Analysis of the Multiplicative Method for Finding Binomial Coefficients

Ogbonnaya C Ngwu
Florida Institute of Technology,
Melbourne, FL. USA.
ongwu2014@my.fit.edu

***Abstract* – The concept of binomial coefficient is really important in the fields of combinatorics, a mathematical field focusing on the study of discrete structures that are countable or finite. Although various method of calculation binomial coefficient given index *n* and *k* exits, this study focuses on one of the most efficient one – the multiplicative formula. The general idea of solutions for binomial coefficients is discussed with major focus on the formula of interest. The algorithm for the multiplicative method is examined to elaborate the ideas behind the method and how part of the algorithm work to achieve efficiency. JProfiler, a java profiling tool, is then used to collect runtime data to analyze the algorithm. The test model is compared to the theoretical model using a regression model and the coefficient of determination, $r^2 = 0.87322$ is used to evaluate the amount of variance in the test model accounted for by the theoretical model. The study concludes the test model has the same complexity as the theoretical model with an error margin of about 12.68%.**

***Keywords* – Binomial coefficient, Multiplicative Method, Profiling, Time complexity, Mathematical Models**

I. INTRODUCTION

Binomials are used to describe the set of polynomials consisting of summation of two terms (each term being a monomial on their own). The algebraic expansion of the power terms of a binomial and the procedure of this expansion is stated by the binomial theorem. Such expansion always results in the production of coefficient in the terms of the expanded polynomial. These coefficients are called binomial coefficients.

In mathematics, binomial coefficients are usually indexed by two positive integers, *n* and *k* with *n* being greater or equal to *k* as a restriction. Binomial coefficients are written in one of the following forms: $\binom{n}{k}$ *or* C_k^n
Where *n* and *k* are the indexing integer.
$\binom{n}{k}$ refers to a specific binomial coefficient in a binomial expansion. It refers to the coefficient of the x^k term of the binomial power $(1+x)^n$.

They are many formulas that can be used to calculate the exact binomial coefficient corresponding to the indexes *n* and *k*. The factorial method simply follows the factorial definition of a binary coefficient:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!} \qquad for\ 0 \le k \le n$$

Computationally, this procedure is quite hectic considering the amount of effort used in multiplicative arithmetic of factorials. Perhaps a better method is the use of the recursive definition:

$$\binom{n}{k} = \begin{cases} \binom{n}{0} = 1 & for\ k = 0 \\ \binom{n-1}{k-1} + \binom{n-1}{k} & for\ 1 \le k \le n-1 \\ \binom{n}{n} = 1 & for\ k = n \end{cases}$$

This solution can be improved on with the application of dynamic programming. It can be observed that the recursive formula computes some sub-problems multiple times. This is not an issue for minute values of *n* but it becomes problematic as the value of *n* increases. At a glance, the recursive solution has two properties of a dynamic programming problem – overlapping sub-problems and optimal substructure. Hence, the solution can be optimized using an array that stores results of sub-problems to prevent recalculations.
Another solution for this problem is the use of Pascal's Triangle. This solution involves construction of a pascal triangle and then binomial coefficients can be looked up at constant time – O(1). However, most computational time is designated to the construction of the triangle. A very efficient way to calculate binomial coefficient is the multiplicative formula:

$$\binom{n}{k} = \prod_{i=1}^{k} \frac{n+1-i}{i}$$

Evaluating the numerator results in the number of choosing *k* items form *n* items while maintaining the selection order (n^k) whereas the denominator is the total sequences of the ways that the *k*-items can be combines without taking order into consideration. The programming solution of this method is explained for better details.

II. Algorithm Overview

The program for computing binomial coefficient using the multiplicative method is show below:

```
public static long  computeCoefficient(int n,int k) {
    long coeff = 1;
    k = k > n-k ? n-k : k;

    for(int i = 1; i <= k; i++, n--) {
        if (n % i == 0) {
            coeff *= n / i;
        }
        else if (coeff % i == 0) {
            coeff = coeff / i * n;
        }
        else {
            coeff = (coeff * n) / i;
        }
    }
    return coeff;
}
```

Fig. 1 Multiplicative method algorithm for finding binomial coefficients.

The solution above is based on the concept of Pascal's triangle. The coefficient of interest is initially initialized to 1. This makes sense since 1 is the first coefficient in every Pascal's triangle. As previously described, the entries of binomial coefficients form the Pascal's triangle. Pascal's triangle is symmetric about a center axis. The coefficient we are evaluating is bound to fall on one side of this triangle. Hence, it makes sense to move k into the left half. This is achieved using the conditional statement:

```
k = k > n-k ? n-k : k;
```

The program above aims to calculate the result using the standard factorial formula:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!} \qquad for\ 0 \le k \le n$$

Observe that this can be rewritten and terms occurring in both numerator and denominator canceled out:

$$\binom{n}{k} = \frac{n(n-1)(n-2)\ldots(n-k)\ldots1}{(n-k)(n-k-1)\ldots1*k(k-1)\ldots1} = \frac{n(n-1)(n-2)\ldots(n-k+1)}{k!}$$

At this point, the numerator and denominator both have equal number of terms (k-terms to be exact). Iteratively, the solution is now simplified as follows;

```
coeff  = 1;
coeff *= n / 1;
coeff *= (n-1) / 2;
coeff *= (n-2) / 3;
.
.
.
coeff *= (n-k+1) / k;
```

The algorithm takes three steps at each iteration:

1. `coeff` is multiplied by `n`
2. `n` is decremented by 1 (`n--`)
3. `coeff` is divided by `i` (`i` runs from 1 to `k`)

They are three conditions for each iteration. The first condition, `if(n%i == 0)`, keeps the result at the smallest possible value given that `n/i` is computable. If the later `n/i` is not computable, then we calculate `coeff/i` based on the second condition, else `if(coeff%i == 0)`. The last condition is used when both ways of keeping the result small is not computable. Here, we are left with the worst choice that involved multiplication first and then finally division.

