

# Using Progressive Stochastic Search to solve Sudoku CSP

Jeetesh Mangwani & Pankaj Prateek

Advisor: Dr. Amitabh Mukherjee[?][?]

Dept. of Computer Science and Engineering

Indian Institute of Technology, Kanpur, India

{jeeteshm, pratikkr, amit} @ cse.iitk.ac.in

April 14, 2012

## ABSTRACT

*Using stochastic search methods to find solutions to Constraining Satisfaction Problems (CSPs) has been reasonably successful in the recent years, with the deterministic search methods performing worse in many combinatorially hard problems. This motivated us to test this ‘superiority’ of Stochastic Search methods over Deterministic Search methods. In this project, we implement Progressive Stochastic Search and Incremental Progressive Stochastic Search as methods to solve Sudoku puzzles of order-2,3 and 4. The results show that these methods do converge to correct solutions, building heuristics during the process, without exploiting any problem-specific standard solving methods. The timings show that deterministic methods have an extremely fast convergence, solving order-2 & 3 puzzles in time less than 20 ms. On the other hand, PSS solves order-2 puzzles in about 11.554  $\mu$ , order-3 puzzles in about 545.850 ms, easy, intermediate & hard order-4 puzzles in about 2, 12 and 30 minutes respectively.*

## 1 Previous Work

This project is based on a 2003 paper titled ‘PSS for solving CSPs’[?], by Bryan Chi-ho Lam and Ho-fung Leung. PSS requires no previous knowledge of the problem and builds heuristics during the period problem is being solved. The paper also suggests a modified PSS algorithm, termed as Incremental Progressive Stochastic Search (IPSS). The results talk about timing comparisons when PSS, IPSS, max-PSS, max-IPSS and LSDL are tested over N-queens problem, permutation generation problem, Latin squares, Quasigroup completion problems and random CSPs.

Stochastic optimization approaches have previously been applied to solve sudoku CSP. Perez and Marwala used Cultural Genetic Algorithm (CGA), Repulsive Particle Swarm Optimization (RPSO), Quantum Simulated Annealing (QSA) and Hybrid Genetic Algorithm with Simulated Annealing (HGASA) to solve sudoku CSP[?].

## 2 Introduction

A CSP is conventionally defined as a problem of finding a consistent assignment, if any, of discrete values to a finite set of variables such that the assignment satisfies a finite set of given constraints over these variables. The characteristic of PSS is that it maintains a list of variables, which dictates the sequence of variables to repair. When a variable is designated to be repaired, it always has to choose a new value even if its original value should give the best cost value.

Intuitively, the search can be thought to be driven by a force so that the search is able to rush through the local minima and plateaus. The search paths are also slightly marked by worsening at every point on the paths as the search proceeds. Random restarts are no longer necessary, and

expensive heuristic learning is replaced by simple path marking.[?]

An order- $N$  sudoku puzzle has  $4N^4$  constraints: Each of the  $N^2$  rows, columns, blocks must have exactly  $N^2$  values, while each of the  $N^4$  cells must be filled with only one value.[?]

### 3 Progressive Stochastic Search

In Progressive Stochastic Search (PSS)<sup>1</sup>, a CSP is modelled as a network of variables, each represented by a cluster of label nodes, each of which correspond to the values present in the domain of the variable. At any moment, only one of these label nodes is active in a cluster (is in on state) while the others are in off state. In other words, this active node corresponds to the current value assigned to the cluster. In the following notes, the terms variables, cells and clusters are used interchangeably.

A constraint  $c \in C$  on two clusters  $x$  and  $y$  is represented as weighed connections between the label nodes of these clusters,  $x_i$  and  $y_j$ . These two label nodes are connected if and only if  $(x = x_i) \wedge (y = y_j)$  is prohibited. Each connection is associated with a weight  $W(x_i, y_j)$  which is initialised to 1. An assignment of values to the variables from their domains is called a state of the problem, with the solution state being the one in which no two on label nodes are connected to each other.

