 0 talls de 40' i 0 talls de 32').

[2] Quan parlem d'una variable definida *sobre* una altra ens referim al fet que el domini de la primera correspon als valors de la segona.

[3] Tot atribut que comparteixi nom en ambdós models implica que té el mateix propòsit.

manera, i tal com es veurà posteriorment, el nostre model ho aconsegueix. Val a afegir que aquest model no requereix estrictament un fitxer .run, amb la qual cosa prescindim d'ell.

1.1.2 Codi

```
set talls;                          # conjunt de talls
set patrons;                        # conjunt total de possibles patrons de tall

param amplada_total_rotllo;         # amplada total dels rotllos
param longitud {patrons};           # longitud en polzades de cada patró de tall
param D {talls,patrons};            # nombre d'unitats de cada tall per cada patró
param comandes {talls};             # nombre de comandes per un tall determinat

var X_D {patrons} >= 0;             # rotllos tallats utilitzant un determinat patró
var unitats {patrons} >= 0;         # unitats sobrants de cada patró

subject to ActualitzarUnitats {i in patrons}: # unitats sobrants de cada patró de tall
      unitats[i] = X_D [i]*(amplada_total_rotllo-longitud[i])/amplada_total_rotllo;
subject to Demanda {i in talls}:              # satisfacció de les comandes
      sum {j in patrons} X_D [j] * D[i,j] >= comandes[i];

minimize MinUnitats:                          # minimitzem el consum de rotllos
      sum {j in patrons} X_D[j];
```

Figura 1: codi corresponent al .mod del model sense generació de columnes

```
set talls := 24 32 40;                  # conjunt de possibles talls
set patrons  := 1 2 3 4 5 6 7 8 9       # conjunt total de possibles patrons de tall
             10 11 12 13 14 15 16;
param amplada_total_rotllo := 100;      # amplada total dels rotllos
param comandes :=                       # nombre de comandes de cada tall
          24     75
          32     110
          40     50;
param longitud :=                       # longituds en polzades de cada patró de tall
1 24   2 56   3 88   4 64   5 96   6 48   7 80   8 88
9 72   10 96  11 40  12 72  13 80  14 32  15 64  16 96;
param D :=                              # nombre d'unitats de cada tall per cada patró
24 1 1       24 2 1       24 3 1       24 4 1       24 5 1       24 6 2
24 7 2       24 8 2       24 9 3       24 10 4      24 11 0      24 12 0
24 13 0      24 14 0      24 15 0      24 16 0      40 1 0       40 2 0
40 3 0       40 4 1       40 5 1       40 6 0       40 7 0       40 8 1
40 9 0       40 10 0      40 11 1      40 12 1      40 13 2      40 14 0
40 15 0      40 16 0      32 1 0       32 2 1       32 3 2       32 4 0
32 5 1       32 6 0       32 7 1       32 8 0       32 9 0       32 10 0
32 11 0      32 12 1      32 13 0      32 14 1      32 15 2      32 16 3;
```

Figura 2: codi corresponent al .dat del model sense generació de columnes

1.2 Model amb generació de columnes

1.2.1 Descripció del model

Aquest segon model resol el CSP amb informació dinàmica, és a dir, donades certes característiques bàsiques del problema, aquest procedeix a intentar resoldre'l definint dinàmicament aquella informació que li manca. De nou, partim amb part de la mateixa informació que ja teníem en resoldre el model bàsic, però tal com es pot observar a la figura 4, aquesta només fa referència a les dades corresponents a les comandes, els tipus de tall i l'amplada total dels rotllos.

Referent a la manera en què es resol el problema mitjançant el present model, cal destacar que es fa per iteracions: tal com es descriu a l'enunciat, intentem trobar aquella solució per la qual els costs reduïts assoleixen un valor ≥ 0, amb la qual cosa es van executant iteracions fins a arribar a aquesta fita. D'acord amb aquest fet, d'ara referenciarem genèricament les iteracions d'aquest procediment.

A banda d'això, pel que fa al mateix model, aquest és més sofisticat: tal com veurem al .run, aquest consta de dos problemes: per una banda, el problema principal, que seria virtualment anàleg a aquell del model bàsic; d'altra banda, el problema dual, que correspondria a aquell que intenta minimitzar els costos reduïts.

Respecte al primer, trobem informació pràcticament igual a la del model anterior: tenim 3 conjunts corresponents als tipus de tall (**talls**), al conjunt total d'índexs de les columnes generades (I_D) i al conjunt de patrons sobre els quals es treballa en una iteració donada (**patrons**). Els únics atributs que val la pena comentar són aquests dos darrers, posat que el primer, I_D, és el que defineix el domini sobre el qual *patrons* pot prendre valors, i és el que precisament ens permet modificar *patrons* dinàmicament sense haver d'incórrer en problemes (si féssim una assignació directa sobre *patrons*, ens seria impossible canviar-li el domini a cada iteració). Referent als paràmetres, trobem els mateixos que abans (*amplada_total_rotllo*, *longitud*, *D*, *comandes*), a excepció d'aquell que, de manera ambivalent, fa referència tant a la iteració en què s'està resolent el problema com a la quantitat màxima de possibles tipus de tall fins ara explorats (**l**). Finalment, referent a les variables, restriccions i funció objectiu, les trobem idèntiques a com estan definides al model bàsic anteriorment presentat.

