School of Informatics

Informatics Project Proposal
Implementing efficient (randomized) algorithms for comparing integers succinctly represented by arithmetic circuits

B181697
May 2021

Abstract

Comparing integers accurately is an essential task in Computing, yet when integers are so big or so small, traditional methods fail. This project focuses on implementing the best know algorithm for checking the equality of two arithmetic circuits with addition and multiplication as operations. Arithmetic circuits or Straight Line Programs are integers represent by a sequence of operations. The best know algorithm has a randomized component that yields the correct result with high probability. We will extensively test it to find the optimal parameter so we can achieve the best results.

Date: Monday 17th May, 2021

Tutor: Dr. Borislav Ikonomov
Supervisor: Dr. Kousha Etessami

Contents

1 Motivation

Comparing integers such as deciding if one is bigger than the other or if they are equal is an essential task in Computing. This project focuses on comparing integers represented succinctly as arithmetic circuits or Straight-Line Programs (SLP) with binary integers as inputs. An example of an arithmetic circuit is the following:

$$x_0 = 1$$

$$x_1 = x_0 + x_0$$

$$x_2 = x_1 * x_1$$

$$x_3 = x_2 - x_0$$

$$\dots$$

$$x_n = \dots$$

We see in the example that the arithmetic circuit has $n+1$ gates or equalities and uses addition, subtraction and multiplication as operations. Formally an n+1-variable arithmetic circuit is a directed acyclic graph made of input nodes labelled by a variable name from the set $x_0, x_1, ..., x_n$, and made of non-input nodes labelled by an operation symbol of in-degree equal to two (Section 7.2.2 [1]). Arithmetic circuits ultimately represent integers broken down into many operations. For simplicity, we will assume that an SLP will start with $x_0 = 1$. Using arithmetic circuits it is easy to build huge integers without writing their values explicitly.

Comparing integers might sound trivial as computers are already able to do this using a Digital Comparator. A Digital Comparator will decide whether an integer is greater than, less than

or equal than the other. The integers are first translated into binary and then using Boolean logic compared. This works perfectly when integers can be stored in 64-bit (standard range) but consider the following arithmetic circuit that only uses addition and multiplication:

$$x_0 = 1$$

$$x_1 = x_0 + x_0$$

$$x_2 = x_1 * x_1$$

$$x_3 = x_2 * x_2$$

$$x_4 = x_3 * x_3$$

$$\dots$$

$$x_n = x_{n-1} * x_{n-1}$$

This arithmetic circuit is equivalent to 2^{2^n}. If we let $n = 10$, then we have $2^{2^{10}}$. This integer requires exponential space to be written in binary and is too big for 64-bit.

In this project, we will focus on big integers. We want to compare them, as their binary representation might not fit in memory we use SLPs with binary integers as inputs to represent them. So far it has been possible to find algorithms capable of comparing certain type of circuits using Number Theory Theorems and Properties. If the arithmetic circuits only use multiplication as an operation there is a deterministic algorithm that can check for equality in polynomial-time, and another deterministic algorithm that checks for inequality assuming that deep number-theoretic conjectures are true [2]. The more challenging case is when we allow addition and multiplication as operations. For equality testing Schönhage [3] describes an efficient polynomial-time algorithm that requires the use of randomization, this is the best we have found so far and de-randomising it will have a deep impact in many fields especially for Polynomial Identity Testing (PIT). For inequality there is no algorithm capable of comparing the two circuits, this problem is in the complexity class $p^{pp^{pp^{pp}}}$ [4] (the fourth level of the counting hierarchy).

The scope of this project is to consider arithmetic circuits with addition and multiplication as operations. We want to implement a working version of the algorithm described by Schönhage [3], such that we can check if two straight-line programs are equal (this problem is defined as equality straight-line program EquSLP). The algorithm is randomized meaning that each time we run it we have to pick a big enough random number to compute a result that will yield the correct answer with high probability.

1.1 Problem Statement

Comparing very big or very small integers is not as straightforward as one might think, the main problem is not being able to write the numbers in 64-bit or even write them in memory. Arithmetic circuits let us represent such huge integers more succinctly. Having a different representation of integers means we also need a different method for comparing them. There are algorithms for comparing such circuits but many have not been implemented and/or properly tested.

