

Hill Climbing Search for Sudoku Puzzle

By Niraj Kumar
BT19CS031

Hill Climbing

Given the size of the state space it is logical to use a search method with a heuristic function to avoid searching less promising sections of the state space . One type algorithm that meets this criteria is hill climbing .

Hill climbing algorithms work by generating a list of successors of the current state, then choosing the one with the lowest heuristic value. In order to apply hill climbing to Sudoku three things must be defined: the start state, the successor function and the heuristic function. One way to go about this is to fill in each box so that it contains the numbers one to n^2 and allow successors to be generated by switching values within the same box (Lewis 391). Instead of filling in the boxes with the numbers one to n^2 the rows will be filled, but the idea is the same.

Let the Start state be defined as the initially puzzle with all of the empty spaces filled in such that each row contains the numbers one to n^2 .

Using this as a start state, the successor function can be defined as swapping any two nonfixed values in the same row. The heuristic can simply be the sum of the number of conflicts that appear within each column and each box. Since each row has exactly the numbers one to n^2 there are no conflicts within the rows. Hill climbing can successfully solve size two puzzles occasionally

Random Restart

The general hill climbing algorithm described above is incomplete. This is because it can get stuck in a local minimum.

One simple way to fix this is to randomly restart the algorithm whenever it goes a while without improving the heuristic value. This is known as random restart hill climbing (Russell and Norvig 114).

This version of hill climbing does not quite suffice to solve the puzzle. The problem comes from the completely random way in which the variables are filled in. One technique for dealing with this is to switch certain values after the initial random placement. Known as post swapping this technique attempts to minimize the number of values that conflict with the values given in the original puzzle (Jones et al., 45).

Another possible solution is to use the constraint propagation algorithm. Applying constraint propagation before filling in any random values prevents some values from being randomly guessed incorrectly.

Also, constraint propagation can be applied in between the assignments of each row to make sure that the future rows agree better with the rows that have already been assigned. This method allows most puzzles of size three to be solved.

Puzzle Difficulties

Puzzle level	Average Number of Givens	Average Number of Nodes Searched			
		BTS	BTS+CP	BTS+CP +MRV	Random Restart Hill Climbing
1	28	4866	3	2	5579.6
4	27	20257	192	137	240340.5
8	25	19963	89	71	-

The chart above indicates the average performance of the various algorithms described on size three puzzles of varying difficulties. Clearly the backtracking search method has an advantage over the hill climbing method.

It should be noted that the level one puzzles could be almost completely solved by the constraint propagation algorithm and required little actual search.

The search algorithm is expensive and avoiding it entirely is very valuable in those cases where it is possible.

It should also be noted that the logic used by the constraint propagation algorithm is very simple and its ability to solve the easy puzzles means that they do not require advanced logic. The graph below shows the number of nodes searched by a backtracking search without constraint propagation or minimum remaining values as a function of the number of givens.

