**DIAGONAL SUDOKU SOLVER**

***15BCE0517 M.S.SANJAY***

**Report submitted for the**

**Final Project Review of**

**Course Code: CSE3013 – Artificial intelligence**

**Slot: B2 + TB2**

**Professor: Dr.W.B.Vasantha**

**Abstract:**

In the last decade, solving the Sudoku puzzle has become every one’s passion. The simplicity of puzzle’s structure and the low requirement of mathematical skills caused people to have enormous interest in accepting challenges to solve the puzzle. Therefore, developers have tried to find algorithms in order to generate the variety of puzzles for human players so that they could be even solved by computer programming.

Sudoku is a partially completed 9x9 grid puzzle. The task is to fill the grid with digits such that each row, each column, and each 3x3 block contains digits from 1 to 9 exactly once. We focus on the Sudoku puzzle because it is both constraint satisfaction problems (CSP) in which we’ve learned in this class, and an NP-complete problem for generalized version with NxN grid in which solving the problem using any currently known algorithm for a reasonable amount of time remains undiscovered. The scope of this project is to minimize time to solve 9x9 grid Sudoku puzzle. The success of this program will be evaluated by using time measurement of how long the algorithm, on average, uses to solve significant amount of puzzles.

1. **Introduction**:

Sudoku is a logic-based, combinatorial number placement puzzle game. The objective of this game is to fill a 𝑁 × 𝑁 grid with digits so that every row, column and inner box is filled with number from 1 to 𝑁 without duplicates. The shape of inner boxes may vary, but all must have exactly 𝑁 blocks. In most situations, a Sudoku game should have a unique solution. However, there can be cases that a Sudoku game has multiple solutions. Usually a game with multiple solution is considered to be more difficult than the one that has a unique solution. Sudoku was originated in the late 16th century. La France developed the embryonic form of the modern Sudoku of 9 by 9 grid with 9 3 × 3 inner boxes, and later, it was believed that the modern Sudoku was mostly designed anonymously by Howard Garns in 1979. However, this game became a worldwide phenomenon not until Nikoli introduced it to Japan in 1984.

There are 3 rules in order to complete and accomplish all of the empty cells:

1) The number must only occur in a column once.

2) The number must only occur in a row once.

3) The number must occur in a sub-grid only once.