Respecte al segon, trobem el següent: el paràmetre corresponent a les variables duals del primer problema (**lambda**); la variable que defineix la informació referent a la nova columna que s'ha d'afegir al paràmetre *D* (**a**); la restricció *LimitAmplada*, que indica que el nou patró de tall ha de respectar el límit de longitud del rotllo; la funció objectiu *CostReduit*, que codifica, idènticament a com es defineix en l'enunciat de la pràctica, la resolució del problema dual.

Ara, passant al fitxer .run, comentarem pas per pas el que s'hi troba codificat:

1. Definim *cplex* com a solver, posat que és un dels solvers que respecta les definicions de variables com a enteres.
2. Definim que l'arrodoniment dels resultats es faci amb 5 decimals de precisió.

3. Carreguem el .mod i .dat (per tal de fer-ho, *AMPL* pressuposa que hom executa el .run des del mateix directori en què es troben els fitxers del model i de les dades).
4. Definim els dos problemes *CSP* i *NovaColumna*, que corresponen al primer problema i al dual. Pel *CSP* indiquem *option presolve 1* amb tal de garantir que el purgat de les columnes es dugui a terme.
5. Definim l a 0 i procedim a definir els patrons inicials amb què resoldrem el problema[4].
6. Fem una primera resolució d'ambdós problemes per tal d'establir inicialment quins patrons fem servir i quins no, informació que vindrà donada a X_D (si x≡0 llavors sabem que aquell patró no s'utilitza). A més a més, la passa intermèdia entre *solve CSP* i *solve NovaColumna* traspassa els valors duals òptims, corresponents a cada tipus de tall, trobats a la restricció *Demanda*. Addicionalment, definim el conjunt *remove*, que serà el que ens ajudarà a emmagatzemar les columnes a purgar.
7. Si els costs reduïts són < 0, llavors entrem en el bucle. Aquí dins el que fem és continuar generant columnes i mirant de trobar la solució òptima, cosa que aconseguim si: definim *remove* com el conjunt buit (ho fem dins el bucle per tal de garantir que, a cada iteració, esborrem només aquelles columnes que **en la mateixa iteració** s'han identificat com a no usades); incrementem l per tal d'incloure una nova columna, diferent de les anteriors; incloem la nova columna l al conjunt de patrons de tall que ara hem de tenir en compte; afegim la nova columna a a D; resolem ambdós problemes amb el nou patró trobat; purguem de *patrons* aquelles columnes que no siguin usades (ho fem en dues passes: la primera posa a *remove* totes les columnes a eliminar, la segona les purga).
8. L'execució acaba quan $CostsReduïts \geq 0$.

[4] Els detalls i motius d'aquesta definició, juntament amb altres particularitats del .run, es troben definides a la secció 6.

1.2.2 Codi

```
param amplada_total_rotllo >= 0;        # amplada total dels rollos
set talls;                              # conjunt de talls
param l integer >= 0;                   # índex de cada iteració (equivalent a la
                                          quantitat de possibles patrons de tall)
set I_D := 1..l;                        # conjunt complet d'índexs de patrons de tall
set patrons within I_D default 1..l;    # conjunt de patrons definits per l'índex "l"
param longitud {patrons};               # longitud en polzades de cada patró de tall
param D {talls,patrons} integer >= 0;   # submatriu de "A" designada "D" a l'enunciat
param comandes {talls} >= 0;            # nombre de comandes per un tall determinat


var X_D {patrons} >= 0;                 # components corresponents a les columnes de I_D
var unitats {patrons} >= 0;             # unitats sobrants de cada patró

subject to Demanda {i in talls}:                # satisfacció de les comandes
   sum {j in patrons} D[i,j] * X_D[j] >= comandes[i];
subject to ActualitzarUnitats {i in patrons}: # unitats sobrants de cada patró de tall
    unitats[i] = X_D[i]*(amplada_total_rotllo-longitud[i])/amplada_total_rotllo;

minimize MinUnitats:                            # minimitzem el consum de rotllos
      sum {j in patrons} X_D[j];
# ============================================================
param lambda {talls} >= 0;              # variable dual

var a {talls} integer >= 0;             # nombre d'unitats de cada tall pel nou patró

subject to LimitAmplada:                # limitació d'amplada del nou patró
   sum {i in talls} i * a[i] <= amplada_total_rollo;

minimize CostReduit:                    # minimitzem els costos reduïts (problema dual)
   1 - sum {i in talls} lambda[i] * a[i];
```

