Progressive Stochastic Search

Application to solving sudoku

Jeetesh Mangwani & Pankaj Prateek

Indian Institute of Technology, Kanpur, India

April 14, 2012

Sudoku

- Sudoku: *Solitary mumber*
- Originated in USA in 1970s
- Gained mainstream popularity in Japan in 1980s

Sudoku

- Sudoku: *Solitary mumber*
- Originated in USA in 1970s
- Gained mainstream popularity in Japan in 1980s

Sudoku

- Sudoku: *Solitary mumber*
- Originated in USA in 1970s
- Gained mainstream popularity in Japan in 1980s

Sudoku: Problem Definition

A Sudoku square of order n consists of n^4 variables formed into a n^2 x n^2 grid such that:

1. Each row of cells contains the integers 1 through n^2 exactly once.
2. Each column of cells contains the integers 1 through n^2 exactly once.
3. Each of the major n x n block contains the integers 1 through n^2 exactly once.

Sudoku: Problem Definition

A Sudoku square of order n consists of n^4 variables formed into a n^2 x n^2 grid such that:

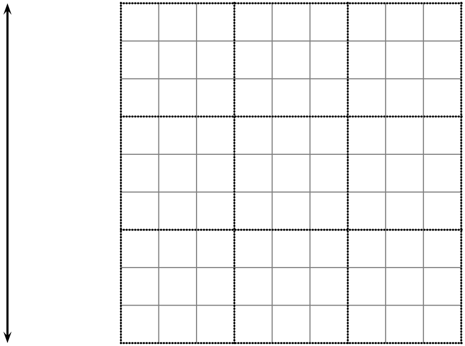
1. Each row of cells contains the integers 1 through n^2 exactly once.
2. Each column of cells contains the integers 1 through n^2 exactly once.
3. Each of the major n x n block contains the integers 1 through n^2 exactly once.

Sudoku: Problem Definition

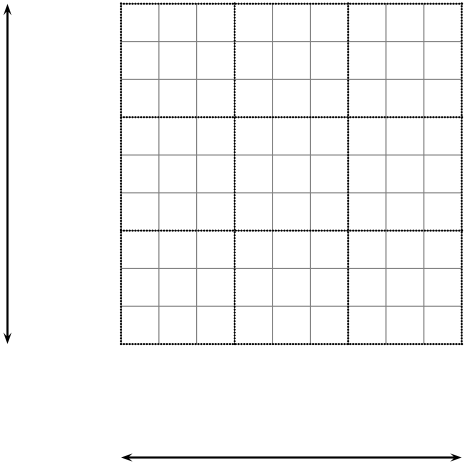
A Sudoku square of order n consists of n^4 variables formed into a n^2 x n^2 grid such that:

1. Each row of cells contains the integers 1 through n^2 exactly once.
2. Each column of cells contains the integers 1 through n^2 exactly once.
3. Each of the major n x n block contains the integers 1 through n^2 exactly once.

A simple illustration of PSS



A simple illustration of PSS



Motivation

- All Sudoku puzzles are not logic-solvable; many involve guesswork
- Proved to be an NP-complete problem: a polynomially bound algorithm for solving all problem instances is not possible
- It is claimed that even for order-3 Sudoku puzzles, there are 6,670,903,752,021,072,936,960 valid arrangements (!)
- After being mesmerised at stochastic algorithms overpowering the deterministic ones for NP-complete problems, we were motivated to dig deeper and test this "superiority' on the Sudoku problem

Motivation

- All Sudoku puzzles are not logic-solvable; many involve guesswork
- Proved to be an NP-complete problem: a polynomially bound algorithm for solving all problem instances is not possible
- It is claimed that even for order-3 Sudoku puzzles, there are 6,670,903,752,021,072,936,960 valid arrangements (!)
- After being mesmerised at stochastic algorithms overpowering the deterministic ones for NP-complete problems, we were motivated to dig deeper and test this "superiority' on the Sudoku problem

Stochastic Search

- Typical SS algorithms first generate a complete initial variable assignment and repair the assignment by heuristic local search with reference to a cost function until a solution is found.
  - Problem: Might get stuck on a plateau or in a local optima
  - Repairs:
    - Random Restart: Information gained in the search process is lost at each restart
    - Use heuristics and associate weights with the constraints violations and define the cost function as a weighed sum of these violations: Learning heuristics may be difficult