1.2 Research Objectives

This project will have as objectives to fully understand the subject and the randomized equality check algorithm for circuits with {+,*} as operations. Implement a working version of it (as describe in [3]) so we can solve EquSLP with a high probability of success. Create families of succinctly represented integers capable of challenging the algorithm and test the performance. A deep analysis of the complexity class of the algorithm and the probabilities of success. Understand the randomized component and find the relation between the size of the selected random integer and the size of the inputs (i.e. what should be the size of the randomized component depending on the size of the input to obtain good probabilities of success).

This will ultimately create a tool that can be used to check for equality between two arithmetic circuits and be a step forward towards solving ACIT (arithmetic circuit identity testing). The project will not address the algorithms for comparing arithmetic circuits when only multiplication is used as an operation. This is being implemented in a parallel project by a fellow student.

1.3 Significance and Novelty

Algorithms have been developed for this intricate problem but they have not been extensively tested and implemented. The project will focus on testing with the goal of finding a relationship or understand more the randomized aspect of the algorithm. Finding insights on this will deeply help better implementations that are more efficient and more accurate so they can be used in real use cases or for further studies connecting to other algorithms or de-randomization of the problem.

Use cases for such an algorithm or similar ones connect to Polynomial Identity Testing and dealing with very small or very large integers. We see in Dr Etessami's paper [2] possible scenarios where one would need to compare extremely large or extremely small numbers, such use cases are for example modelling genomic sequences or computing the maximum probability parsing for stochastic context-free grammars (SCFG). There are other examples that deal with only arithmetic circuits with addition and multiplication such as exact numerical arithmetic in game theory for computing Nash equilibrium with three or more players.

1.4 Feasibility

Implementing the algorithm is very straightforward. The real challenge is creating challenging test cases to analyse performance and understand the relation between the size of the input and the size of the randomised component. There are no explicit examples of very similar arithmetic circuits that could trick or really push the algorithm. Creating such tests that try to break the algorithm could be a very time-consuming task that could yield unsuccessful results. We will focus on experimental results and even if finding very challenging tests is not possible the data that would be gathered is still of extreme value, not being able to challenge the algorithm would indicate that it is very robust and performs very well under usual use cases.

1.5 Beneficiaries

This project will be of great value for researchers in the field and for concrete applications of Polynomial Identity Testing. It will develop more experimental result and hypothesis for possible lower bounds on the randomized component of the algorithm. This will help future research or applications for big integer comparisons to have more accurate and better-performing algorithms.

2 Background and Related Work

We are going to define the following two problems as presented by Allender et. al. [4]:

- EquSLP (equality straight-line program) : given a straight-line program representing an integer N , decide whether $N = 0$. This translates to the problem of testing equality of two arithmetic circuits that represent integers.
- ACIT (arithmetic circuit identity testing) : given a straight-line program representing a polynomial $f \in \mathbb{Z}[X_1, ..., X_k]$, decide whether $f = 0$. This translates to the problem of testing equality of two arithmetic circuits that represent polynomials.

These are the problems we want to solve. They have a direct connection to Polynomial Identity Testing (PIT) which is the problem of deciding given a polynomial if it is equal to the zero polynomial or if two arithmetic circuits representing polynomials are equal (Section 7.2.2 [1] and Section 12.4 [5]). PIT is a central problem in Complexity Theory and algorithm design. It has connections to many algorithms based on identity testing such as primality testing algorithms. PIT was used to prove Complexity classes equalities such as IP = PSAPCE or MIP = NEXPTIME (Chapter 4 [6]). We can see that PIT is equivalent to ACIT.

As PIT is so important there has been a lot of work around finding a deterministic algorithm for it. Finding one or doing some progress towards it would be a breakthrough. Unfortunately, there hasn't been much progress. The latest addition was done by Kabanets and Impagliazzo [7] showing that a deterministic algorithm will give circuit lower bounds on arithmetic complexity. So far many randomized algorithms have been proposed such as the Schwartz-Zippel algorithm [8, 9, 10] that is based on the concept of replacing the variables in the polynomial by random values from a large enough domain, we will get a zero value if the polynomial is the zero polynomial with a high probability. Another randomized algorithm is the Agrawal-Biswass Algorithm [11] which is more sophisticated and uses the Chinese-Reminder Theorem to reduce the degree of the polynomial.

