

On Matrices, Automata, and Double Counting

Nicolas Beldiceanu, Mats Carlsson, Pierre Flener, and Justin Pearson

June 17, 2010

Double Counting

How many 1s?

$$\begin{pmatrix} 1 & & 1 & & 1 \\ 1 & & & 1 & \\ & 1 & & 1 & \\ & & 1 & & 1 \\ & & & & & 1 \\ 1 & & & & 1 & 1 \end{pmatrix}$$

Double Counting

Count Columns

$$\begin{pmatrix} 1 & & & 1 & 1 & \\ 1 & & 1 & & & \\ & 1 & & 1 & & \\ & 1 & & 1 & & \\ & & & & 1 & \\ 1 & & & 1 & 1 & \end{pmatrix}$$

(3)

Gives 3 +

Double Counting

Count Columns

$$\begin{pmatrix} 1 & & & 1 & & & 1 \\ 1 & & & & 1 & & \\ & 1 & & & & 1 & \\ & & 1 & & & & 1 \\ & & & 1 & & & \\ 1 & & & & & 1 & 1 \end{pmatrix}$$
$$(3 \quad 2 \quad 1 \quad \quad \quad)$$

Gives $3 + 2 + 1 +$

Double Counting

Count Columns

$$\begin{pmatrix} 1 & & & 1 & & 1 \\ 1 & & & & 1 & \\ & 1 & & & & 1 \\ & & 1 & & & \\ & & & 1 & & \\ & & & & & 1 \\ 1 & & & & 1 & 1 \end{pmatrix}$$
$$(3 \quad 2 \quad 1 \quad 3 \quad \quad)$$

Gives $3 + 2 + 1 + 3 +$

Double Counting

Count Columns

$$\begin{pmatrix} 1 & & & 1 & & & 1 \\ 1 & & & & 1 & & \\ & 1 & & & 1 & & \\ & & 1 & & 1 & & \\ & & & & & & 1 \\ 1 & & & & 1 & & 1 \end{pmatrix}$$
$$(3 \ 2 \ 1 \ 3 \ 1 \ 3)$$

Gives $3 + 2 + 1 + 3 + 1 + 3 = 13$

Double Counting

- Double counting is an old idea in combinatorics.
- Find two ways of counting things.
- Used all over the place in combinatorics.
 - Given a graph the number of nodes with odd degree is even.

General Idea

- Lots of problems involve constraints on rows and columns of matrices.
- Although there is some work on global constraints on matrices, there is still a lot of work that needs to be done.
- We apply double counting ideas to matrix problems with constraints specified by automata on the rows and global cardinality constraints on the columns.

Plan of the talk

- Some more advanced double counting.
- Deriving constraints via counter automata.
- Cardinality Automata.
- Brief summary of some experiments.

How many times does a word appear?

- Suppose that for each row we have an upper bound and lower bound on the number of times a particular word can be.
- In this example we'll take BOB.

$$\begin{pmatrix} B & O & B & O \\ B & O & B & \\ & & B & O & B \end{pmatrix}$$

How many times does a word appear

<i>B</i>	<i>O</i>	<i>B</i>	<i>O</i>	.	.
<i>B</i>	<i>O</i>	<i>B</i>	.	.	.
<i>B</i>
.	.	<i>B</i>	<i>O</i>	<i>B</i>	.
.
.
<hr/>					
$\#(B)$	$\#(O)$	$\#(B)$			

How many times does a word appear

<i>B</i>	<i>O</i>	<i>B</i>	<i>O</i>	.	.
<i>B</i>	<i>O</i>	<i>B</i>	.	.	.
<i>B</i>
.	.	<i>B</i>	<i>O</i>	<i>B</i>	.
.
.
<hr/>					
$\#(B)$	$\#(O)$	$\#(B)$			
	$\#(B)$	$\#(O)$	$\#(B)$		

How many times does a word appear

<i>B</i>	<i>O</i>	<i>B</i>	<i>O</i>	.	.
<i>B</i>	<i>O</i>	<i>B</i>	.	.	.
<i>B</i>
.	.	<i>B</i>	<i>O</i>	<i>B</i>	.
.
.
<hr/>					
$\#(B)$	$\#(O)$	$\#(B)$			
	$\#(B)$	$\#(O)$	$\#(B)$		
		$\#(B)$	$\#(O)$	$\#(B)$	

How many times does a word appear

<i>B</i>	<i>O</i>	<i>B</i>	<i>O</i>	.	.
<i>B</i>	<i>O</i>	<i>B</i>	.	.	.
<i>B</i>
.	.	<i>B</i>	<i>O</i>	<i>B</i>	.
.
.
$\#(B)$	$\#(O)$	$\#(B)$			
	$\#(B)$	$\#(O)$	$\#(B)$		
		$\#(B)$	$\#(O)$	$\#(B)$	
			$\#(B)$	$\#(O)$	$\#(B)$

How many times does a word appear?