Stochastic Search

- Typical SS algorithms first generate a complete initial variable assignment and repair the assignment by heuristic local search with reference to a cost function until a solution is found.
    - Problem: Might get stuck on a plateau or in a local optima
    - Repairs:
        - Random Restart: Information gained in the search process is lost at each restart
        - Use heuristics and associate weights with the constraints violations and define the cost function as a weighed sum of these violations: Learning heuristics may be difficult

Stochastic Search

- Typical SS algorithms first generate a complete initial variable assignment and repair the assignment by heuristic local search with reference to a cost function until a solution is found.
  - Problem: Might get stuck on a plateau or in a local optima
  - Repairs:
    - Random Restart: Information gained in the search process is lost at each restart
    - Use heuristics and associate weights with the constraints violations and define the cost function as a weighed sum of these violations: Learning heuristics may be difficult

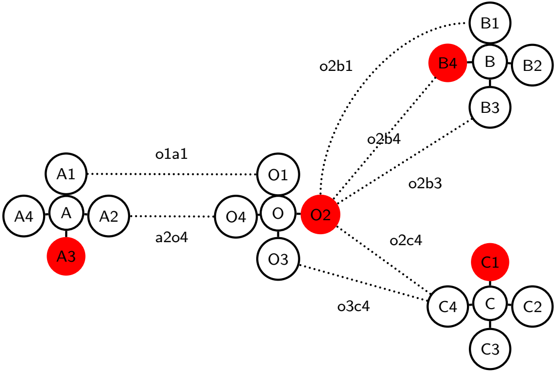
Stochastic Search

- Typical SS algorithms first generate a complete initial variable assignment and repair the assignment by heuristic local search with reference to a cost function until a solution is found.
  - Problem: Might get stuck on a plateau or in a local optima
  - Repairs:
    - Random Restart: Information gained in the search process is lost at each restart
    - Use heuristics and associate weights with the constraints violations and define the cost function as a weighed sum of these violations: Learning heuristics may be difficult

Stochastic Search

- Typical SS algorithms first generate a complete initial variable assignment and repair the assignment by heuristic local search with reference to a cost function until a solution is found.
  - Problem: Might get stuck on a plateau or in a local optima
  - Repairs:
    - Random Restart: Information gained in the search process is lost at each restart
    - Use heuristics and associate weights with the constraints violations and define the cost function as a weighed sum of these violations: Learning heuristics may be difficult

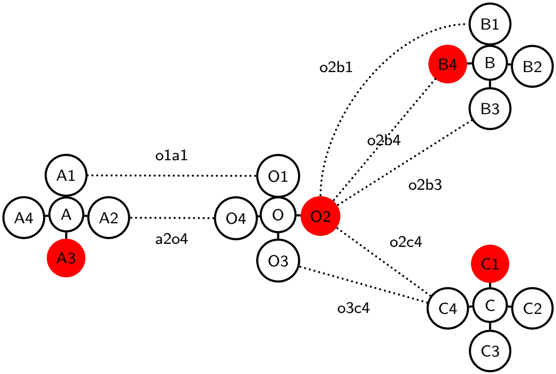
A simple illustration of PSS



A simple illustration of PSS

Step 1

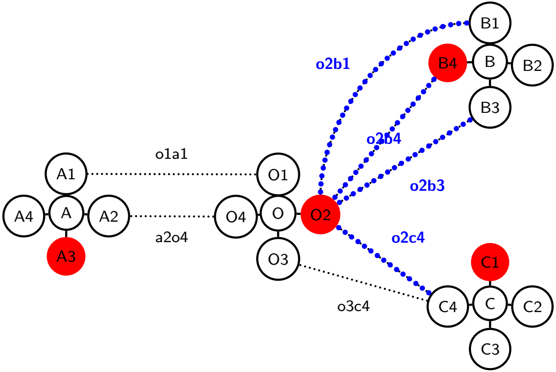
Pick up the head cluster O of queueQ.



