BASIC COMPUTING OF SOLUTION DENSITY

Reporter: Jinling Xing

April 17, 2018

Polytechnique Montréal

OUTLINES

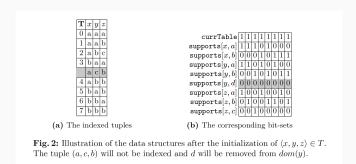
- 1 Introduction
- 2 Compact Table
 - 2.1 currTable
 - 2.2 Algorithm of simplified version of CT
- 3 CT with solution density
 - 3.1 The counting of nbOfOnes
 - 3.2 CT with solution density
 - 3.3 The maxOfCount
- 4. Experiment
- 5. Conclusion

1 INTRODUCTION

INTRODUCTION

Basic computing of **Solution density**, the number of supports & currTable for each pair of (x, a) in the compactTable constraint of Oscar.

2 COMPACT TABLE



For example, the solution density of **supports[x,a] & currTable** is the **number of ones** after computing, at this case the result is 4.

2.2 ALGORITHM OF SIMPLIFIED VERSION OF CT

```
Algorithm 1. Class ConstraintCT
 1 Method updateTable()
       foreach variable x \in scp do
           mask \leftarrow 0
 3
 4
           if |\Delta_x| < |dom(x)| then
                                                       // Incremental Update
               foreach value a \in \Delta_x do
                  mask \leftarrow mask \mid supports[x, a] // Use supports^*[x, a] in CT^*
 6
                                                      // ~: bitwise negation
               mask ← ~mask
           else
                                                       // Reset-based Update
 8
               foreach value a \in dom(x) do
10
                  mask \leftarrow mask \mid supports[x, a]
                                                             // | : bitwise or
           currTable \leftarrow currTable \& mask
                                                           // & : bitwise and
11
12 Method filterDomains()
       foreach variable x \in scp do
           foreach value a \in dom(x) do
14
               if currTable & supports[x, a] = 0 then
                  dom(x) \leftarrow dom(x) \setminus \{a\}
17 Method enforceGAC()
       updateTable()
       filterDomains()
```

Modify the Method filterDomains() and use the Method nbOfOnes()

3 CT WITH SOLUTION DENSITY

3.2 THE COUNTING OF NBOFONES

Count the number of elements in intersection between currTable and bs(bitset to intersect) and return number of elements in intersection.

```
/**
  * count the number of ones of supports and currTable
  */
def nb0fOnes(bs: BitSet): Int = {
  var count = 0
  var i : Int = nNonZero /*bit length of nWords*/
  while(i > 0) {
    i -= 1
    val offset = nonZeroIdx(i)
    count += java.lang.Long.bitCount(words(offset) & bs.words(offset))
  }
  count
}
```

sd is a 2-dimension array: Array[Array[]], I used ArrayBuffer to store the counts in sd.

```
//sd is the solution density
val sd = Array.tabulate(x.length)(i => Array.tabulate(spans(i))(v => new ArrayBuffer[Int]()))
/* Create the final support bitSets and remove any value that is not supported */
for {
  varIndex <- variableValueSupports.indices</pre>
  valueIndex <- variableValueSupports(varIndex).indices</pre>
  if (varValueSupports(varIndex)(valueIndex).nonEmpty) {
    variableValueSupports(varIndex)(valueIndex) = new
        validTuples.BitSet(varValueSupports(varIndex)(valueIndex))
    val count = validTuples.nb0f0nes(variableValueSupports(varIndex)(valueIndex))
    sd(varIndex)(valueIndex) += count //sd is the solution density
    println(" var index " + varIndex + " value " + valueIndex + " count " + count )
  }else {
    /* This variable-value does not have any support, it can be removed */
    x(varIndex).removeValue(valueIndex)
```

3.3 THE MAXOFCOUNT

Find the max count value of each variable. We can get a 3*6 2-dimension array. Variables: val x = Array.fill(5)(CPIntVar(0 to 5)). When the size of the sd>=1, we will escape the empty ArrayBuffer.

```
// Print two dimensional array and find the max
for {
  varIndex <- variableValueSupports.indices</pre>
  var max0fCount = 0
  for {valueIndex <- variableValueSupports(varIndex).indices} {</pre>
    if(sd(varIndex)(valueIndex).size >= 1) {
      val c = sd(varIndex)(valueIndex).head
      //print(" " + c )
      maxOfCount = java.lang.Math.max(maxOfCount, c)
      print(" " + sd(varIndex)(valueIndex))
  print(" Max " + max0fCount + "\n")
```

4. EXPERIMENT

4. EXPERIMENT

The origin table is on the left, which is a Array[Array[]] format table, including 3 variables and the domain of each variable is between 0-5. The output of count is on the right, for each X[i] and the value in its domain.

```
val tableGround: Array[Array[Int]] = Array(
    Array(1, 2, 3),
    Array(2, 3, 4),
    Array(1, 1, 1),
    Array(0, 1, 5),
    Array(1, 2, 5),
    Array(5, 5, 5)
}
```

```
Figure: Origin table
```

```
var index 0 value 0 count 1 var index 0 value 1 count 3 var index 0 value 2 count 1 var index 0 value 5 count 1 var index 1 value 1 count 2 var index 1 value 2 count 2 var index 1 value 3 count 1 var index 1 value 5 count 1 var index 2 value 1 count 1 var index 2 value 4 count 1 var index 2 value 4 count 1 var index 2 value 5 count 3
```

Figure: The count of each (x,a)

4. EXPERIMENT

Table: The count Array[Array[]] format matrix

Variable	Count				MaxCount
X[0]	1	3	1	1	3
X[1]	2	2	1	1	2
X[2]	1	1	1	3	3

ArrayBuffer(1) ArrayBuffer(3) ArrayBuffer(1) ArrayBuffer(1) Max 3 ArrayBuffer(2) ArrayBuffer(2) ArrayBuffer(1) ArrayBuffer(1) Max 2 ArrayBuffer(1) ArrayBuffer(1) ArrayBuffer(3) Max 3

5. CONCLUSION

5. CONCLUSION

- In this project, I want to compute the basic **solution density**, the number of ones in of each pair(x,a) after calculating currTable & Supports(x,a).
- The modification of **Method** *filterDomains()* and use the **Method** *nbOfOnes()* to count the number of ones in currTable & Supports(x,a).
- · Construct the 2-dimension array "sd" to store the "counts" and find the "max" of the count.
- Future work: Based on the maximum value and its index to modify the search.

Questions?