Artificial Intelligence Report 2nd Week-5th Problem

Le Kien Trung
03-120291, Department of Mechano-Informatics

November 11th, 2012

1 Constrain Satisfaction Problem (CSP)

As the name suggests, CSPs are search problems whose variables must follow some given rules. CSPs' characteristics and solving strategies are very different from general search problems. We will not consider all cases of CSPs here, but normally an CSP can be represented by the following notations:

- Variables: $\{x_1, x_2, ..., x_n\}$ (n is a finite number)
- Domains: $\{a_1, a_2, ..., a_m\}$ (m is also a finite number)
  Each variable may have its own distinguished domains.
  E.g: x_1 is a digit, x_2 is a character.
- Constrains: Conditions that bind variables together.
  There are two kind of constrains, implicit and explicit.
  E.g: $x_1 + x_2 = 2$ is an implicit while $(x_1, x_2) \in \{(0, 2), (1, 1), (2, 0)\}$ is explicit.

General search strategies, such as Depth First or Breath First or even A* search, do not work for CSPs. Solutions for CSPs are the solutions itself, not the path to them, namely, how to get a solution is irrelevant. Therefore, DFS or BFS won't work, as solutions for CSPs lay on the lowest level of the search tree, and in almost all cases the depth of the search tree of an CSP is too big for those algorithms to handle. However, by utilizing the truth maintenance system, we can modify those algorithms and make them work with CSPs. Let's first talk a little bit about TMS.

2 Truth Maintenance System

A truth maintenance system, or TMS, is a knowledge representation method for representing both beliefs and their dependencies. The name truth maintenance is due to the ability of these systems to restore consistency.

An TMS maintains consistency between old believed knowledge and current believed knowledge in the knowledge base (KB) through revision. If pushing the current knowledge to KB causes contradictory, it will be replaced by other knowledge. The record of which knowledge contradicts with which in KB is kept, so we can avoid considering the same pair of current knowledge and old believe knowledge more than once.

A well defined algorithm based on TMS is dependency-directed backtracking, or backtracking in short. Supposed that each variable has its own domain. The algorithm will recursively assigning variables to values in their corresponding domains. When all variables are assigned without contradiction, a solution is reached. Until here, it looks just like DFS or BFS (depend on the order of assignments), however, actually it does not. Each time it encounters a variable with their new value, it investigates if that assignment violates truth in current knowledge base by checking constrains of the problem. If that assignment passes the test, then new assignment will be conducted. On the contrary, the current assignment will be removed, and the false system will become true again. By doing that, the truth system is consistent, so we can expect to find solution some way down the road.

3 Examples

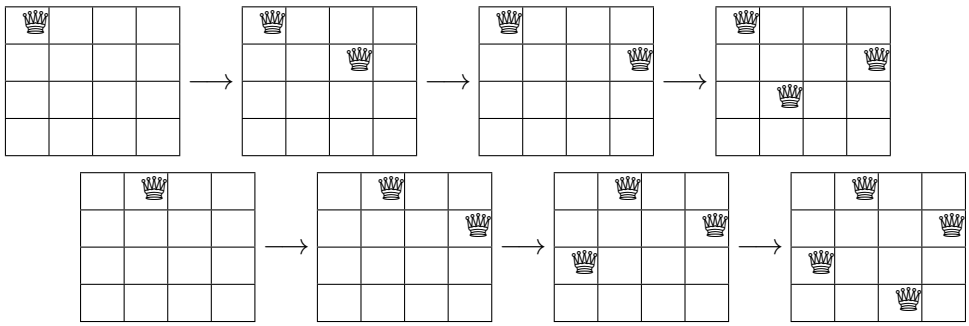
3.1 NQueen

Analysing NQueen problem as an CSP, we have:

- Variables: $\{x_i\},\ i \in \overline{1,n}$, in which x_i value is the column number of the queen placed in row #i
- Domains: $x_i \in \overline{1,n},\ \forall i \in \overline{1,n}$
- Constrains: No queen is in a threatening position against another. Mathematically speaking:

$$\begin{cases} x_i \neq x_j, & \forall i,j \in \overline{1,n} \\ |i-j| \neq |x_i - x_j| & \forall i,j \in \overline{1,n} \end{cases}$$

Backtracking algorithm can be easily applied to this problem. The following figures with show its process step by step when n=4.



1. Place 1st queen on to 1st column
2. Place 2nd queen on to 3rd column (1st and 2nd column cause contradictory)
3. Place 2nd queen on to 4th column (no available grid for 3rd queen $\rightarrow$ backtrack)
4. Place 3rd queen on to 2nd column
5. Place 1st queen on to 2nd column (no available grid for 4th, 3rd, 2nd $\rightarrow$ backtrack)
6. Place 2nd queen on to 2nd column
7. Place 3rd queen on to 1st column
8. Place 4th queen on to 3rd column

Backtracking algorithm computational cost is the same with depth first search. Because the its implementation uses recursion, it cannot find solution with a large n ($\geq$25). Moreover, when n$\geq$15, it takes a lot of time to find all the solutions.