Figura 3: codi corresponent al .mod del model amb generació de columnes

```
param amplada_total_rollo := 100;       # amplada total dels rotllos
param talls := 24 32 40;                # conjunt de possibles talls
param comandes :=                       # nombre de comandes de cada tall
          24     75
          32     110
          40     50;
```

Figura 4: codi corresponent al .dat del model amb generació de columnes

```
option solver cplex;
option solution_round 5;
model cg.mod
data cg.dat
problem CSP: X_D, unitats, MinUnitats, Demanda, ActualitzarUnitats;
	option presolve 1;
problem NovaColumna: a, CostReduit, LimitAmplada;

let l := 0;
for {i in talls} {
	let l := l + 1;
	let D[i,l] := floor(amplada_total_rollo/i);
	let {j in talls: i <> j} D[j,l] := 0;
	let longitud[l] := i * D[i,l];
};

# necessitem una primera iteració per trobar els valors de X_D =/= 0
solve CSP;
let {i in talls} lambda[i] := Demanda[i].dual;
solve NovaColumna;

set remove;

repeat {
	let remove := {};
	let l := l + 1;
	let patrons := patrons union {l};
	let {i in talls} D[i,l] := a[i];
	let longitud[l] := sum {i in talls} i * D[i,l];
	# resolem amb el nou patró trobat
	solve CSP;
	let {i in talls} lambda[i] := Demanda[i].dual;
	solve NovaColumna;
	# ara podem procedir a purgar aquells patrons que no utilitzem
	for {i in patrons} if X_D[i] == 0 then let remove := remove union {i};
	let patrons := patrons diff remove;
} while (CostReduit < 0);
```

Figura 5: codi corresponent al .run del model amb generació de columnes

2. Contrastar la implementació de l'algoritme de generació de columnes

Procedim a solucionar el present problema mitjançant el model sense generació de columnes i veiem la següent solució amb els *displays* de les variables corresponents.

```
CPLEX 20.1.0.0: optimal solution; objective 76.25
3 dual simplex iterations (0 in phase I)
```

Figura 6: solució del model sense generació de columnes

```
ampl: display X_D;
X_D [*] :=
 1   0
 2   0
 3   0
 4   0
 5  50
 6   0
 7   0
 8   0
 9   0
10   6.25
11   0
12   0
13   0
14   0
15   0
16  20
;
```

Figura 7: valor de l'atribut "X_D" a la solució

```
ampl: display unitats;
unitats [*] :=
 1  0
 2  0
 3  0
 4  0
 5  2
 6  0
 7  0
 8  0
 9  0
10  0.25
11  0
12  0
13  0
14  0
15  0
16  0.8
;
```

Figura 8: valor de l'atribut "unitats" a la solució

Vista aquesta, procedim a solucionar el problema amb el model amb generació de columnes i visualitzem la solució amb els corresponents *displays*.

```
CPLEX 20.1.0.0: optimal solution; objective 80.41666667
0 dual simplex iterations (0 in phase I)
CPLEX 20.1.0.0: optimal integer solution; objective -0.08333333
2 MIP simplex iterations
0 branch-and-bound nodes
CPLEX 20.1.0.0: optimal solution; objective 76.25
4 dual simplex iterations (0 in phase I)
CPLEX 20.1.0.0: optimal integer solution; objective 0
1 MIP simplex iterations
0 branch-and-bound nodes
ampl: display X_D;
X_D [*] :=
1   6.25
2  20
4  50
;

ampl: display unitats;
unitats [*] :=
1  0.25
2  0.8
4  2
;

ampl: display a;
a [*] :=
24  1
32  1
40  1
;
```

Figura 8: solució del model amb generació de columnes, i valors dels atributs "X_D", "unitats" i "a"

Comparant les dues solucions, primer de tot cal observar que el valor de la funció objectiu és idèntic, la qual cosa ens incita a esperar que el funcionament d'ambdós sigui correcte. Respecte al display de les variables, la primera diferència que podem identificar és que tenim una variable "*a*" en el programa de generació de columnes que no existeix a l'altre. Això és degut al fet que aquesta variable ens indica el valor de la nova columna generada. Per altra banda, a les variables "X_D" i "*unitats*" es veu com a l'algorisme amb generació de columnes no compta amb fileres de 0 i, en canvi, a l'altre sí. Referent a les diferències entre models, vegeu la secció 6.

3. Execució de l'algoritme

En aquesta secció, visualitzarem informació diversa sobre com evoluciona l'execució del programa durant les diverses iteracions executades.

