Assignment Problem: A Branch and Bound Approach

Sumanth

June 18, 2025

Abstract

The Assignment Problem involves assigning n workers to n tasks to minimize the total cost, where each worker is assigned exactly one task. The **branch and bound** approach explores a state space tree, using cost bounds to prune paths that cannot yield better solutions than the current minimum. This document details the problem formulation, pseudocode, complexity analysis, practical applications, and implementation considerations, including matrix reduction for bounding and the contributor's note on optimization with the Hungarian Algorithm.

1 Introduction

The Assignment Problem seeks to optimally assign n workers to n tasks such that each worker performs exactly one task, minimizing the total cost based on a given cost matrix. The **branch and bound** approach efficiently navigates the solution space by estimating lower bounds on the cost of partial assignments and pruning unpromising branches. This method is widely used in scheduling, resource allocation, and planning, offering a flexible alternative to other methods like the Hungarian Algorithm.

2 Problem Formulation

Given an $n \times n$ cost matrix $\text{cost}[i][j]$, where $\text{cost}[i][j]$ represents the cost of assigning task j to worker i, find a permutation of tasks $\{t_1, t_2, \ldots, t_n\}$ such that each worker i is assigned task t_i, and the total cost:

$$\sum_{i=1}^{n} \text{cost}[i][t_i]$$

is minimized. The output is the minimum total cost and the corresponding assignment.

2.1 Key Idea

The algorithm uses **branch and bound** to explore a state space tree, where each node represents a partial assignment of tasks to workers. At each node:

- Branch by assigning an unassigned task to an unassigned worker.
- Compute a lower bound on the cost of completing the assignment using matrix reduction.
- Prune the node if its bound exceeds the current minimum cost.

Nodes are processed using a priority queue to explore the most promising paths first.

3 Assignment Algorithm

The algorithm uses a priority queue to process nodes with the lowest bounds first, calculating bounds via matrix reduction. Below is the pseudocode:

3.1 Bound Calculation

The `computeBound` function estimates the minimum additional cost by:

- Reducing the cost matrix by subtracting the minimum value from each row and column.
- Summing the subtracted values to form a lower bound on the cost of completing the assignment.

The reduced matrix reflects the minimum cost for unassigned tasks and workers.

3.2 Hungarian Algorithm Optimization

As noted by the contributor, the Hungarian Algorithm solves the Assignment Problem in $O(n^3)$ time, offering a polynomial-time alternative if branch and bound pruning is insufficient.

4 Complexity Analysis

- **Time Complexity**: $O(n!)$ in the worst case, as all permutations may be explored. Effective pruning significantly reduces the number of nodes in practice.
- **Space Complexity**: $O(n^2)$ for the cost matrix and reduced matrices, plus $O(n)$ for the priority queue and assignment storage.

5 Practical Applications

The Assignment Problem is used in:

- **Job Scheduling**: Assigning workers to tasks to minimize completion costs.
- **Task Distribution**: Optimizing workload allocation in distributed systems.
- **Resource Allocation**: Matching resources to projects for cost efficiency.
- **Robotics and AI**: Planning task assignments for multi-agent systems.

6 Example

Given a 3×3 cost matrix:

	T_1	T_2	T_3
W_1	10	12	8
W_2	9	11	7
W_3	8	10	6

Algorithm 1 Assignment Problem (Branch and Bound)

```
Input: Cost matrix $\text{cost}[1\ldots n][1\ldots n]$
Output: Minimum cost, assignment
Initialize priority queue $Q$ with root node $(\text{level} = 0, \text{cost} = 0, \text{bound} = \text{computeBound}(\text{cost}), \text{assigned} = \emptyset)$
Initialize minCost $\leftarrow \infty$, bestAssignment $\leftarrow \emptyset$
while $Q$ is not empty do
    $u \leftarrow Q.\text{extractMin}()$
    if $u.\text{bound} \geq \text{minCost}$ then
        continue                                        ▷ Prune node
    end if
    if $u.\text{level} = n$ then
        if $u.\text{cost} < \text{minCost}$ then
            minCost $\leftarrow u.\text{cost}$
            bestAssignment $\leftarrow u.\text{assigned}$
        end if
        continue
    end if
    for each unassigned task $j$ do
        $v \leftarrow$ new node
        $v.\text{level} \leftarrow u.\text{level} + 1$
        $v.\text{assigned} \leftarrow u.\text{assigned} \cup \{(u.\text{level} + 1, j)\}$
        $v.\text{cost} \leftarrow u.\text{cost} + \text{cost}[u.\text{level} + 1][j]$
        Create reduced matrix redCost by setting row $u.\text{level} + 1$ and column $j$ to $\infty$
        $v.\text{bound} \leftarrow v.\text{cost} + \text{computeBound}(\text{redCost})$
        if $v.\text{bound} < \text{minCost}$ then
            $Q.\text{insert}(v)$
        end if
    end for
end while
Return minCost, bestAssignment
Procedure computeBound(matrix):
Initialize bound $\leftarrow 0$
Copy matrix to temp
for each row $i$ do
    Subtract minimum value in row $i$ from all elements in row $i$
    bound $\leftarrow$ bound $+$ minimum value
end for
for each column $j$ do
    Subtract minimum value in column $j$ from all elements in column $j$
    bound $\leftarrow$ bound $+$ minimum value
end for
Return bound
```

The algorithm assigns $W_1 \to T_3$, $W_2 \to T_1$, $W_3 \to T_2$, with costs $8+9+10=27$. The branch and bound tree prunes nodes with bounds exceeding 27.

7 Limitations and Extensions

- **Limitations**:
  - Exponential worst-case complexity for large n.
  - Memory-intensive for large matrices and deep branching.
- **Extensions**:
  - **Hungarian Algorithm**: Use for $O(n^3)$ time complexity.
  - **Parallel Processing**: Distribute branch exploration for large instances.
  - **Approximation Algorithms**: Apply for near-optimal solutions in time-sensitive applications.

8 Implementation Considerations

- **Bound Tightness**: Use matrix reduction to compute tight bounds, enhancing pruning efficiency.
- **Priority Queue**: Implement a min-heap (e.g., `std::priority`$_q ueueinC++)toprocessnodeswiththe$
  $Handlelargecostswithappropriatedatatypes(e.g.,$ `long long`$)toavoidoverflow.$
- **Matrix Management**: Store reduced matrices efficiently to minimize memory usage.
- **Output Format**: Represent assignments as worker-task pairs for clarity.

9 References

- GeeksForGeeks: Assignment Problem
- Wikipedia: Assignment Problem