Pràctica MiniZinc: Festival de música de cambra
Programació Declarativa: Aplicacions: 2023

Aniol Molero Grau i Bernat Comas Machuca

18 de gener de 2024

Índex

1 Introducció

L'objectiu d'aquesta pràctica és confeccionar els horaris d'un festival de música de cambra a partir d'una sèrie de musics, peces i restriccions econòmiques que ens venen donades. Tots els problemes tenen en comú que ens interessa minimitzar la seva durada global, tocant tantes peces en paral·lel com ens sigui possible sense infringir cap de les restriccions que se'ns donen.

Les dades que obtenim són: el nombre de peces, amb la seva durada i instruments necessaris per tocar-la, el nombre de músics disponibles amb els instruments que saben tocar i el seu salari, i finalment el pressupost i el nombre de peces que s'hauran de tocar abans que d'altres. L'objectiu doncs serà que es toquin totes les peces que se'ns donen de manera que un músic mai hagi de tocar dues peces simultàniament i al mateix temps sense sobrepassar el pressupost de contractació de músics, tot respectant les precedències que se'ns indiquen. A l'apartat 5 es desenvolupa la part extra, on s'inclou alguna restricció addicional a què de moment no farem referència.

Per altra banda, també sabem que les peces només podran començar i durar múltiples de 5 minuts.

2 Metodologia i Output

2.1 Metodologia

Aprofitant la restricció sobre la durada de les peces (les peces només poden començar i durar múltiples de 5 minuts) treballarem en tota la codificació amb els temps dividits entre 5, i en el moment de mostrar per pantalla els tornarem a multiplicar per 5. L'únic impacte que té aquesta metodologia en les entrades és que haurem de dividir les durades que ens donen inicialment entre 5 abans de fer càlculs.

Per tal d'acotar els valors que poden prendre les s (els temps d'inici), calculem un upper bound (UB) que en diem "dummy". Aquest UB es calcula com a la suma de les durades de totes les peces. Ho podem fer d'aquesta manera perquè sabem segur que existirà una solució que serà la concatenació de totes les activitats una darrera l'altra sense descansos, i també sabem que cap alternativa que introdueixi descansos podrà oferir una durada del festival inferior a aquesta. Veurem que aquesta norma s'haurà de canviar a la part extra i caldrà redefinir el càlcul de l'UB dummy.

2.2 Output

Per generar l'output hem decidit mostrar-lo de la mateixa manera que es diu a l'enunciat de la pràctica, ja que ens ha semblat manera entenedora i clara.

En primer lloc, mostrem l'hora d'inici i fi del festival i el cost de contractació. Seguidament, mostrem per cada peça l'hora d'inici, hora de fi i el conjunt de músics que la toquen amb l'instrument que utilitzen. A continuació podem veure un exemple de l'output que generarà el minizinc.

```
Hora d'inici del festival: 10h 0min
Hora d'acabada del festival: 10h 45min
Cost de contractacio: 88
Peca 1:
  Inici: 10h 0 min
  Final: 10h 25 min
  Music 5 toca instrument 1
  Music 6 toca instrument 2
  Music 7 toca instrument 3
  Music 8 toca instrument 1
Peca 2:
  Inici: 10h 0 min
  Final: 10h 5 min
  Music 1 toca instrument 1
  Music 2 toca instrument 3
  Music 3 toca instrument 2
  Music 4 toca instrument 1
Peca 3:
  Inici: 10h 25 min
  Final: 10h 45 min
  Music 6 toca instrument 2
  Music 8 toca instrument 2
Peca 4:
  Inici: 10h 25 min
  Final: 10h 40 min
  Music 3 toca instrument 2
  Music 7 toca instrument 3
----------
```

3 Viewpoints

3.1 Viewpoint Principal: Start times

En el viewpoint principal utilitzarem les següents variables:

1. Start times: Array de mida nPeces que contindrà a cada posició el temps d'inici de la peça que ocupa aquella posició en l'assignació inicial. Pot prendre valors entre 0 i dummyUB.
2. Músics Assignats: Matriu de mida nPeces x nMusics que contindrà en cada posició el nombre de l'instrument que cada músic toqui en una peça en concret. En cas que un músic no toqui cap instrument en aquella peça en aquella posició hi haurà un 0. Pot prendre valors entre 0 i nInstruments
3. Preu: Variable auxiliar que contindrà el total que cobraran tots els músics que participin en un festival en concret. Pot prendre valors entre 0 i pressupost.
4. Max Value: Variable auxiliar utilitzada només per mostrar l'output, contindrà l'instant de temps que correspon al moment de finalització de l'última peça que es toca en el concert, és a dir, a l'instant de finalització del concert. Contindrà el valor final, és a dir, multiplicat per 5. És un enter en què no s'especifica cap cota, però es calcula a partir de la variable s i constants fent servir la fórmula s[i]*5+d[i]*5. Això vol dir que pot prendre valors entre 0*5+0*5=0 i last_tasca(s)*5+last_tasca(d)*5, que serà dummyUB*5+last_tasca(d)*5.

A més d'aquestes variables fem servir també un array precalculat que s'anomena poolInstruments. Aquest array ens servirà per saber quin és el màxim nombre de cada

tipus d'instrument que tenim disponible. D'aquesta manera tractarem els instruments com a recursos renovables. Podem saber el màxim nombre de cada instrument que tindrem disponibles en un instant si sumem els músics que saben tocar aquell instrument. Tot i això, no els podem tractar com a recursos independents, ja que un mateix músic pot tenir diverses especialitats, el que complica més el problema.