The clusters<sup>2</sup> are initialised using their minimum conflicting values in accordance with the given values in the sudoku puzzle, until the first complete assignment is reached.

All clusters are then added to a list  $Q$ . In each iteration, the head of the queue  $h$  is removed and ‘repaired’ to a new value  $n_h$  (its minimum conflicting value among all values other than the present value) even if the present value  $p_h$  gives a better cost, unless the present value is the only possible value in the domain. Each cluster, with its on value connected to this new value  $n_h$ , i.e. having the same on node as the just-switched-on label node of  $h$ , is appended to  $Q$ , if it is not already present. We then increase the weights of all the connections between the on label nodes and  $p_h$  by 1.

These iterations are continued till the queue is empty. In this situation, PSS is in a solution state.

### 4 Incremental PSS

Incremental PSS<sup>3</sup> is a variant of PSS, which finds a consistent partial assignment and extends it until a complete solution is found. In IPSS, the underlying network architecture is same as that of PSS with a difference that the state represents a partial assignment of values to variables from their respective domains.

IPSS divides the set of clusters into two subsets,  $Q_{PSS}$  which contains the clusters with one on label node and  $Q_{IPSS}$  which contains the rest. Initially, all clusters are in  $Q_{IPSS}$  which are selected and moved to  $Q_{PSS}$  one by one. While moving a cluster  $i$  to  $Q_{PSS}$ , its label node  $m_i$  with the minimum conflicting value is turned on.

The list  $Q_{PSS}$  is initialised to empty. Any cluster in  $Q_{IPSS}$ , having its on label node connected to  $m_i$  is appended to  $Q$ , unless it is already present. Then the convergence step in PSS is applied to

