Kendall Weistroffer
AI Assignment #1
Due: 10.9.2016

1. *Define what a CSP is:*

Constraint Satisfaction Problems, or CSPs, are problems in which we must find either states or objects that satisfy given constraints. By using CSPs, we are able to solve a large amount of problems more efficiently by using a more general set of heuristics rather than problem-specific heuristics to find a solution to a given problem. A CSP consists of three sections:

1. A set of variables
2. A set of domains, where each variable gets its own domain
3. A set of constraints which determine which combinations of variables are allowed

For each variable there consists a domain that specifies what possible values that variable can have. However, constraints limit the possibilities for what values a variable can be in order to satisfy a problem based on the variable's relationship with other variables. A constraint will usually be defined in two parts:

1. Scope: The variables that are being constrained by this particular constraint
2. Relation: Defines which values the in-scope variables are able to take on

An example of a constraint would be:

alldiff(x0,x1,x2,x3)

Which constrains the variables x0, x1, x2, and x3 to be all different values, meaning that:

x0 != x1 , x0 != x2, x0 !=x3, x1 != x0, x1 != x2, x1 !=x3, x2 != x0, x2 != x1, x2 != x3, x3 != x0, x3 != x1, x3 != x2

If the values that are assigned to variables do not violate any of the defined constraints, then that assignment is determined to be a consistent, legal, assignment. Assignments can either be partial (in which values are assigned to some of the variables) or complete (in which values are assigned to all of the variables). In order to solve a CSP, we must have a consistent, complete assignment in which every variable is assigned a value and those values do not violate any constraints.

2. *Discuss what a Sudoku is:*

Sudoku is a single-player puzzle game that consists of a grid of individual boxes and some prefilled numbers. The goal of the game is to fill in the rows, columns, and blocks (equal subsections of the grid) so that there is only one instance of each number in each row, column, and block. In most situations the puzzle will take place on a 9x9 grid, blocks will be 3x3 subsections of the 9x9 grid, and the numbers in play will range from 1-9. However, for this assignment we are dealing with a "Hexiduko" grid, in which the game is played on a 16x16 grid (made of 4x4 blocks), there are 16 numbers in play (usually either 1-16, 0-F, or 0-15), but the overall rules of the game remain the same (one instance of a number to each row/column/block).

So why are Sudoku's instances of CSPs and why are we dealing with a Sudoku-eque problem to learn about CSPs? There are constraints placed upon what values can be assigned to each variable within each row, column, and block of the playing grid. One approach could be where we use a brute-force approach to solving this problem, where we try every outcome of placing numbers on the grid until we find a solution that meets the criteria for solved Sudoku. Although the brute-force approach produces a solution in the end, it is highly inefficient and can take hundreds of millions of steps to solve a 9x9 Sudoku problem. By using CSP-based approach, we can eliminate many possible outcomes that violate the Sudoku rules, thus speeding up the process and producing a much more efficient solution.

For a basic 9x9 Sudoku problem, our CSP would look like this:

Variables: The 81 squares on the 9x9 grid

Domains/Values: {1,2,3,4,5,6,7,8,9}

Constraints: There can only be one instance of a number in each row, column, and block

For a 16x16 "Hexidoku" problem, our CSP would look like this:

Variables: The 256 squares on the 16x16 grid

Domains/Values: {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16} or {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15} or {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}, depending on how you choose to set up the problem

Constraints: There can only be one instance of a number/letter in each row, column, and block

Here, the process of producing a solution is sped up by narrowing down the domain for each variable based on the given constraints. For example, when choosing what number to place in a certain box (or what value to assign to a particular variable), we eliminate the pre-existing values that are found in that same box's corresponding block, row, and column from the variable's domain using arc or path consistency. This significantly narrows down the domain for each variable, resulting in significantly less values to check for each variable than if we were applying a brute-force approach. Although when dealing with Sudoku, or the Sudoku-eque problem of Hexidoku we are dealing with a specific amount of variables (either 81 or 256) and specific domains (numbers 1-9, 0-15, 1-16, or numbers/letters 0-F), the constraint-solving mechanisms like the AllDiff constraint, arc consistency, and path consistency that are used in this Sudoku/Hexidoku-instance apply generally to all CSPs.