3.1.1 Restriccions Necessàries

1. Successió

   La clàusula de successió és molt simple. Per totes les peces on tinguem prioritat, el temps d'inici d'un successor serà major o igual que el temps de fi del seu predecessor. Amb això forcem que es respectin les prioritats definides per l'enunciat.

```
/* Constraint que assegura que es respecta la prioritat */
constraint forall(i in 1..nPrecs) (
  s[succ[i]] >= s[pred[i]]+d[pred[i]]
);
```

2. Que cada peça tingui els instruments que necessita durant l'execució.

   Per assegurar que totes les peces tenen els instruments que necessiten, comptem per cada peça i instrument, que el nombre de músics que tenen assignat aquell instrument en aquella peça sigui igual al nombre d'instruments d'aquell tipus que requereix la peça. Per fer-ho utilitzem el predicat count_eq, que força que el nombre d'elements de la llista que ocupa el primer paràmetre (en el nostre cas tindrà per cada músic l'instrument que toca en aquella peça o bé 0), que són iguals a j (cada instrument), ha de ser igual al nombre d'instruments d'aquell tipus que requereix la peça.

```
/* Que cada pe a tingui els instruments que necessita durant l'
    execucio */
constraint forall(i in 1..nPeces, j in 1..nInstruments)(
    count_eq([musicsAssignats[i,k]|k in 1..nMusics], j, requereix[i,
    j])
);
```

3. Que cada músic només toqui instruments que sap tocar.

   Per fer que cada músic toqui només algun instrument dels que sap tocar, forcem que saptocar d'aquell músic amb l'instrument que té assignat a qualsevol de les peces sigui true. Hem d'assegurar-nos que l'instrument que té assignat no sigui 0 (no té instrument assignat) quan generem la clàusula.

```
/* Que un music toqui un instrument que li toca. */
constraint forall(i in 1..nPeces, j in 1..nMusics)(
    if musicsAssignats[i,j]!=0 then saptocar[j,musicsAssignats[i,j]]
     endif
);
```

4. Un músic només pot tocar una peça a la vegada.

   Per assegurar que un músic no pot tocar dos instruments a la vegada, utilitzarem el predicat disjunctive entre els temps d'inici i les durades de les dues peces que utilitzen el mateix músic. El disjunctive s'assegura que dues activitats amb els seus inicis i duracions no se superposin.

```
/* Si dues peces s n concurrents, llavors un mateix m sic no pot
    tocar a les dues */
constraint forall(i in 1..nPeces, m in 1..nMusics, j in (i+1)..
    nPeces)(
      if musicsAssignats[i,m] != 0 /\ musicsAssignats[j,m] != 0 then
        disjunctive([s[i],s[j]],[d[i],d[j]])
      endif
);
```

3.1.2 Restriccions Implicades

1. Càlcul del Lower Bound

   Els càlculs de lower bound en els problemes de minimització de temps sempre són restriccions implicades (ja que podriem utilitzar com a LB dummy el 0). En aquest cas per trobar el LB ens adonem que la pool d'instruments ens marca clarament un mínim de temps que haurà de durar el festival. Aquest es pot calcular si sabem quina és la durada total de les peces que requereixen un instrument en concret i es divideix pel nombre d'instruments d'aquell tipus que tenim disponibles. Llavors estem segurs de què la durada del festival serà major al màxim d'aquests càlculs per tots els instruments.

   Una ampliació d'aquesta restricció que ens acostaria més al mínim real (augmentant el LB) seria fer sumes d'aquests màxims tenint en compte els músics que toquen diversos instruments i la seva concurrència, però hem pensat que seria massa costós i llarg per valdre la pena.

```
/* Lower bound per el maxim de temps */
constraint max([s[i]+d[i]| i in 1..nPeces]) > max([sum([d[p]*
    requereix[p,i]/poolInstruments[i]| p in 1..nPeces]) | i in 1..
    nInstruments]);
```

2. Activitat a temps 0.

   Forçant que s'iniciï una activitat a temps 0 no ens estem perdent cap solució òptima, ja que com hem explicat, afegir espais entre les peces no pot fer que millori una solució de temps mínim. El mateix passa si afegim espai a l'inici, amb la diferència que ni tan sols en la part extra ens pot ser útil afegir un espai al principi.

   Per aquesta raó hem afegit aquesta clàusula que força que com a mínim una activitat comenci en temps 0.

```
/* La primera can    ha de comen ar al principi del festival */
    constraint min([s[i] | i in 1..nPeces]) = 0;
```

3. Si dos músics toquen el mateix, tindrà prioritat el que cobra menys.

   Restricció pensada per trencar simetries. En cas que dos músics toquin els mateixos instruments voldrà dir que són completament intercanviables i, per tant, que sempre podrem obtenir les solucions simètriques amb els dos intercanviats. Per aquesta raó només ens interessa una de les dues solucions, i hem decidit trencar la simetria quedant-nos amb la solució que té el menor cost.

   Això no vol dir que sempre agafem primer el que cobra menys, perquè això podria fer que no es trobés una solució amb un cost dintre del pressupost, ja que, per exemple, si primer es toca una cançó curta i abans que aquesta acabi, en comença

una de molt llarga, si a la curta hi assignem el músic més barat i a la llarga el més car, ens podríem passar de pressupost. El que estem amb aquesta restricció és que, al final, l'estona que ha tocat el músic més barat ha de ser superior o igual al temps que ha tocat el músic car.

```
/* Si dos m sics son exactament iguals farem servir m s el que
    cobra menys */
constraint forall(i in 1..nMusics, j in i..nMusics)(
  if forall(k in 1..nInstruments)(saptocar[i,k] = saptocar[j,k])
    then
    if salari[i] <= salari[j] then
      sum([d[p] | p in 1..nPeces where musicsAssignats[p,i]>0]) >=
    sum([d[p] | p in 1..nPeces where musicsAssignats[p,j]>0])
    else
      sum([d[p] | p in 1..nPeces where musicsAssignats[p,i]>0]) <=
    sum([d[p] | p in 1..nPeces where musicsAssignats[p,j]>0])
    endif
  endif
);
```

4. Cumulative, que no s'assignin més instruments que músics que el saben tocar.

   Un predicat que des del principi ens ensumàvem que milloraria els resultats si l'introduíem és el cumulative. Aquest predicat força que a partir d'unes activitats amb uns determinats start times, durades i consums de recursos, el consum total no superi un límit fix establert. La quantitat d'instruments disponibles era l'excusa perfecta per aplicar aquest predicat i ens ha millorat els resultats notablement.