<sup>1</sup>For details, consult the paper[?] by Lam and Leung

<sup>2</sup>cluster : a cell in the sudoku puzzle

<sup>3</sup>For details, consult the paper[?] by Lam and Leung

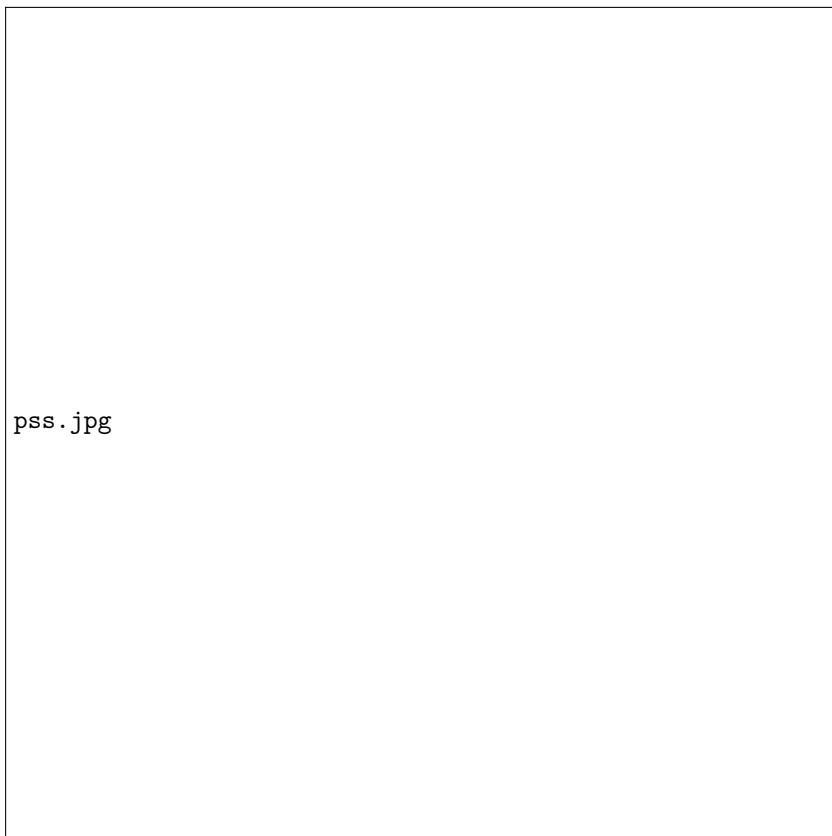


Figure 1: PSS Network

$Q_{PSS}$  until it becomes empty. This is followed by moving next head cluster and conflicting clusters from  $Q_{IPSS}$  to  $Q_{PSS}$ . Convergence is achieved once both the lists are empty.

## 5 Implementation

The code has been written in C++ and compiled using GNU C++ v4.3.2. The code was compiled and tested on Intel i7 @2GHz machine. The code can be found in [?][?]. The order-2 and 3 puzzles were generated using generator available at [?]. The order-4 puzzles were taken from [?]. The Norvig-like Python code was taken from [?] and edited to get it working on order-2 and order-3 puzzles.

## 6 Results

PSS, IPSS and Norvig's code were tested on the generated puzzle set, while the remaining results are from [?].

Table 1: Average Time required to solve a puzzle

Algorithm/Code	4X4	9X9	16X16
<b><i>Stochastic Algorithms</i></b>			
PSS[?]	11.554 $\mu$	545.850 ms	902.599 s
IPSS[?]	12.515 $\mu$	7.260 s	-
Cultural Genetic Algorithm (CGA)[?]	-	28 s	-
Quantum Simulated Annealing (QSA)[?]	-	65 s	-
Repulsive Particle Swarm Optimization (RPSO)[?]	-	Unable	-
Hybrid Genetic Algorithm with Simulated Annealing (HGASA)[?]	-	1.447 s	-
<b><i>Deterministic Algorithms</i></b>			
Norvig's Code (python)[?]	0.539 ms	18.500 ms	-

