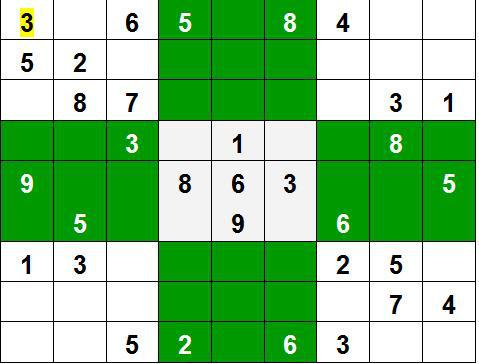
**Back Tracking Search For Csps & Cryptarithmetic Problem**

**Back Tracking Search For Csps**

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree).

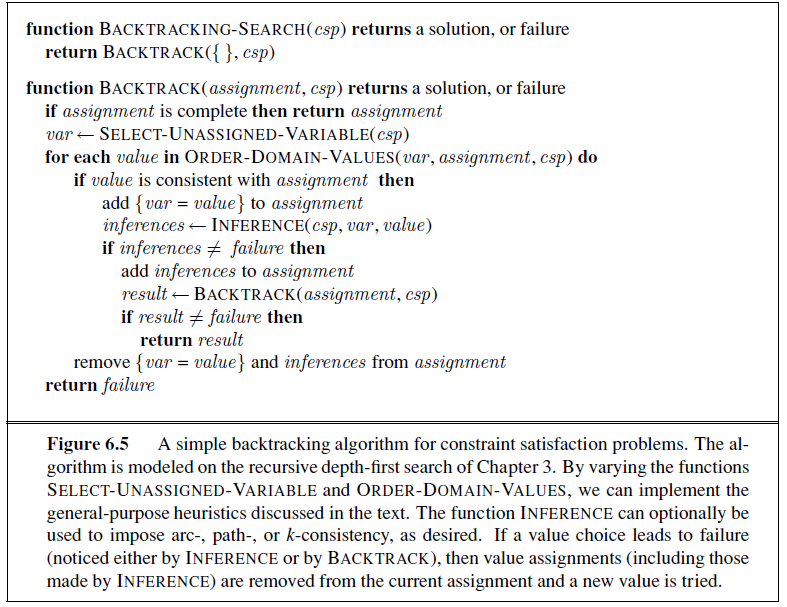
For example, consider the SudoKo solving Problem, we try filling digits one by one. Whenever we find that current digit cannot lead to a solution, we remove it (backtrack) and try next digit. This is better than naive approach (generating all possible combinations of digits and then trying every combination one by one) as it drops a set of permutations whenever it backtracks.



There are three types of problems in backtracking –

1. Decision Problem – In this, we search for a feasible solution.
2. Optimization Problem – In this, we search for the best solution.
3. Enumeration Problem – In this, we find all feasible solutions.

The term backtracking search is used for a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign. The algorithm for backtracking is given below.



It repeatedly chooses an unassigned variable, and then tries all values in the domain of that variable in turn, trying to find a solution. If an inconsistency is detected, then BACKTRACK returns failure, causing the previous call to try another value.

**Eg.**

N-Queens. Place N Queens on an N X N chess board so that no Queen can attack any other Queen.

■ N Variables, one per row. Value of Qi is the column the Queen in row i is placed.

■ Constrains:

Vi ≠ Vj for all i ≠ j (cannot put two Queens in same column)

|Vi-Vj| ≠ |i-j| (Diagonal constraint) (i.e., the difference in the values assigned to Vi and Vj can’t be equal to the difference between i and j.