3. *Results:*

After selecting both an easy and difficult Hexidoku problem from the internet, I encoded each problem

into CSP format and used the Minion CSP solver software to find solutions to each problem. For the selected easy Hexidoku problem, the grid started off with 106 pre-encoded values ranging from 0-15, and resulted in a solve time of 0.164210 seconds, and a total wall time of 0.169328 seconds. For the selected difficult Hexiduko problem, the grid started off with 102 pre-encoded values also ranging from 0-15, and resulted in a solve time of 5.870435 seconds and a total wall time of 5.907730 seconds. In comparison, the easy Hexidoku problem was able to be solved 5.706225 seconds faster than the hard Hexidoku problem, and had a 5.738402 second quicker wall time overall. Then, I tested each problem to see if the found solution was the only solution to that particular Hexidoku problem using Minion's -findallsols switch flag. According to Minion's website, the -findallsols switch flag "find[s] all solutions and count[s] them". When applying the -findallsols switch flag, the easy Hexidoku problem the solve-time increased from 0.164210 seconds to 0.236887 seconds (a difference of 0.072677 seconds) and a total wall time increased from 0.169328 seconds to 0.341607 seconds (a difference of 0.172279 seconds). Applying the switch flag to the hard Hexidoku problem, the solve-time increased from 5.870435 seconds to 168.570081 seconds (a difference of 162.699646 seconds), and total wall time increased from 5.907730 seconds to 170.849435 seconds (a difference of 164.941705 seconds). Although only one solution was able to be found for both Hexidoku problems, reaching this conclusion for the hard Hexidoku problem took 168.333194 seconds longer than reaching this conclusion for the easy Hexidoku problem.

So why did it take so much longer for the hard Hexidoku problem to be solved than the easy Hexidoku problem? In the easy Hexidoku problem we start with 106 pre-encoded values, whereas in the hard Hexidoku problem we start with 102 pre-encoded values. Although starting with four more encoded values may not seem as that big of an advantage to most, it means that there are more possibilities that each grid square can be, resulting in more checks having to be made, more searching, and more backtracking, resulting in a higher run time/solve-time.

Works Cited:

Ercsey-Ravasz, Mária, and Zoltán Toroczkai. "The Chaos Within Sudoku." *Sci. Rep. Scientific Reports* 2 (2012): n. pag. Web.

"Minion Manual." *Constraint Modelling*. N.p., n.d. Web. 02 Oct. 2016.

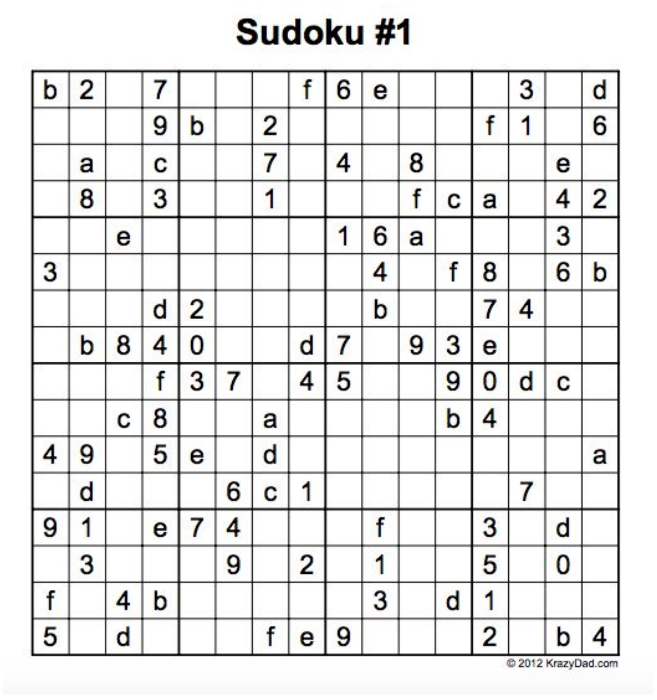
Russell, Stuart Jonathan., and Peter Norvig. "Constraint Satisfaction Problems." *Artificial Intelligence: A Modern Approach*. 3rd ed. Upper Saddle River: Pearson, 2010. 202-27. Print.

Solving Sudoku Using Constraint Satisfaction. Perf. Brandon Adame. *Youtube*. N.p., 23 Apr. 2012. Web. 2 Oct. 2016.