## 6.1 Progressive Stochastic Search

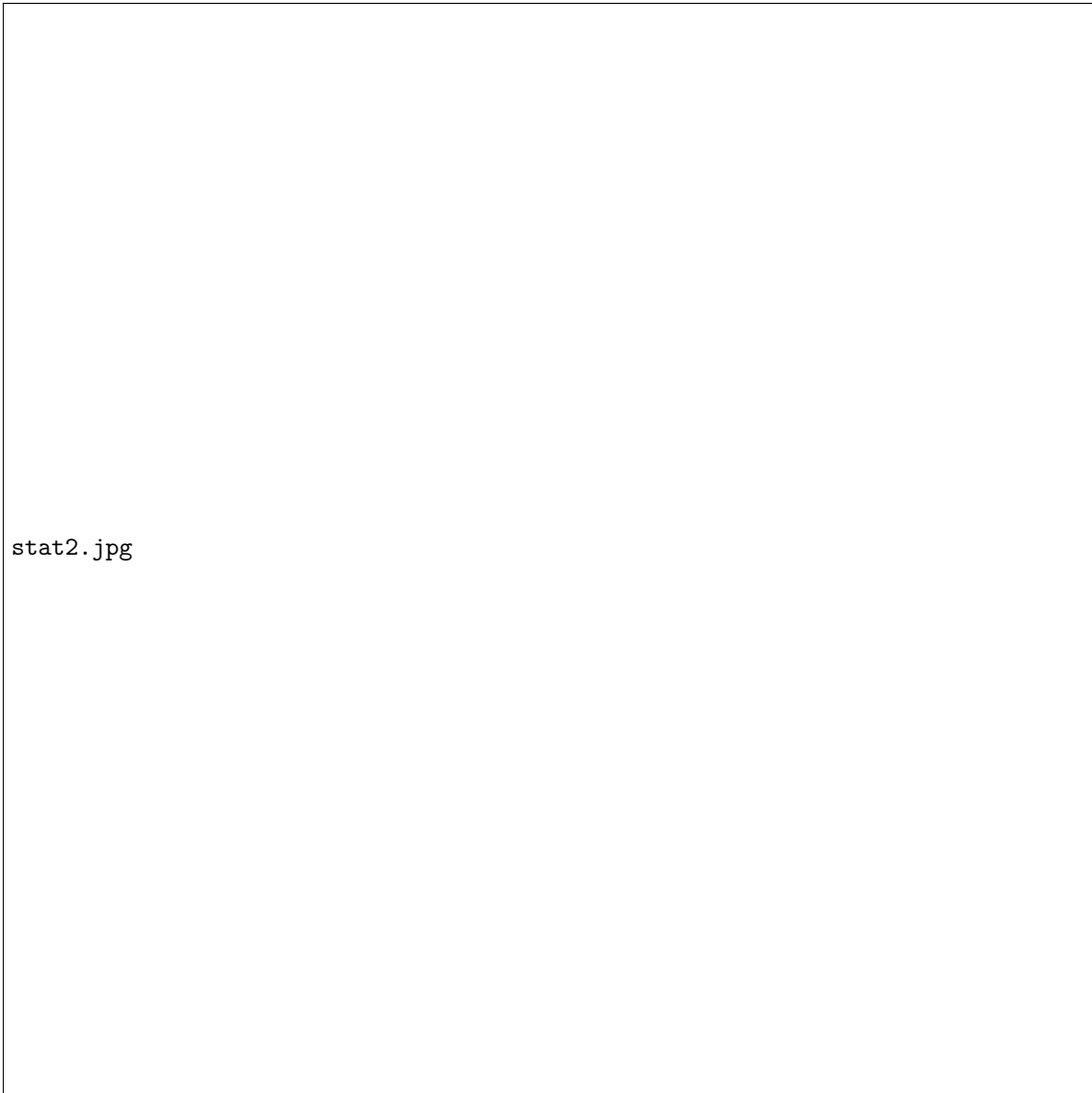