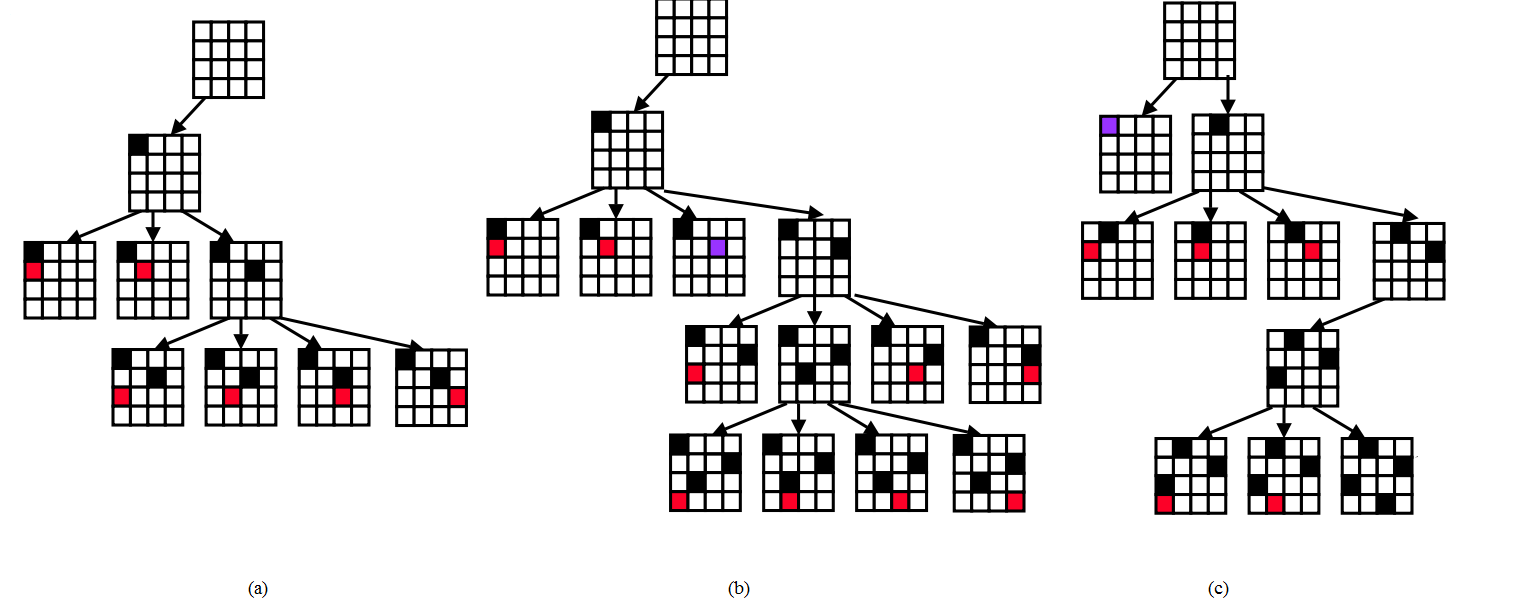


Fig. 4X4 Queen Problem solution using BackTracking.

Black color indicates queen position which is fixed i.e; no clashes.

Red color indicates current queen position that have some clashes with existing queens on the board.

Purple color indicates queen position that is not fixed i.e going to change as there is no futher possibilities.

**Backtracking with Forward Checking:**

●Forward checking is an extension of backtracking search that employs a “modest” amount of propagation (lookahead).

●When a variable is instantiated we check all constraints that have only one uninstantiated variable remaining. ●For that uninstantiated variable, we check all of its values, pruning those values that violate the constraint.

Eg.

4X4 Queens

■ Q1,Q2,Q3,Q4 with domain {1..4}

■ All binary constraints: C(Qi,Qj)

