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

# Informe de PdA: Constraints i SAT: BuscaMines

Cels Esteva Planas
DNI: 41679856S
ID d'estudiant: u1978909
Correu electrònic: u1978909@campus.udg.edu
GEINF
Universitat de Girona

# Índex

# 1 ViewPoint

## 1.1 Explicació del ViewPoint

El ViewPoint del buscamines és una matriu de mida del buscamines original. Allà on hi ha mines queda true, on no, queda false.

```
    val e = new ScalAT("BuscaMines")

    val buscamines = buscaMinesToArray(test)
    val n = buscamines._1
    val m = buscamines._2
    val mines = buscamines._3
    val tauler: Array[Array[Int]] = e.newVar2DArray(n, m)
    val mineSweeper = buscamines._4
    for(i <- 0 until n){
      for(j <- 0 until m){
        addConstraint(i,j);
      }
    }

    def addConstraint(row: Int, col: Int): Unit = {
      val centerElement = mineSweeper(row)(col)
      centerElement match {
        case "-" => // do nothing (cas en el que no es sap res)
        case "x" | "X" => e.addClause(-tauler(row)(col) :: List()) //marcar
            casella mineSweeper(row)(col) == false
        case n => addConstraint3x3(centerElement,row,col) //exactlyN //get
            the 3x3 list and then add an exactlyK (K == n)
      }
    }

    def addConstraint3x3(centerElement: String, row: Int, col: Int) = {
      e.addClause(-tauler(row)(col) :: List()) //element conte numero -> per
          tant fals.
      val elements = getSurroundingElements(row,col).flatten //get
          surroundingElements retorna els elements de tauler(row)(col)
      val el = centerElement.toInt;

      if(el == 0){
        for(clause <- elements) e.addClause(-clause :: List())
      }
      else if(el == 1){
        e.addEOQuad(elements)
        //e.addEOLog(elements)
      }
      else if(el==elements.length) {
        for (clause <- elements) e.addClause(clause :: List())
      }
      else if(el > elements.length){
        e.addClause(List()) //trivialment fals.
      }
      else if(el < 0){
        e.addClause(List()) //trivialment fals.
```

```
44        }
45        else{
46          e.addEK(elements,el)
47          //e.addAMK(elements,el)
48          //if(el>0)
49          //e.addALK(elements,el)
50        }
51      }
52
53      def getSurroundingElements(row: Int, col: Int) = {
54        val edgeElements = scala.collection.mutable.ListBuffer[List[Int]]()
55        for (di <- -1 to 1) {
56          val edgeElementsRow = scala.collection.mutable.ListBuffer[Int]()
57          for (dj <- -1 to 1) {
58            val rowD = row + di
59            val colD = col + dj
60            if(rowD>=0 && rowD<n && colD >= 0 && colD < m && !(rowD == row &&
                  colD == col))
61              edgeElementsRow += tauler(rowD)(colD)
62          }
63          edgeElements += edgeElementsRow.toList
64        }
65        edgeElements.toList
66      }
```

### 1.1.1 Restriccions

No hi ha restriccións implicades. Les restriccions són molt bàsiques, es va per a tot el buscamines i s'agafen els elements del centre.

- Per a tots els que l'element del centre és '-', no es fa res.
- Per a tots els que l'element del centre és 'X', és marca la posició de l'element del centre com a no mina.
- Per a tots els que la casella tingui un numero (i per tant hi hagi un cert nombre de mines al voltant), s'agafa la posició del centre i es posa com a no mina, després es crida a la funció getSurroundingElements, que retorna els elements del voltant de l'element del centre. Segons si l'element del centre és 0, 1 n o (n i elementsDelVoltant.length==n) es fa una de les següent coses amb les posicións del voltant:
  - 0: es posen tots a negatiu: no hi ha cap mina al voltant.
  - 1: es posa un exactly one.
  - n si elementsDelVoltat.length == n: es posen tots a true, hi ha tantes mines com elements al voltant.
  - n: es posa un exactly K on K és n.
- Si es menciona quantes mines hi ha en total, es fa el següent segons les mines:
  - 0: tot negat
  - 1: es posa un exactly one

- n si tauler.length == n: tot positiu
- n: exaclty K

## 1.2 Execució del ViewPoint

No entenc a que us referiu amb feu instancies tant grans com podeu, si les de la pàgina web no tarden res fins i tot amb les grans. Hem de fer un generador de buscamines? Com garantiriem que només tenen una sol·lució?

| | AMOQuad temps | AMOQuad size | AMOLog temps | AMOLog size |
|---|---|---|---|---|
| 350 | 165 - 78 | 3535 - 10319 | 162 - 75 | 3607 10277 |
| 349 | 163 - 74 | 3028 - 8919 | 160 - 74 | 3111 8877 |
| 300 | 157 - 74 | 2805 - 8447 | 163 - 78 | 2907 8365 |
| 299 | 180 - 96 | 3059 - 8902 | 162 - 75 | 3126 8896 |
| 200 | 165 - 76 | 2992 - 8615 | 161 - 74 | 3055 8630 |
| 199 | 159 - 75 | 2357 - 6975 | 156 - 71 | 2438 6948 |

Taula 1: Taula d'execucions, les mides estan en format (variables, clàusules) i el temps en ms. El primer temps temps engloba crear el fitxer dimacs i trobar la solució, el segon només trobar la solució.

Hi ha molt poca variació de temps perquè només hi ha unes poques clàusules que fan servir el constraint EOLog/Quad. El logarítmic és més ràpid però útilitza més variables (per a codificar afageix log variables). En canvi útilitza menys clausules i per tant fa les unit propagation més ràpid.