Hexidoku Problems Sourced From: http://krazydad.com/hexsudoku/

Appendix:

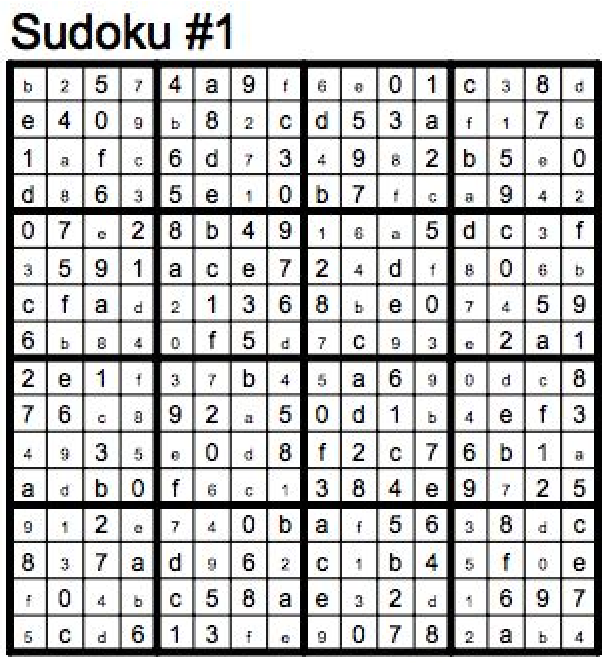
Easy Hexidoku:



Sudoku #1

b	2		7				f	6	e				3		d
			9	b		2						f	1		6
	a		c			7		4		8				e	
	8		3			1				f	c	a		4	2
		e						1	6	a				3	
3									4		f	8		6	b
			d	2					b			7	4		
	b	8	4	0			d	7		9	3	e			
			f	3	7		4	5			9	0	d	c	
		c	8			a					b	4			
4	9		5	e		d									a
	d				6	c	1						7		
9	1		e	7	4				f			3		d	
	3				9		2		1			5		0	
f		4	b						3		d	1			
5		d				f	e	9				2		b	4

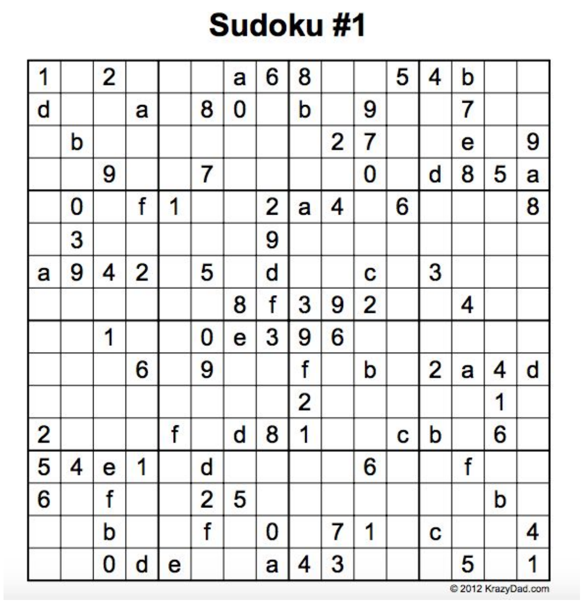
© 2012 KrazyDad.com



Sudoku #1

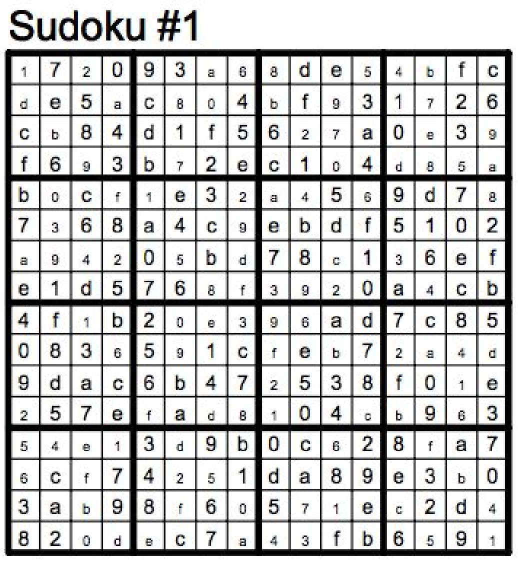
b	2	5	7	4	a	9	f	6	e	0	1	c	3	8	d
e	4	0	9	b	8	2	c	d	5	3	a	f	1	7	6
1	a	f	c	6	d	7	3	4	9	8	2	b	5	e	0
d	8	6	3	5	e	1	0	b	7	f	c	a	9	4	2
0	7	e	2	8	b	4	9	1	6	a	5	d	c	3	f
3	5	9	1	a	c	e	7	2	4	d	f	8	0	6	b
c	f	a	d	2	1	3	6	8	b	e	0	7	4	5	9
6	b	8	4	0	f	5	d	7	c	9	3	e	2	a	1
2	e	1	f	3	7	b	4	5	a	6	9	0	d	c	8
7	6	c	8	9	2	a	5	0	d	1	b	4	e	f	3
4	9	3	5	e	0	d	8	f	2	c	7	6	b	1	a
a	d	b	0	f	6	c	1	3	8	4	e	9	7	2	5
9	1	2	e	7	4	0	b	a	f	5	6	3	8	d	c
8	3	7	a	d	9	6	2	c	1	b	4	5	f	0	e
f	0	4	b	c	5	8	a	e	3	2	d	1	6	9	7
5	c	d	6	1	3	f	e	9	0	7	8	2	a	b	4