A simple illustration of PSS

Step 2

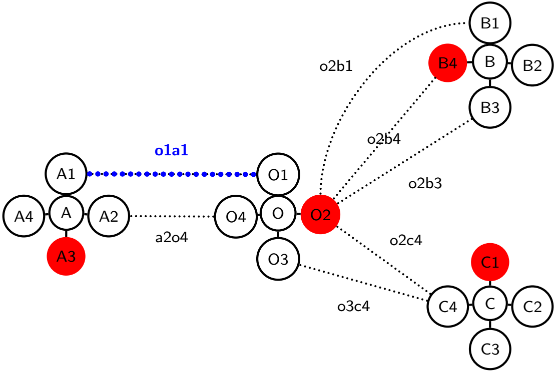
$conflicts_{O2} = o2b1 + o2b3 + o2b4 + o2c4 \neq 0$



A simple illustration of PSS

Step 3

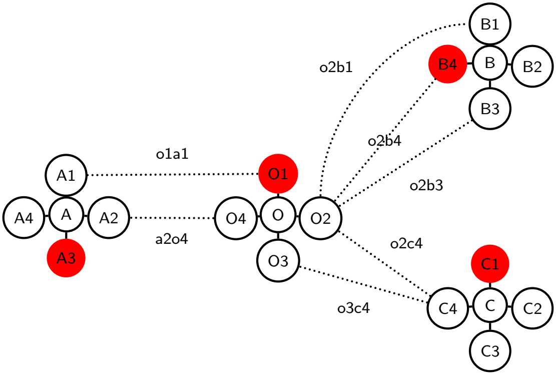
$conflicts_{O1} = o1a1$



A simple illustration of PSS

Step 4

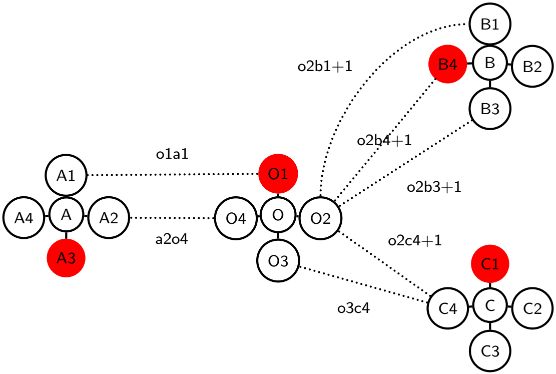
$value_O := O_1$



A simple illustration of PSS

Step 5

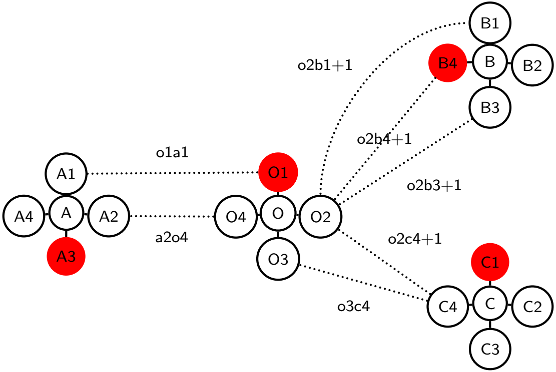
$o2b1 := o2b1 + 1$ $o2b3 := o2b3 + 1$
$o2b4 := o2b4 + 1$ $o2c4 := o2c4 + 1$



A simple illustration of PSS

Step 7

Append A_1 to the queue



Progressive Stochastic Search

- Maintains a list of variables, which dictate the sequence of variables to be repaired
- When a variable is designated to be repaired in PSS, it always chooses a new value even if its original value gives a better cost
- Search paths slightly marked by worsening at every point on the paths
- Intuitively, it's driven by a "force" so that the search is able to "rush through" the local minima and plateaus

Progressive Stochastic Search

- Maintains a list of variables, which dictate the sequence of variables to be repaired
- When a variable is designated to be repaired in PSS, it always chooses a new value even if its original value gives a better cost
- Search paths slightly marked by worsening at every point on the paths
- Intuitively, it's driven by a "force" so that the search is able to "rush through" the local minima and plateaus

Progressive Stochastic Search

- Maintains a list of variables, which dictate the sequence of variables to be repaired
- When a variable is designated to be repaired in PSS, it always chooses a new value even if its original value gives a better cost
- Search paths slightly marked by worsening at every point on the paths
- Intuitively, it's driven by a "force" so that the search is able to "rush through" the local minima and plateaus

Progressive Stochastic Search

- Maintains a list of variables, which dictate the sequence of variables to be repaired
- When a variable is designated to be repaired in PSS, it always chooses a new value even if its original value gives a better cost
- Search paths slightly marked by worsening at every point on the paths
- Intuitively, it's driven by a "force" so that the search is able to "rush through" the local minima and plateaus