3.1 Valor de la funció objectiu a cada iteració

A continuació, visualitzarem com evoluciona el valor de la funció objectiu per cada iteració executada. Abans de prosseguir, cal recordar que es resolen dos problemes, amb la qual cosa tindrem dues funcions objectiu, una referent al problema del *CSP* (que minimitza la quantitat de patrons de tall que hem d'utilitzar) i l'altra referent al problema dual (que minimitza els costos reduïts).

```
*****************************
======== Nova iteracio ========
*****************************

****************************************************
Valor de la funció objectiu al principi de la iteracio
----------------------------------------------------
MinUnitats = 80.4167

****************************************************
CPLEX 20.1.0.0: optimal solution; objective 76.25
4 dual simplex iterations (0 in phase I)
CPLEX 20.1.0.0: optimal integer solution; objective 0
1 MIP simplex iterations
0 branch-and-bound nodes
************************************************
Valor de la funció objectiu al final de la iteracio
------------------------------------------------
MinUnitats = 76.25

************************************************
Fi de l'execució
```

Figura 9: evolució del valor de la funció objectiu a cada iteració executada

Tal com s'observa, per les dades amb què treballem, el programa només requereix una única iteració per assolir el resultat òptim. A més a més, podem veure que, si la visualització del valor de la funció objectiu es fa per separat mitjançant el comandament ≪*display MinUnitats;*≫, aquest coincideix amb el resultat obtingut pel comandament ≪*solve CSP;*≫ executat al fitxer .run.

3.2 Valor del cost reduït en la passa 3

A continuació, visualitzarem el valor del cost reduït a cada iteració executada.

```
*****************************
======== Nova iteracio ========
*****************************

*********************
Valor del cost reduit
---------------------
CostReduit = -0.0833333

*********************
CPLEX 20.1.0.0: optimal solution; objective 76.25
4 dual simplex iterations (0 in phase I)
CPLEX 20.1.0.0: optimal integer solution; objective 0
1 MIP simplex iterations
0 branch-and-bound nodes
*********************
Valor del cost reduit
---------------------
CostReduit = 0

*********************
Fi de l'execució
```

Figura 10: evolució del valor del cost reduït a cada iteració executada

Tal com podem observar, a cada iteració executada, el cost reduït augmenta fins a assolir un valor ≥ 0 (en aquest cas 0). De nou, podem comprovar la coincidència entre el resultat de ≪*solve NovaColumna;*≫ amb el comandament ≪*solve CostReduit;*≫.

3.3 Nova columna determinada

A continuació, visualitzarem la nova columna generada a cada iteració executada.

```
*****************************
======== Nova iteracio ========
*****************************

*****************************
Valor de la nova columna per 4
------------------------------
a [*] :=
24  1
32  1
40  1
;

*****************************
CPLEX 20.1.0.0: optimal solution; objective 76.25
4 dual simplex iterations (0 in phase I)
CPLEX 20.1.0.0: optimal integer solution; objective 0
1 MIP simplex iterations
0 branch-and-bound nodes
Fi de l'execució
```

Figura 11: evolució de la nova columna generada a cada iteració executada

De nou, podem constatar que, per la iteració executada, el model genera una columna nova i diferent de totes les incloses en la matriu *D* (podem afirmar-ho sabent que, inicialment, la matriu conté patrons de tall que corresponen a un únic tipus de tall, mentre que l'afegida és, en aquest sentit, completament diferent).

Dada pel model sense generació de columnes, veiem que no només coincideix el valor de la funció objectiu, sinó que també ho fan els patrons de tall: si mirem la figura 7, podem observar que els patrons de tall emprats corresponen al conjunt $\{5, 10, 16\}$ que, si consultem el codi .dat de la figura 2, corresponen a la següent matriu *D*:

	5	10	16
24	1	4	0
32	1	0	3
40	1	0	0

Figura 12: patrons corresponents a la solució donada pel model bàsic

La present matriu representa la quantitat que, per cada tipus de tall, correspon a cada patró de tall. Veient aquests patrons, i sabent que el model amb generació de columnes comença amb aquells que només tenen en compte un tipus de tall, veiem que 2 de les 3 columnes originals es mantenen, mentre que s'abandona la que prioritza el tipus de tall de 40' a favor del patró de tall que els barreja, amb la qual cosa veiem que ambdues solucions són anàlogues.

4. Comparació el valor de la funció objectiu després de l'arrodoniment

La solució obtinguda per l'algorisme de generació de columnes és la següent:

```
Solucio sense arrodoniment

X_D [*] :=
1   6.25
2  20
4  50
;

MinUnitats = 76.25

Arrodonim la solucio obtinguda

X_D [*] :=
1   7
2  20
4  50
;

MinUnitats = 77
```

Figura 13: comparació dels efectes de l'arrodoniment sobre la variable X_D

Tal com es pot observar, inicialment gairebé totes les quantitats són enteres tret d'aquella corresponent al patró de tall 1, que fa que el valor de la funció objectiu no sigui enter. Vist això, si arrodonim els valors de X_D amb la funció *ceil,* veiem que el valor de la funció objectiu també es veu arrodonida a l'alça. Sabent que la funció objectiu correspon a la suma de quantitats de X_D, podem comprovar immediatament la correctesa de la propagació de l'arrodoniment.

