Monster ("Mega") Sudoku

- NxN grid, N = pq a composite number > 9
- N symbols; generate them "odometer" style
  - 1 … 9 A B … Z 11 12 … 19 1A … 1Z 21 … 9Z A1 … A9 AA … ZZ 111 112 … 9ZZ A11 … ZZZ 1111 … ZZZZ 11111 … ZZZZZ …
- N blocks, each with p rows and q columns
  - The N blocks fit regularly into the NxN grid
  - p blocks fit across the NxN grid rows (p blocks x q columns = N)
  - q blocks fit down the NxN grid columns (q blocks x p rows = N)
- Some elements of the NxN grid already have symbols
- Fill in the rest of the NxN grid with symbols under constraints
  - No symbol appears twice in any row
  - No symbol appears twice in any column
  - No symbol appears twice in any block
  - Often called the "AllDiff" constraint

Examples

			1			4		7			8
	4				1				B		2
		5		A			3		4		
	2	4		7	B				6	C	
A							8		2		5
			C		6			4		B	
	9		4			6		B			
7		8		2							1
	1	6				5	B		3	2	
		1		3			2		A		
4		7				A				6	
5			8		4			3			

easy

8	F		C						A						6
			A				F				B	7	4	D	
B		4				D	6		7			0		5	
1							0	3		9	2				
					1	F	D		3	0			E	7	4
	1		6				C		B			A		3	
	C		D			6	3		5			9	2		
9		3	4	E		2				7	D				
				5	7				8		C	3	0		A
		E	2			4		7	1			F		6	
	5		3			8		9				E		C	
7	0	6			C	9		D	E	3					
				D	E		4	0							2
	7		8			C		4	2				B		5
	2	9	E	B				5				4			
6						7						1		8	3

hard

You Will Write Code:

- Code that inputs a Monster Sudoku puzzle
  - Input parameters N, p, q to define the grid and blocks
  - Which symbols already are on which grid elements

- Code that generates a random Monster Sudoku puzzle
  - Input parameter M the number of symbols initially on grid
  - Symbols are chosen and placed randomly respecting constraints

- Code that solves a Monster Sudoku puzzle
  - Node consistency, arc consistency, path consistency (6.2)
  - Backtracking search (6.3)
  - Variable and value ordering: minimum-remaining values, degree heuristic, least-constraining-value (6.3.1)
  - Forward checking (6.3.2)

- Extra Bonus Credit:
  - Local search for CSPs: min-conflict heuristic (6.4)

You Will Analyze:

- For each value of N there is a “hardest” value of M
  - For each of an increasing series of N, find the corresponding M
  - How does M change as N increases?

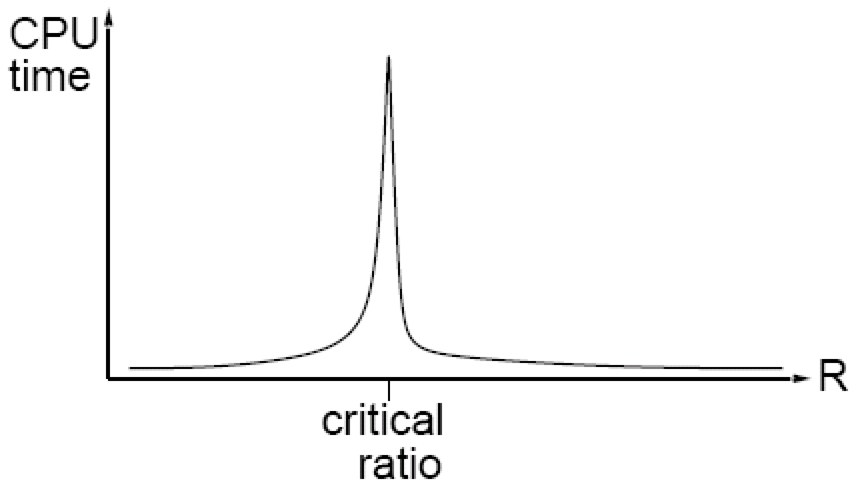
- What is the biggest N for which you reliably solve the “hardest” M?
  - How does solution time grow with increasing N for “hardest” M?
  - What is the relative contribution of the various CSP heuristics?