**2. Literature Review Summary Table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Authors and Year (Reference)* | *Title (Study)* | *Concept / Theoretical model/ Framework* | *Methodology used/ Implementation* | *Dataset details/ Analysis* | *Relevant Finding* | *Limitations/ Future Research/ Gaps identified* |
| *Rohit Iyer, Amrish Jhaveri, Krutika Parab*  *(2013)* | *A Review of Sudoku Solving using Patterns* | *This paper attempts to feed the tips meant for Human Sudoku solving by detecting different patterns, to a machine and also using its power to guess rapidly and backtrack, to gain a remarkable improvement in Sudoku solving than a simple Backtracking approach.* | *A novel technique for very fast Sudoku solving using recognition of various patterns like Naked Singles, Hidden Singles, Locked Candidates, etc. is reviewed by conducting experiments and plotting the observations. Evaluation of the technique in solving random set of Sudoku puzzles collection show that the rate of solving can be greatly improved.* |  | *Sudoku solving using patterns surely lowers the execution time required to solve huge number of Sudoku’s.* | *The motive of this paper was to review the performance of a brute-forced based Sudoku Solver using pattern matching. This review used limited number of patterns that were comparatively easier to detect than some complex patterns. However, detecting more patterns may give considerably better results.* |
| *Jeﬀrey O. Pfaﬀmann and William J. Collins*  *(2007)* | *Teaching Artiﬁcial Intelligence Across the Computer Science Curriculum using Sudoku as a Problem Domain* | *framework constraining, there was not much recourse: An ad-hoc solution without backtracking would not have met the goal of the project, would not have allowed the solution of “diabolical” Sudoku puzzles, and would have either required a substantial knowledge of Sudoku or tempted students to scour the Web for an already existing program.* | *Sudoku as a problem domain for a constraint satisfaction project where the students can have a real research experience. They were required to create a puzzle solver using a variety of constraint satisfaction techniques and a puzzle generator using a technique of their own determination. Then using the solver and generator results.* |  | *Paper illustrates that Sudoku is an excellent problem domain choice for teaching AI approaches in both introductory- and upper-level computer science courses.* | *Analysing the performance measures of different algorithms in further research can be explored.* |
| *Lekha.K, Vadivu.G*  *(2010)* | *PERFORMANCE ANALYSIS OF SUDOKU PUZZLE SOLVER* | *They have presented an algorithm called SPS (Sudoku Puzzle Solver). The purpose of this paper is to implement a more efficient algorithm SPS, and compare it with other Sudoku solver algorithms named as brute force algorithm and backtracking.* | *Performance measures of SPS algorithm and Brute force and back tracking algorithms are compared and their implementations are done for solving the Sudoku puzzle.*  *This SPS algorithm is a general purpose algorithm that can be employed in to any level of missing values. The results have proved that the SPS algorithm solves the puzzle faster and more effective than the brute force algorithm and backtracking algorithm. The performance is analysed based on the number of clock ticks used to execute the algorithms.* |  | *This paper shows that SPS algorithm is more appropriate than backtracking algorithm and brute force algorithm, to solve any Sudoku puzzles.* | *Future work includes study of number of clues and run time of each difficulty level and comparing Brute Force and Backtracking algorithm based on each difficulty level.* |
| *Tristan Cazenave*  *(2000)* | *A search based Sudoku solver* | *They detail constraint satisfaction search algorithms used to solve Sudoku problems. They showed that a good value ordering heuristic helps solving problems. They also show that Limited Discrepancy search is a good alternative to the traditional Forward Checking algorithm. They also found a phase transition for 25x25 Sudoku.* | *There is a clear phase transition for Sudoku problems of size 25x25. – The min-domain-sum value ordering heuristic works better than the lexicographic ordering of values. – Limited Discrepancy Search is a good alternative to the Forward Checking algorithm for Sudoku.* |  | *They have shown a phase transition for 25x25 Sudoku at 51 % of holes. They have also shown that the min-domain-sum value ordering heuristic and that Limited Discrepancy Search are useful for solving Sudoku problems.* | *Future work include search algorithm to generate Sudoku problems with a unique solution, ﬁnding the minimum number of pre-assigned variables necessary to have unique solutions, and improving the search algorithm.* |
| *Ricardo Soto, Broderick Crawford, Cristian Galleguillos, Fernando Paredes, and Enrique Norero*  *(2015)* | *A Hybrid* all different *-Tabu Search Algorithm for Solving Sudoku Puzzles* | *The work of Tabu algorithm is illustrated here and the performance measures are compared with respect to other algorithms and the efficient algorithm can be known.* | *They introduced a new hybrid that smartly integrates a global constraint, namely, the* all different *constraint, in a classic tabu search procedure. The* all different *constraint comes from the constraint* *programming world and has specially been designed for the efficient domain reduction of variables involved in constraints that must be pairwise different. This global constraint works perfectly for Sudokus since all the puzzle constraints can be expressed as a pairwise comparison. We implement the* all different *constraint following Puget’s approach, which identifies Hall intervals and then filters the domains.* |  | *They illustrate interesting experimental results where our proposed algorithm out performs the best results previously reported by hybrids and approximate methods.* | *interesting research direction will be the study of variations of the classic* all different *constraint working in conjunction with approximate methods. An example is the symm* all different *constraint ,which will be useful in the resolution of the well-known round-robin tournament.* |