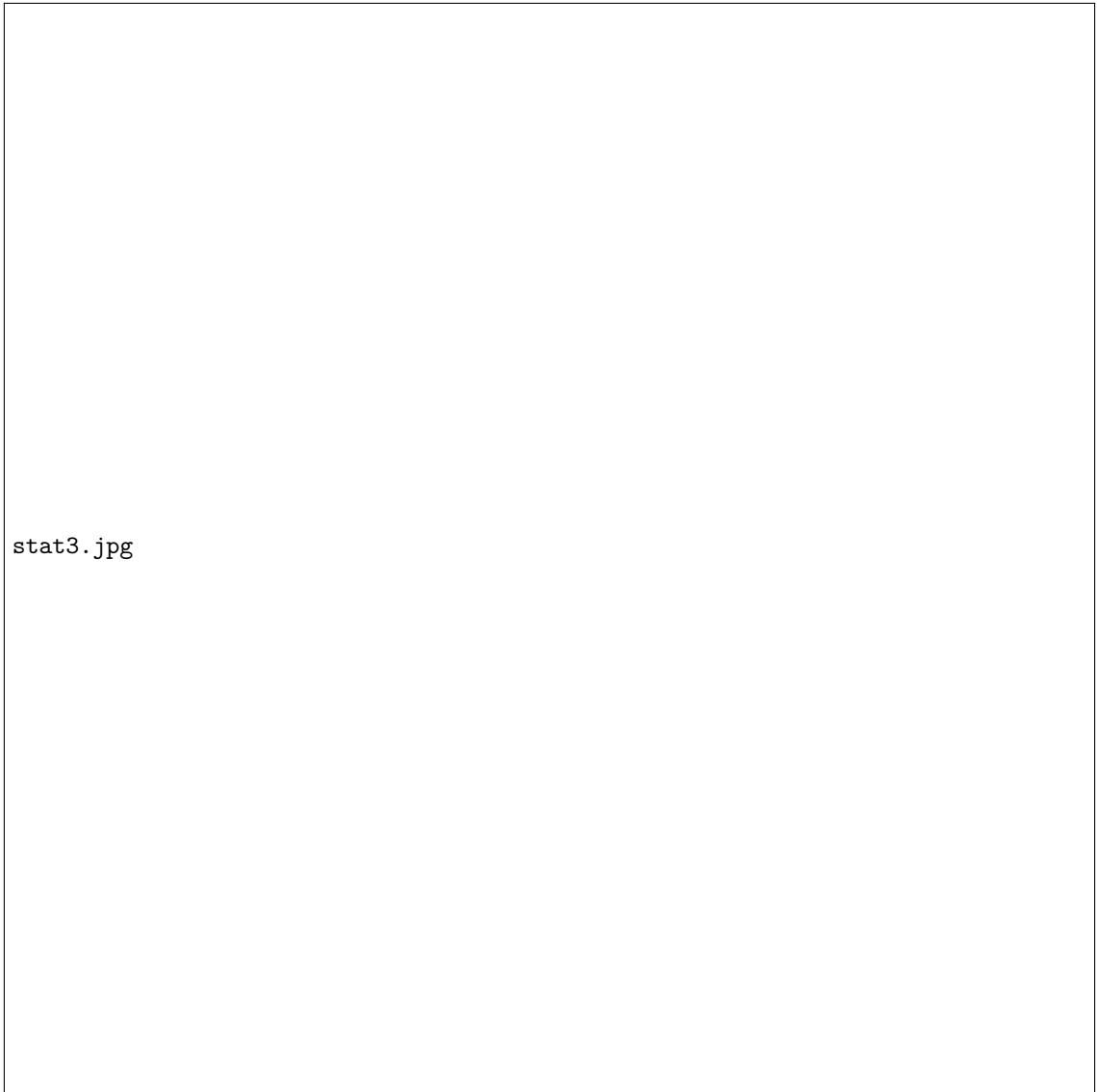
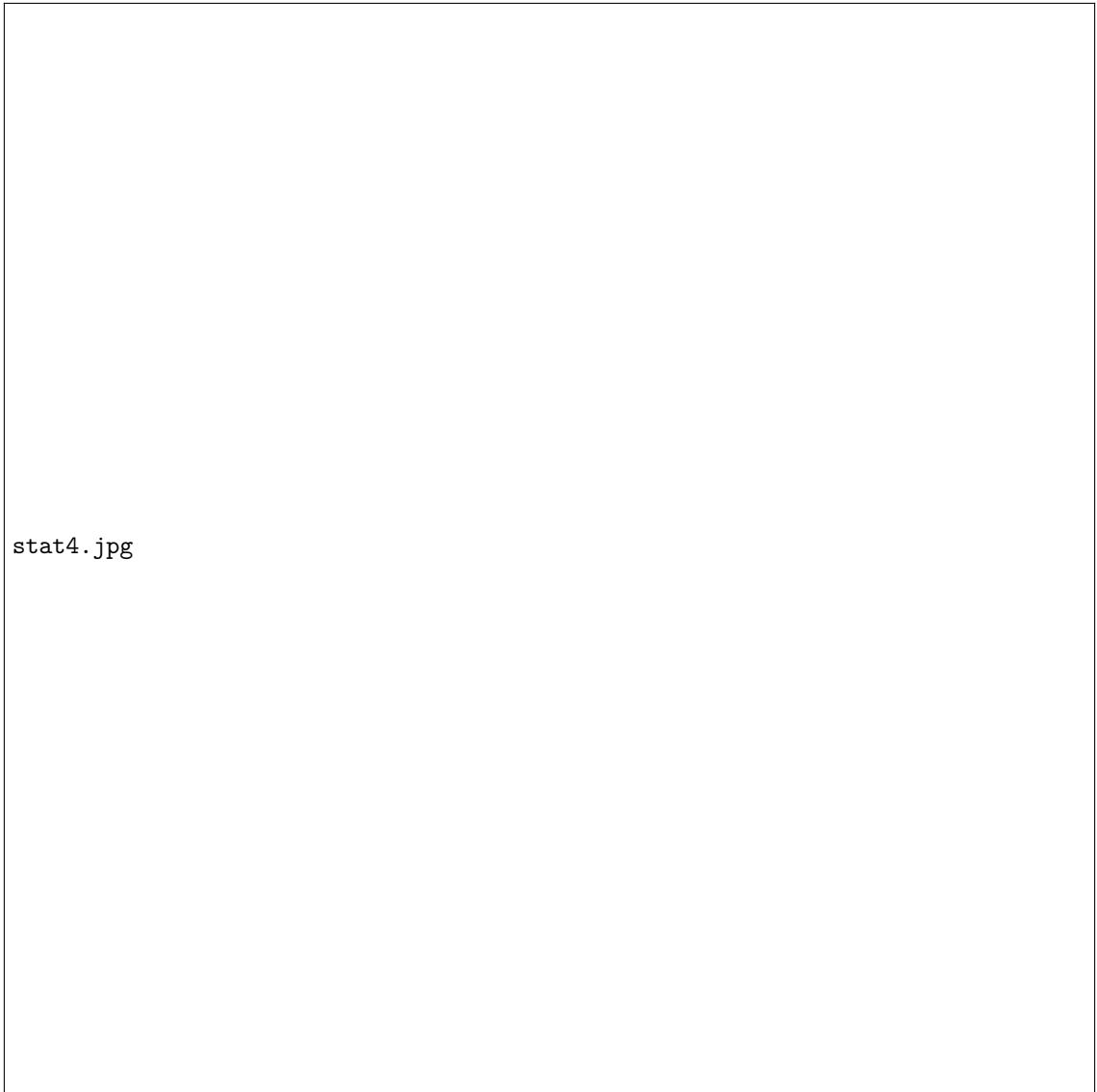