Performance of min-conflicts

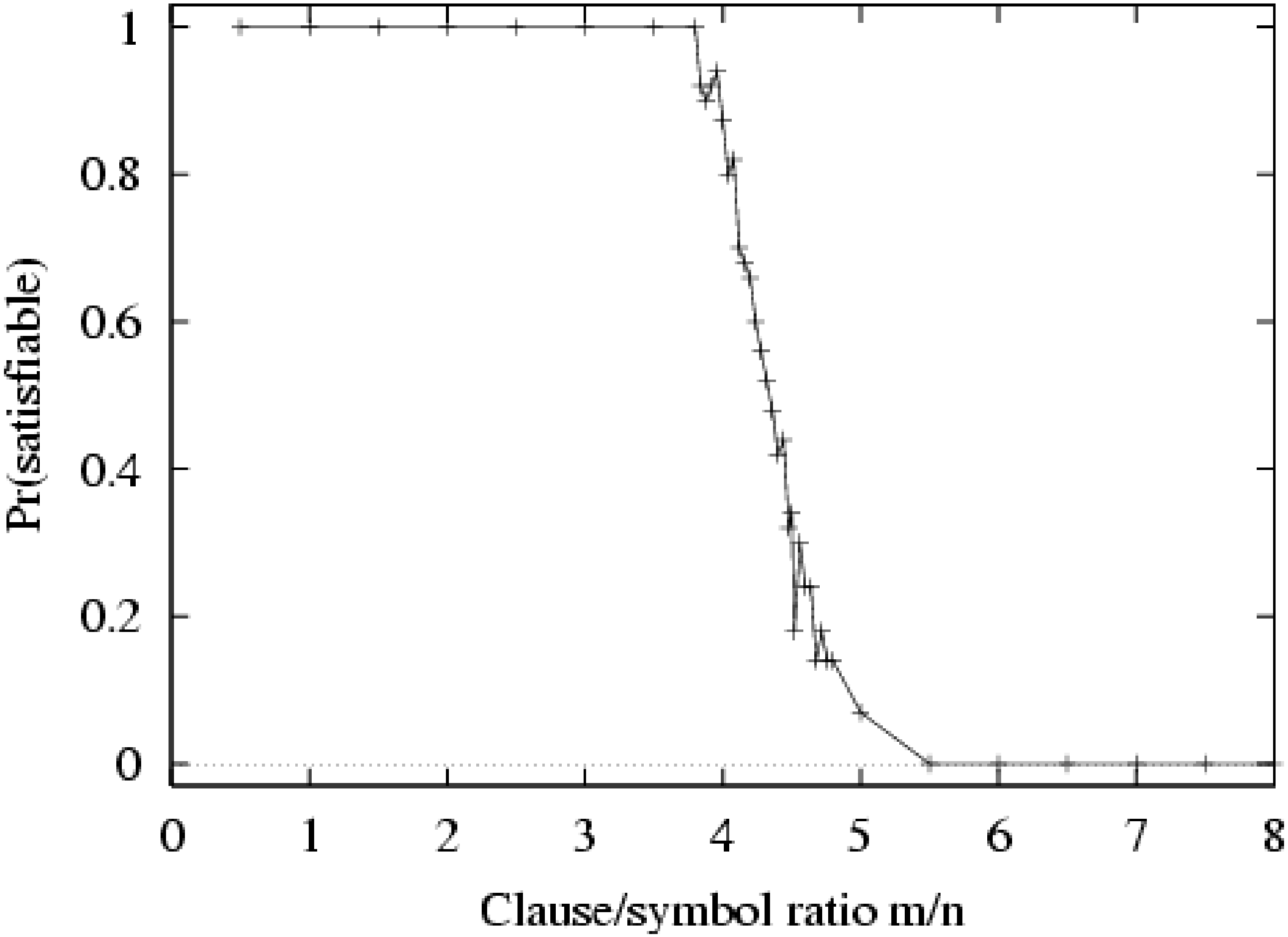
Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)

The same appears to be true for any randomly-generated CSP
except in a narrow range of the ratio

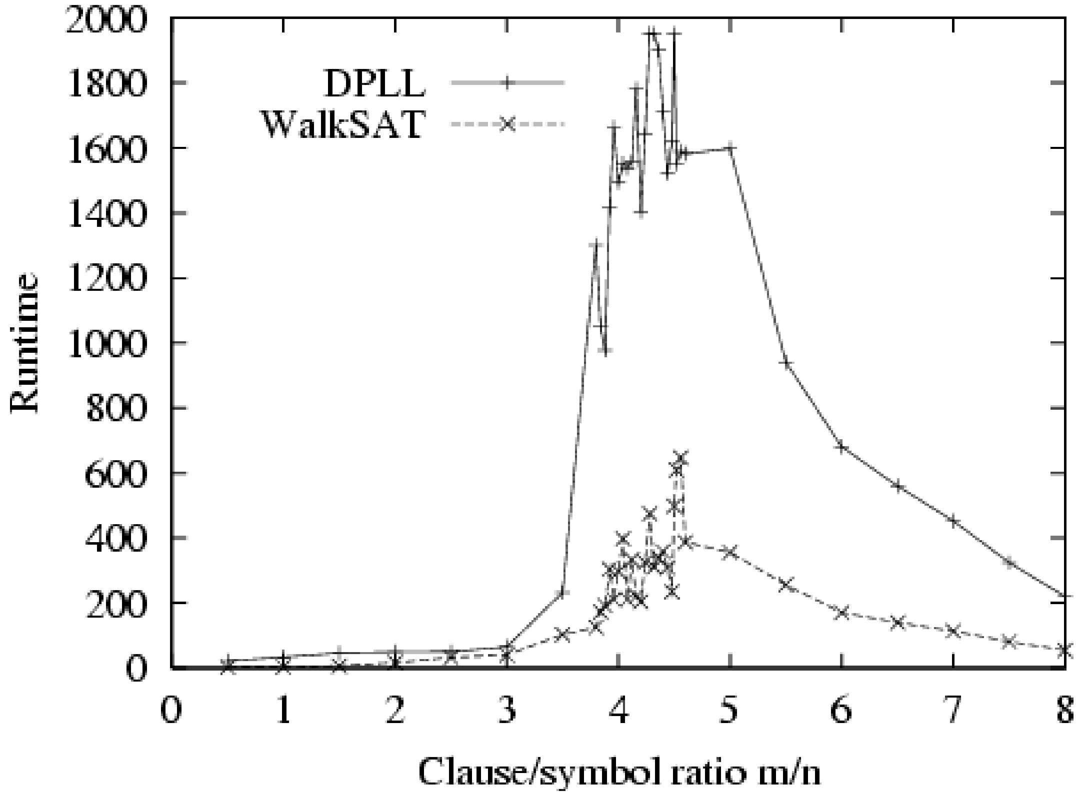
$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



Hard satisfiability problems



Hard satisfiability problems



- Median runtime for 100 satisfiable random 3-CNF sentences, $n = 50$