CS3052 — Computational Complexity

Practical 1 : Analysis of Matrix Multiplication

Susmit Sarkar ss265@st-andrews.ac.uk
Adriana Wilde agw5@st-andrews.ac.uk

Due: 2018-03-09 (Fri Week 6) 21:00

Key Points

This is **Practical 1**, in which I ask you to analyse the complexity of matrix multiplication algorithms.

Due date: Friday Week 6 (MMS is authoritative)

Credit: This practical is worth 50% of the coursework.

Submission: Upload a zipfile with your report in PDF and any supplementary information (code, data, spreadsheets, etc.) to MMS

Purpose

This practical will help practice and develop your skills in:

- analysing scaling behaviour of real-world algorithms;
- analysing theoretical asymptotic complexity;
- researching complexity improvements for particular data sets;
- understanding how and when asymptotic complexity improvements matter.

For this, we will study a simple but very important and widely-used problem, that of calculating the Matrix Multiplication of matrices.

Overview and Background

Matrix multiplication is a fundamental operation of linear algebra, and various applications in engineering, computational science, and computational finance require solving this problem numerically. Thus matrix multiplication algorithms are a workhorse of packages for signal processing, data mining, simulations, and so on.

Matrices are mathematically defined as rectangular tables of numbers, arranged in rows and columns, for example:

$$\mathbf{A} = \begin{bmatrix} 1 & -5 & 10 \\ 12 & 2 & -30 \end{bmatrix}$$

They are written here with bold capitals, and the elements are written indexed by the row and column $\mathbf{A}_{ij}$. We will count starting with 0. (For example, with the matrix above, $\mathbf{A}_{12} = -30$).

For simplicity, we consider matrices with the number of rows equal to the number of columns (so-called square matrices). Two matrices $\mathbf{A}$ and $\mathbf{B}$, with the same number of rows (and columns) n are multiplied to give $\mathbf{A} \cdot \mathbf{B}$ according to the following specification:

$$(\mathbf{A} \cdot \mathbf{B})_{ij} = \sum_{k=0}^{n-1} \mathbf{A}_{ik} \times \mathbf{B}_{kj}$$

As could be expected from such a fundamental problem in so many areas, much effort has been expended in finding good algorithms to calculate matrix product, under various assumptions. In particular, in many practical cases, especially in data science, the matrices have a large proportion of elements which are zero (so-called sparse matrices). For concreteness, we will say a matrix is sparse if the majority of elements are zero. This practical aims to explore matrix multiplication algorithms, particularly in the sparse matrix case.

A basic algorithm for matrix multiplication, often taught in introductory linear algebra classes and introductory numerical programming classes, is to use the definition above directly. We will call this the **Basic Algorithm**. An example implementation is given in the `multiply` method in the supplied file `BasicMultiplier.java`.

Specification

Task 1

Analyse the Basic Algorithm for Matrix Multiplication for the worst-case time complexity. You may use the provided code in `BasicMultiplier.java` (or any alternative implementation you write or find on the web). This task is to give a theoretical worst-case guarantee, and will be assessed by the report.

Task 2

Analyse and estimate the average-case performance of the Basic Algorithm by generating random test inputs of various sizes and doing data analysis. You will have to generate harness code to generate random matrices, to find the time taken for the multiplication procedure, and then do some data analysis to find the behaviour as a function of input size. Submit your harness code, any scripts/spreadsheet input used for the data analysis, and the results of that data analysis. This task will be assessed by the report together with any supplementary material submitted.

Task 3

Research data structures and algorithms that can save time taken, or memory used, or both, for sparse matrix multiplication. Note that we care about the sparse case, so you should think carefully about how you can save time or memory in the case most elements are zero. In your report, describe the results of your research and your sources. This must be backed up by an implementation of at least one idea, in any programming language of your choice. You can either write your own, or reuse code from elsewhere with proper attribution. The assessment will be based on the report.

Task 4

Investigate in what situations your implementation from Task 3 does better than the Basic Algorithm, and in what situations (if any) the Basic Algorithm wins. This will involve similar data analysis as Task 2, but you will have to decide the characteristics of the input you want to modify. Describe in the report your testing strategy and your results.

Task 5 – Extension

Give a theoretical justification for your results from Task 4, doing a complexity analysis based on input sizes and the characteristics of sparse matrices. This will be assessed by our judgement on how you understand the analysis, so simply copying down complexity analyses from published sources is unlikely to convince us that you understand the analysis and principles behind it.

Notes

- You may wish to implement your own versions, or use code downloaded from the web. If you wish to program or use programs in a language other than Java, that's absolutely fine. Whether you program in Java or something else, it should be possible to build and run your programs from the command line on the lab machines. It should be clear how to compile and run your programs, from your report or a README or similar.
- In Tasks 2 and 4, you are asked to find experimentally the average case complexity. There are various good tools that can be used for data analysis, including Excel, MATLAB, and R (roughly in that order of power & complexity). You are free to use any tool, as long as we understand how you calculated the empirical average case complexity.
- Sparse Matrix Multiplication has been very well-explored, both in the academic and in the popular scientific literature. We are asking you to do your own research, so please make use of any and all resources you can find, with proper attribution. However, for getting high marks you have to convince us you understood and can apply that research.

Submission

The primary output will be a **report**, which should be in PDF. You are encouraged (but not required) to use LaTeX. We will also require supplementary files to understand how you did your experiments and your analysis. As noted in the individual tasks, submit your code, any data analysis scripts, the data itself and the results of the analysis. Remember that being able to independently reproduce both your experiments and your data analysis is fundamental to good science.

For programs and scripts, submit the **source code** you wrote (e.g. `.java` files).

For **data**, present it in whatever format your tools use, text-based being good for accessibility. CSV or JSON are suggested standard alternatives.

Make a **zip archive** of all of the above, and submit via MMS by the deadline.

Marking

We are looking for:

- correct, properly justified, and well-argued theoretical arguments for the worst-case time complexity;
- a careful analysis of the average cases, using a range of inputs as appropriate;
- evidence of researching appropriate sources and a critical analysis of the sources for sparse matrix multiplication; and
- demonstrated understanding of key characteristics of the algorithms, data structures, and input characteristics for comparing complexities;

Tasks 1 and 2 serve as the basic requirements, and it will be possible to get marks up to 10 doing just those. The quality of research and critical analysis in Tasks 3 and 4 count for the rest of the marks. It is possible to get high marks (up to 18) even without attempting Task 5, and related work going beyond the specification will be rewarded. The standard mark descriptors in the School Student Handbook will apply:
https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors

Lateness

The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):
https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties

Good Academic Practice

The University policy on Good Academic Practice applies:
https://www.st-andrews.ac.uk/students/rules/academicpractice/