| *Zhe Chen*  *(2009)* | *Heuristic Reasoning on Graph and Game Complexity of Sudoku* | *Experimental results show that all the puzzles can be solved fast using the proposed heuristic reasoning, and that the proposed game complexity metrics can discriminate difﬁculty levels of puzzles perfectly.* | *They proposed to focus on the essential graph structure underlying the Sudoku puzzle. First, They formalized Sudoku as a graph. Then a solving algorithm based on heuristic reasoning on the graph is proposed. The related r-Reduction theorem, inference theorem and their properties are proved, providing the formal basis for developments of Sudoku solving systems. In order to evaluate the difﬁculty levels of puzzles, a quantitative measurement of the complexity level of Sudoku puzzles based on the graph structure and information theory is proposed.* |  | *They provided a formal basis for studies and developments of Sudoku solving systems and complexity evaluation.* | *To ﬁnd a stronger rule by extending r-Reduction, in order to solve all the puzzles using a single rule.* |
| *Manav B. Sanghavi, Aniket K. Rupani, Mahek S. Maniar, Sai Deepthi Pabba*  *(2015)* | *Solving Sudoku from an Image using Modular Architecture Approach* | *They have proposed a modular architecture for solving a Sudoku puzzle from an image. The advantage of following this architecture is its modularity: if a module’s implementation needs to be improved, it can be changed relatively easily, as long as the input to the module and output from the module remains unchanged. Such an architectural model would be useful for developing applications which work on Sudoku puzzles, as the modular nature allows for quickly customizing the nature of the output supplied by simply changing parameters or modules of the architecture.* | *Proposes a method for extracting and solving a Sudoku puzzle captured in an image. AI techniques can then be applied to solve the Sudoku puzzle. A modular architecture is created for this purpose. Modules can be replaced as needed, making it easier to improve and maintain an application using the proposed architecture* |  |  | *To find a concrete and efficient implementation of the proposed architecture as well as apply and compare various Sudoku solving techniques to find an optimal Sudoku solver for the proposed architecture.* |
| *Uwe Pfeiﬀer, Tomas Karnagel and Guido Scheﬄer*  *(2005)* | *A Sudoku-Solver for Large Puzzles using SAT* | *They present some ideas for reducing the number of clauses, for improving the encoding, and for the selection of suitable SAT solvers.* | *The aim of our work was to build a Sudoku solver for participating in a local competition at the TUD where Sudoku puzzles with 3 ≤ n ≤ 15 had to be solved using SAT solvers. The goal of the competition was to build a correct system, which should solve as many Sudoku’s as fast as possible given a certain time-out per puzzle. This paper presents the architecture of our solver and the decisions that were made during development.* |  | *They found the encoding of quite appropriate for Sudoku puzzles. Pre-processing had a major impact and should be investigated further. We would have a better understanding to what extends the ordering of the input clauses inﬂuences the performance.* | *We would also like to rank puzzles according to their diﬃculty, but we don’t know how to measure the diﬃculty. We would also be interested in puzzles with n > 15.*  *Further research can be done by using all these constraints.* |
| *Urszula Boryczka, Przemysław Juszczuk*  *(2012)* | *SOLVING THE SUDOKU WITH THE DIFFERENTIAL EVOLUTION* | *To test if the Differential Evolution is an efﬁcient methodforsolvingcombinatorialproblems-especiallytheSudokupuzzle.Proposed algorithm will be evaluated on different sets of example problems and different difﬁculty puzzle levels.* | *In this paper, they present the application of the Differential Evolution (DE) algorithm to solving the combinatorial problem. The advantage of the DE algorithm is its capability of avoiding so-called “local minima” within the considered search space. The DE algorithm applies the selection operator that selects from the child population only the offspring with the greater value of the ﬁtness function in comparison to their parents. An algorithm applied to a combinatorial optimization problem: Sudoku puzzle is presented. Sudoku consists of a nine by nine grid, divided into nine three by three boxes. Each of the eighty-one squares should be ﬁlled in with a number between one and nine.* |  | *Mutation schema has signiﬁcant impact on the quality of created solution.* | *Next goal is to adapt similar mutation into the continuous optimization problem.* |
| *Seyed Mehran Kazemi, Bahare Fatemi*  *(2014)* | *A Retrievable Genetic Algorithm for Efficient Solving of Sudoku Puzzles* | *They examined the problem of proposing an algorithm to solve Sudoku puzzles. They introduced genetic algorithms and then propose a way of applying this algorithm to the problem of solving Sudoku puzzles efficiently. They evaluate out model on Sudoku puzzles having different difficulties and compare our results with another GA-based method.* | *Several methods and algorithms have been proposed and used in different software’s to efficiently solve Sudoku puzzles. Various search methods such as stochastic local search have been applied to this problem. Genetic Algorithm (GA) is one of the algorithms which have been applied to this problem. In these works, chromosomes with little or no information were considered and obtained results were not promising. In this paper, we propose a new way of applying GA to this problem which uses more-informed chromosomes than other works. Then they used the optimized values of the parameters to solve various puzzles and compare our results to another GA-based method for solving Sudoku puzzles.* |  | *Various methods and algorithms have been proposed to solve Sudoku puzzles. One of these methods is to formulate the problem as a genetic algorithm and try to solve it. Previous attempts of using genetic algorithm seem inefficient because of including only little information in their chromosomes.* | *We can use a different objective function in our parameter optimization which also includes the number of generations it takes to solve the puzzle. We can do the same experiments to see if it can improve our proposed method in terms of the average number of generations for solving the puzzles or not. We may also consider other criterion such as median or standard deviation of they are of high importance to us.* |