Hard Hexidoku



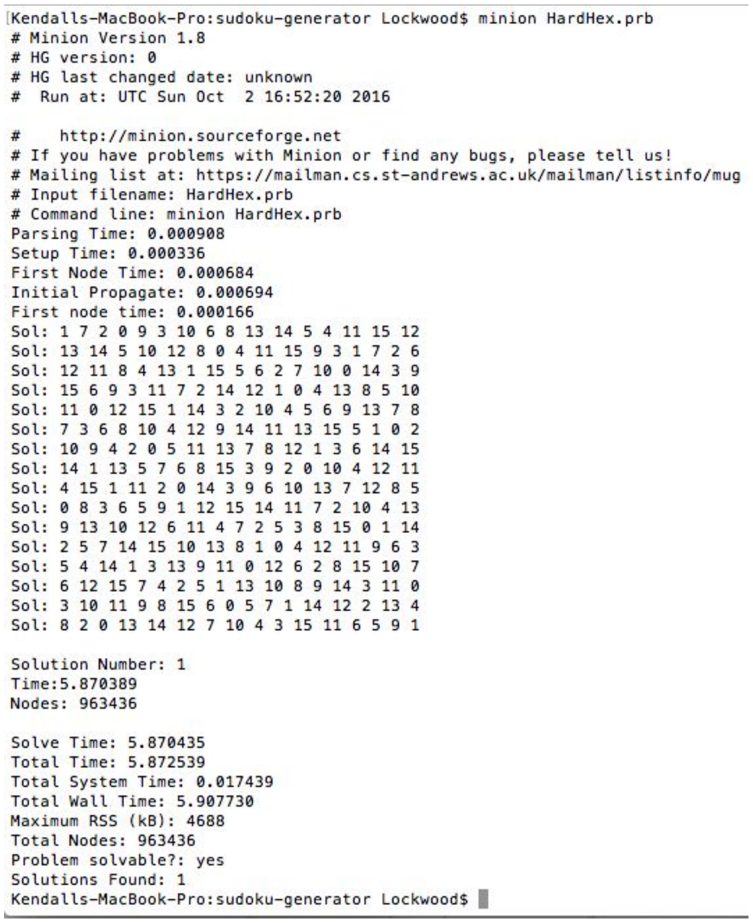
Sudoku #1

1		2				a	6	8			5	4	b		
d			a		8	0		b		9			7		
	b								2	7			e		9
		9			7					0		d	8	5	a
	0		f	1			2	a	4		6				8
	3						9								
a	9	4	2		5		d			c		3			
						8	f	3	9	2			4		
		1			0	e	3	9	6						
			6		9			f		b		2	a	4	d
								2						1	
2				f		d	8	1			c	b		6	
5	4	e	1		d					6			f		
6		f			2	5								b	
		b			f		0		7	1		c			4
		0	d	e			a	4	3				5		1



