Pràctica de PDA, Constraints i SAT (2024-2025)

Data màxima de lliurament Moodle: 20 de Gener, lliurament únic.

Possible entrevista de lliurament: a concretar amb cada grup.

Aquesta pràctica constarà de dues parts. La primera part, la de Minizinc, tindrà el major pes (60%) i la segona part, de SAT, contarà un 40%. La pràctica es realitzarà per parelles.

Part Minizinc (Torneig al desert), fins a 6 punts

El problema a resoldre

Una empresa encarregada d'orgnitzar tornejos esportius ens ha encarregat que l'ajudem amb un software genèric que li permeti muntar tornejos per un nombre d'equips, estadis i dies variable. El requisit principal és que tots els equips s'han d'enfrontar entre si un sol cop, que cada equip jugui un cop a cadascun dels estadis donats i que cada equip jugui un partit per dia. A més a més volen poder tenir el control d'especificar "calendaris parcials" de manera que hi hagi partits prefixats en dia i estadi, o bé equips que han de jugar en un dia i estadi concret.
Com que el torneig mou molts seguidors dels equips i les capacitats dels estadis és limitada es voldrà tenir l'opció de maximitzar la venta d'entrades global del tornei.
També, per tal d'estalviar quilòmetres a jugadors i seguidors, es voldrà tenir l'opció de minimitzar el quilometratge total recorregut.
Així doncs sabrem:

- El nombre d'equips participants
- El nombre de dies del torneig
- El nombre d'estadis del torneig
- Les capacitats dels estadis
- El nombre de seguidors de cada equip
- Les distàncies entre els estadis
- Els partits prefixats totalment o parcial (si només s'informa d'un dels dos equips del partit)

Així doncs, la nostra tasca és fer un model en Minizinc que ens munti el torneig satisfent les restriccions indicades amb les següents variants d'optimització:

- Només trobar una solució correcte.
- Minimitzant tots els kilòmetres recorreguts per l'equip (no s'ha de tenir en compte el nombre de seguidors).
- Maximitzant el nombre d'espectadors que podran entrar a tots els partits.
- Trobant la solució òptima que minimitzi els kilòmetres i per la solució òptima segons kilòmetres també maximitzi els espectadors.
- Trobant la solució òptima que maximitzi els espectadors i per aquesta que minimitzi també els kilòmetres.

Finalment, fixeu-vos que quan minimitzeu els kilòmetres recorreguts, pot passar que un equip en faci pocs però que un altre en faci molts i això no és just. Com es podria controlar d'alguna manera que aquesta diferència no fos exagerada però intentant minimitzar igualment la suma de kilòmetres totals?

Dades d'entrada

Es proporciona un conjunt de 9 instàncies del problema plantejat de duresa creixent. La instància `t1fix.dzn` es mostra a continuació a tall d'exemple:

```
nmachesperday = 4;
nmatchesperstadium = 4;
ndays = 7;
nteams = 8;
nstadiums = 7;

fixes=[| {},   {8},  {},    {}, {7,5},  {},  {}|
        {2}, {1,5},  {},    {},    {},  {},  {}|
        {7},    {}, {8},    {},    {},  {},  {}|
         {},    {},  {},    {},   {2}, {5}, {1}|
        {8},    {},  {},    {},    {}, {1},  {}|
         {},    {},  {}, {5,4},    {},  {},  {}|
       { 4},    {},  {},    {}, {1,3},  {},  {}|];

distancies = [|  0, 10, 20, 30, 40, 50, 60|
                10,  0, 15, 25, 35, 45, 55|
                20, 15,  0, 12, 22, 32, 42|
                30, 25, 12,  0, 18, 28, 38|
                40, 35, 22, 18,  0, 16, 26|
```

```
                50, 45, 32, 28, 16,  0, 14|
                60, 55, 42, 38, 26, 14,  0|];


tifosi = [10025,20770,2876,15009,7750,30380,3043,25001];
capacitats = [40000,35000,20000,25000,10000,18000,23000];
```