Advantages: PSS

- Random restarts are no longer necessary
- Expensive heuristic learning is replaced by simple path marking

Advantages: PSS

- Random restarts are no longer necessary
- Expensive heuristic learning is replaced by simple path marking

Modelling Sudoku as a CSP

- The Sudoku puzzle will be modelled as a grid of n^4 cells - each of which represents an integer variable which initially will have a domain of 1 through n^2
- Constraints can be added in the form of "alldifferent" constraints and using the pre-filled cells in the grid
- Domain sizes of some variables reduced
- The problem remains to finding a bijection between n^2 variables and n^2 values satisfying the constraints

Modelling Sudoku as a CSP

- The Sudoku puzzle will be modelled as a grid of n^4 cells - each of which represents an integer variable which initially will have a domain of 1 through n^2
- Constraints can be added in the form of "alldifferent" constraints and using the pre-filled cells in the grid
- Domain sizes of some variables reduced
- The problem remains to finding a bijection between n^2 variables and n^2 values satisfying the constraints

Modelling Sudoku as a CSP

- The Sudoku puzzle will be modelled as a grid of n^4 cells - each of which represents an integer variable which initially will have a domain of 1 through n^2
- Constraints can be added in the form of "alldifferent" constraints and using the pre-filled cells in the grid
- Domain sizes of some variables reduced
- The problem remains to finding a bijection between n^2 variables and n^2 values satisfying the constraints

Modelling Sudoku as a CSP

- The Sudoku puzzle will be modelled as a grid of n^4 cells - each of which represents an integer variable which initially will have a domain of 1 through n^2
- Constraints can be added in the form of "alldifferent" constraints and using the pre-filled cells in the grid
- Domain sizes of some variables reduced
- The problem remains to finding a bijection between n^2 variables and n^2 values satisfying the constraints

Procedure

- Initially, logically-deductible values are filled in their correspoding squares
- This reduces the search space to the point where only guesswork can lead to progress
- It is here that PSS takes over the reign and completes the partial assignment to completeness

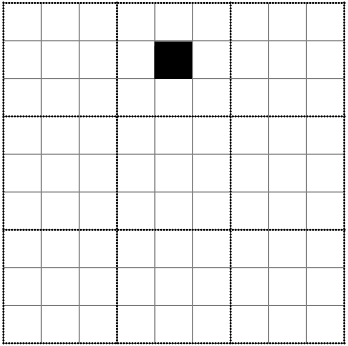
Procedure

- Initially, logically-deductible values are filled in their correspoding squares
- This reduces the search space to the point where only guesswork can lead to progress
- It is here that PSS takes over the reign and completes the partial assignment to completeness

Procedure

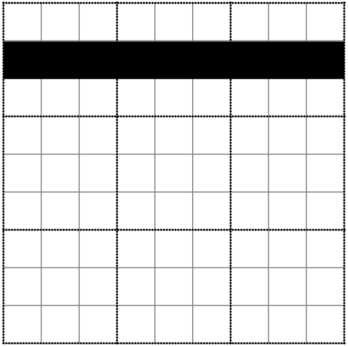
- Initially, logically-deductible values are filled in their correspoding squares
- This reduces the search space to the point where only guesswork can lead to progress
- It is here that PSS takes over the reign and completes the partial assignment to completeness