Sudoku #1

1	7	2	0	9	3	a	6	8	d	e	5	4	b	f	c
d	e	5	a	c	8	0	4	b	f	9	3	1	7	2	6
c	b	8	4	d	1	f	5	6	2	7	a	0	e	3	9
f	6	9	3	b	7	2	e	c	1	0	4	d	8	5	a
b	0	c	f	1	e	3	2	a	4	5	6	9	d	7	8
7	3	6	8	a	4	c	9	e	b	d	f	5	1	0	2
a	9	4	2	0	5	b	d	7	8	c	1	3	6	e	f
e	1	d	5	7	6	8	f	3	9	2	0	a	4	c	b
4	f	1	b	2	0	e	3	9	6	a	d	7	c	8	5
0	8	3	6	5	9	1	c	f	e	b	7	2	a	4	d
9	d	a	c	6	b	4	7	2	5	3	8	f	0	1	e
2	5	7	e	f	a	d	8	1	0	4	c	b	9	6	3
5	4	e	1	3	d	9	b	0	c	6	2	8	f	a	7
6	c	f	7	4	2	5	1	d	a	8	9	e	3	b	0
3	a	b	9	8	f	6	0	5	7	1	e	c	2	d	4
8	2	0	d	e	c	7	a	4	3	f	b	6	5	9	1