   Per tenir el màxim nombre de cada instrument que tenim disponible hem utilitzat la constant explicada anteriorment poolInstruments. Com hem dit, aquesta constant ens indica un upper bound màxim d'instruments que podrem fer servir a la vegada, per tant, en cap moment podrem emprar a la vegada més instruments d'un tipus dels que ens indica aquest array. Afegirem doncs un cumulative diferent per cada un dels instruments.

```
/* Assegura que per cada instrument en cap moment s'assignin m s
    peces que el toquen que els musics que el saben tocar. */
constraint forall(i in 1..nInstruments) (
  cumulative(s,d,[requereix[p,i]|p in 1..nPeces],poolInstruments[i])
);
```

3.1.3 Restriccions Descartades

Al llarg de la confecció del model de MiniZinc hem tingut diverses idees, algunes que han avançat més i altres que hem vist que no tindrien futur i hem optat per no desenvolupar. A continuació expliquem quines són les restriccions que teníem desenvolupades, però que no hem acabat introduint perquè no ajudaven al model o empitjoraven el rendiment.

1. Global cardinality constraint.

   Assegurem que cada peça tindrà assignats el nombre d'instruments que són requerits. La constraint global cardinality assegura que cada element de la llista del segon paràmetre apareixerà a la llista del primer un nombre de vegades igual al mateix índex en l'array del tercer paràmetre. No podem utilitzar el predicat global cardinality closed perquè no existeix l'instrument 0.

```
/* Assegura que apareixeran el nombre d'instruments que son
    requerits. */
constraint global_cardinality([musicsAssignats[i,j]|i in 1..nPeces,
    j in 1..nMusics],[i|i in 1..nInstruments],[sum([requereix[p,i] |
     p in 1..nPeces])|i in 1..nInstruments]);
```

2. Les peces començaran a l'instant 0 o a l'instant on acaba una altra peça.

   Comprovem que tots els temps d'inici siguin 0 o bé existeixi un altre peça que acabi en aquell instant.

   La idea darrera d'aquesta restricció és que, deixant de banda la part extra, en una organització òptima d'un festival no hi hauria d'haver mai espais buits, ja que seria temps extra innecessari. A més, sempre intentarem tocar les cançons tan aviat com puguem. Per tant, com que l'única cosa que pot provocar un canvi en les cançons que podem tocar en un moment donat és que hagi acabat una cançó, no té sentit que provem de fer que una cançó comenci en algun altre instant (excepte a l'inici del festival, és clar), ja que si no la podíem tocar abans, si els músics disponibles no han canviat, és evident que ara tampoc podrem tocar-la.

```
/* Una can    o b   comen a al principi, o b   comen a quan una
    altra can    acaba */
constraint forall(i in 1..nPeces)(
  s[i] = 0 \/ exists(j in 1..nPeces where j != i)(s[j] + d[j] = s[i
    ])
);
```

3. Als músics que no sàpiguen tocar cap dels instruments requerits per una peça hauran de tenir forçosament musicsAssignats a 0.

```
constraint forall(p in 1..nPeces, m in 1..nMusics) (
  let {
    var bool: foundInstrument = false;
  } in
  if not exists(i in 1..nInstruments) (saptocar[m, i] /\ requereix[p
    , i]>0) then
    musicsAssignats[p, m] = 0
  endif
);
```

4. Si dues peces són concurrents, un mateix músic no pot tocar a les dues.

   Per assegurar que un mateix músic no pugui tocar dues peces a la vegada en aquest cas ho fem obligant que la multiplicació entre els instruments de les dues peces que toca aquest músic sigui 0. Posteriorment, vam veure que la multiplicació és molt ineficient i la vam substituir per la restricció que es troba a "Restriccions necessàries"al punt 4.

```
/* Constraint que assegura que un m sic no pugui de tocar dos peces
     simult niament */
constraint forall(i in 1..nPeces, m in 1..nMusics, j in 1..nPeces)(
      % Si dues peces s n concurrents, llavors un mateix m sic no
    pot tocar a les dues
      if s[i] <= s[j] /\ s[i] + d[i] > s[j] /\ i!=j then
    musicsAssignats[i,m]*musicsAssignats[j,m]=0 endif
);
```

5. No es poden executar a la vegada peces que requereixin més músics que els que tenim.

   Simètricament al que hem fet amb els instruments utilitzant un cumulative per obligar que no es passi del nombre d'instruments de cada tipus que tenim, vam pensar que ho podríem provar amb el nombre màxim de músics que tenim. Com era previsible, no redueix el temps de cerca i fins i tot l'empitjora, així que ràpidament la vam descartar.

```
/* Constraint per dir que no es poden a la vegada peces que
    requereixin m s m sics que els que tenim. */
%constraint cumulative(s,d,[sum([requereix[p,i] | i in 1..
    nInstruments])|p in 1..nPeces],nMusics);
```

6. Noves variables de upper i lower bounds per cada peça.

   Aquesta idea de lower i upper bounds recursiu no ens ha arribat a funcionar degut a la sintaxi del minizinc. En aquest cas com que estem definint els bounds d'starting times a partir dels valors màxims de s dels seus predecessors, ens diu que s'ha trobat una inconsistència i que el model és no satisfactible.

```
/* C LCULS DE LOWER BOUNDS I UPPER BOUNDS */
constraint forall(i in 1..nPeces) ( % LB
    s[i] >= max([s[pred[j]]+d[pred[j]] | j in 1..nPrecs where succ[j
    ]==i])
);

constraint forall(i in 1..nPeces) ( % UB
    s[i] <= min([s[i]-d[pred[j]] | j in 1..nPrecs where succ[j]==i])
);
```

3.2 Viewpoint Alternatiu: Is Playing

En el viewpoint alternatiu utilitzarem les següents variables:

1. Is Playing: Matriu de mida nPeces x 0..dummyUB que contindrà a cada posició un 0 si la peça no s'està executant i un 1 si la peça s'està executant. Pot prendre valors 0 o 1 segons si la peça s'executa.
2. Músics Assignats: Matriu de mida nPeces x nMusics que contindrà en cada posició el nombre de l'instrument que cada músic toqui en una peça en concret. En cas que un músic no toqui cap instrument en aquella peça en aquella posició hi haurà un 0. Pot prendre valors entre 0 i nInstruments
3. Preu: Variable auxiliar que contindrà el total que cobraran tots els músics que participin en un festival en concret. Pot prendre valors entre 0 i pressupost.
4. Max Value: Variable auxiliar que contindrà l'instant de temps que correspon al moment de finalització de l'última peça que es toca en el concert, és a dir, a l'instant de finalització del concert. Contindrà el valor final. És un enter sense especificar cap cota, però es calcula a partir de les variables max_values. Per tant, pot prendre valors entre 0 i dummyUB.
5. Max Values: Array de variables que conté els temps de fi de cada una de les peces. Es calcula buscant el màxim temps on s'està executant una peça, ja que calcular-lo amb el min_values i durada empitjora el rendiment. Pot prendre valors entre 0 i dummyUB.

6. Min Values: Array de variables que conté el temps més petit on s'està executant una peça en concret. Necessari per calcular el temps d'inici de les peces.

A més d'aquestes variables fem servir de la mateixa manera que en el viewpoint principal l'array precalculat poolInstruments. La seva funció serà definir el màxim nombre d'instruments de cada tipus que tenim disponible en un instant com a màxim, i els podrem tractar com a recursos renovables.

3.2.1 Restriccions Necessàries

1. Prioritat

   Aquesta restricció ens assegura que es respecta la prioritat de peces indicada per l'enunciat. Només cal indicar que totes les variables x que estiguin a 1 del successor han de ser posteriors a les del predecessor. La clàusula que podem veure implica per tots els temps posteriors a l'execució del successor d'una peça no es pot estar executant la peça original.

```
/* Constraint que assegura que es respecta la prioritat */
constraint forall(i in 1..nPrecs, j in 0..dummyUB, t in j..dummyUB)
    (
    /* x[succ[i],j] -> not x[pred[i],t] */
    /* si el successor s'est  tocant a j, no pot ser que el
    predecessor es toqui a t (j o m s tard)  */
    not x[succ[i],j] \/ not x[pred[i],t]
);
```

2. Que cada peça tingui els instruments que necessita durant l'execució.

   Exactament la mateixa constraint que teniem al viewpoint 1. Per assegurar que totes les peces tenen els instruments que necessiten, comptem per cada peça i instrument, que el nombre de músics que tenen assignat aquell instrument en aquella peça sigui igual al nombre d'instruments que requereix la peça. Per fer-ho utilitzem el predicat count_eq, que força que el nombre de músics assignats a la peça ï"que toquen un instrument concret, sigui igual al nombre d'aquell instrument que requereix la peça.

```
/* Que cada pe a tingui els instruments que necessita durant l'
    execucio */
constraint forall(i in 1..nPeces, j in 1..nInstruments)(
    count([musicsAssignats[i,k]|k in 1..nMusics], j) == requereix[i,
    j]
);
```

3. Durada de les peces

   Per forçar que una peça duri d[i] instants hem de comptar que el nombre de variables de temps posades a 1 de la peça en qüestió sigui igual als temps que ha de durar.

```
/* Que cada  c a n    duri el que ha de durar */
constraint forall(i in 1..nPeces)(
    count([x[i,t]|t in 0..dummyUB], true) == d[i]
);
```

4. Que un músic toqui un instrument que sap tocar

   També una clàusula que és igual en els dos viewpoints. Per forçar que cada músic toqui un instrument dels que sap tocar forcem que saptocar d'aquell músic amb l'instrument que té assignat a qualsevol de les peces sigui true. Hem d'assegurar-nos que l'instrument que té assignat no sigui 0 (no té instrument assignat) quan generem la clàusula.

```
/* Que un music toqui un instrument que li toca. */
constraint forall(i in 1..nPeces, j in 1..nMusics)(
    if musicsAssignats[i,j]!=0 then saptocar[j,musicsAssignats[i,j]]
     endif
);
```

5. Que un músic no pugui tocar dues peces a la vegada.

   Per tal d'impossibilitar que un músic toqui dues peces a la vegada obliguem a que un músic estigui assignat com a molt a una peça simultània. Ens ajudem del predicat count per obtenir les peces que s'executen en un instant, i d'aquests, forcem a que com a molt una tingui el nombre de músics major a 1.

```
constraint forall(t in 0..dummyUB, m in 1..nMusics) (
    count(i in 1..nPeces where x[i,t])(musicsAssignats[i,m]>0)<=1
);
```

6. Evitem preempció

   Per tal d'evitar que una peça es pugui parar de tocar i reprendre en un altre moment utilitzem aquesta clàusula. Com que sabem quin serà l'inici i el final de la peça buscant en quin temps es troben la primera i última x d'una peça en concret, la seva resta +1 (ja que el final és l'últim temps on es toca) haurà de ser igual a la durada de la peça.

```
constraint forall(p in 1..nPeces)(
  d[p] = max_values[p]-min_values[p]+1
);
```

3.2.2 Restriccions Implicades

1. Activitat a temps 0.

   Semblantment a com fem en el viewpoint principal, forçant que s'iniciï una activitat a temps 0 no ens estem perdent cap solució òptima, ja que com hem explicat, afegir espais entre les peces no pot fer que millori una solució de temps mínim.

```
/*  La primera can    ha de comen ar al principi del festival */
constraint exists(i in 1..nPeces)(x[i,0]);
```

3.2.3 Restriccions Descartades

1. Lower bound d'instruments

   Seguint una idea similar a la clàusula que hem afegit al viewpoint principal vam provar aquesta clàusula, que posa un lower bound al temps d'execució trobant el mínim temps que duraria executar cada peça en funció dels músics que saben tocar un instrument. En aquest viewpoint, però, no ajuda a reduïr el temps de cerca.

```
constraint max([t| t in 1..dummyUB, p in 1..nPeces where x[p,t]]) >
    max([sum([d[p]*requereix[p,i]/poolInstruments[i]| p in 1..nPeces
    ]) | i in 1..nInstruments]);
```

2. Temps d'inici d'una peça

   Hem pogut observar que forçar que una peça comenci només en temps 0 o en temps de finalització d'una altra peça ens evita simetries no desitjades. Per exemple si estem tocant dues peces simultàniament i una és més llarga que l'altra, trobarem diferents maneres de posicionar-les entre elles, que no ens interessen, ja que no poden millorar el temps de finalització. Per aquesta raó ens quedarem per exemple només amb la versió on totes dues comencen a la vegada.