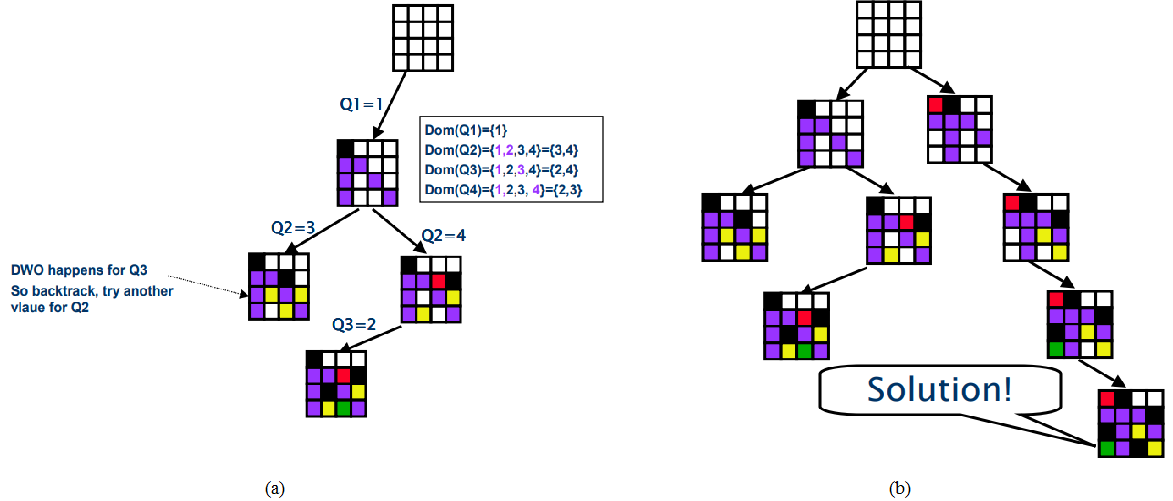
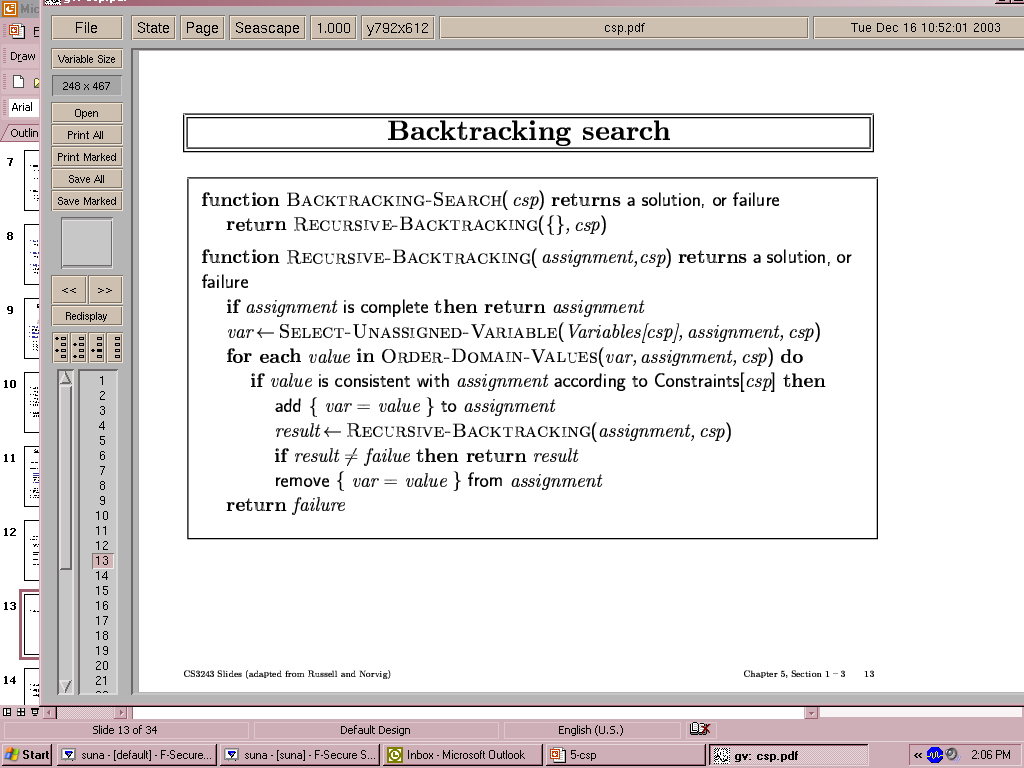
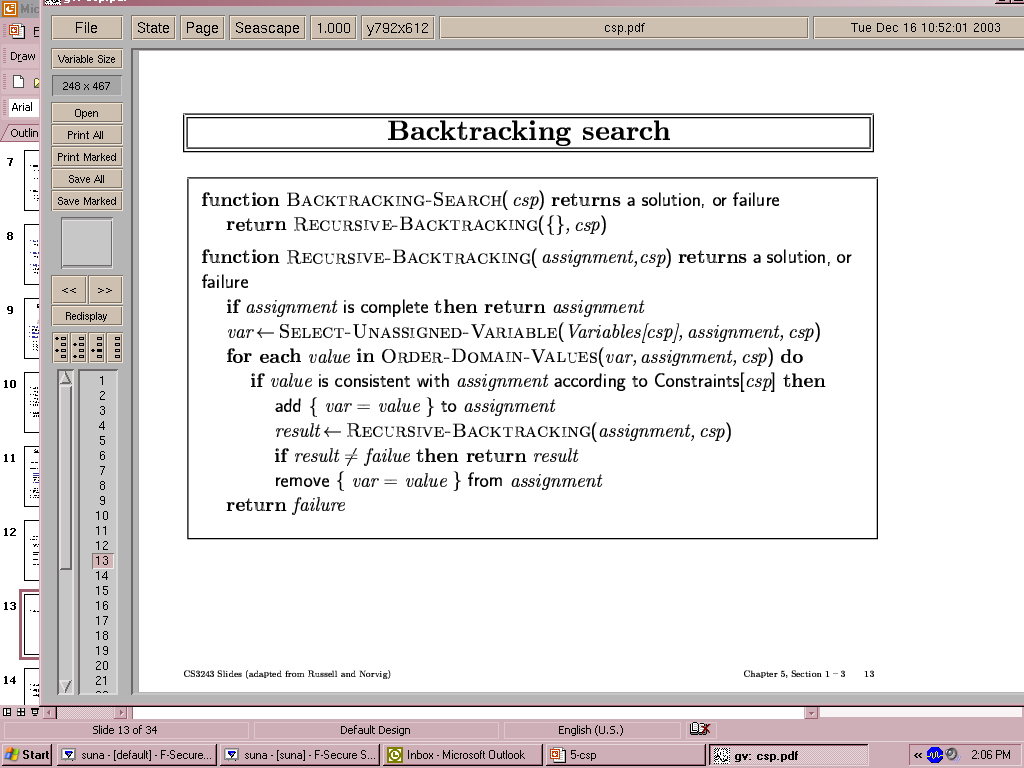