VI. STOCHASTIC HILL CLIMBING

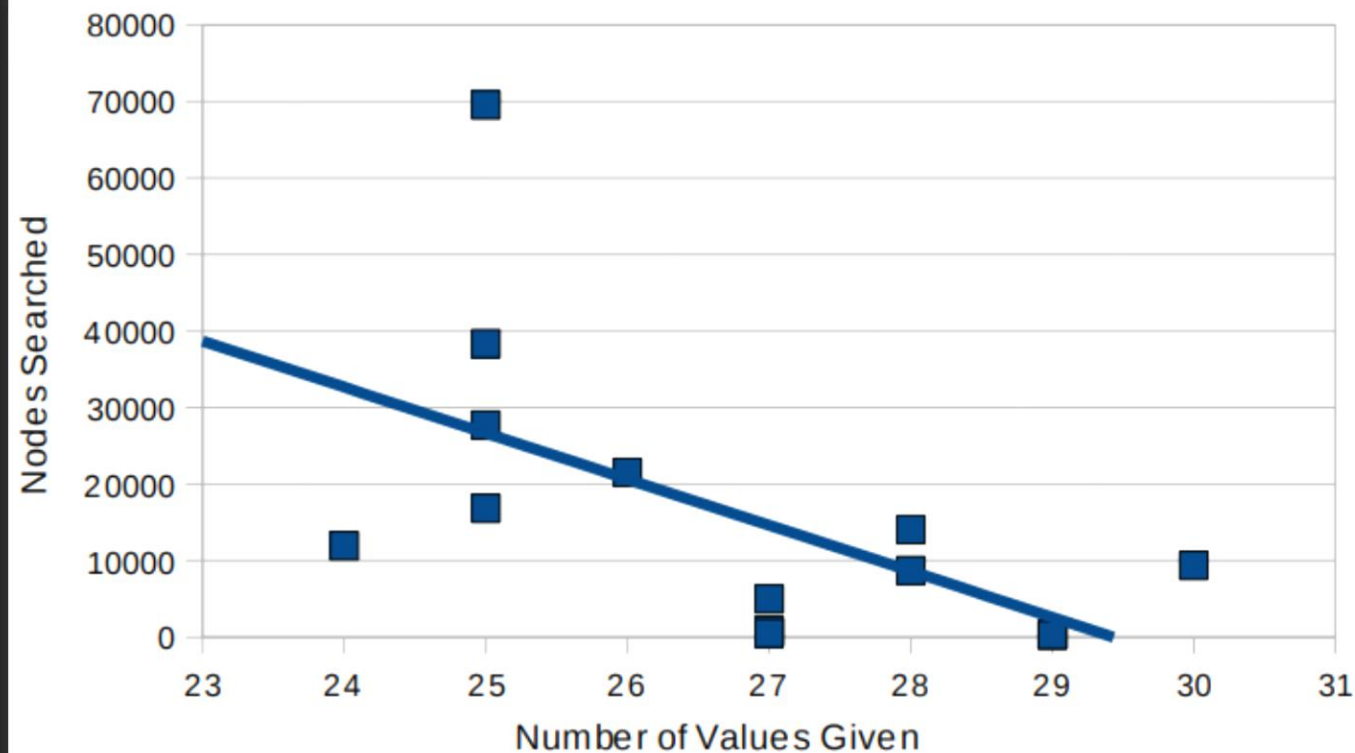
A. Pseudocode

```
X = Fill the empty cells randomly
flag = true
while flag is true
    flag = false
    for i=0 to 8
        for j= 0 to 8
            i= random btw 0 to 8
            j= random btw 0 to 8
            for k = 1 to 9
                oldScore = evaluate(X)
                temp = X[i][j]
                X[i][j] = k
                newScore = evaluate(X)
                if oldScore > newScore
                    X[i][j] = temp
            else
                flag = true
        end second for
    end first for
end while
```

Structure of Algorithm In this algorithm

We first randomly filled the empty cells of the Sudoku problem. Then until no local optimization is possible, we randomly selected a cell and tried to insert values between 1 to 9 to that cell. During the insertion step, we considered the improvement at the end of that insertion

In order to make comparison between different algorithms and studying their effectiveness on solving the Sudoku problems visually and numerically, a Graphical User Interface (GUI) is built in Java language. This GUI enables the user to run any algorithm we implemented. It is also possible that the level of difficulty puzzle be chosen. In addition, the designed GUI is constructed to receive input from the user to tweak the setting of bees population



The graph definitely indicates a correlation between the number values given in the puzzle and the time required to solve it. Puzzles with more values given can be solved after a shorter search.

Conclusion

Although the Sudoku problem is a difficult constraint satisfaction problem, it is not completely invulnerable to search methods. Puzzles of size three or less can be solved very quickly and consistently.

The backtracking search can consistently solve size three Sudoku puzzles after considering fewer than 200 states. Considering that there are 6,670,903,752,021,072,936,960 valid Sudoku puzzles, searching only 200 of them to find a solution is excellent.

Random restart hill climbing is also successful on easy puzzles of size three, but is unable to solve more difficult puzzles.

β -HILL CLIMBING ALGORITHM FOR SUDOKU PUZZLE

β -Hill Climbing is the recent metaheuristic algorithm. It is an extended version of hill climbing (i.e., the base version of the local search algorithm). In this section, β -Hill Climbing is pseudo-coded in Algorithm 1 and the flowchart that shows how β -Hill Climbing algorithm is processed through the search is given in Fig. 1.

As shown in β -Hill Climbing pseudo-code at Algorithm 1 Step1, the search is initiated with a totally random solution. Initially, the algorithm generates integer numbers in the range of 1 - 9 and fulfills the empty cells by these generated digits. Note that the initial solution may have replicate digits in each row, columns and block. However, the given digits that are predefined in advance cannot be changed or moved.

The initial solution is evaluated using the objective function discussed in Eq. (2). Thereafter, the algorithm will process the improvement loop.