   Aquesta clàusula força que si una peça s'està tocant, o bé sigui en temps 0, o bé ja s'estigués tocant prèviament o hi ha una altra peça que ha acabat just en aquell instant.

```
/* Una can    o b   comen a al principi, o b   comen a quan una
    altra can    acaba */
constraint forall(p in 1..nPeces, t in 0..dummyUB)(
   not x[p,t] \/ t=0 \/ x[p,t-1] \/ exists(i in 1..nPeces)(x[i,t-1]
    /\ not x[i,t])
);
```

3. No hi hagi forats buits

   Amb aquest constraint volíem assegurar que no deixem cap temps on cap peça està en execució. És implicat perquè estem fent minimització del temps.

```
/* No hi pot haver forats buits! */
constraint forall(t in 0..dummyUB where t < max_value) (
  exists(p in 1..nPeces)(x[p,t])
);
```

4 Experiments

En aquesta secció mostrarem resultats empírics d'execució dels nostres viewpoints amb MiniZinc. Les proves que durem a terme seran les següents:

1. Comparació dels diferents viewpoints
2. Comparació de diferents solvers
3. Taula final d'execucions de tots els festivals

4.1 Metodologia d'experimentació

Per provar els nostres viewpoints hem utilitzat els diferents solvers del MiniZinc IDE. En concret ens centrarem en els solvers Gecode 6.3.0, COIN-BC 2.10.11/1.17.9 i OR-Tools CP-SAT 9-8-3296. En un principi pensàvem fer servir també el Chuffed 0.13.1, però l'hem descartat perquè acaba l'execució en molt poques instàncies. Per altra banda, no podem provar totes les execucions amb cadascun perquè a causa de les limitacions d'alguns d'ells, n'hi ha que no acaben en hores el que altres fan en segons. Les proves les farem amb 1 i 8 threads. No provem amb altres nombres majors d'1 perquè quan ho fem ens apareix un missatge en gris que ens indica que s'empren 8

threads igualment. Pel solver OR-Tools hem optat per no fer ús de l'opció de "Free Search", ja que en totes les proves que hem fet ens empitjoraven els resultats.

Després de fer proves amb totes les estratègies de cerca de MiniZinc, ens hem quedat amb int_search de les variables s (ja que són les que volem minimitzar), amb tria per smallest (tria la variable amb domini més petit), i constraint choice indomain_min, que assigna a una variable el seu valor més petit dins el domini (lògicament, perquè volem que totes comencin tan aviat com sigui possible). Per agafar les mostres hem utilitzat una mitjana de tres execucions de cada un.

Totes les proves s'han dut a terme amb una màquina amb les següents característiques:

- Processador: 12th Gen Intel Core i5-12400F
- Memòria: 16.0 GB DDR4
- Targeta Gràfica: NVIDIA GeForce RTX 3060
- Memòria Física: SSD M.2

4.2 Comparació dels diferents viewpoints

Per testejar els diferents viewpoints hem decidit comparar els temps d'execució del solver OR-Tools CP-SAT 9-8-3296. Haviem trobat interessant provar també el Gecode, el Chuffed o algun dels altres que provem a la secció següent però finalment hem optat per no fer-ho degut a la enorme diferència que hi ha entre les seves execucions, tal i com podrem comprovar.

Els temps d'execució dels festivals entre els viewpoints són els que es poden veure a la figura 1. A la figura 2 podem veure en un gràfic la comparació per cada instància del temps que tarda a ser resolta pel solver escollit en cada un dels viewpoints que hem creat.

Instance \ Viewpoint	Viewpoint 1	Viewpoint 2
0	0,340	0,421
1	0,393	6,659
2	0,369	7,936
3	0,818	28,844
4	0,578	11,350
5	0,388	47,280
6	1,358	40,560
7	1,217	41,526
8	-	-
9	2,829	333,000
10	1,618	62,000

Figura 1: Resultats d'execució dels viewpoints .

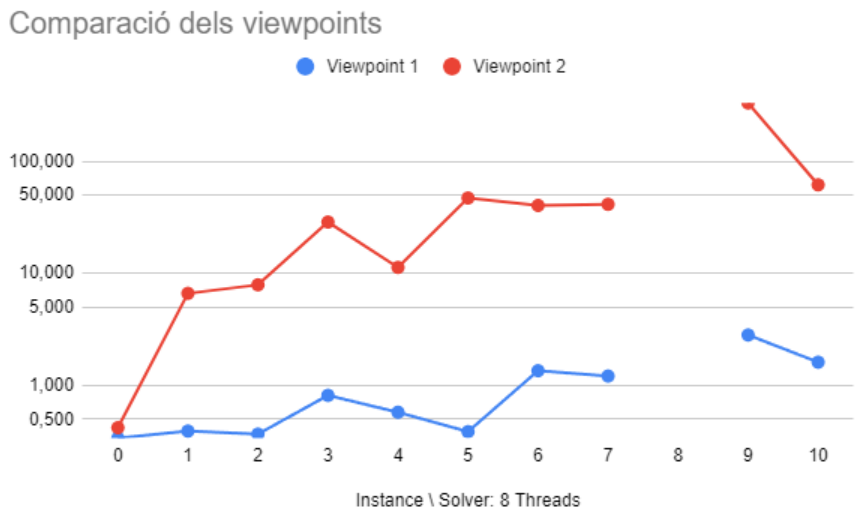

Figura 2: Gràfic d'execució dels festivals segons el viewpoint. L'escala de l'eix vertical és logarítmica per tal de facilitar la comparació dels resultats en totes les instàncies.