These algorithms work and have good probabilities of success but Allender et. al. [4] shows that ACIT is polynomial-time equivalent to EquSLP. Why is this so important? We can prove that EquSLP is a special case of ACIT. EquSLP is in coRP as we can use the algorithm expressed by Schönhage [3] to solve it, and ACIT is also in coRP [12, 8, 10, 9]. This indicates that if we find a deterministic algorithm for the special case EquSLP of ACIT, we would be able to find circuit lower bounds. Using Schönhage [3] simpler algorithm for EquSLP would help us solve ACIT more easily without having to implement more complex algorithms such as the two mentioned

above (Schwartz-Zippel and Agrawal-Biswass).

This project focuses on implementing the algorithm described by Schönhage [3] that focuses on calculating each gate of the arithmetic circuit modulo a random number. We will describe this algorithm further:

First note the following Lemma [3]: there exits a constant $c > 0$ such that for any integers $x \neq y$ of length n generated by a straight-line program with operations $\{+,*\}$, there are more than $2^{cn}/cn$ integers $R < 2^{cn}$ with $x \not\equiv y \bmod R$. This Lemma ensures that we can select a suitable random integer R and gives us an upper bound $R < 2^{cn}$.

The algorithm: randomly chooses a suitable large enough prime number p (this will work with a non-prime, but prime numbers give better results). Compute both arithmetic circuits mod p, compare the result for both in polynomial time. If we have the same value ($x \equiv y \bmod p$) then the algorithm answers yes for equality with high probability. If they are not equal ($x \not\equiv y \bmod p$) then the algorithm always answers no for equality. Hence if $x = y$ then the algorithm always answers yes, if $x \neq y$ then the algorithm answers no with high probability.

Consider the following circuit: $x_0 = 1, x_1 = x_0 + x_0, x_2 = x_1 * x_1$, and select a random integer R, then the algorithm will compute $x_0 \equiv 1 \bmod R$, $x_1 \equiv 2 \bmod R$, $x_2 \equiv 4 \bmod R$. We get an exact comparison if the gates never get bigger in value than the random integer R for both circuits, but as we are comparing huge integers to ensure this we would have to pick a huge random number as well which is not efficient.

Why R being prime is an advantage? If R is not prime then we can have the following case $x = R + 1$ and $y = 2R + 1$. This will give us $x \equiv 1 \bmod R$ and $y \equiv 1 \bmod R$ but clearly $x \neq y$. As R is random the chances of it being prime are $1/k$ when R is k bits. If R is prime then the chances of delivering the wrong answer are very small as $a * b \equiv 0 \bmod p$ (p prime) if and only if $a \equiv 0 \bmod p$ or $b \equiv 0 \bmod p$.

3 Programme and Methodology

This algorithm has never been fully implemented and tested. We will work with very big integers so we will need the programming language of choice to be able to handle big int numbers. We need a big int package that is honest and does not use rounding as we need exact values for comparison. Python can handle arbitrarily big numbers without having to explicitly state it. The GMPY extension module for Python gives us access to GMP (multiple-precision arithmetic library). Having this library we can create multi-precision integers. Hence we will use Python to implement and test the algorithm.

A challenging aspect of the Project is designing tests for the algorithm. This has never been done before, we want to understand the relation between the random prime and the input size. We consider the number of gates of the circuit as the circuit size and we call it n. What is the smallest number of bits $K(n)$ for generating a large enough random number such that we have a high probability of success? We know the upper theoretical bound and we are going

to test it but finding a function $K(n)$ that indicates the minimum number of bits needed will create a lower bound which is more useful for performance purposes. Hence having quality tests and gathering a large amount of data is needed for understanding this relationship if it even exists. We will try finding this function by applying Statistical techniques, for example, logistic regression.

We want to find a family of succinctly represented integers that are big and that we already know if they are equal or not, so we can test the algorithm and relate the performance with the input size and the size of the random prime. An example of such a family is the second arithmetic circuit in the Motivation Section (2^{2^n}). This family inspires another one that can be defined by induction: Base Case $2^{2^0} - 1 = 1$, Inductive Step $2^{2^i} - 1 = (2^{2^{i-1}} - 1) * (2^{2^{i-1}} + 1)$, where $i = 0, 1, 2, ..., n$.