La solució òptima, pel que fa a entrades venudes màximes (dit d'una altra manera a mínimització de seguidors que es queden sense entrada), seria (no té perquè ser única):

```
      : Es 1  | Es 2  | Es 3  | Es 4  | Es 5  | Es 6  | Es 7  |
-------------------------------------------------------------------
Dia  1:       |  6- 8 |       |  1- 2 |  5- 7 |       |  3- 4 |
Dia  2:  2- 3 |  1- 5 |       |       |  4- 8 |  6- 7 |       |
Dia  3:  1- 7 |  2- 4 |  3- 8 |       |       |       |  5- 6 |
Dia  4:       |       |  4- 7 |       |  2- 6 |  3- 5 |  1- 8 |
Dia  5:  5- 8 |       |       |  3- 6 |       |  1- 4 |  2- 7 |
Dia  6:       |  3- 7 |  1- 6 |  4- 5 |       |  2- 8 |       |
Dia  7:  4- 6 |       |  2- 5 |  7- 8 |  1- 3 |       |       |
-------------------------------------------------------------------

KMs TOTALS: 1452
SEGUIDORS FORA: 233497
----------
==========
Finished in 718msec.
```

Què es demana

Es demana que entregueu un model en Minizinc **en un arxiu anomenat calendari.mzn** que resolgui **correctament** el problema plantejat. L'output del Minizinc ha de ser com a l'exemple mostrat anteriorment, si trobeu maneres més "maques" i amb més informació, millor. Per optar a una bona nota, és necessari que **intenteu sol·lucionar quantes més instàncies millor, i en el menor temps possible**. Per això haureu de definir un bon model i provar diferents configuracions (solvers diferents, considerar mútiples threads en els solvers que ho permetin, us de restriccions globals, implicades, trencament de simetries si n'hi ha, estratègies de cerca, restarts, ...).
Un mínim per aprobar seria que el vostre sistema fos capaç de resoldre les primeres instàncies en un temps "raonable" (<5 min.) i que la sortida fos bona i el document

explicatiu ben treballat. Possiblement en algunes instàncies no pogueu certificar l'òptim en un temps límit (Posem com a temps límit 1h), llavors es valorarà quan bona és la millor solució trobada fins al moment.
D'aquesta pràctica s'espera que experimenteu amb diferents configuracions i expliqueu en un informe quines proves heu fet i quins resultats heu obtingut. De cares a la nota, **l'informe és tan o més important que el model**. Concretament heu d'entregar un fitxer PDF fet en LaTeX on es responguin les següents qüestions:

- Explicació del **model** triat: variables, dominis, i restriccions. Expliqueu clàrament el **viewpoint** i si n'heu necessitat més d'un en el model, expliqueu-los i indiqueu quin **channeling** heu fet servir. Si heu fet més d'un model es valorarà positivament, entregueu els que tingueu i indiqueu quin és el millor amb comparacions experimentals.

  Indiqueu el nombre de variables i de restriccions del model (en funció del nombre d'equips, d'estadis, ...).

- Sobre les vostres **restriccions**, indiqueu quines **globals** heu fet servir, quines **reificacions**, etc. Heu afegit **variables auxiliars**?

- Heu calculat algun **upper bound** o **lower bound** de la funció objectiu? Quins i com?

- Heu trobat **restriccions implicades**, quines? i **simetries**, com les heu trencat? Quin guany de rendiment heu obtingut?

- Per a les estrategies del **cerca** del Minizinc, quines proves heu fet i quins resultats heu obtingut? Quin solver us ha anat millor? Per què? Doneu-ne les taules amb temps de cada instància per les diferents opcions.

Com podeu suposar la nota dependrà de la correctesa i completesa dels diferents punts requerits, així com també de la quantitat d'instàncies que resolgueu òptimament o amb solucions properes a l'òptim. Per això calen taules amb les millors solucions i temps en que s'han aconseguit o indicant "t.o." si heu exaurit el temps límit d'una hora. Cal que indiqueu la màquina que heu fet servir.

Part SAT, fins a 4 punts

Exercici 1, fins a 1 punt

En aquesta part es demana que amplieu l'objecte ScalAT amb els mètodes per a codificar cardinality constraints que es descriuran a continuació per a poder donar un model i resoldre el problema del *buscamines* (descrit a més envall).
Implementeu dins de la llibreria ScalAT les següents codificacions de constraints:

- La funció `addAMOLog(x)`, que codifiqui una restricció de tipus At-Most-One ($x[0] + x[1] + x[2] + \ldots + x[size - 1] \leq 1$) amb l'encoding logarítmic.
- La funció `addAMK(x,K)`, que codifiqui una restricció de tipus At-Most-K ($x[0] + x[1] + x[2] + \ldots + x[size - 1] \leq K$) fent servir sorting networks. La implementació de la sorting network ja ve donada.
- La funció `addALK(x,K)`, que codifiqui una restricció de tipus At-Least-K ($x[0] + x[1] + x[2] + \ldots + x[size - 1] \geq K$) fent servir sorting networks.
- La funció `addEK(x,K)`, que codifiqui una restricció de tipus Exactly-K ($x[0] + x[1] + x[2] + \ldots + x[size - 1] = K$) fent servir sorting networks.