Tal com podem veure a les taules dels resultats de cada viewpoint, és fàcil adonar-se que el primer viewpoint és molt més ràpid que el segon en tot tipus d'instàncies. Per això, al següent apartat farem referència només al primer viewpoint, ja que és amb el que ens quedaríem si n'haguéssim d'escollir un.

4.3 Comparació de diferents solvers

Per tal de comparar els diferents solvers hem provat la seva execució amb un sol thread i amb 8 threads. Les taules es poden veure a la Figura 3 i Figura 4. El solver anomenat OR-WO Chan és l'OR-Tools sense el channeling que hem incorporat.

Instance \ Solver: 1 Thread	OR-Tools CP-SAT 9-8-3296	Gecode 6.3.0	COIN-BC 2.10.11/1.17.9	OR-WO Chan
0	0,332	0,330	0,463	0,367
1	1,547	0,607	1,975	1,458
2	0,372	163,000	2,556	0,440
3	2,297		23,390	2,330
4	12,320		3,390	11,608
5	0,393	5,776	1,782	0,411
6	36,546			45,285
7				
8				
9	5,190			4,888
10				

Figura 3: Resultats d'execució dels diferents solvers amb 1 thread i el viewpoint 1.

Instance \ Solver: 8 Threads	OR-Tools CP-SAT 9-8-3296	Gecode	COIN-BC 2.10.11/1.17.9	OR-WO Chan
0	0,340	0,313	0,463	0,330
1	0,393	0,472	1,975	0,429
2	0,369	40,141	2,556	0,377
3	0,818		23,390	0,872
4	0,578		3,390	0,609
5	0,388	0,507	1,782	0,400
6	1,358			1,322
7	1,217			1,648
8				
9	2,829			3,100
10	1,618			1,998

Figura 4: Resultats d'execució dels diferents solvers amb 8 threads i el viewpoint 1.

Els apartats que veiem buits són perquè no han acabat en 30 min. L'única excepció és el festival 8 en OR-Tools, vegeu 4.5.

A partir d'aquestes taules en podem treure gràfics que ens permetran veure com canvia el temps de resolució en funció de la instància i del solver. Podem veure aquests gràfics a les figures 5 i 6

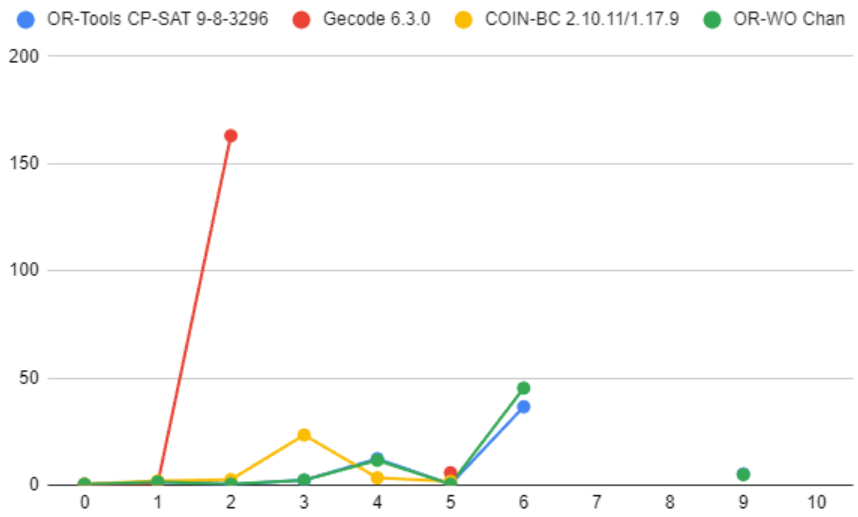

Figura 5: Gràfic d'execució dels diferents solvers amb 1 thread i el viewpoint 1.

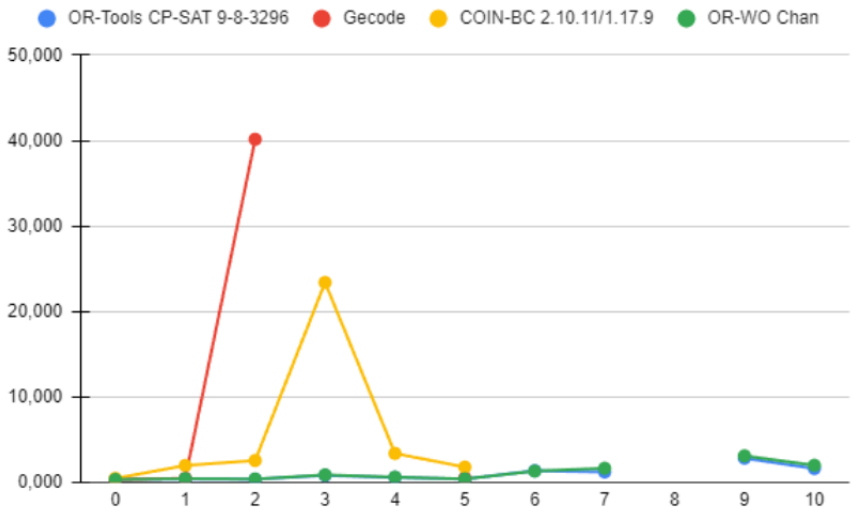

Figura 6: Gràfic d'execució dels diferents solvers amb 8 threads i el viewpoint 1.