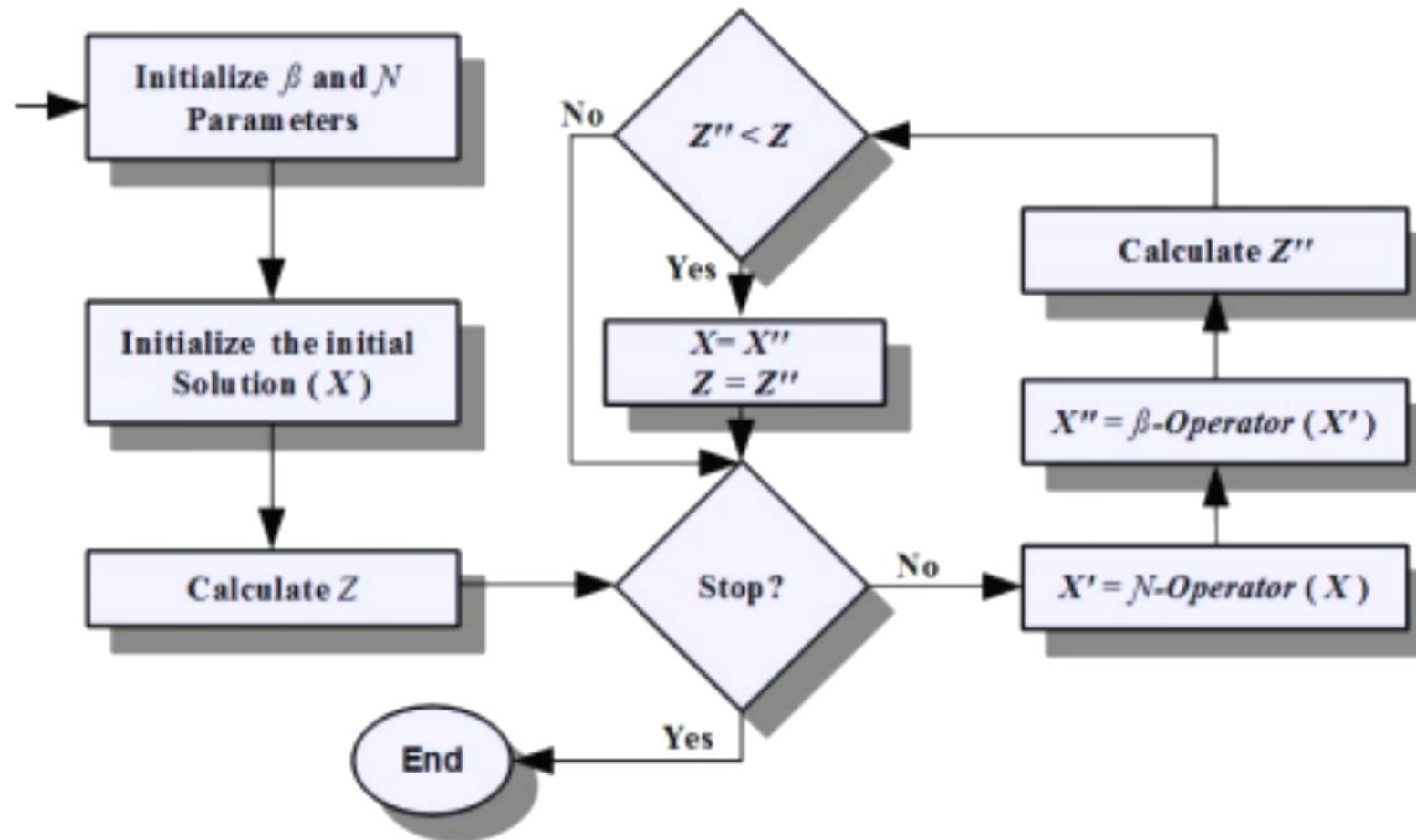


Fig. 1. β -Hill Climbing algorithm flowchart.

Fig. 1. β -Hill Climbing algorithm flowchart.

Algorithm 1 β -hill climbing pseudo-code

```
1.  $x_{ij} = \begin{cases} \in \{1 \dots 9\}, & x_{ij} = 0 \\ x_{ij}, & x_{ij} > 0 \end{cases}$       {Initialize  $X$ }  
2. Calculate  $Z$   
3.  $itr = 0$   
4. while ( $itr \leq Max\_Iterations$ ) do  
5.    if ( $rnd \leq \mathcal{N}$ ) then  
6.      $X' = \mathcal{N}$ -operator ( $X$ )  
7.     if ( $rnd \leq \beta$ ) then  
8.       $X'' = \beta$ --operator ( $X'$ )  
9.     end if  
10.  end if  
11.  Calculate  $Z''$   
12.  if ( $Z'' < Z$ ) then  
13.     $X = X''$   
14.     $Z = Z''$   
15.  end if  
16.   $itr = itr + 1$   
17. end while
```

11' $rnd = rnd + 1$

12' $itr = itr + 1$

13' $itr = itr + 1$

EXPERIMENTAL RESULTS

In this section, the performance of the β -Hill Climbing algorithm for Sudoku puzzle is experimentally evaluated.

Fig. 2 shows the benchmark of the Sudoku which is taken from [18]. Again all cases of Sudoku puzzle have a unique solution.

	5		3		6			7
				8	5		2	4
	9	8	4	2		6		3
9		1			3	2		6
	3						1	
5		7	2	6		9		8
4		5		9		3	8	
	1		5	7				2
8			1		4		7	

Fig. 2. Example of easy sudoku puzzle problem.