A simple illustration of PSS



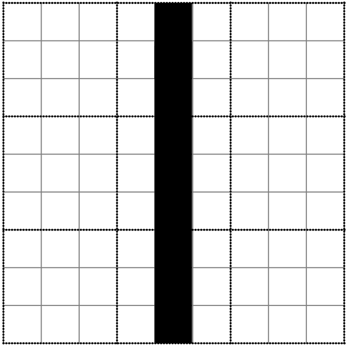
$Domain = Domain - v$

A simple illustration of PSS



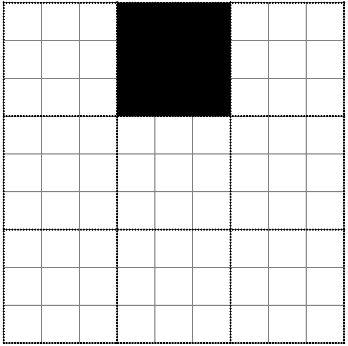
$Domain = Domain - v$

A simple illustration of PSS



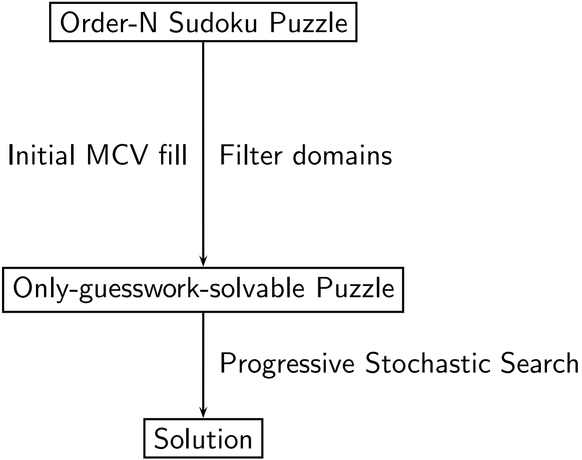
$Domain = Domain - v$

A simple illustration of PSS



$Domain = Domain - v$

A simple illustration of PSS



Implementation

1. typedef struct cluster{
   bool domain[N*N]; domain[i]==true means i is in the domain of this cell
   num value; value is the current value of this cell
   bool given; given==true means that this cell was already given in the original problem
   bool queued; queued==true means that this cell is currently queued
   }cell;
2. cell matrix[N*N] The main array acting as the workplace
3. long long int rwt[N*N][N*N][N*N][N*N]; rwt[r][i][j][v] is the weight of constraint between ith and jth cell in rth row for the value v
   long long int cwt[N*N][N*N][N*N][N*N]; cwt[r][i][j][v] is the weight of constraint between ith and jth cell in rth column for the value v
   long long int bwt[N*N][N*N][N*N][N*N]; bwt[r][i][j][v] is the weight of constraint between ith and jth cell in rth block for the value v
4. int blockno=((int)(i/N))*N + (int)(j/N);
   int ijno=((int)(i%N))*N + (int)(j%N);

Implementation

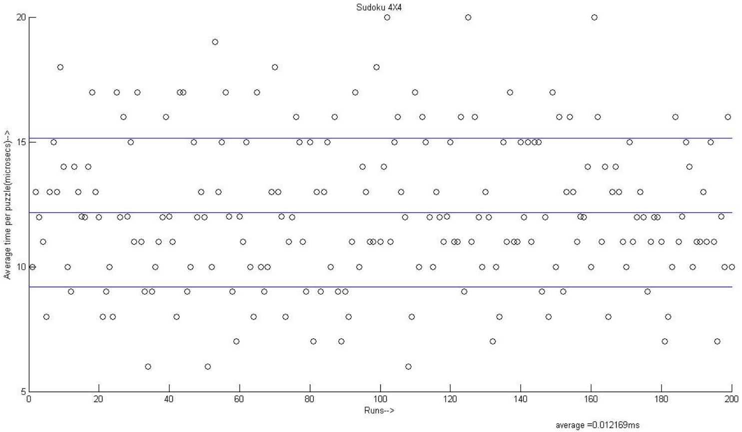
1. typedef struct cluster{
   bool domain[N*N]; domain[i]==true means i is in the domain of this cell
   num value; value is the current value of this cell
   bool given; given==true means that this cell was already given in the original problem
   bool queued; queued==true means that this cell is currently queued
   }cell;
2. cell matrix[N*N] The main array acting as the workplace
3. long long int rwt[N*N][N*N][N*N][N*N]; rwt[r][i][j][v] is the weight of constraint between ith and jth cell in rth row for the value v
   long long int cwt[N*N][N*N][N*N][N*N]; cwt[r][i][j][v] is the weight of constraint between ith and jth cell in rth column for the value v
   long long int bwt[N*N][N*N][N*N][N*N]; bwt[r][i][j][v] is the weight of constraint between ith and jth cell in rth block for the value v
4. int blockno=((int)(i/N))*N + (int)(j/N);
   int ijno=((int)(i%N))*N + (int)(j%N);