**3. Objective of the project:**

To create a Python program that takes in an incomplete Sudoku grid and returns the same grid with all the completed values. The project uses a search algorithm (Constraint Satisfaction Problem (CSP) to solve a puzzle, and return the puzzle solution.

Constraint satisfaction problems (CSPs) are mathematical problems defined as a set of objects whose [state](https://en.wikipedia.org/wiki/State_(computer_science)) must satisfy a number of [constraints](https://en.wikipedia.org/wiki/Constraint_(mathematics)) or [limitations](https://en.wikipedia.org/wiki/Limit_(mathematics)). CSPs represent the entities in a problem as a homogeneous collection of finite constraints over [variables](https://en.wikipedia.org/wiki/Variable_(mathematics)), which is solved by [constraint satisfaction](https://en.wikipedia.org/wiki/Constraint_satisfaction) methods. CSPs are the subject of intense research in both [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence) and [operations research](https://en.wikipedia.org/wiki/Operations_research), since the regularity in their formulation provides a common basis to analyse and solve problems of many seemingly unrelated families. [CSPs often exhibit high complexity](https://en.wikipedia.org/wiki/Complexity_of_constraint_satisfaction), requiring a combination of [heuristics](https://en.wikipedia.org/wiki/Heuristics) and [combinatorial search](https://en.wikipedia.org/wiki/Combinatorial_search) methods to be solved in a reasonable time. The [Boolean satisfiability problem](https://en.wikipedia.org/wiki/Boolean_satisfiability_problem) (SAT), the [satisfiability modulo theories](https://en.wikipedia.org/wiki/Satisfiability_modulo_theories" \o "Satisfiability modulo theories) (SMT) and [answer set programming](https://en.wikipedia.org/wiki/Answer_set_programming) (ASP) can be roughly thought of as certain forms of the constraint satisfaction problem

**4. Innovation component in the project:**

The general rule to solve Sudoku is:

Each of nine block has to contain all numbers from 1 to 9 within its square. Also each number can appear once in a row, column or a box.

In addition to this rule, we add a diagonal constraint that each digit should appear only once in a diagonal.

**5. Work done and implementation**

For the first model, we cast the Sudoku puzzle as a constraint satisfaction problem

• The variables are each cell on the grid, so there are 81 variables: Xij = the cell value at row i and column j, A ≤ i ≤ I, 1 ≤ j ≤ 9.

• The domain of each cell is 1, 2, 3... 9.

• The constraints are:

1. Unary Constraint: m constraints, f(Xij ) : xij = initially assigned value, where m is number of initially assigned cells

2. Binary Constraint: 810 constraints, f(Xij , Yij ): return x ! = y, for – Digits in the same row have to be distinct (8 constraints per tile) – Digits in the same column have to be distinct (8 constraints per tile) – Digits in the same block have to be distinct (4 more constraints per tile) There are 20 constraints per tile. Therefore, there are 20 \* 81 / 2 = 810 constraints in total where factor of 1/2 accounts for double counting.

CODE:

def solvedvalnum(array):

return len([pt for pt in array.keys() if len(array[pt]) == 1])

def valgrid(grid):

array = dict(zip(boxes, grid))

empty\_box\_value = 'asdfghjkl'

for pt, val in array.items():

if(val == '.'):

enumval(array, pt, empty\_box\_value)

return array

def completelysolved(array):

return all(len(array[pt]) == 1 for pt in array.keys())

def simplifysudoku(array):

stalled = False

while not stalled:

solved\_values\_before = solvedvalnum(array)

array = eliminate(array)

array = nooption(array)

array = strategynakedtwin(array)

solved\_values\_after = solvedvalnum(array)

stalled = solved\_values\_before == solved\_values\_after

if len([pt for pt in array.keys() if len(array[pt]) == 0]):

return False

return array

filled = []

def ddisp(array):

width = 1+max(len(array[s]) for s in boxes)

border = '+'.join(['-'\*(width\*3)]\*3)

for r in horizontal:

print(''.join(array[r+c].center(width)+('|' if c in '36' else '')

for c in vertical))

if r in 'CF': print(border)

return

horizontal = 'qwertyuio'

def chknk(value\_list, tuple):

return value\_list.count(tuple) > 1 or value\_list.count(strrev(tuple)) > 1

vertical = 'asdfghjkl'

def mate(w, x):

return [p+q for p in w for q in x]

def strrev(string):

return string[::-1]

boxes = mate(horizontal, vertical)

ppoint = [mate(rs, cs) for rs in ('qwe','rty','uio') for cs in ('asd','fgh','jkl')]

