Backtracking Algorithms for the Constraint Satisfaction Problem

Aditya Khaire
Department of Artificial Intelligence
Australian National University
u5657642@anu.edu.au

Abstract

Constraint Satisfaction Problem (CSP) are defining as the problems of assigning values to the variables that satisfy all the constraints. There are basic five tree search for CSP are Naïve Backtracking (BT), Backjumping (BJ), Conflict-directed Backjumping (CBJ), Backmarking (BM), Forward Channel (FC). (Prosser, 1993) The paper is described specifically on Backward moves used for solving the CSP problem. The problem considered is N-Queen problem with different board size to compare the results of the experiment performed on the Backtracking algorithms.

Key words: Constraint Satisfaction Problem, Naïve Backtracking, Backjumping, Conflict-directed Backjumping.

Introduction

The approach of this paper is as there are lot of different technique used for forward checking but, not much efforts are applied to improve the backward search. As with Naïve Backtracking algorithm at the dead end of the search when it has no value left to assign it backtracks one steps at a time and checks for consistency check with the neighboring nodes. The approach has less benefit as the problem size starts to increase. The two new approach defined is this paper are Backjumping and Conflict-directed Backjumping where they try to find the exact cause of conflict and jumps directly to that node. The algorithms BT → BJ → CBJ can be seen as more informative than other as they move forward. BJ algorithm tries to jumps back as the current variable exhausted all its domain value to the past variable which is the deepest variable from where conflict started and as with CBJ it tries to jump to the variable which is exactly causing conflict for the current variable and it hops back to all the past variables till it reaches to the conflict node. There is small difference in algorithm between BJ and CBJ but, because of that difference there is a lot of improvement in the results. The problem used for experiments is N-queen problem as it specifically a type of search tree algorithm. For the experiment, the size of the board is changed and results regarding the number of backtracks, visited nodes and the time for execution are used as the parameters for comparison. The next sections are as followed, section 1 defines the parameter used in the paper, section 2 gives idea about algorithms used in the paper, section 3 and 4 discuss about the experiment and results from that experiment and final section 5 give a brief summary and further studies in that direction

1.Definitions and conventions

Constraint Satisfaction Problem are defined as < V, D, C >

- V → is a set of variable $\{v_1, v_2, \ldots, v_n\}$
- D → is a set of valid values for variables $\{d_1, d_2, .., d_n\}$
- C → Constraints conditions on the variable $\{C_{1,2}, C_{2,3} \ldots, C_{i,j}\}$

The constraint is a relation between the two variables v_i and v_j for N-Queen problem constraint considered are binary constraints on the variables.

Domain: is a fixed set bag of values available to each variable before the start of the problem

Current domain: set of values changes according to the instantiation and if the domain is empty then there is no valid values for the variable.

The complete assignment of the queens on the board is to be said as all the variable have completed consistency check.

The variables used during the course of the paper are,

- *v[i]* → current variable need to instantiated.
- *v[h]* → past variable with a value assigned from the domain.
- *v[j]* → future variable.
- *n* x *n* size of the board is used with rows and columns used for calculating the constraints.

2.Backtracking Algorithms

Naïve Backtracking Algorithm

The Constraint satisfaction problem starts with instantiating variables with a value from its domain and check for

consistency with all the past variables and if all the values assigned are consistent then there is a solution for the problem. If the current variable *v[i]* has a current domain as empty set it will backtrack to the past variable *v[h]* and will try to instantiated a new value from the current domain to the past variable that will now be a current variable. The algorithm checks consistency for the variable with all the domain values and if it fails, it again backtracks to the next past variable connected to the current this goes on till all the nodes are exploited. In BT algorithm terminates in two conditions a) all the variables are exploited and if it has reached to the initial node after backtracking and it displays as ***no solution.*** b) if all the variables are assigned values from there respective domain.

The following is the pseudo code for the backtracking,

```
1.FUNCTION backtracking (i, consistent) = FALSE
2.  BEGIN
3.     h ← i – 1
4.     current-domain[i] ← domain[i]
5.     current-domain[h]←remove(v[h],current-domain(h) )
6.     current-domain[h] ≠ nil
7.     return[h]
8.  end
```

(Prosser, 1993)

Line on 1 checks for the consistency and if its true backtracking is skipped and next variable is chosen for instantiation.