5. Aplicació del mètode de generació de columnes a problemes de *Set Partitioning* i de *Set Covering*

Aquest punt del treball és l'únic amb el qual s'ha hagut de consultar articles acadèmics. Per tant, tota la bibliografia tracta dels articles acadèmics que s'han consultat per realitzar aquest apartat.

La generació de columnes és un mètode de programació lineal dissenyat per a resoldre problemes que tenen un gran nombre de variables, però també una certa estructura. Quan es complementa amb una tècnica adequada de programació entera, la generació de columnes ha demostrat la seva eficàcia en la resolució de molts problemes a gran escala.

El *Set Covering Problem* (d'ara endavant *SCP*) adaptat al *CSP* consisteix, donada una col·lecció de patrons de tall i els costos associats a aquests, en trobar una partició de cost mínim del conjunt inicial. Formalment, definint "n" com el nombre total de patrons de tall que satisfan la restricció, el problema de partició de conjunts es pot escriure de la següent manera (Gilmore and Gomory):

$$(P_2) \qquad \min \sum_{j=1}^{n} x_j$$

$$\text{s.t.} \sum_{j=1}^{n} a_{ij} x_j \geq n_i, \quad i = 1, \ldots, m, \quad \text{(demand)}$$

$$x_j \in \mathbb{Z}_+, \; j = 1, \ldots, n,$$

$$\sum_{i=1}^{m} w_i a_{ij} \leq W.$$

Figura 14: formulació del SCP adaptat al CSP segons Gilmore & Gomory

On "X_j" és el nombre de vegades que cada patró j-èssim s'utilitza, "a_{ij}" és la quantitat de l'ítem i-èssim present al patró de tall j-èssim, "W" és l'amplada total del rotllo, "n_i" és la demanda de l'ítem i-èssim, "w_i" és l'amplada de l'ítem i-èssim i cada columna a *P2* representa un patró de tall.

Si tinguéssim moltes variables, encara que tinguéssim una manera de generar totes les columnes de patrons de tall, l'algorisme del símplex estàndard necessitaria calcular el cost reduït per a cada variable no bàsica, la qual cosa és impossible si "n" és enorme.

Cal dir, que el nombre de variables diferents de zero (les variables base) és igual al nombre de restriccions. Encara que el nombre de variables possibles pugui ser gran, només necessitem un petit subconjunt d'aquestes columnes en la solució òptima. En el cas del problema de *CSP*, el nombre d'articles sol ser molt de menor que el nombre de patrons de tall.

La idea principal de la generació de columnes tracta en començar amb un subconjunt factible del problema *P*, sent $P \subset \{1,\ldots, n\}$:

$$
\begin{aligned}
(RLPM) \qquad & \min \sum_{j\in\mathcal{P}} x_j \\
& \text{s.t.} \sum_{j\in\mathcal{P}} a_{ij}x_j \geq n_i, \quad i = 1,\ldots,m, \\
& \qquad x_j \geq 0,\ j \in \mathcal{P}, \\
& \pi^T = c_B^T B^{-1}
\end{aligned}
$$

Figura 15: formulació del SCP, adaptat al CSP i basada en la generació de columnes, segons Gilmore & Gomory

On "π" és la solució dual, "c" els costos reduïts i "B" la base actual. Aleshores, solucionar el problema dual es redueix a:

$$
\begin{aligned}
& \max \sum_{i=1}^{m} n_i\pi_i \\
& \text{s.t.} \sum_{i=1}^{m} a_{ij}\pi_i \leq 1, \quad j \in \mathcal{P}, \\
& \qquad \pi_i \geq 0,\ i = 1,\ldots,m.
\end{aligned}
$$

Figura 16: formulació del problema dual del SCP de la formulació de la figura 15

Vist això, només ens cal trobar una columna que pugui millorar la solució de la relaxació lineal anterior. El *Knapsack Subproblem* (d'ara endavant *KS*) és una manera de trobar una columna que ens generi aquesta millora:

$$
\begin{aligned}
& \min\ 1 - \sum_{i=1}^{m} \bar{\pi}_i y_i = 1 - \max \sum_{i=1}^{m} \bar{\pi}_i y_i \\
& \text{s.t.} \sum_{i=1}^{m} w_i y_i \leq W, \\
& \qquad y_i \in \mathbb{Z}_+,\ i = 1,\ldots,m,
\end{aligned}
$$

Figura 17: formulació del KS pel SCP adaptat al CSP segons Gilmore & Gomory

on y_i representa una columna i la constricció $w_i y_i \leq W$ comporta que la i-èssima columna sigui satisfactible respecte a l'amplada del rotllo consumida pel patró de tall.