def search(array):

reduced\_values = simplifysudoku(array)

if reduced\_values is False:

return False

if completelysolved(reduced\_values):

return reduced\_values

unfilled\_box = selectempty(reduced\_values)

for val in reduced\_values[unfilled\_box]:

new\_values = reduced\_values.copy()

new\_values[unfilled\_box] = val

solution = search(new\_values)

if solution:

return solution

diagright = [sum([mate(r, str(i + 1)) for i, r in enumerate(horizontal)], [])]

hori\_val = [mate(r, vertical) for r in horizontal]

def valeliminate(curr\_value, value\_to\_eliminate):

return curr\_value.replace(value\_to\_eliminate, '');

vert\_val = [mate(horizontal, c) for c in vertical]

diagleft = [sum([mate(r, str(i + 1)) for i, r in enumerate(strrev(horizontal))], [])]

def strategynakedtwin(array):

for unit in unitlist:

strategynakedtwin = nkfind(array, unit)

if len(strategynakedtwin) > 0:

for pt in unit:

array = disposenk(array, unit, strategynakedtwin)

return array

unitlist = hori\_val + vert\_val + ppoint + diagright + diagleft

def enumval(array, pt, val):

array[pt] = val

if len(val) == 1:

filled.append(array.copy())

return array

def selectempty(array):

return min((len(array[pt]), pt) for pt in boxes if len(array[pt]) > 1)[1]

boxes = dict((s, [u for u in unitlist if s in u]) for s in boxes)

neighbour = dict((s, set(sum(boxes[s],[]))-set([s])) for s in boxes)

def nkfind(array, unit):

tuples = {}

for pt in unit:

if len(array[pt]) == 2:

tuples[pt] = array[pt]

tuple\_values = list(tuples.array())

strategynakedtwin = [tuple for tuple in tuple\_values if chknk(tuple\_values, tuple)]

return ''.join(set(strategynakedtwin))

def disposenk(array, unit, strategynakedtwin):

for pt in unit:

if len(array[pt]) > 2:

for naked\_twin in strategynakedtwin:

array = enumval(array, pt, valeliminate(array[pt], naked\_twin))

return array

def findsudoku(grid):

return search(valgrid(grid))

def eliminate(array):

solved\_values = [pt for pt in array.keys() if len(array[pt]) == 1]

for pt in solved\_values:

value\_to\_eliminate = array[pt]

for peer in neighbour[pt]:

enumval(array, peer, valeliminate(array[peer], value\_to\_eliminate))

return array

def nooption(array):

for unit in unitlist:

for digit in '123456789':

dplaces = [pt for pt in unit if digit in array[pt]]

if len(dplaces) == 1:

enumval(array, dplaces[0], digit)

return array

if \_\_name\_\_ == '\_\_main\_\_':

diag\_sudoku\_grid = '....12.3......97....9....7...4..2...1...5...4...6..1...7.1..2..7.5...1.9....7..2.'

ddisp(findsudoku(diag\_sudoku\_grid))

from visualize import visualize\_assignments

visualize\_assignments(filled)

**Methodology to be adapted:**

Methodology:

Unique missing candidate: The unique missing candidate is used when any row, column or box is missing only one single digit.

Naked Singles: This method is useful when we find a square that can only take one single value, once the contents of other squares in the same row, column and box are considered.

Hidden Singles: The hidden single method is similar to the naked single method but the way to find the way to find the empty square is different. When there is only one square in the row, column or box that can take a certain number, then the square must take that number.

Locked candidate: If a candidate belongs to a row or column then we can remove this candidate as a possible one with other boxes that the row (or column) connected with them

Naked Pairs, Triplets: These techniques are very similar to the naked single technique, but in this method we find the same two candidates in two squares. By using this information we can find a possible candidate to other squares

MINIMUM REQUIREMENT:

Processor: Pentium4/1.2Ghz Celeron/duron processor

CPU speed: 1.4 ghz

RAM: 512 MB

Hard disk: 80 GB.