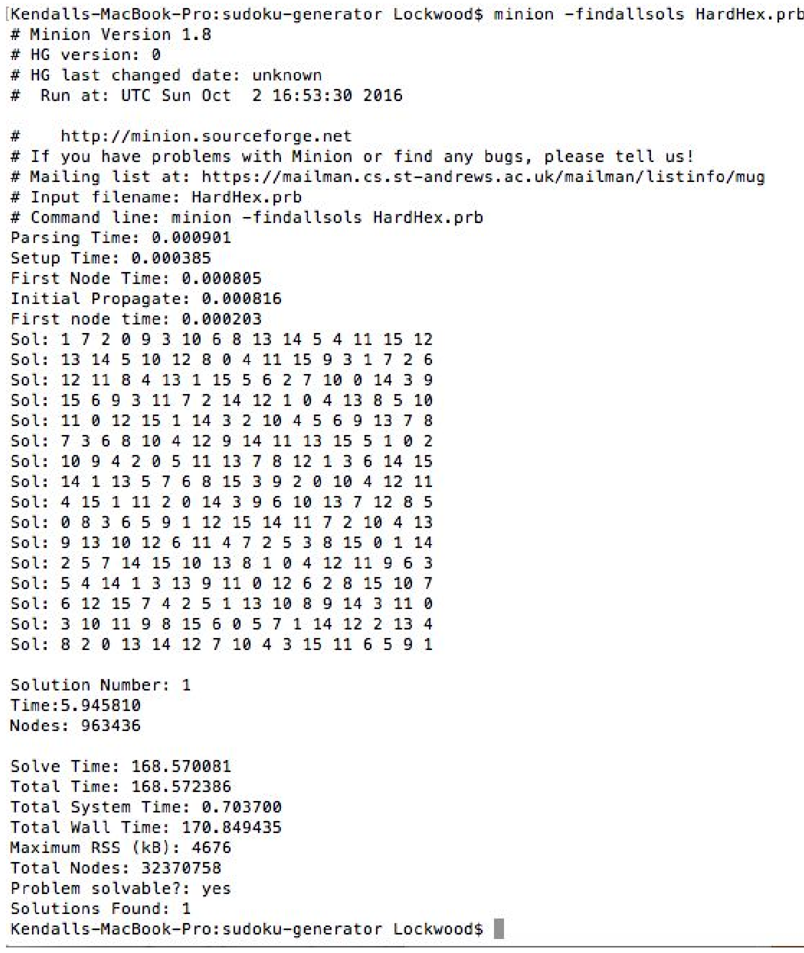


```
[Kendalls-MacBook-Pro:sudoku-generator Lockwood$ minion HardHex.prb
# Minion Version 1.8
# HG version: 0
# HG last changed date: unknown
#  Run at: UTC Sun Oct  2 16:52:20 2016

#    http://minion.sourceforge.net
# If you have problems with Minion or find any bugs, please tell us!
# Mailing list at: https://mailman.cs.st-andrews.ac.uk/mailman/listinfo/mug
# Input filename: HardHex.prb
# Command line: minion HardHex.prb
Parsing Time: 0.000908
Setup Time: 0.000336
First Node Time: 0.000684
Initial Propagate: 0.000694
First node time: 0.000166
Sol: 1 7 2 0 9 3 10 6 8 13 14 5 4 11 15 12
Sol: 13 14 5 10 12 8 0 4 11 15 9 3 1 7 2 6
Sol: 12 11 8 4 13 1 15 5 6 2 7 10 0 14 3 9
Sol: 15 6 9 3 11 7 2 14 12 1 0 4 13 8 5 10
Sol: 11 0 12 15 1 14 3 2 10 4 5 6 9 13 7 8
Sol: 7 3 6 8 10 4 12 9 14 11 13 15 5 1 0 2
Sol: 10 9 4 2 0 5 11 13 7 8 12 1 3 6 14 15
Sol: 14 1 13 5 7 6 8 15 3 9 2 0 10 4 12 11
Sol: 4 15 1 11 2 0 14 3 9 6 10 13 7 12 8 5
Sol: 0 8 3 6 5 9 1 12 15 14 11 7 2 10 4 13
Sol: 9 13 10 12 6 11 4 7 2 5 3 8 15 0 1 14
Sol: 2 5 7 14 15 10 13 8 1 0 4 12 11 9 6 3
Sol: 5 4 14 1 3 13 9 11 0 12 6 2 8 15 10 7
Sol: 6 12 15 7 4 2 5 1 13 10 8 9 14 3 11 0
Sol: 3 10 11 9 8 15 6 0 5 7 1 14 12 2 13 4
Sol: 8 2 0 13 14 12 7 10 4 3 15 11 6 5 9 1

Solution Number: 1
Time:5.870389
Nodes: 963436

Solve Time: 5.870435
Total Time: 5.872539
Total System Time: 0.017439
Total Wall Time: 5.907730
Maximum RSS (kB): 4688
Total Nodes: 963436
Problem solvable?: yes
Solutions Found: 1
Kendalls-MacBook-Pro:sudoku-generator Lockwood$ 
```



```
[Kendalls-MacBook-Pro:sudoku-generator Lockwood$ minion -findallsols HardHex.prb
# Minion Version 1.8
# HG version: 0
# HG last changed date: unknown
#  Run at: UTC Sun Oct  2 16:53:30 2016

#    http://minion.sourceforge.net
# If you have problems with Minion or find any bugs, please tell us!
# Mailing list at: https://mailman.cs.st-andrews.ac.uk/mailman/listinfo/mug
# Input filename: HardHex.prb
# Command line: minion -findallsols HardHex.prb
Parsing Time: 0.000901
Setup Time: 0.000385
First Node Time: 0.000805
Initial Propagate: 0.000816
First node time: 0.000203
Sol: 1 7 2 0 9 3 10 6 8 13 14 5 4 11 15 12
Sol: 13 14 5 10 12 8 0 4 11 15 9 3 1 7 2 6
Sol: 12 11 8 4 13 1 15 5 6 2 7 10 0 14 3 9
Sol: 15 6 9 3 11 7 2 14 12 1 0 4 13 8 5 10
Sol: 11 0 12 15 1 14 3 2 10 4 5 6 9 13 7 8
Sol: 7 3 6 8 10 4 12 9 14 11 13 15 5 1 0 2
Sol: 10 9 4 2 0 5 11 13 7 8 12 1 3 6 14 15
Sol: 14 1 13 5 7 6 8 15 3 9 2 0 10 4 12 11
Sol: 4 15 1 11 2 0 14 3 9 6 10 13 7 12 8 5
Sol: 0 8 3 6 5 9 1 12 15 14 11 7 2 10 4 13
Sol: 9 13 10 12 6 11 4 7 2 5 3 8 15 0 1 14
Sol: 2 5 7 14 15 10 13 8 1 0 4 12 11 9 6 3
Sol: 5 4 14 1 3 13 9 11 0 12 6 2 8 15 10 7
Sol: 6 12 15 7 4 2 5 1 13 10 8 9 14 3 11 0
Sol: 3 10 11 9 8 15 6 0 5 7 1 14 12 2 13 4
Sol: 8 2 0 13 14 12 7 10 4 3 15 11 6 5 9 1

Solution Number: 1
Time:5.945810
Nodes: 963436

Solve Time: 168.570081
Total Time: 168.572386
Total System Time: 0.703700
Total Wall Time: 170.849435
Maximum RSS (kB): 4676
Total Nodes: 32370758
Problem solvable?: yes
Solutions Found: 1
Kendalls-MacBook-Pro:sudoku-generator Lockwood$ 
```