Christian B. Mendl, Richard M. Milbradt

Tutorial 1 (Matrix chain ordering problem)
Matrix multiplication is associative, so given three matrices A, B and C, $(AB)C = A(BC)$. However, when it comes to the computational cost of these two expressions, one may be more efficient than the other.

(a) Write down the element-wise notation of matrix multiplication and consider its computational cost.

(b) Given three matrices $A \in \mathbb{R}^{10\times 30}$, $B \in \mathbb{R}^{30\times 2}$ and $C \in \mathbb{R}^{2\times 7}$, compare the computational cost of $(AB)C$ and $A(BC)$.

(c) If we add one more matrix D to the product chain, how many orderings to compute $ABCD$ are there? (It may be useful to draw a tree diagram).

(d) Argue why using a brute force approach to select the least expensive ordering requires an exponential run-time in the number of matrices.

(e) One recursive algorithm to select the best ordering works as follows:

1. Take the chain of matrices and split it into two subsequences.
2. Find the minimum cost of multiplying each subsequence.
3. Compute the final cost of multiplying the whole matrix chain.
4. Repeat for each possible split of the chain and choose the one with minimum cost.

Is this any better than the brute force approach? Why / why not?

(f) Say we want to find the optimal ordering of the matrix product $ABCDE$ and we use the above algorithm. Identify any redundant operations performed and propose a solution.

(g) Analyze how the following algorithm fits with the answer to the previous question:

```
// Matrix A[i] has dimension dims[i-1] x dims[i] for i = 1..n
MatrixChainOrder(int dims[])
{
    // length[dims] = n + 1
    n = dims.length - 1;
    // m[i,j] = Minimum number of scalar multiplications (i.e., cost)
    // needed to compute the matrix A[i]A[i+1]...A[j] = A[i..j]
    // The cost is zero when multiplying one matrix
    for (i = 1; i <= n; i++)
        m[i, i] = 0;

    for (len = 2; len <= n; len++) { // subsequence lengths
        for (i = 1; i <= n - len + 1; i++) {
            j = i + len - 1;
            m[i, j] = MAXINT;
            for (k = i; k <= j - 1; k++) {
                cost = m[i, k] + m[k+1, j] + dims[i-1]*dims[k]*dims[j];
                if (cost < m[i, j]) {
                    m[i, j] = cost;
                    s[i, j] = k; // index of the subsequence split that achieved minimal cost
                }
            }
        }
    }
}
```

Solution

(a) Given a matrix-matrix multiplication $C = AB$, where $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$, each element in C is given by:

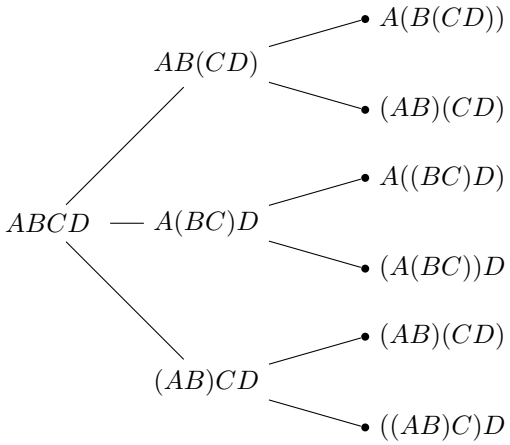
$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

Therefore, the computational cost of computing C is $\mathcal{O}(mnp)$.

(b) For $(AB)C$ we first compute AB, with a cost of $10 \times 30 \times 2 = 600$. The resulting matrix is in $\mathbb{R}^{10 \times 2}$, so multiplying it with C has a cost of $10 \times 2 \times 7 = 140$. So computing $(AB)C$ takes $600 + 140 = 740$ operations.

Similarly, $A(BC)$ takes $30 \times 2 \times 7 + 10 \times 30 \times 7 = 2520$ operations.

(c) There are five different orderings:



(d) From the previous tree diagram one can see that including further matrices increases the depth of the tree. Therefore, the number of possible orderings increases exponentially with the number of matrices.

(e) No, it is not. Recursively computing the minimum cost for each subsequence means that will end up computing the cost of each possible ordering.

(f) Consider two splits of the main chain: $A(BCDE)$ and $(ABCD)E$. The algorithm has to compute the minimum cost of $BCDE$ and $ABCD$. For each of them it will need the minimum cost of BCD. A naive implementation of the algorithm would perform the computation twice. However, a simple fix is to store in memory the minimum cost of each subsequence that has already been computed. It turns out that this reduces the runtime to $\mathcal{O}(n^3)$, n being the number of matrices in the sequence.

(g) The code finds the optimal order for a sequence of length n using previously computed optimal sequences of length $n - 1$ that have been stored in memory.