On line 4 the current domain[i] is reset to the domain[i] and from line 5 to 7 it does the consistency check if it is failed and it initializes with the past variable *h* and removes its domain from the array and checks if it is not *nil*. This function is repeated number whenever current variable fails the consistency check and backtracks to the past variable.

Backjumping Algorithm

The procedure first implement by Gaschnig,1979 in an attempt to reduce the nodes within the search tree. At dead-end, backup to the most recent variable that eliminated some value in the domain of the current dead end variable. In BJ every variable keeps the record of the consistency check with every variable and if *v[i]* fails the consistency check with past variables. The BJ jumps back to the past variable *v[h]*. *v[h]* is the deepest variable in the search tree that performed a consistency check with the *v[i]*. The BJ doesn't perform consistency check with all the variables and thus saves number of backtracking steps.

$$h \leftarrow max\text{-}check[i]$$

The following is the pseudo code for the backtracking, (Williams, 2010) (8-Queen Puzzle, 2016) (Frost, 1997)

```
1.FUNCTION Backjumping (i, consistent) = FALSE
2.  BEGIN
3.     h ← max–check[i]
4.     FOR j ← h + 1 to i
5.     DO BEGIN
6.     max-check[i] ← 0;
7.     current-domain[i] ← domain[i]
8.     END;
9.    current-domain[h]←remove(v[h],current-domain(h) );
10.   current-domain[h] ≠ nil;
11.   return[h]
12.  END;
```

(Prosser, 1993)

The deepest variable *h* to be chosen is from the max-check[i] is the backtracking point. From line 4 to 8 it needs to reset all the variables *j* that are in future from *h* as these all variables need to recalculate after changing the assigned value of the *h*.

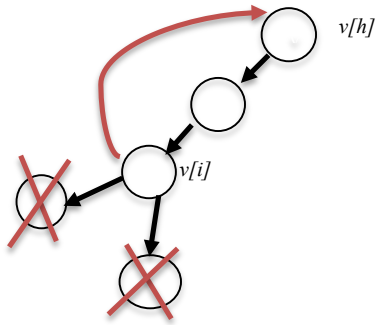

Figure 1: Backjumping

From the figure 1 as v[i] has no left consistency it backtracks to past variable v[h]. Thus BJ keeps all the set of variables that are in conflict with current variable v[i] and discards is when the backjump is performed.

Conflict-directed Backjumping Algorithm

The CBJ is similar to the BJ in jumping back to the past variable but, the main difference is BJ once jumps from *v[i]* to *v[h]* it then backtracks from that point, if the past variable *v[h]* fails the consistency check. As with CBJ it performs the backjump across conflicts which involve both *v[h]* and *v[i]*. This is achieved by recording all the variables that fail the direct consistency check with current variable. If no consistent value remaining with *v[h]* it again jumps back to the conflict variable.

CBJ maintains a *conflict-set[i]* for each variable and when a consistency check is failed the past variable is added to the set. When *v[i]* comes to the dead end the *max-list[i]* has the variable which has maximum conflict with current variable and CBJ jump backs to the exact variable. For the Backjumping algorithm to work as Conflict-directed modify BJ to include conflict set that include not only the variables with a direct conflict but also any subsequent variables with no consistent solution.

$$h \leftarrow max\text{-}list(conf\text{-}set[i])$$

```
1.FUNCTION Conflict-directed Backjumping (i, consistent) = FALSE
2.  BEGIN
3.      h ← max-list(conf-set[i]);
4.      FOR j ← h + 1 TO i
5.      DO BEGIN
6.          Conf-set[j] ← {0};
7.          Current-domain[j] ← domain[j]
8.          END;
9.   current-domain[h]←remove(v[h],current-domain(h) );
10. current-domain[h] ≠ nil;
11. return[h];
12. END; (Prosser, 1993)
```

Basic CBJ algorithm is a similar to BJ the changes are on line 3 where the past variable *v[h]* is a variable having maximum conflict with the current variable *v[i]*. From line 6 to 8 it reset all the future conflict set from the variable *h* till the current variable *i* and line 9 to 11 backtracking of the variable *h* and returning its current domain for further instantiation.