The two parameters of the proposed algorithm are studied using various values as shown in Table I. This table includes the values of the two parameters (i.e., β , and N), the minimum time consumed to solve the problem, the minimum number of iterations to reach the solution, and rate of success to solve the Sudoku game over 10 runs.

TABLE I. RESULTS OF β -HILL CLIMBING WITH ITS COLLABORATIVE PARAMETERS

Scenario	Parameter values		Results		
	N	β	Iterations	Time(s)	Success Rate (%)
Sen#1	0.01	0.5	19	2	100
Sen#2	0.1		35	2	100
Sen#3	0.3		25	2	100
Sen#4	0.5		38	6	100
Sen#5	0.7		155	3	100
Sen#6	0.9		1289	10	90
Sen#7	0.3	0.01	5299	39	10
Sen#8		0.1	91	9	40
Sen#9		0.3	14	4	100
Sen#10		0.5	25	2	100
Sen#11		0.7	145	6	100
Sen#12		0.9	163	3	100

Sen#13	0.01	103	3	100
Sen#11	0.1	142	6	100

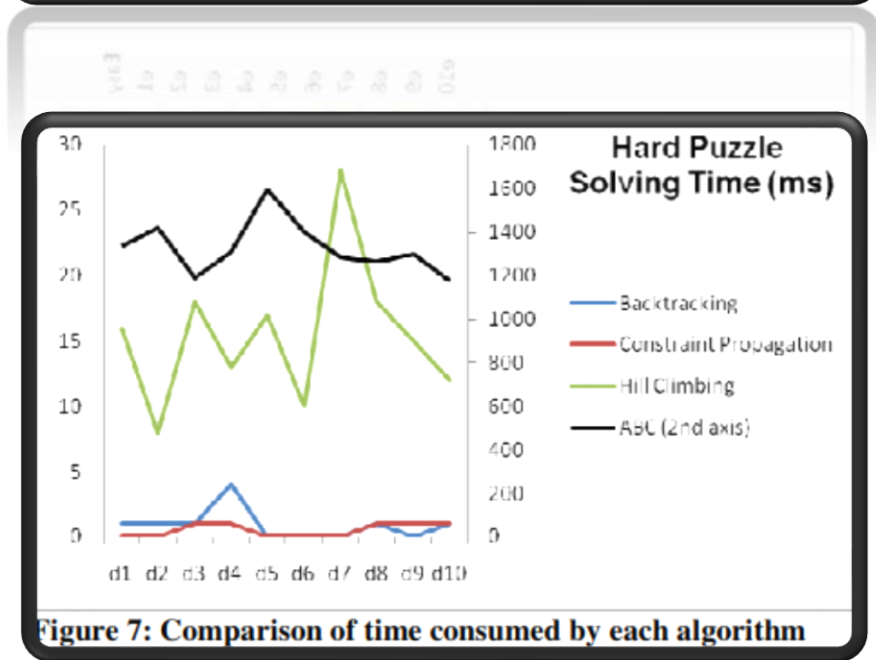
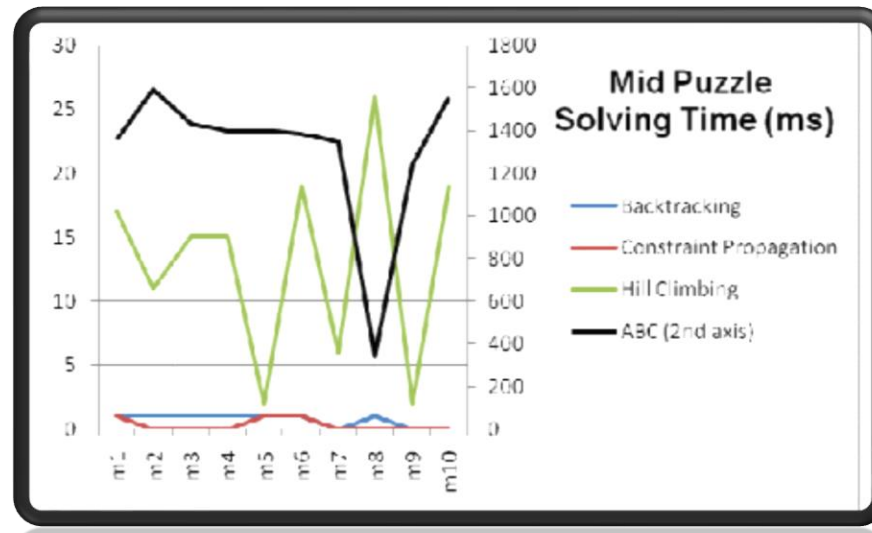
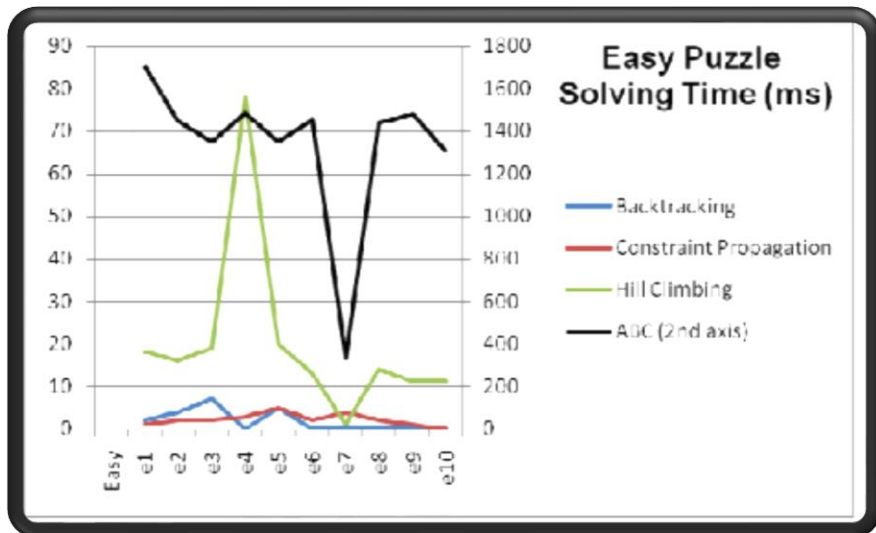
The α parameter is studied with various values in six experimental scenarios (i.e., Sen#1 - Sen#6). The value of the β parameter is fixed to 0.5. The higher value of α parameter leads to higher rate of exploitation. It can be observed from the results, the α parameter with lower values leads to better results, where the first three scenarios solve the game within two seconds and minimum number of iterations