Figure 2: 4X4 Sudoku: Average Time =  $11.554 \mu$



stat3.jpg

Figure 3: 9X9 Sudoku: Average Time = 545.850 ms



stat4.jpg

Figure 4: 16X16 Sudoku: Average Time = 902.599 s

## 6.2 Incremental Progressive Stochastic Search

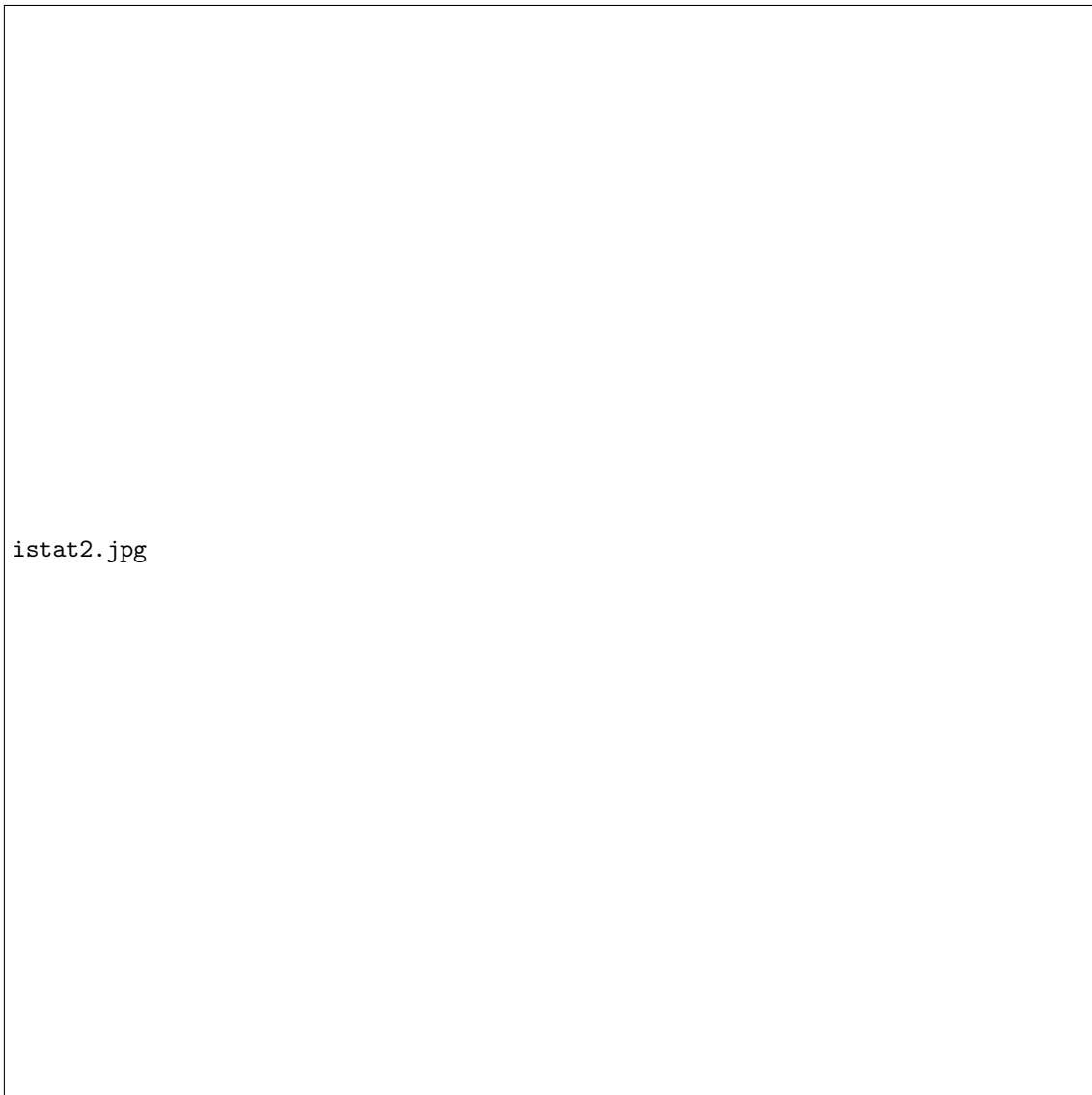
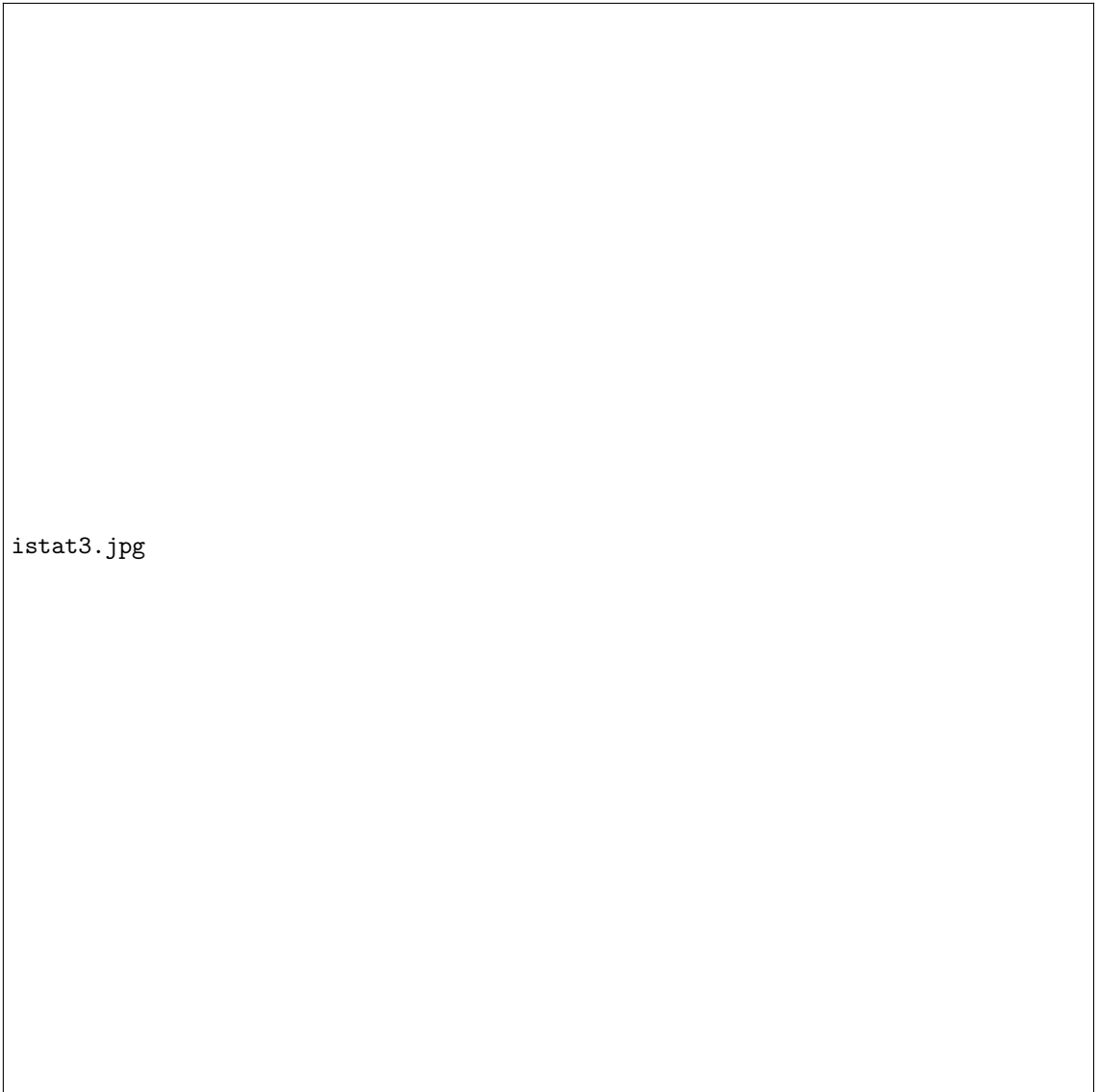