D'altra banda, el *Set Partitioning Problem* (d'ara endavant *SPP*) consisteix a determinar com es poden dividir els elements d'un conjunt (*S*) en subconjunts més petits. Tots els elements de *S* han d'estar continguts en una i només una partició. Formalment, el *SPP* es pot escriure de la següent manera:

$$\begin{aligned} &\min_x \quad \sum_{j \in N} c_j x_j \\ &\text{s.t.:} \qquad \sum_{j \in N} a_{ij} x_j = 1, \quad \forall i \in T \\ &x_j \in \{0,1\}, \quad \forall j \in N, \end{aligned}$$

Figura 18: formulació del SPP

On "*T*" són les tasques a complir exactament una vegada, "*N*" els subconjunts de tasques factibles, "*c*" el cost d'acomplir la tasca j-èssima, "*a*" el paràmetre binari que indica si la tasca i-èssima és al subconjunt j-èssim i "*x*" és una variable binària que indica si el subconjunt j-èssim està seleccionat o no. El *SPP* consisteix a seleccionar subconjunts en "*N*" tals que cada tasca "*T*" pertanyi exactament a un dels subconjunts seleccionats i es minimitzi la suma dels costos d'aquests subconjunts.

Pel *SPP*, amb solució dual "*u*", el subproblema de generació de columnes és el següent:

$$\begin{aligned} \text{[CG]} \qquad \bar{c}_p \;=\; &\min \quad c - \sum_{i \in M} u_i a_i \\ &\text{s.t.} \quad (c,a) \in \mathcal{P}, \end{aligned}$$

Figura 19: formulació del problema dual del SPP adaptat a la generació de columnes

On "*P*" seria el subconjunt de columnes.

Qualsevol columna amb cost reduït negatiu és "favorable". Sovint, buscarem la més negativa, amb la qual cosa acabem resolent un problema d'optimització. Addicionalment, per tal de començar necessitem una solució inicial factible per a calcular el primer conjunt de valors duals. Això ens proporcionarà una columna (c,a) per afegir al problema principal *SPP*, sent "*c*" els costs associats a "*a*".

6. Diferències/ampliacions/variants entre el model bàsic i el model implementat

Són diversos els detalls a comentar de la implementació de l'algorisme de generació de columnes aquí realitzat. La primera d'elles correspon a la primera passa de l'algorisme: mentre que a l'enunciat es fa una distinció entre n_0 i l, i aprofita aquesta distinció per definir $I_D = \{1,..., n_0\}$, nosaltres hem optat per fusionar n_0 i l en aquesta darrera i, per tant, hem optat per definir $I_D = \{1,..., l\}$. En el nostre cas, l es defineix dinàmicament durant el primer bucle del .run, i pren valors des de l'1 fins a $|talls|$ (pel nostre cas, $|talls| = 3$, i tot i que inicialitzem $l = 0$, aquesta inicialització és virtualment inexistent posat que sempre tindrem, com a mínim, 1 tipus de tall). A més a més, aquesta inicialització també estableix que les primeres columnes corresponen als patrons de tall en què es prioritza únicament maximitzar un tipus de tall, el que correspon a la decisió més simple que es pot prendre. Per tant, la nostra passa 1 difereix del que tenim establert a l'enunciat.

D'altra banda, la implementació feta comprèn diverses passes que l'algorisme, tal com ve definit, no es troben especificades: el fet que, per exemple, hàgim de fer una resolució inicial d'ambdós problemes per tal de poder decidir si hem d'entrar dins el bucle de la passa 3, n'és un exemple. Com a detalls menors també hauríem de tenir en compte la nomenclatura de les noves columnes, posat que si bé al mateix enunciat s'especifica que, a la passa 3, la nova columna pren per valor $j = n_0 + l$, en el nostre cas hem optat per anomenar-la $j = l + 1$, garantint també que $l = l + 1$ per la mateixa iteració. A banda d'aquestes qüestions, la implementació aquí presentada no difereix més de la proposada al mateix enunciat.

Addicionalment, també cal comentar la diferència referent a la naturalesa de la implementació del purgat de columnes: mentre que a l'enunciat del treball es comenta l'ús d'una màscara que s'encarregui de fer ignorar segons quines columnes, nosaltres hem optat per treballar amb conjunts dinàmics que, a cada passa vagin canviant de mida i que, per tant, vagin tenint un impacte sobre la resta de paràmetres i variables del problema. Per tal de fer aquesta implementació, tal com s'ha comentat a la secció 1, tenim 3 conjunts crucials per tal de garantir aquest purgat: I_D, que defineix el domini complet sobre el qual *patrons* pot prendre valors, conjunt del qual anirem retirant columnes mitjançant el conjunt *remove*. El conjunt I_D ens ajuda a no haver de definir exactament quin és el domini de *patrons*, posat que si es fes així, no es podria modificar dinàmicament el seu domini. Dit això, si I_D és el conjunt que habilita a *patrons* a poder anar augmentant de domini i prenent nous valors, el conjunt *remove* habilita a *patrons* a poder perdre elements del seu domini i, per tant, cadascun d'aquests representa una d'aquestes components necessàries pel purgat.