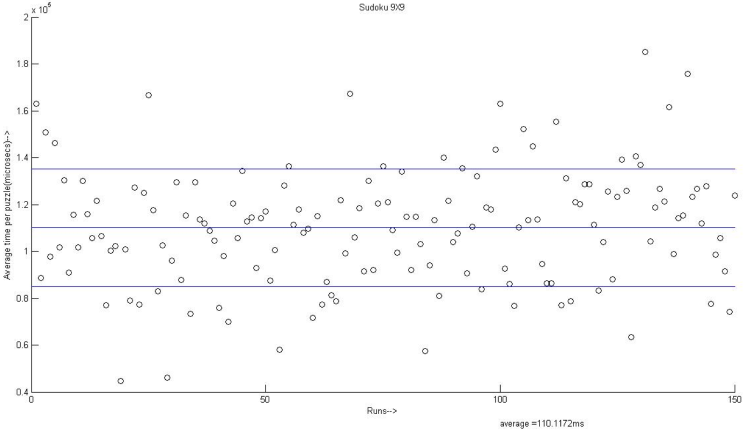
Implementation

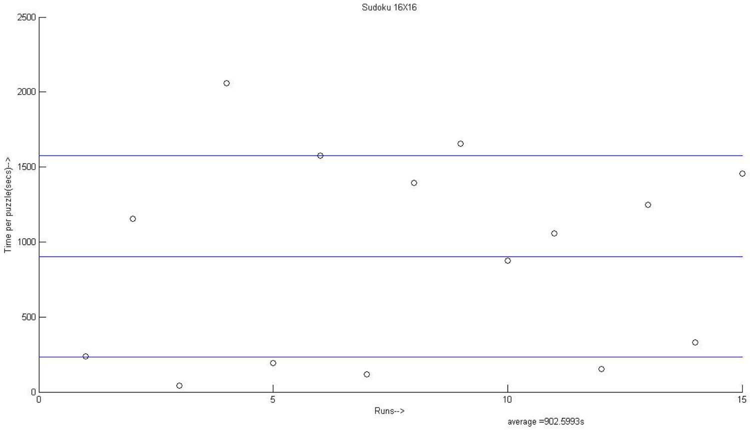
1. typedef struct cluster{
   bool domain[N*N]; domain[i]==true means i is in the domain of this cell
   num value; value is the current value of this cell
   bool given; given==true means that this cell was already given in the original problem
   bool queued; queued==true means that this cell is currently queued
   }cell;
2. cell matrix[N*N] The main array acting as the workplace
3. long long int rwt[N*N][N*N][N*N][N*N]; rwt[r][i][j][v] is the weight of constraint between ith and jth cell in rth row for the value v
   long long int cwt[N*N][N*N][N*N][N*N]; cwt[r][i][j][v] is the weight of constraint between ith and jth cell in rth column for the value v
   long long int bwt[N*N][N*N][N*N][N*N]; bwt[r][i][j][v] is the weight of constraint between ith and jth cell in rth block for the value v
4. int blockno=((int)(i/N))*N + (int)(j/N);
   int ijno=((int)(i%N))*N + (int)(j%N);

Implementation

1. typedef struct cluster{
   bool domain[N*N]; domain[i]==true means i is in the domain of this cell
   num value; value is the current value of this cell
   bool given; given==true means that this cell was already given in the original problem
   bool queued; queued==true means that this cell is currently queued
   }cell;
2. cell matrix[N*N] The main array acting as the workplace
3. long long int rwt[N*N][N*N][N*N][N*N]; rwt[r][i][j][v] is the weight of constraint between ith and jth cell in rth row for the value v
   long long int cwt[N*N][N*N][N*N][N*N]; cwt[r][i][j][v] is the weight of constraint between ith and jth cell in rth column for the value v
   long long int bwt[N*N][N*N][N*N][N*N]; bwt[r][i][j][v] is the weight of constraint between ith and jth cell in rth block for the value v
4. int blockno=((int)(i/N))*N + (int)(j/N);
   int ijno=((int)(i%N))*N + (int)(j%N);


Sudoku 4X4
Average time per puzzle(microsecs)-->
Runs-->
5
10
15
20
0
20
40
60
80
100
120
140
160
180
200
average =0.012169ms


Sudoku 9X9
x 10^5
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
Average time per puzzle(microsecs)-->
0
50
100
150
Runs-->
average =110.1172ms


Sudoku 16X16
2500
2000
1500
1000
500
0
Time per puzzle(secs)-->
0
5
10
15
Runs-->
average =902.5993s

Acknowledgements

- 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.
- Rhydian Lewis, 2007. On the Combination of Constraint Programming and Stochastic Search. The Sudoku CaseProceedings of the 4th international conference on Hybrid metaheuristics.
- Helmut Simonis, 2005, Sudoku as a Constraint Problem, IC-Parc, Imperial College, London