Similarity, the β parameter is studied with different values using six experimental scenarios (i.e., Sen#7 - Sen#12). It should be noted that the higher the value of β parameter the higher of exploration will be .

Apparently, the experimental scenario (Sen#10) solves the game within two seconds and with the minimum number of iterations. Fig. 3 provides a solution for Sudoku puzzle given in Fig 2.

2	5	4	3	1	6	8	9	7
7	6	3	9	8	5	1	2	4
1	9	8	4	2	7	6	5	3
9	8	1	7	5	3	2	4	6
6	3	2	8	4	9	7	1	5
5	4	7	2	6	1	9	3	8
4	7	5	6	9	2	3	8	1
3	1	9	5	7	8	4	6	2
8	2	6	1	3	4	5	7	9

Fig. 3. A solution for Sudoku puzzle given in Fig. 2.



Following figures show the value of time performance and memory amount needed by each algorithm (Backtracking, Constraint Propagation, Hill Climbing and ABC algorithms) to carry out the task in different set of environment, i.e. three different levels of difficulty.

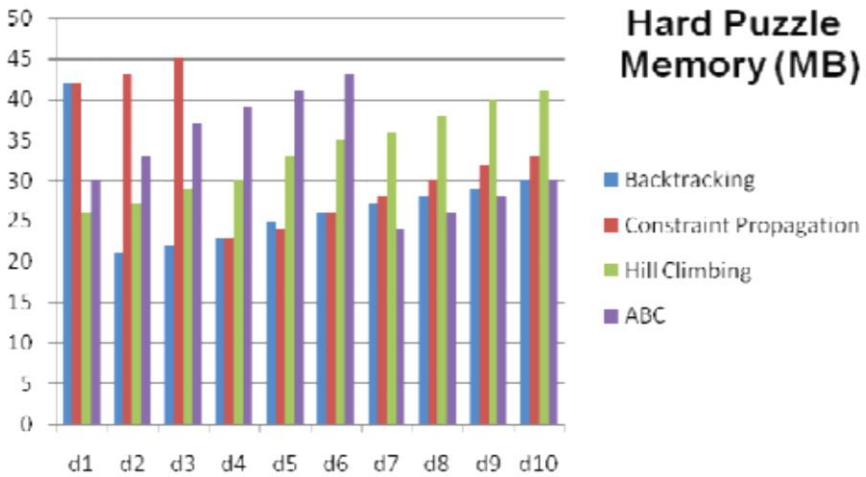
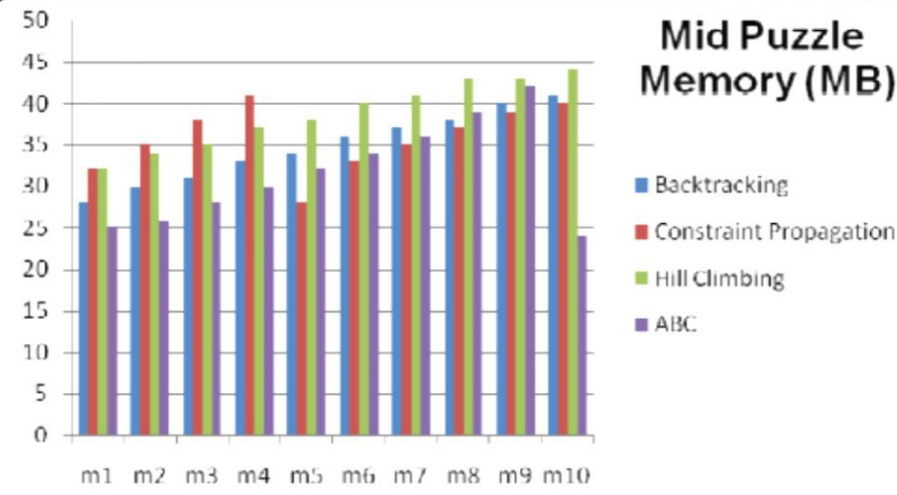
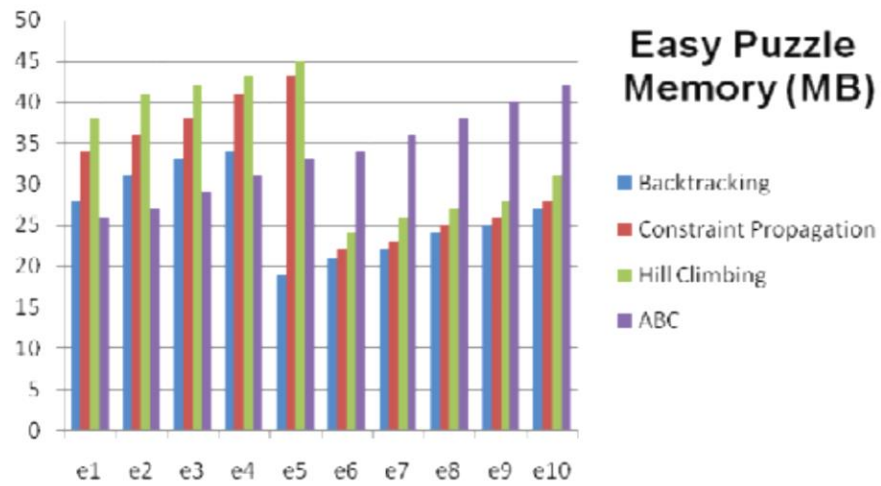


Figure 8: Comparison of memory consumed by each algorithm

The facts shown above tell us that the most time demanding algorithm is the ABC, on the other hand, the least time-demanding algorithm is the Constraint Propagation

On the other hand, Constraint Propagation can be said to be the most memory demanding algorithm. As it requires extra memory for domain values in the heap and for recursive function calls in the stack, this is expected