Com podem veure als resultats experimentals, l'OR-Tools amb channeling és significativament millor que tota la resta de solvers per instàncies grans. Això es pot deure a diverses raons[2]. En primer lloc, té l'opció de multithreading, cosa que permet paral·lelitzar càlculs. A més té les seves pròpies implementacions dels constraints globals, que poden diferir de la resta de solvers, i causa un gran impacte en el nostre viewpoint escollit (el primer) a causa del gran nombre de restriccions globals que tenim (cumulative, disjunctive, count...).

Observem també com és obvi que el channeling que hem triat és millor que no utilitzarne, doncs l'estratègia consisteix en provar valors petits d'inici dels festivals primer.

4.4 Taula final d'execucions de tots els festivals

A la següent taula podem veure els resultats òptims que hem trobat per a cada festival. En el cas de la instància 8 no podem assegurar que sigui la millor solució, però és la millor que hem aconseguit trobar. Malgrat que no s'indiqui a la taula, tots els festivals comencen a les 10:00. Cal tenir en compte que en alguns casos el cost es podria millorar, però això no era un requisit, només calia mantenir-se dins del pressupost, i intentar calcular, per la durada mínima, el seu cost mínim, feia el problema molt més difícil de computar:

Festival	Hora d'acabada	Cost
0	10:25	80
1	12:10	444
2	13:50	534
3	12:50	770
4	11:50	607
5	13:55	408
6	15:05	1300
7	14:10	1368
8*	14:25	2081
9	18:15	1430
10	14:10	1908

Figura 7: Solucions òptimes dels diferents festivals

4.5 Sobre la instància 8

La instància 8 és l'única instància que no arriba a trobar una solució final. Tot i això en pocs segons (menys de 10 s) trobem una solució que ens diu que el festival pot començar a les 10:00 i acabar a les 14:25, l'execució no acaba en més de dues hores. Això probablement es deu al fet que, per demostrar que una solució és la millor, hem de provar totes les opcions possibles fins a determinar que no hi ha cap manera alternativa que sigui millor, és a dir, que cal explorar tot l'espai de cerca.

Aquest fet provoca que demostrar que una solució és la millor és molt més difícil que trobar-la, de manera que en alguns casos concrets, on a causa de les dades d'entrada l'espai de cerca és molt gran, sigui massa costós arribar a explorar tot l'espai de cerca.

```
Hora d'inici del festival: 10h 0min
Hora d'acabada del festival: 14h 25min
Cost de contractació: 2081
```

Figura 8: Resultats del festival8 sense finalitzar l'execució

5 Part extra

La part extra d'aquesta pràctica consisteix a implementar una nova restricció. Aquesta nova restricció determina que no es poden programar peces on es toqui l'instrument número 2 més de 50 minuts seguits. Això significa que en cas que dues peces s'estiguin tocant al mateix temps amb l'instrument 2, la durada entre el principi de la primera i el final de l'última no pot ser superior a 50, i el mateix passa si són més de dues peces seguides.

Tal com hem explicat a la introducció, aquesta nova restricció ens força a canviar el dummy UB. Per fer l'ajustament requerit tindrem en compte que les activitats no poden durar més de 50 minuts. En cas que ho fessin hem introduït una nova restricció que farà que la resolució falli. Tenint això en compte, sabem que existirà una solució que consistirà a posar totes les peces una darrere l'altra afegint per cada una de les que fan servir l'instrument 2 un temps de 5 minuts de descans. Probablement aquest UB sigui millorable, però ens quedem amb aquest dummy UB per la seva rapidesa a ser calculat.

La nostra implementació es basa en la següent idea. Per tal d'evitar que hi hagi 50 minuts de música amb l'instrument 2 seguits, comprovem si per cada peça que toca l'instrument 2 té una peça que també el toca a l'inici de la primera + 50 minuts. En cas que no sigui així sabem que no portarem més de 50 minuts seguits.

Altrament, haurem de forçar que entre l'inici de la peça i l'inici + 50 minuts hi hagi algun instant de temps on totes les peces que s'executen no necessitin l'instrument 2.

Totes aquestes restriccions les hem de fer tenint en compte que estem treballant amb els temps dividits entre 5 per les raons explicades a la introducció, per tant, estarem sumant 10 en comptes de 50.

A més d'això hem decidit afegir una altra restricció que força que qualsevol peça que faci servir l'instrument 2 duri menys de 50 minuts, per tal de complir amb els requisits de l'enunciat.

```
/* Cap peca que fa servir l'instrument 2 pot durar mes de 50.*/
constraint forall(p in 1..nPeces) (
 % Fem els calculs sempre amb temps % 5
 if  d[p]>10 /\ requereix[p,2]>0 then
  false
  endif
);

constraint forall(p in 1..nPeces where requereix[p,2]>0) ( /* Per cada
    peca que requereixi instrument 2 */
  /* Existeix una peca que requereixen l'instrument 2 que acaba entre la
    primera i la primera+50 */
  if exists(k in 1..nPeces where s[k]+d[k]>=s[p]+10 /\ s[k]<=s[p]+10)(
    requereix[k,2]>0) then% s[p]+50 existeix una peca on requereix[k,2]
    exists(i in 1..nPeces where requereix[i,2]>0 /\ s[i]+d[i]>s[p] /\ s[i
    ]+d[i]<=s[p]+10)(
      /* On totes les peces que es toquen en aquell moment no necessiten
    l'instrument 2 */
      forall(j in 1..nPeces where s[j] <= s[i]+d[i] /\ s[j]+d[j] > s[i]+d
    [i])( /* Que comenci abans d'aquell moment o en aquell i acabi
    despres d'aquell moment*/
            requereix[j,2]=0
      )
    )
  endif
);
```

A la següent taula podem veure els resultats d'execució dels diferents festivals, juntament amb el seu cost i hora d'acabada. Les execucions les hem fet utilitzant el solver OR-Tools CP-SAT 9-8-3296 i amb 8 threads.