Take the minimum of each count and this will tell you how many times the word can appear in that position. Sum all these up and you will get a lower bound on the number of times the word can occur.

$$\begin{array}{cccccc} B & O & B & O & . & . \\ B & O & B & . & . & . \\ B & . & . & . & . & . \\ . & . & B & O & B & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ \hline 3 & 2 & 3 & & & \end{array}$$

How many times does a word appear?

Take the minimum of each count and this will tell you how many times the word can appear in that position. Sum all these up and you will get a lower bound on the number of times the word can occur.

<i>B</i>	<i>O</i>	<i>B</i>	<i>O</i>	.	.
<i>B</i>	<i>O</i>	<i>B</i>	.	.	.
<i>B</i>
.	.	<i>B</i>	<i>O</i>	<i>B</i>	.
.
.
<hr/>					
3	2	3			
	0	0	0		

How many times does a word appear?

Take the minimum of each count and this will tell you how many times the word can appear in that position. Sum all these up and you will get a lower bound on the number of times the word can occur.

$$\begin{array}{cccccccc} B & O & B & O & . & . & & \\ B & O & B & . & . & . & & \\ B & . & . & . & . & . & & \\ . & . & B & O & B & . & & \\ . & . & . & . & . & . & & \\ . & . & . & . & . & . & & \\ \hline 3 & 2 & 3 & & & & & \\ & 0 & 0 & 0 & & & & \\ & & 3 & 2 & 1 & & & \end{array}$$

How many times does a word appear?

Take the minimum of each count and this will tell you how many times the word can appear in that position. Sum all these up and you will get a lower bound on the number of times the word can occur.

<i>B</i>	<i>O</i>	<i>B</i>	<i>O</i>	.	.
<i>B</i>	<i>O</i>	<i>B</i>	.	.	.
<i>B</i>
.	.	<i>B</i>	<i>O</i>	<i>B</i>	.
.
.
<hr/>					
3	2	3			
	0	0	0		
		3	2	1	
			0	0	0

Double Counting Properties

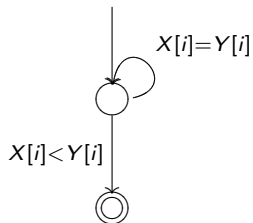
- In the paper there are lots of double counting properties derived including:
 - Bounds on stretch lengths
 - Bounds on the number of stretches
- These properties are partly motivated by problems in shift scheduling.

Deriving properties.

- Obviously these properties can be used directly by the user.
- Here we derive them from automata that describe the row constraints.
- The automata constraint of Beldiceanu et al. allows you to describe a constraint by describing the constraint checker as a finite automata with counters.

Automata

- Assignments correspond to paths to final states.



Deriving Properties

- Generally we take conjunctions (products) of automata on the rows.
- We then annotate the automata with extra counter variables that correspond to the double counting properties, number of occurrences of a word, , we identified earlier.
- We then use CP search to discover properties of the automata.
- In our experiments we looked for patterns up to word length 3.
- It turns out that there are lots of patterns out there and they have a big impact on the search (more later).

Row Automata

- We have a bunch of automata, one for each row.
- We can run them in parallel.
- We build an automaton that consumes whole row.
 - Given R automata the row automaton has states

$$(s_1, \dots, s_r)$$

where each element is a state of a row automaton.

$$(s_1, \dots, s_r) \rightarrow (t_1, \dots, t_r)$$

if for each i there exists a transition $s_i \rightarrow t_i$.

- Process algebra people would call this synchronous parallel composition.

Cardinality Automata

- Parallel composition in general too big.
- We make an abstraction when the automata are identical.
- We count the number of automata that are in a particular state (still too big).
- Define transitions to work on state counts.
- The can all be coded as a bunch of linear constraints (generalises work of Côté, Gendron, and Rousseau).
- More importantly we can then add cardinality constraints on the columns to the linear constraints which increases propagation between rows and columns.

Results

- We tested these methods on problems from the NSPLib repository.
- NSLib contains lots of instances of nurse scheduling problems.
- Label a matrix each nurse is labelled with a shift.
 - Rows represents nurses.
 - Columns represent days
- Hard constraints, include coverage constraints (cardinality constraints) on the columns.
- Also constraints on the rows on the number of types of shifts, plus constraints on shift stretches.

- Three different experiments (actual numbers in the paper):
 - String properties extracted from product automata on the rows.
 - Cardinality Automata
 - Combination of the two.
- String properties perform well, cardinality automata rarely beats the string properties.
- Combination of the two perform well.
- We do best on unsatisfiable instances.

Conclusion

- Automata are not just for deriving filtering algorithms, but here we use them to define necessary conditions.
- Adding more string properties is not hard and can be done in incremental way.
- Automata provide a declarative specification which encourages reuse and not just algorithm engineering.

The End

Thank You