OS: Any windows version

Language:Python 3.4 and above

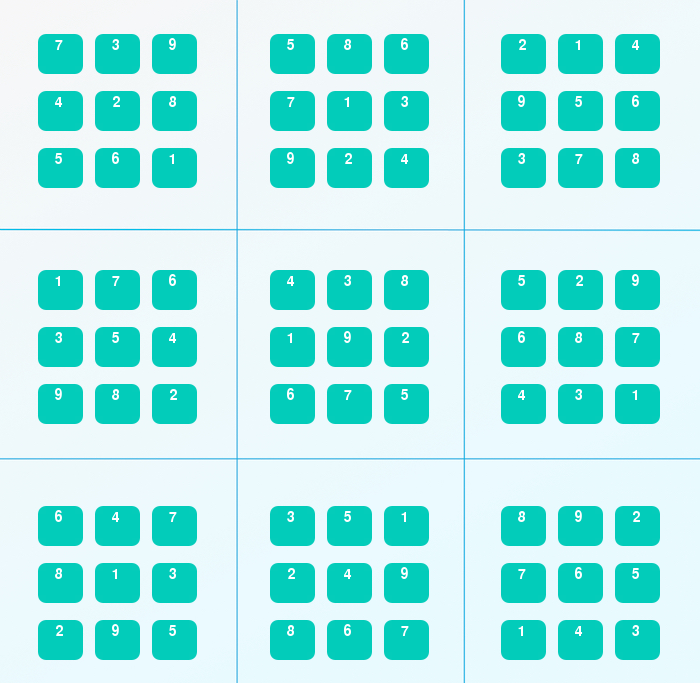
**Dataset used:**

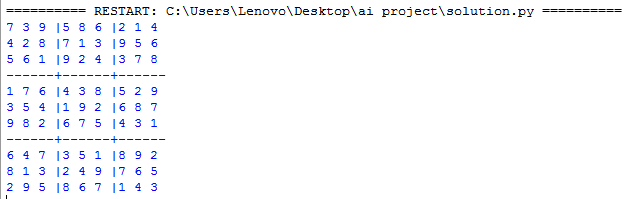
We have no need to use any external dataset . We are also not referring any external project on internet.We plan to solve for diagonal values of Sudoku also in addition to horizontal and vertical values.

**Tools used:**

**Python 3.4**

**6. Screenshot and demo**





**7.Results and discussions:**

Constraint propagation is the idea of applying the same constraint as many times as possible until a solution is obtained, or the constraint can no longer be applied to refine the solution.

In sudoku, we apply naked twins as a strategy to reduce the number of possibilities. Naked twins is the strategy is illustrated in the figure below. The strategy is to identify a pair of boxes belonging to the same set of peers that have the same 2 numbers as possibilities, and eleminate these two numbers from all the boxes that have these two boxes as peers.

This method was implemented in Python using a set intersection function. First we identify all boxes that have only 2 elements. Next we identify which boxes among these have the same elements to get naked twins. Once we get the naked twins, we remove the corresponding digits from all the boxes that are peers to both the twins.

By using these strategies the Sudoku was solved.

**8. References**

1. I. L. Ist, I. Lynce, and J. Ouaknine. Sudoku as a sat problem. In Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics, AIMATH 2006, Fort Lauderdale, pages 1–9. Springer, Jan. 2006.

2. Sato, Yuji “Solving Sudoku with Genetic Operations that Preserve Building Blocks,” Computational Intelligence and Games (CIG), 2010 IEEE Symposium.

3. Mantere, T., Koljonen, J.: Solving, rating and generating Sudoku puzzles with GA, Proc. of IEEE Congress on Evolutionary Computation (CEC 2008), IEEE, New York, pp. 1382-1389 (2007).

4. Mandal, SatyendraNath., Sadhu, Soumi.: An Efficient Approach to Solve Sudoku Problem by Harmony Search Algorithm, An International Journal of Engineering Sciences ISSN: 2229-6913 Issue September 2011, Vol. 4, pp. 312-323 (2011)

5. Volume 6146, 2010, pp 461-467. Sudoku Using Parallel Simulated Annealing, Zahra Karimi-Dehkordi, Kamran Zamanifar, Ahmad Baraani-Dastjerdi, Nasser Ghasem-Aghaee, First International Conference, ICSI 2010, Beijing, China, Proceedings, Part II, June 12-15, (2010).

6. Mantere, T., Koljonen, J.: Solving and analyzingSudokus with cultural algorithms, Proc. of IEEE Congress on Evolutionary Computation (CEC 2008), IEEE, New York, pp. 4053-4060 (2008).

7. Pereira, M. R., Vargas, P. K., França, F. M., de Castro, M. C. S., & de Castro Dutra, I. Applying Scheduling by Edge Reversal to constraint partitioning. In Computer Architecture and High Performance Computing, 2003. Proceedings. 15th Symposium on (pp. 134-141). IEEE, (2003, November).