3.2 Verbal Arithmetic

Verbal Arithmetic is another well-known example of CSP. Supposed we have to solve the following verbal arithmetic problem:

$$\text{send + more = money}$$

- Variables: Value for each characters x_i and we need to define extra value for carry bits c_j. In this cases, x_i should be values of s,e,n,d,m,o,r,n,y
- Domains: Domain for characters' value is from 1 to 9, while domain for carry bits is 0 and 1.
- Constrains: Sum of the first string and the second string will be equal to the third string, given that each characters are assigned a distinct number. In this problem, if we define c_i as carry bits of formula i, and s_j as the first, second and third string (send, more, money) then the constraints can be written as following:

$$x_i \neq x_j,\ \forall i \neq j$$
$$s_1(i) + s_2(i) + c_i = s_3(i) + 10 * c_{i+1}$$

Using those notations, we can easily solve the verbal arithmetic problem. Let's denote C_i as the carry of column i.

1. $M = C_5 = 1$
2. $O = 0,\ S = 9,\ C_4 = 0$ (11 $\geq$ S+1+C_4 = O + 10, O $\neq$ 1)
3. $N = E + 1,\ C_3 = 1$ (N $\neq$ E & N $\leq$ E+1)
4. $R = 8,\ C_2 = 1$ (if C_2 = 0 then N+R % 10=E then 1+R % 10 = E then R = 9 (but S = 9) so C_2 = 1 and R = 8)
5. D+E = 10 + Y $\geq$ 12 (Y $\geq$2)
6. Only two pairs left to produce sum bigger than 12: (5,7) and (6,7)
7. D = 7 (N = E+1 $\neq$ R=8)
8. E = 5 (N = E+1 $\neq$ D=7)
9. Y = 2

By constantly checking the constraints in each assignment, we can avoid going deeply into the search tree.

3.3 Sudoku

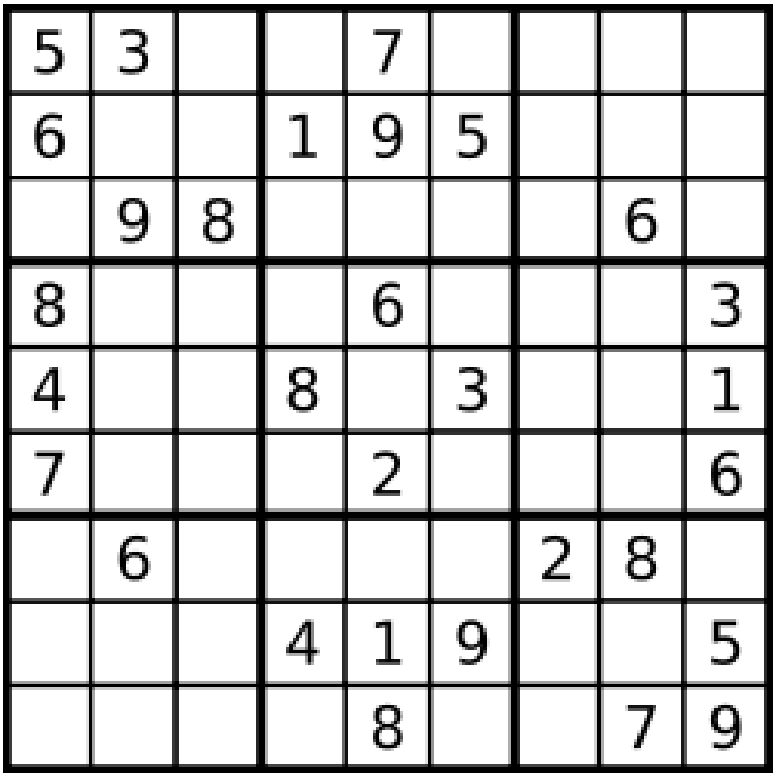


Figure 1: Sudoku puzzle (From Wikipedia)

Consider Sudoku as an CSP, we have:

- Variables: Values of each empty grids
- Domain: {1,2,3,4,5,6,7,8,9}
- Constrains: Each column, each row, and each of the nine 33 sub-grids that compose the grid (also called ”boxes”, ”blocks”, ”regions”, or ”sub-squares”) contains all of the digits from 1 to 9.

The backtracking implementation of sudoku is very similar to nqueen. We just need to gradually assign each grids to a number in {1,2,3,4,5,6,7,8,9}. However, sudoku is too complicated to show a the backtracking process step by step. Moreover, without modification, it may take several ten seconds to solve super hard sudoku. In the next report, I will write the implemented codes for all the problems here, including sudoku and show how we can improving backtracking by several simple tricks.

4 Conclusion

The implemenatation of TMS, backtracking is an well-defined tool to solve CSPs problem. However, it is not a powerful method in term of speed and memory. There are several ways to improve backtracking algorithm:

- Choose the order of assigment. For instance, in the verbal arithmetic example, assignment did not go from right to left but left to right.
- Forward checking: If we do this, there is no need of backtracking.
- Represent the problem in the best suitable ways.

It is all about the actual implementation so I will leave the details for the next report.