Chapter 6 Branch & Bound (B&B)

Key Points

- Understand the pruning strategy of B&B
- Master the framework of B&B
    - FIFO
    - Priority Queue
- Learn from examples
    - 0-1 Knapsack
    - TSP
    - MCP
    - SSSP

General Thought

B&B vs. BT

- Similar to BT, B&B searches solution in solution space tree
- Goal of search
    - BT: all feasible solutions
    - B&B: one feasible solution or the **optimal solution**
- Search Strategy
    - BT: skipped DFS
    - B&B: BFS or best first search
- The functions for search
    - BT
        - constraint function
        - bounding function
    - B&B
        - constraint function
        - bounding function
        - priority function (always = bounding function)
    - Characteristic:
        - BT: efficient in space
        - B&B: much faster, without an exhaustive search on the average

B&B Features

- B&B searches
    - BFS or best first search
    - avoid invalid and sub-optimal search by pruning functions
- Estimate value of node
    - pick the node with extremum to search
    - continually adjust the search direction
- Design of bunding function bases on the objective function

- B&B applies to solve optimization problem
- Each active node has only one chance to be an extending node

Search procedure

- once being an extending node, the node produces all its sons as candidate active nodes
    - infeasible or sub-optimal ones are **discarded**
    - remaining sons: inserted into an active node list
- choose next active node in the active node list as an **extending node** and repeat the above step
- **Until:** the **optimal solution** is found or **the active node list** is empty

Two kinds of B&B

- FIFO
    - Use **FIFO** rule to choose an **active node** in the list to be the next **extending node**
- Priority Queue
    - Use **Priority Queue** to choose an **active node** in the list to be the next **extending node**
- To accelerate the search, each active node maintains a bound value
- The active node with the **optimal bound value** in the list is chosen as the next **extending node**

B&B Framework

- For a specific problem, define its solution space
    - $X = (x_1, \cdots, x_n), x_i \in S_i = \{a_{i_1}, \cdots, a_{i_{ri}}\}$
    - Solution space: $S_1 \times S_2 \times \cdots \times S_n$
- Determine a solution-space tree to search
- Traverse the solution-space tree in BFS (FIFO Queue) or Best First Search (Priority Queue) and avoid invalid search by pruning functions (determine B&B efficiency)
    - Constraint function
    - Bounding function, optimal bound
    - Priority function = bounding function (priority queue)

Complexity of B&B

- Efficient on the average
    - the search adapts to the branch that includes the optimal solution
    - many branches can be terminated very early
- Many NP-hard problem can be solved by B&B efficiently on the average
- A very large tree may be generated in the worst case
- The worst case time complexity is still exponential

0-1 Knapsack

B&B ALG Design for 0-1 Knapsack

- Solution
- Search space
- Node
- Feasible solution: a solution that satisfies weight constraint
- Optimal solution: the feasible solution with **maximal** value
- BFS or best first search
    - Constraint function: weight constraint

Bounding/Priority Function

- Sort items in descending order in items of $v_i/w_i, i = 1, 2, \cdots, n$
- At node $< x_1, x_2, \cdots, x_i >$
- **Bound** bestp: the maximum knapsack value found so far
    - current value + rest value = total value
- **Bounding/Priority function** to compute **the upper bound of the knapsack**
    - up, based on the current item selection

ALG Comparison for 0-1 Knapsack

- DP if the C is small
- If the C is large
    - If space is limited, BT is preferred
    - If search speed is important, B&B is preferred

Details about B&B 0-1 Knapsack

- 寻找上界：一路左子树走最大
    - 对于0-1背包，还可以最后按比例放进去
- 判断是否加入活跃节点列表：是否可行 <=> 是否超重
- 判断拿出哪个节点，选上界最大的
- 如果一个节点算下来bestp（上界），且为叶节点，那么bestp就是这个解对应的值
    - 如果其他节点的上界 < 该叶节点的bestp，那么最优值已经找到

Traveling Salesman Problem (TSP)