Festival	Temps Extra	Temps normal	Fi extra	Fi normal	Cost extra	Cost normal
0	0,343	0,340	10:25	10:25	79	80
1	0,454	0,393	12:10	12:10	440	444
2	0,451	0,369	13:55	13:50	562	534
3	1,247	0,818	12:50	12:50	770	770
4	0,733	0,578	11:50	11:50	586	607
5	0,491	0,388	14:05	13:55	431	408
6	1,385	1,358	15:20	15:05	1303	1300
7	2,388	1,217	14:15	14:10	1361	1368
8						
9	21,885	2,829	18:25	18:15	1423	1430
10	3,490	1,618	14:45	14:10	1909	1908
1 Extra	0,821	0,419	12:30	12:20	486	487

Figura 9: Taula d'exeució dels festivals en la part extra comparats amb la part normal

I en el següent gràfic podem veure representats els temps d'execució de cada festival.

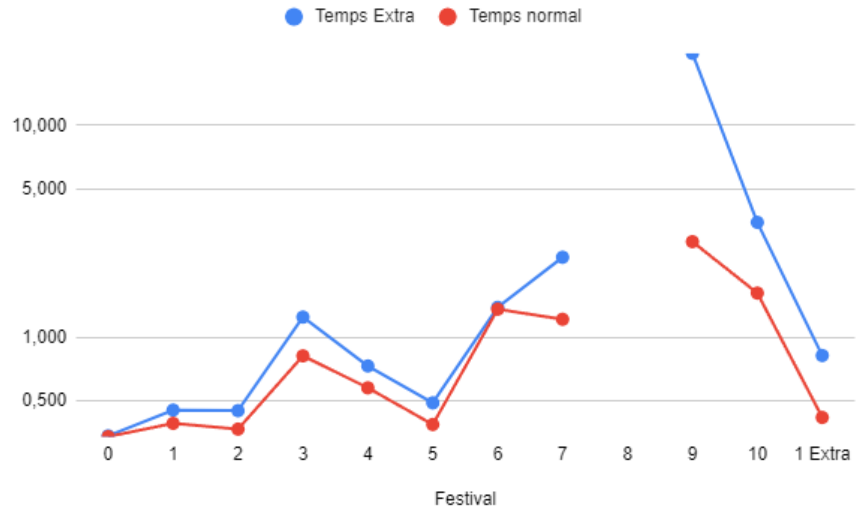

Figura 10: Gràfic on es pot veure l'evolució del temps d'execució segons el festival.

Aplicant aquesta nova restricció evidentment canvien els resultats i el rendiment. Per exemple en la peça 2 la durada mínima és 5 minuts més alta que sense la restricció, i això es deu a una pausa de músics que toquen l'instrument 2. Com és evident els festivals podran o bé augmentar en temps o bé quedar-se iguals que estaven (com en el cas de la peça 0), ja que afegir noves restriccions en cap cas pot millorar el temps mínim del festival.

Per altra banda, aquesta nova restricció no sempre vol dir que el rendiment empitjora. Tot i que en molts casos com el festival 7 o el 9 veiem que és cert, en el festival 0 ens dona resultats molt similars. No veiem cap cas de millora en tots els festivals que tenim.

5.1 Festival 1 Extra

Per comprovar la correctesa de la part extra hem creat un nou fitxer anomenat "festival1extra.dzn". Aquest fitxer s'ha creat a partir del festival 1 perquè sabem que dura més de 50 minuts (2h 10 l'òptim), modificant totes les peces perquè facin ús de l'instrument 2. D'aquesta manera ens assegurem que es facin com a mínim dues pauses on no es toqui cap peça (2h10 min = 130 min i 130/50=2.6).

Els resultats finals són els següents:

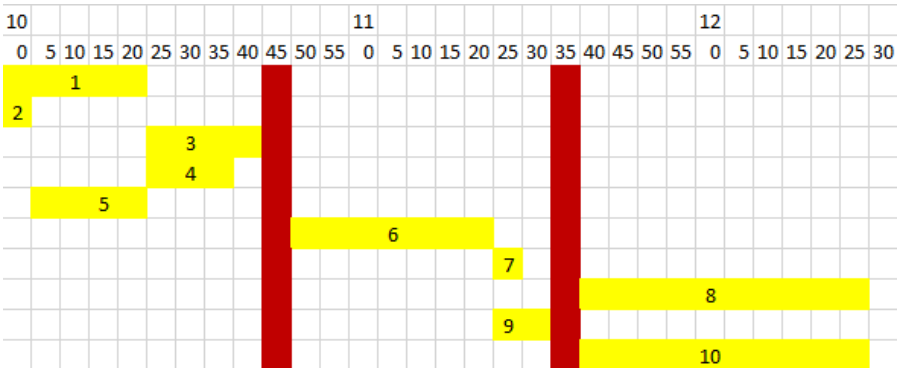

Figura 11: Taula d'execució de les peces amb el fitxer festival1extra.dzn

Podem veure que ens ha estat suficient amb dues parades d'execució per encabir totes les peces.

6 Conclusions i suggeriments

Al llarg de la pràctica hem pogut veure com es desenvolupen diferents constraints, viewpoints i solvers en un problema de planning. Ens hem adonat clarament que el solver OR-Tools CP-SAT 9-8-3296 és molt millor que qualsevol dels altres que tenim per problemes minimament complicats.

Com a suggeriment estaria bé tenir més resultats òptims dels problemes suggerits per poder validar-ne la correctesa i comprovar que els temps que surten són raonables.

Per millorar els resultats de la nostra pràctica, per exemple resolent la instància 8, es podria provar viewpoints diferents, sat solvers millors o constraints que no se'ns hagin acudit.

En conclusió ha estat una pràctica molt enriquidora ja que hem après molt però en alguns moments ha estat feixuc haver d'esperar minuts i a vegades hores a veure si una execució acabava.

7 Bibliografia

Referències

[1] Varis, *The MiniZinc Handbook 2.8.2* (Enllaç a la pàgina)

[2] Varis *Solving Technologies and Solver Backends - The MiniZinc Handbook 2.8.2* (Enllaç a la pàgina)

[3] Mateu Villaret et al., *Privat. Apunts de classe: Programació Declarativa. Aplicacions.*