CS312: Artificial Intelligence Laboratory
AO* Algorithm

Soumya Srividhya Doma, 170010038
Anudeep Tubati, 170010039

OVERVIEW

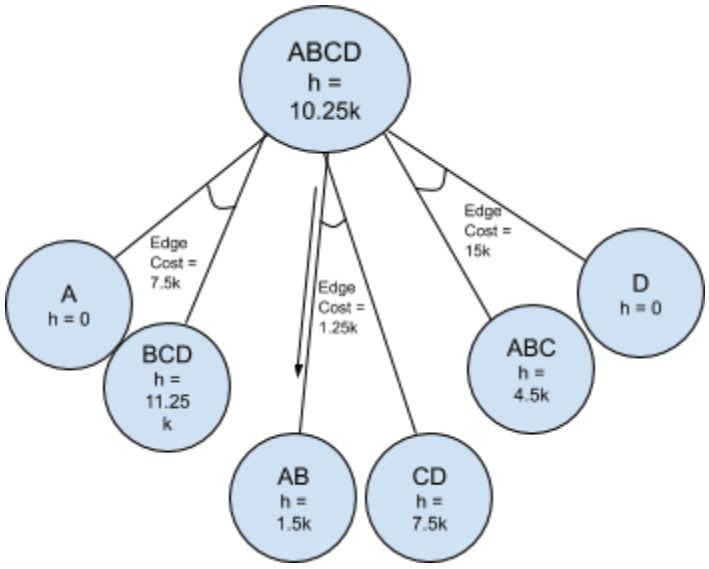
This report elaborates on the implementation of the AO* algorithm on the domain **Optimal Matrix Multiplication**.

I. DOMAIN

Given an *n* number of matrices, return the cost (or order) of multiplying them to achieve the least number of operations.

For eg., consider 4 matrices A (10x30), B (30x5), C (5x60), D (60x25). If the order is ((AB)C)D, the number of calculations is 10x30x5 + 10x5x60 + 10x60x25 = 19500. Whereas, (A(BC))D gives 30x5x60 + 10x30x60 + 10x60x25 = 42000.

We construct the graph with each node as the matrices it has to multiply. Hence root will be all the matrices. For the example above, the graph would look like this (only first level shown)



II. HEURISTIC FUNCTIONS USED

a. Underestimating Heuristic

Consider a node with matrices A, B and C. Since 2 of them have to be multiplied in some way or the other (i.e., (AB) or (BC)), we take the **minimum** of all such pairs.

underestimate(node):
    return min{numOps(i, i+1) | i from 1 to n - 1} // n = number of matrices in *node*

b. Overestimating Heuristic

Consider a node with matrices A, B and C. Since 2 of them have to be multiplied in some way or the other (i.e., (AB) or (BC)), we take the **maximum** of all such pairs.

overestimate(node):
    return max{numOps(i, i+1) | i from 1 to n - 1} // n = number of matrices in *node*

Note - numOps(A, B) returns the number of operations required to compute the product of A and B

III. AO* ALGORITHM ANALYSIS

	Underestimate	Overestimate
A (40x20) B (20x30) C (30x10) D (10x30)	26000	36000
A (40x20) B (20x30) C (30x10) D (10x50) E (50x30) F (30x20)	43000	55000
A (68x34) B (34x51) C (51x17) D (17x51)	127738	176868
A (84x42) B (42x63) C (63x21) D (21x105) E (105x63)	379701	472311

As we can observe from the table, the overestimating heuristic gives erroneous results. This is owed to the fact that it picks the minimum of children, who are evaluated by taking the **max** of the number of calculations in each pair product of the child. Since the heuristic gives a max of such pairs, it might be possible that the optimal product was on a different pair and hence lesser. However, if the other node picked by max is solved, then the solution is back propagated and hence the lesser cost is missed out.

The underestimating heuristic solves this by taking the **min** of the number of calculations in each pair product of the child. This way, if the selected node wasn't actually optimal, its cost will increase when its successors' cost is known. The new cost will be greater than the underestimated cost of a possibly optimal solution, as the underestimated cost just selects the **min** pair products and the new cost will include that minimum plus the operations for other matrices. This will make the algorithm pick different nodes, till it finds an optimal node.