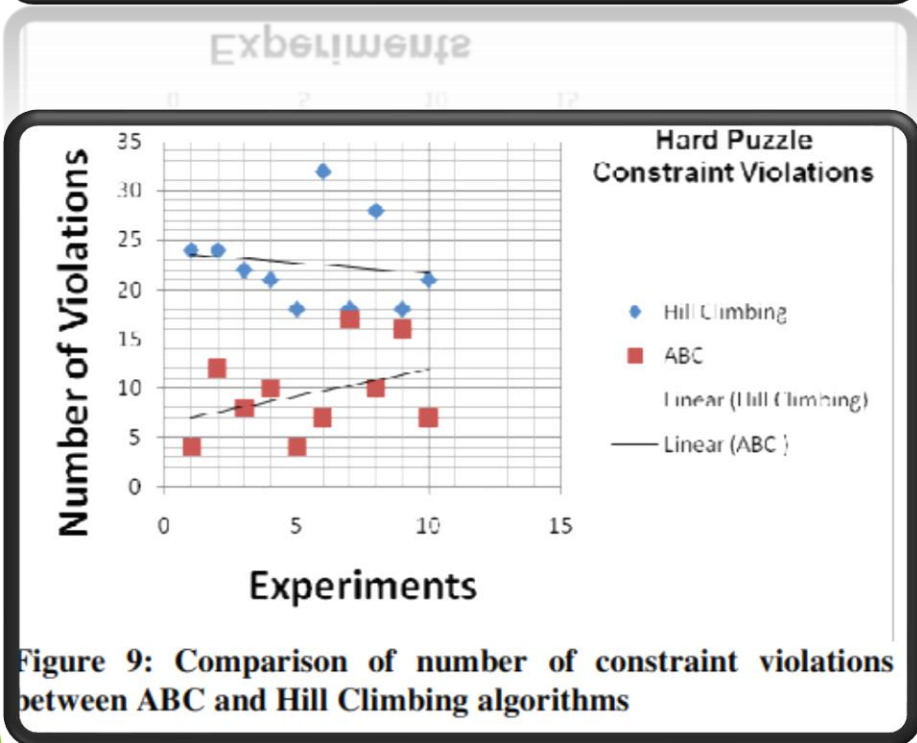
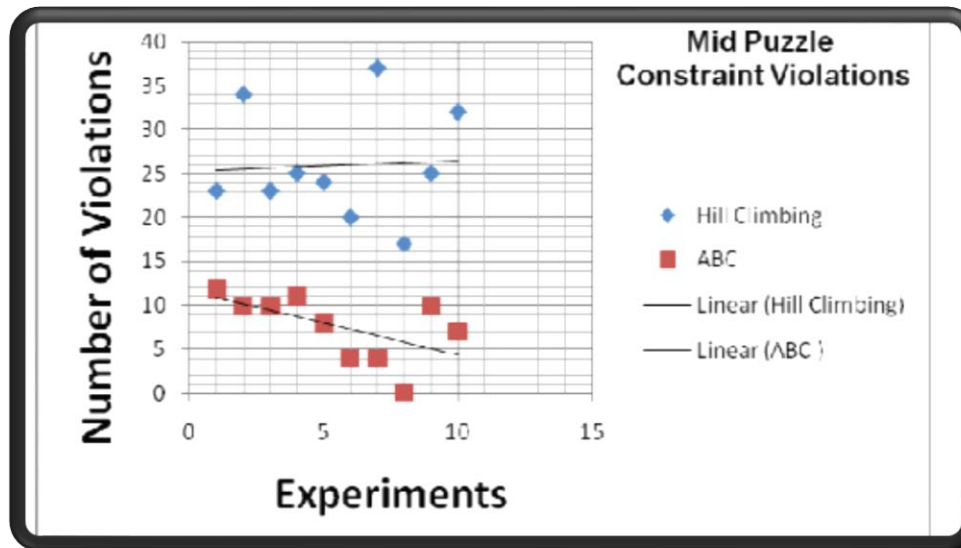
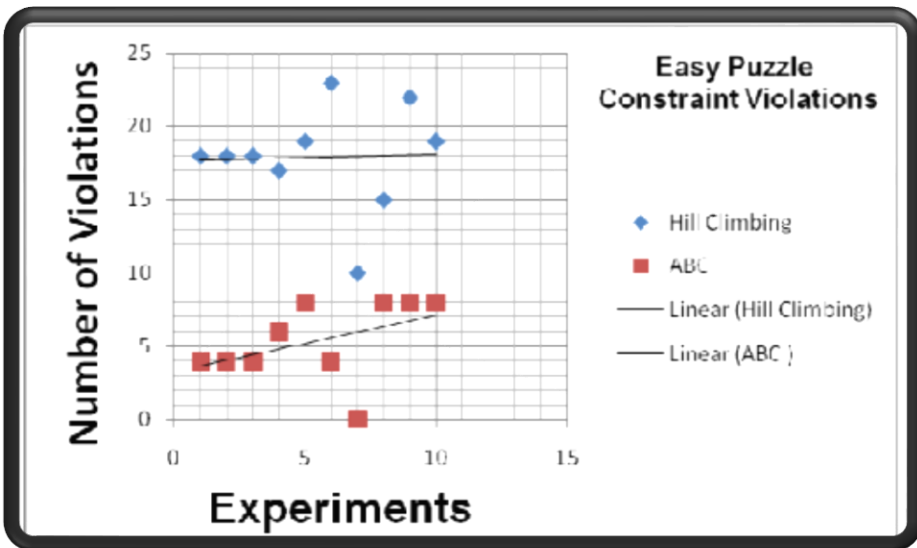


Figure 9: Comparison of number of constraint violations between ABC and Hill Climbing algorithms

Since backtracking and Constraint Propagation algorithms did not give violation containing solutions, they are not included in the scatter graphs. ABC algorithm is detected to be more accurate than Hill Climbing algorithm. This performance increase mainly obtained from the existence and efforts of the Onlooker bees. Onlooker bees helped ABC to avoid the search sticking at local maximums.