Exercici 2, fins a 3 punts

Codifiqueu amb ScalAT un solucionador del Buscamines. Es tracta d'una variant estàtica:

- El joc consisteix en un tauler (graella) de $N \times M$ caselles. Algunes de les caselles contenen mines i les altres no.
- Tenim com a entrada un tauler parcialment solucionat, on es dona un conjunt de caselles que no contenen mina. Cada casella pre-inicialitzada pot contenir:
  - Un número del 0 al 8, que indica el nombre de caselles adjacents, incloent diagonals, que contenen una mina.
  - Una `X`, que indica que a la casella no hi ha una mina, però no dona pistes sobre el nombre de mines adjacents.
- Una instància sempre contindrà prou informació perquè es pugui deduïr el contingut de totes les caselles que falta omplir. Per tant, hi haurà una única solució.
- En algunes instancies s'indicarà el nombre de mines total que conté el tauler (*nmines*), ja que en aquestes instàncies serà una informació rellevant (altrament no les podríem resoldre).

Es demana una aplicació en Scala, que faci servir una codificació a SAT amb la llibreria ScalAT per resoldre el problema del Buscamines. Un fitxer d'instància tindrà el següent format:

```
N  M  nmines
casella_1_1 casella_1_2 ... casella_1_M
...
casella_N_1 casella_N_2 ... casella_N_M
```

On N, M i *nmines* són nombres enters; *nmines* val -1 si la instància no especifica quin és el nombre de mines esperat; `casella_i_i` val `0-8` si està pre-inicialitzada i ens dona una pista, val `X` si està pre-inicialitzada però no ens dona cap pista, i val `-` si cal resoldre-la.
Exemple d'instància:
(correspon a la instancia 37 de lloc web `www.janko.at/Raetsel/Minesweeper/index.htm`)

```
5 5 -1
- - - - 0
1 4 - 4 -
- - - - 3
1 - 4 - -
- 0 - - 2
```

Exemple de solució (`o` significa mina, `X` significa no mina):

```
X X o X X
X X o X X
X o o o X
X X X o o
X X X X X
```

Es proporciona un conjunt d'instàncies perquè pogueu validar el vostre programa. Totes provenen del lloc web `www.janko.at/Raetsel/Minesweeper/index.htm`, on podeu trobar més istàncies, i podeu entrar les solucions manualment per comprovar si són correctes.

Què es demana

Caldrà que lliureu un enllaç al vostre Github privat, i m'hi dongueu accés, on hi haurà d'haver l'ScalAT ampliat i un objecte `BuscaMines` (podeu fer servir el `Queens` com a llavor) que codifiqui i resolgui el *busca mines* usant ScalAT.
Caldrà també lliurar (el podeu posar al Github) un informe PDF (fet amb LaTeX):

1. Doneu i expliqueu el vostre model: viewpoint, restriccions bàsiques, restriccions implicades que hagueu trobat, trencament de simetries (si n'hi ha) etc.

2. Òbviament fareu servir cardinality constraints. Cal que proveu les dues configuracions següents del vostre model:

   Conf1 feu servir només codificacions quadràtiques pels AMOs i EOs que us apareguin al model.

   Conf2 Feu servir només codificacions logarítmiques pels AMOs i EOs que us apareguin al model.

   - Quines diferències observeu entre les dues configuracions pel que fa a la mida?
   - Quina de les dues configuracions és més ràpida? Per què creieu que passa?

3. Mireu de resoldre instàncies tan grans com pogueu, podeu fer servir un *timeout* de 10 minuts). Doneu una taula de temps per instància. A la taula s'hi ha de poder veure què us ha suposat de millora de temps les millores que hagueu proposat al model (és a dir, feu columnes amb els temps per la codificació bàsica, per la codificació amb restriccions implicades, etc)

Vols afegir algun encoding diferent de *cardinality constraint*? Consulta'n a [1].

Referències

[1] Miquel Bofill, Jordi Coll, Peter Nightingale, Josep Suy, Felix Ulrich-Oltean, and Mateu Villaret. SAT encodings for pseudo-boolean constraints together with at-most-one constraints. *Artif. Intell.*, 302:103604, 2022.