Figure 5: 4X4 Sudoku: Average Time = 12.515  $\mu$ s





istat3.jpg

Figure 6: 9X9 Sudoku: Average Time = 7.260 s

## References

- [1] Bryan Chi-ho Lam and Ho-fung Leung, 2003. Progressive Stochastic Search for Solving Constraint Satisfaction Problems. Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI03) 1082-3409/03, Pages 487-491.
- [2] Rhydian Lewis, 2007. On the Combination of Constraint Programming and Stochastic Search. The Sudoku Case Proceedings of the 4th international conference on Hybrid metaheuristics.
- [3] <http://norvig.com/sudoku.html>
- [4] Meir Perez and Tshilidzi Marwala, 2008. Stochastic Optimization Approaches for Solving Sudoku. CoRR, journals/corr/abs-0805-0697.
- [5] <http://home.iitk.ac.in/~jeeteshm/cs365/projects>
- [6] <http://home.iitk.ac.in/~pratikk/cs365/projects>

- [7] <http://ostermiller.org/qqwing/>
- [8] <http://www.menneske.no/sudoku/4/eng/>
- [9] <http://pastebin.com/kamVh7Cx>
- [10] <http://www.cse.iitk.ac.in/users/cs365/>
- [11] <http://www.cse.iitk.ac.in/users/amit//>