CS3310 Final Report

Chris Saba

Professor Tannaz R. Damavandi

Sudoku

Sudoku was coded using branch and bound backtracking. The naive approach is to generate all possible configurations of numbers from 1 to 9 to fill the empty cells. Try every configuration one by one until the correct configuration is found, i.e. for every unassigned position fill the position with a number from 1 to 9. After filling all the unassigned position check if the matrix is safe or not. If safe print else recur for other cases.

Backtracking works sudoku in a similar fashion to N Queens problem.

1. Create a function that checks after assigning the current index the grid becomes unsafe or not. Keep HashMap for a row, column and boxes. If any number has a frequency greater than 1 in the HashMap return false, otherwise return true; HashMap can be avoided by using loops.
2. Create a recursive function which takes a grid.
3. Check for any unassigned location. If present then assign a number from 1 to 9, check if assigning the number to current index makes the grid unsafe or not, if safe then recursively call the function for all safe cases from 0 to 9. if any recursive call returns true, end the loop and return true. If no recursive call returns true, then return false.
4. If there is no unassigned location, then return true.

0-1 Knapsack

The backtracking-based solution works better than brute force by ignoring infeasible solutions. We can do better (than backtracking) if we know a bound on best possible solution subtree rooted with every node. If the best in subtree is worse than current best, we can simply ignore this node and its subtrees. So, we compute bound (best solution) for every node and compare the bound with current best solution before exploring the node. This results in the Best-First algorithm for. Branch and bound is very useful technique for searching a solution but in worst case, we need to fully calculate the entire tree. At best, we only need to fully calculate one path through the tree and prune the rest of it.

Travelling Salesman Problem (TSP)

Utilizing Branch and Bound method, for current node in tree, we compute a bound on best possible solution that we can get if we down this node. If the bound on best possible solution itself is worse than current best (best computed so far), then we ignore the subtree rooted with the node.

Note that the cost through a node includes two costs.

1) Cost of reaching the node from the root (Cost computed when we reach a node)

2) Cost of reaching an answer from current node to a leaf (Compute a bound on this cost to decide whether to ignore subtree with this node or not) AKA Pruning.

The TSP has a much larger time complexity than that of 01-Knapsack, and Sudoku. However, all three problems are NP-Complete, they share the NP hard constraint, in that the optimal solution isn’t global. TSP has worst case time complexity of O((n-1)!/2) Whereas 01 knapsack worst case is (nW) where n is number of items, W is capacity. Sudoku has a complexity of O(n^m) n is number of possibilities of each square, and m is blank spaces. Sudoku however, seems to calculate almost instantly, because of the backtracking algorithm cuts this time extremely well, but not globally.