Pel que fa a diferències entre el model sense generació de columnes i el seu alternatiu, val la pena esmentar, recuperant la qüestió de la nomenclatura de les columnes generades, que els índexs corresponent als patrons de tall no coincideixen entre models. En aquest sentit, per tal de saber, pel model de generació de columnes, de quin patró de tall es parla, s'ha de visualitzar la matriu *D*, la qual indica la quantitat de talls de cada tipus. A més a més, la

diferència de visualitzacions entre atributs com "*unitats*" és deguda al fet que el model sense generació de columnes parteix d'un conjunt ja definit de dades, mentre que l'altre les ha de generar dinàmicament, amb la qual cosa no li calen aquelles columnes i/o fileres corresponents a 0s.

7. Exemples de funcionament del model

En la següent taula, es poden veure els valors de la variable X_D segons el nombre de comandes de cada tall. Val a dir que aquests jocs de proves es basen en les dades originals del problema aquí estudiat, és a dir, es mantenen els tipus de tall i l'amplada total del rotllo amb què treballem.

	Generació de columnes	Sense generació de columnes	Script manual AMPL
param comandes := 24 75 32 110 40 50;	X_D [*] := 1 6.25 2 20 4 50;	X_D [*] := 1 0 2 0 3 0 4 0 5 50 6 0 7 0 8 0 9 0 10 6.25 11 0 12 0 13 0 14 0 15 0 16 20;	Cut [*] := 1 6.25 2 20 3 0 4 50;
param comandes := 24 90 32 70 40 80;	X_D [*] := 1 5 3 5 4 70;	X_D [*] := 1 0 2 0 3 0 4 0 5 70 6 0 7 0 8 0 9 0 10 5 11 0 12 0 13 5 14 0 15 0 16 0;	Cut [*] := 1 5 2 0 3 5 4 70;

param comandes := 24 70 32 100 40 80;	X_D [*] := 2 10 3 5 4 70;	X_D [*] := 1 0 2 0 3 0 4 0 5 70 6 0 7 0 8 0 9 0 10 0 11 0 12 0 13 5 14 0 15 0 16 10;	Cut [*] := 1 0 2 10 3 5 4 70;
param comandes := 24 0 32 40 40 60;	X_D [*] := 2 13.3333 3 30;	X_D [*] := 1 0 2 0 3 0 4 0 5 0 6 0 7 0 8 0 9 0 10 0 11 0 12 0 13 30 14 0 15 0 16 13.3333;	Cut [*][5] := 1 13.3333 2 30;

Figura 20: taula corresponent als resultats dels jocs de proves aquí presentats

Podem veure com tots els resultats són coherents, ja que coincideixen entre si (obviant les diferències comentades a la secció 6). Si més no, també s'han contrastat amb un script del manual d'AMPL que es troba en el següent enllaç per acabar de constatar que els resultats són definitivament correctes.

[5] (s'ha adaptat l'input ja que no processa valors amb 0s i per tant dona error al adjudicar 0 al tall 24)