The reduction of the problem into k-terms and the use of conditions to reduce result size all contribute to making this algorithm efficient. The algorithm is then analyzed for performance.

II. Time Complexity: Sample, Measure, Result and Error Analysis

A profiler is used to obtain runtime date required to measure the performance of the algorithm. For this, JProfiler, a commercially licenses profiling tool for java programs and applications, is used. With JProfiler, it is possible to isolate a single method and test its performance without interference from other parts of the code or even the garbage collector. The sample is incremented ranges from 10 to 200000 for n and 5 to 100000 for k. For each test input the value of k is always half the value of n. This criterion was chosen in order to access performance of the algorithms at midpoint instead of endpoints. The data collected was recorded in Excel. To unify the input size, n and k is added to form the independent variable (n+k). Time was measured in microseconds (µs). Below is the graph of the performance of the algorithm:

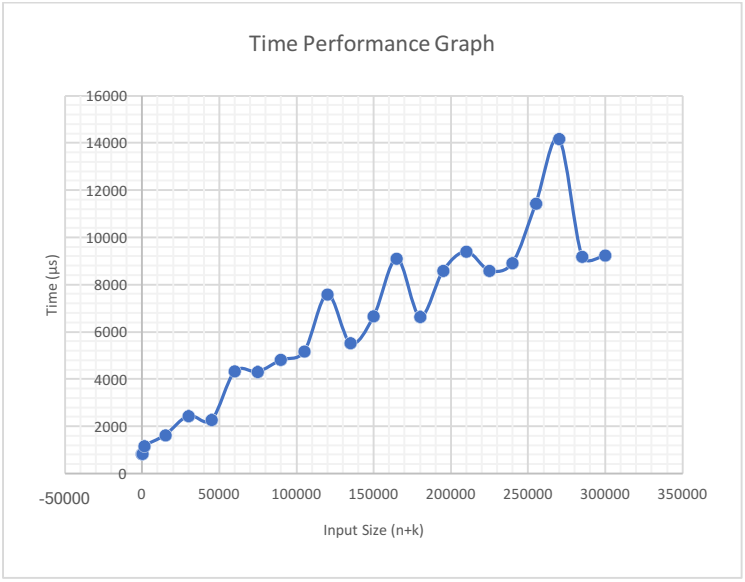

Fig. 2 Time performance graph of multiplicative method.

Theoretically, the performance of the algorithm used is given by O(k). The algorithms time complexity is linear. To compare the test model to the theoretical model, a graph of the theoretical model is superimposed on the previous graph as show below:

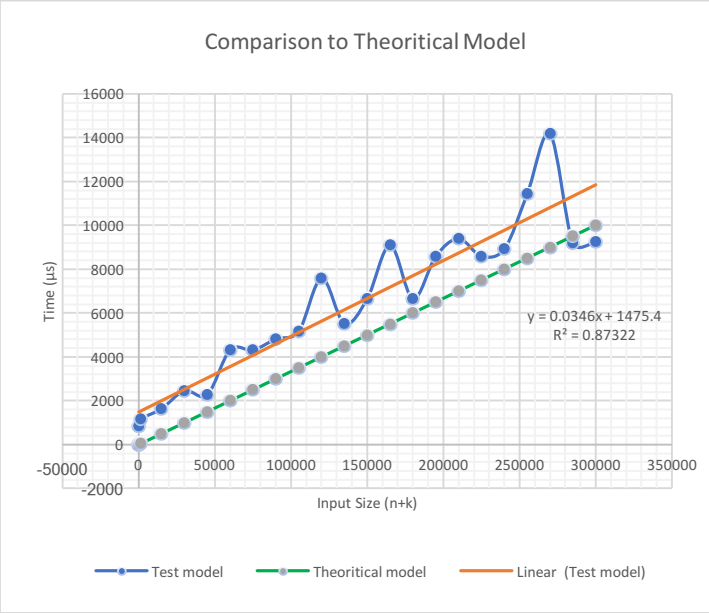

Fig. 3 Comparison of the test model to the theoretical model.

Through observation, it is quite easy to recognize that the test model is quite close to the theoretical model. A line of best fit is then plotted and the coefficient of determination is obtained. From the graph above, the line of best fit is parallel to the theoretical model with a vertical displacement of approximately 1500µs. The coefficient of determination, r^2 = 0.87322, implying that the graph of y = 0.0346x +1475.4 (regression model) accounts for 87.32% of the variance of the test model. Since the regression model is parallel to the theoretical model, it is concluded that 87.32% of the variance in the test model is accounted for by the theoretical model. In other words, the test model matched the theoretical model with a non-significant error of approximately 12.68%. A numerical error analysis can be performed to support this data.

REFERENCE:

[1] J. L. Gross, *Combinatorial methods with computer applications*. Chapman & Hall: CRC Press, 2007.