B&B ALG Design for TSP

- Modeling: city set $C = \{c_1, c_2, \cdots, c_n\}$, distance $d(c_i, c_j) = d(c_j, c_i)$
- Solution: n-dimensional vector $< x_1, x_2, \cdots, x_n >, x_i \in \{1, 2, \cdots, n\}, x_i = j \iff$ the i_{th} city that the salesman travels is city j
- Node: $< x_1, x_2, \cdots, x_i >$ (partial vector)
- Search space: a tree of (n-1)! leaves, referred to as a permutation tree
- Optimal solution: $k_1, k_2, \cdots, k_n$, a permutation of $1, 2, \cdots, n$, so that

$$\min\left\{\sum_{i=1}^{n-1} d(c_{k_i}, c_{k_{i+1}}) + d(c_{k_n}, c_{k_1})\right\}$$

Bounding/Priority Function

- Constraint: the i_{th} vertex should be connected to the $(i-1)$th vertex on the chosen path
  有路可走
    - If the extending node is a parent of a leaf, the chosen vertex should also connected to vertex x_1 能回去
- Bound *bestc*: the cost of the found traveling route with minimum cost
- Bounding/Priority function *lcost* = *cc* + *rcost*
    - *cc*: current cost
    - *rcost*: the sum of the minimal costs of the outgoing edges of the remaining vertices and the last vertex on the current path
    - *lcost*: the lower bound of the potential route in the subtree rooted at the present node *lcost* = *cc* + *rcost*

B&B with Priority Queue for TSP

- parent of a leaf node

  if append to **the active node list**
    - feasible conditions
        - edge to next vertex
        - able to get back to origin
    - bounding condition
        - potential cost <= bestc
- other inner node

  if append to **the active node list**
    - feasible conditions
        - edge to next vertex
    - bounding condition
        - potential cost <= bestc

every time pick a node with optimal value (potential) as an extending node

- a leaf node: feasible/optimal solution
- a non-leaf node: extending it, and add the feasible sons to the active list

Maximum Clique Problem (MCP)

- Problem: undirected graph $G =< V, E >$, find the maximum clique of G
- Complete subgraph of $\mathrm{G}: \mathrm{U} =< \mathrm{V}', \mathrm{E}' >, \mathrm{V}' \subseteq \mathrm{V}, \mathrm{E}' \subseteq \mathrm{E}, \quad \forall u, v \in V', (u, v) \in E'$
- Clique of G: a complete subgraph of G which is not included in another larger complete graph of G
- Maximum clique of G: the clique with maximum vertices

MCP Analysis

- Solution: n-dimensional vector $< x_1, x_2, \cdots, x_n >$, $x_k = 1$ iff vertex k belongs to MCP
- Brute force search: for any subset of vertices, check if it is a clique
    - 2^n subsets
    - exponential time

Design B&B ALG for MCP

- A subset solution space tree
- Node $< x_1, x_2, \cdots, x_k >$
    - The vertices $1, 2, \cdots, k$ are checked and the vertex i is in the clique if $x_i = 1$
- Constraint: the vertex being checked is connected to each vertex in the current clique

Bounding/Priority Function

- Bound *bestn*: the number of vertices in the MC found so far
- Bounding/Priority function: the upper bound of the # of vertices in the MC extended from the current clique

  $$\mathrm{F = C_n + n - k}$$

  C_n: the # of vertices in the current clique (initiated as 0)

  k+1: the level of the node being considered
- The worst case: $O(n2^n)$

B&B with Priority Queue for MCP

- non-leaf node
    - feasible condition
        - has edge connected to the nodes in current clique
    - bounding condition
        - upper bound of the vertices $\geq$ bestn so far

  if both are satisfied insert to the active node list
- save the solution so far (of a node)
    - with node
        - a ptr to the parent node
        - a tag shows that it's the l/r child

every time pick a node with optimal value (potential) as an extending node

- a leaf node: feasible/optimal solution
- a non-leaf node: extending it, and add the feasible sons to the active list

Single Source Shortest Path (SSSP)

Compute Short(A,F) -- B&B

- Basic idea
    1. Start from the source s and the queue begins with only s
    2. Delete/pick one node from the queue and extending it
        - Each neighbor contributes to one son
    3. If not being pruned, its sons are inserted into the queue
    4. Repeat steps 2-3 until the queue is empty

Pruning functions

- Node i in the solution space tree denotes the distance to vertex v $dist(v)$
- If $dist(v) \geq$ the current shortest path from the source to the destination, prune the subtree rooted at i

The ways to select a node from the queue

- FIFO (FIFO queue)
- The node denoting the current shortest path (priority queue)