8. Apèndix

8.1 Canvis o afegits que s'han hagut de fer al llarg de la realització del treball

Un dels afegits rellevants a tenir en compte és el script dedicat a mostrar la informació referent a la secció 3, i mostra cada un dels valors de les variables demanades durant l'execució del programa de manera detallada i amb un format fàcil de llegir. S'ha utilitzat per a la secció indicada i per debugar el programa. Cal destacar que, a més a més de la informació demanada, conté una visualització de la matriu D en forma matricial, enlloc de la visualització clàssica mostrada per la comanda *display*.

```
option solver cplex;
option solution_round 5;
model cg.mod
data cg.dat
problem CSP: X_D, unitats, MinUnitats, Demanda, ActualitzarUnitats;
	option presolve 1;
problem NovaColumna: a, CostReduit, LimitAmplada;

let l := 0;
for {i in talls} {
	let l := l + 1;
	let D[i,l] := floor(amplada_total_rotllo/i);
	let {j in talls: i <> j} D[j,l] := 0;
	let longitud[l] := i * D[i,l];
};

solve CSP;
let {i in talls} lambda[i] := Demanda[i].dual;
solve NovaColumna;

set remove;

repeat {
	printf "*******************************\n";
	printf "======== Nova iteracio ========\n";
	printf "*******************************\n\n";
	let remove := {};
	printf "*********************************************************\n";
	printf "Valor de la funció objectiu al principi de la iteracio\n";
	printf "---------------------------------------------------------\n";
	display MinUnitats;
	printf "*********************************************************\n";
	printf "*********************\n";
```

```
	printf "Valor del cost reduit\n";
	printf "---------------------\n";
	display CostReduit;
	printf "*********************\n";
	let l := l + 1;
	let patrons := patrons union {l};
	let {i in talls} D[i,l] := a[i];
	let longitud[l] := sum {i in talls} i * D[i,l];
	printf "*******************************\n";
	printf "Valor de la nova columna per %d\n",l;
	printf "-------------------------------\n";
	display a;
	printf "*******************************\n";
	solve CSP;
	let {i in talls} lambda[i] := Demanda[i].dual;
	solve NovaColumna;
	printf "*****************************************************\n";
	printf "Valor de la funció objectiu al final de la iteracio\n";
	printf "-----------------------------------------------------\n";
	display MinUnitats;
	printf "*****************************************************\n";
	printf "*********************\n";
	printf "Valor del cost reduit\n";
	printf "---------------------\n";
	display CostReduit;
	printf "*********************\n";
	for {i in patrons} if X_D[i] == 0 then let remove := remove union {i};
	let patrons := patrons diff remove;
} while (CostReduit < 0);
printf "Fi de l'execució\n\n";
display D;
for {i in patrons} printf "\t%d",i;
printf "\n";
for {i in talls} {
	printf "%d",i;
	for {j in patrons}
		printf "\t%d",D[i,j];
	printf "\n";
}
printf "\n";
printf "Solucio sense arrodoniment\n\n";
display X_D;
display MinUnitats;
printf "Arrodonim la solucio obtinguda\n\n";
let {i in patrons} X_D[i] := ceil(X_D[i]);
display X_D;
display MinUnitats;
```

Figura 21: codi referent a les visualitzacions de la secció 3

8.2 Ús solvent dels recursos d'informació

La nostra cerca d'informació s'ha centrat en journals acadèmics. Per tant, ens hem centrat en aquells que els autors tenien més citacions o estaven més relacionats amb el mètode de generació de columnes.

(1)

Agarwal, Y.; Mathur, K.; Salkin, M. A Set-Partitioning-Based Exact Algorithm for the Vehicle Routing Problem. *Networks* **1989**, *19*, 731–749. https://doi.org/10.1002/net.3230190702.

(2)

Rönnberg, E.; Larsson, T. All-Integer Column Generation for Set Partitioning: Basic Principles and Extensions. *European Journal of Operational Research* **2014**, *233* (3), 529–538. https://doi.org/10.1016/j.ejor.2013.08.036.

(3)

Larsson, T.; Quttineh, N.-H. Column Generation Extensions of Set Covering Greedy Heuristics. *Operations Research Letters* **2022**, *50* (6), 738–744. https://doi.org/10.1016/j.orl.2022.10.014.

(4)

Clautiaux, F.; Guillot, J.; Pesneau, P. Exact Approaches for Solving a Covering Problem with Capacitated Subtrees. *Computers & Operations Research* **2019**, *105*, 85–101. https://doi.org/10.1016/j.cor.2019.01.008.

(5)

Tahir, A.; Desaulniers, G.; El Hallaoui, I. Integral Column Generation for Set Partitioning Problems with Side Constraints. *INFORMS Journal on Computing* **2022**, *34* (4), 2313–2331. https://doi.org/10.1287/ijoc.2022.1174.

(6)

Integral column generation for the set partitioning problem | Elsevier Enhanced Reader. https://doi.org/10.1007/s13676-019-00145-6.

(7)

Diaby, M. Linear Programming Formulation of the Set Partitioning Problem. *Int. J. Operational Research Int. J. Operational Research* **2010**, *8*, 399–427. https://doi.org/10.1504/IJOR.2010.034067.

(8)

Beliën, J.; Demeulemeester, E. On the Trade-off between Staff-Decomposed and Activity-Decomposed Column Generation for a Staff Scheduling Problem. *Ann Oper Res* **2007**, *155* (1), 143–166. https://doi.org/10.1007/s10479-007-0220-2.

(9)

ResearchGate. https://www.researchgate.net/publication/334060304_Integral_column_generation_for_the_set_partitioning_problem/link/5fda056a45851553a0bf02fe/download (accessed 2023-01-05).

(10)

Hoffman, K.; Padberg, M. Set Covering, Packing and Partitioning ProblemsSet Covering, Packing and Partitioning Problems. In *Encyclopedia of Optimization*; Floudas, C. A., Pardalos, P. M., Eds.; Springer US: Boston, MA, 2009; pp 3482–3486. https://doi.org/10.1007/978-0-387-74759-0_599.

(11)

van Krieken. Solving Set Partitioning Problems Using Lagrangian Relaxation.