3.N-Queen Problem

As experiments were performed on a single problem that is N-Queen problem. The reason for choosing the problem was a) as the algorithm discussed were more related to binary constraint satisfaction problem. b) N-queen are a type of a tree search algorithm and Backtracking algorithms which is similar to depth first search algorithm generate better results. What steps need to be performed for consistency check between two queen can be analyzed using an example. The problem considered is a *4 x 4* Queen problem as follows,

	1	2	3	4
1			Q	
2	Q			
3				Q
4		Q		

F (Norvig, 2003)igure 2: 4 x 4 Queen Problem

In the game of chess, queen can move vertically, horizontally and diagonally and this makes it more powerful than any other piece. The problem is to place all the queen in a non attacking position from each other, the above provided is one of the solutions.
Queens can be placed into two columns without any conflicts if two conditions are satisfied,

1. Both the queens cannot be in the same row i.e. (1,1) and (1,2) as both of them are in same row.
2. All the elements that are diagonal to the queen (2,1) are under attack, so no other queen can be placed in that blocks. (8-Queen Puzzle, 2016)

If all the queens satisfy above two conditions, then the solution is said to be constraint satisfied solution. There is distinct solution for different size of the queen problems. For 4-queen it has two solutions and as the size of the problem increase the number of solution also starts to increase. The backtracking algorithms checks for these conditions after every variable is instantiated and if it returns *FALSE* there is a conflict between the current variable *v[i]* and the past variable *v[h]*. Backjumping and Conflict-directed Backjumping collects these records of conflict and use it, when all the domain value of the current variable *v[i]* is exhausted. *V[i]* then jumps to one of the *v[h]* stored in its record list and changes the assigned value of that past variable to make the variables consistent.

4.Results and Analysis

All the experiments are executed on the Apple Desktop and Partch server and algorithms were complied on Python version 3.3. The output values are the average of three outputs. The results taken from the three algorithms are in tabular form,

Sr.No	Algorithms	Parameters			
		Board Size	No. of Backtracks	Nodes visited	Time (ms)
1	BT	8	105	113	7.036
2		17	5357	5374	532.267
3		21	8541	8562	218.323
4		35	-	-	-
1	BJ	17	77	94	10.211
2		21	211	232	38.56
3		200	2975	3175	118879.8297
4		1000	-	-	-
1	CBJ	17	5	22	4.828
2		21	11	32	6.719
3		200	44	244	3408.353
4		1000	199	1199	1618877.50

Figure 3: Analysis of three backtracking Algorithms

The size of Board for Naïve Backtracking is less than BJ and CBJ because of memory issue in python. From looking at the table, CBJ takes 5 backtracks to reach solution as against BT takes 5357 and BJ takes 77 that's also much reduce number of backtracks than Naïve BT. According to the results CBJ is performing better than other two algorithms in all the parameters.

5.Conclusion

The backtracking search tree algorithms have performed according to the prediction. Backjumping and Conflict-directed Backjumping are performing much better than the naïve Backtracking and the reason is they are calculating the exact node from where the consistency is failed for the current variables. The CBJ algorithm, which stores conflict data for every variable still it does not require large space as it discards the array of data after every backtracking thus reduces the space complexity of the algorithm.

The future scope of this algorithm is to combine backtracking algorithms with the Forward channel, Arc consistency algorithms in order to improve the variable selection criteria in combination with backtracking. These combination has been proposed in "Hybrid Algorithms for the Constraint Satisfaction Problem" (Prosser, 1993) Journal this can be a good reference for the algorithms.

References

8-Queen Puzzle. (2016, May 20). Retrieved May 22, 2016, from

Wikipedia:
https://en.wikipedia.org/wiki/Eight_queens_puzzle

Frost, D. H. (1997). *Algorithms and Heuristics for Constraint Satisfaction Problems.* Retrieved May 10, 2016, from http://www.ics.uci.edu/~csp/R69.pdf

Norvig, S. R. (2003). *Artificial Intillignece A modern Approch.* New Jersey, USA: Pearson Education.

Prosser, P. (1993). Hybrid Algorithms For Constraint Satisfaction problem. *Computational Intelligence , 9* (3), 32.

Williams, B. C. (2010, Septmember 29). *Solving Constraint problem using Conflict and Backjumping.* Retrieved May 7, 2016, from http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-410-principles-of-autonomy-and-decision-making-fall-2010/lecture-notes/MIT16_410F10_lec07a.pdf