Python Code For Hill Climbing Approach

[Github Link](#) [\[click here\]](#)

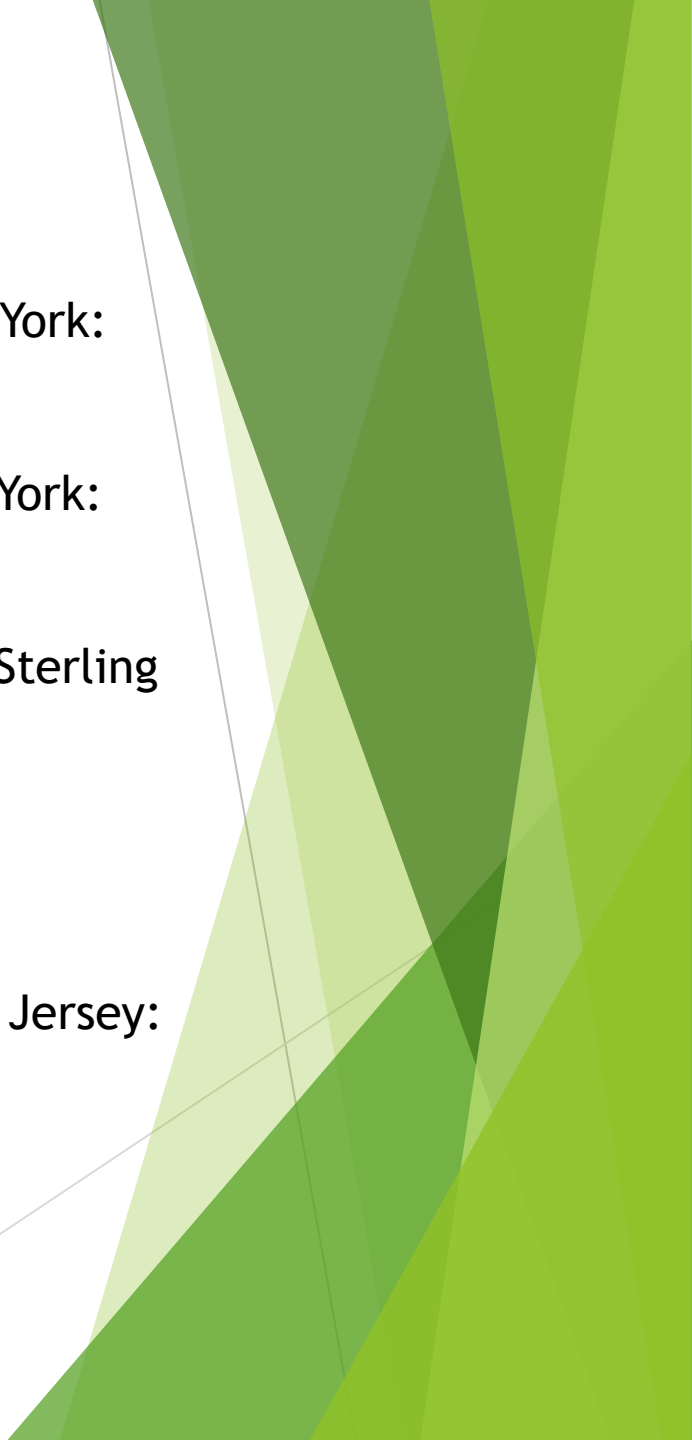
CONCLUSION AND FUTURE WORK

In this paper, β -Hill Climbing, a recent meta-heuristic local search algorithm, is investigated for Sudoku puzzle. β Hill Climbing algorithm is an extended version of hill climbing algorithm in which a new operator called β operator is introduced. This operator empowers the algorithm to escape the local optima by means of adding exploration capability to the basic hill climbing.

Sudoku puzzle is a popular game in the artificial intelligent field where the player attempt to fulfill the empty cells of a board of size 9X9 by a digits from the range 1-9 in the condition that these digits are distributed in each row, columns, and block of size 3X3 without repeating any

References

1. Felgenhauer, Bertram and Frazer Jarvis. “Enumerating possible Sudoku grids.” 20 June 2005. 06 April 2009.
<http://www.afjarvis.staff.shef.ac.uk/sudoku/sudoku.pdf>.
2. Jones, S.K., P.A. Roach, and S. Perkins. “Construction of Heuristics for a Search Based Approach to Solving Sudoku.” In the Proceedings of the 27th SGA International Conference on Innovative Techniques and Applications of Artificial Intelligence. pp. 3749. 2007.
3. Lewis, Rhyd. “Metaheuristics can solve sudoku puzzles.” Journal of Heuristics. Vol 13, (2007): 387401. 06 April 2009

- 
4. Longo, Frank. Green Belt Sudoku: Martial Arts Sudoku, Level 4: NotSoEasy. New York: Sterling Publishing Company, 2006.
 5. Longo, Frank. Red Belt Sudoku: Martial Arts Sudoku, Level 8: Super Tough. New York: Sterling Publishing Company, 2006.
 6. Longo, Frank. White Belt Sudoku: Martial Arts Sudoku, Level 1: Easy. New York: Sterling Publishing Company, 2006.
 7. Lynce, I., J. Ouaknine. "Sudoku as a SAT problem." In: Proceedings of the 9th Symposium on Artificial Intelligence and Mathematics, 2006.
 8. Russell, Stuart and Peter Norvig. Artificial Intelligence: A Modern Approach. New Jersey: Prentice Hall, 2003.

Thank you