Figure: FC illustration: color values are removed from domain of each row (blue, then yellow, then green)



**Cryptarithmetic Problem:**

Cryptarithmetic Problem is a type of constraint satisfaction problem where the game is about digits and its unique replacement either with alphabets or other symbols. In cryptarithmetic problem, the digits  (0-9) get substituted by some possible alphabets or symbols. The task in cryptarithmetic problem is to substitute each digit with an alphabet to get the result arithmetically correct. We can perform all the arithmetic operations on a given cryptarithmetic problem.

The **rules or constraints** on a cryptarithmetic problem are as follows:

There should be a unique digit to be replaced with a unique alphabet.

The result should satisfy the predefined arithmetic rules, i.e., 2+2 =4, nothing else.

Digits should be from 0-9 only.

There should be only one carry forward, while performing the addition operation on a problem.

The problem can be solved from both sides, i.e., lefthand side (L.H.S), or righthand side (R.H.S)

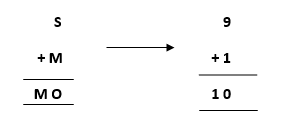
Let’s understand the cryptarithmetic problem as well its constraints better with the help of an example:

Given a cryptarithmetic problem, i.e., S E N D + M O R E = M O N E Y



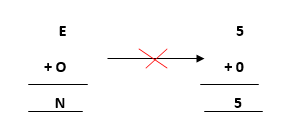
In this example, add both terms S E N D and M O R E to bring M O N E Y as a result. Follow the below steps to understand the given problem by breaking it into its subparts:

Starting from the left hand side (L.H.S) , the terms are S and M. Assign a digit which could give a satisfactory result. Let’s assign S->9 and M->1.

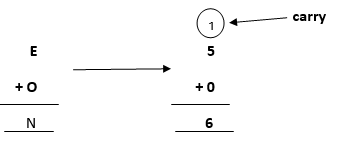


Hence, we get a satisfactory result by adding up the terms and got an assignment for O as O->0 as well.

Now, move ahead to the next terms E and O to get N as its output.

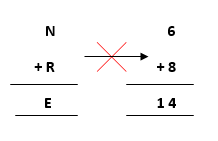


Adding E and O, which means 5+0=0, which is not possible because according to cryptarithmetic constraints, we cannot assign the same digit to two letters. So, we need to think more and assign some other value.



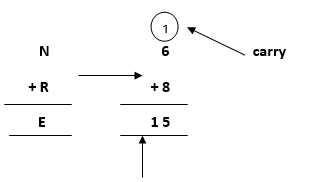
Note: When we will solve further, we will get one carry, so after applying it, the answer will be satisfied.

Further, adding the next two terms N and R we get,



But, we have already assigned E->5. Thus, the above result does not satisfy the values because we are getting a different value for E. So, we need to think more.

Again, after solving the whole problem, we will get a carryover on this term, so our answer will be satisfied.



            where 1 will be carry forward to the above term

Let’s move ahead.

Again, on adding the last two terms, i.e., the rightmost terms D and E, we get Y as its result.



where 1 will be  carry forward to the above term

Keeping all the constraints in mind, the final resultant is as follows:



Below is the representation of the assignment of the digits to the alphabets.