4 Evaluation

Evaluation will focus on two main aspects. The first would be to calculate experimentally the success probability of the algorithm using the theoretical boundary for the randomised component. This is done by running the algorithm many times with different random numbers and counting the number of times we get a wrong result to compare to the number of times we have the correct one. The second would be to gather data on the relationship between the input size and the size of the randomized component and being able to find a function $K(n)$ that relates the two. If possible we will have a lower bound on the number of bits needed for the random number.

5 Expected Outcomes

This project has as expected outcome to identify a relationship between the input size and the randomized element size in the algorithm described by Schönhage [3] for checking equality of two arithmetic circuits with {+,*} as operations. I expect to be able to create a function $K(n)$ that reflects this relation. This function will act as a lower bound on the randomized element size, this will have a great impact on research and practical usage. It will be possible to implement, using the lower bound, the most efficient algorithm we have ever found for this problem.

6 Research Plan, Milestones and Deliverables

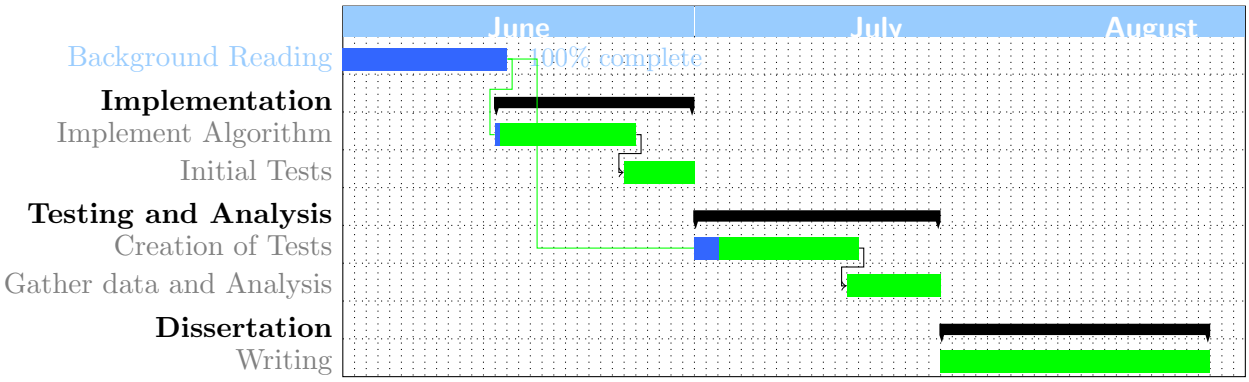

Figure 1: Gantt Chart of the activities defined for this project.

Milestone	Week	Description
M_1	2	Literature Review and Feasibility study completed
M_2	4	Implementation of the algorithm and first tests
M_3	6	Creation of tests and gather data
M_4	7	Analysis of the data
M_5	10	Submission of dissertation

Table 1: Milestones defined in this project.

Deliverable	Week	Description
D_1	4	Algorithm Code
D_2	7	Evaluation Report of Algorithm and Tests
D_3	10	Dissertation

Table 2: List of deliverables defined in this project.

References

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, Cambridge, 2009.

[2] Kousha Etessami, Alistair Stewart, and Mihalis Yannakakis. A Note on the Complexity of Comparing Succinctly Represented Integers, with an Application to Maximum Probability Parsing. *ACM Transactions on Computation Theory*, 6(2):1–23, May 2014.

[3] Arnold Schönhage. On The Power of Random Access Machines. In *Automata, languages and programming*, number 71 in ICALP'79, pages 520–529. LNCS, h.a. maurer edition, 1979.

[4] Eric Allender, Peter Burgisser, Johan Kjeldgaard-Pedersen, and Peter Bro Miltersen. On the Complexity of Numerical Analysis. *Electronic Colloquium on Computational Complexity*, 37(1):1–10, 2005.

[5] Avi Wigderson. *Mathematics and computation*. Princeton University Press, Princeton, NJ, 2019.

[6] Amir Shpilka and Amir Yehudayo. Arithmetic Circuits: a survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, page 123.

[7] Valentine Kabanets and Russell Impagliazzo. Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds. *computational complexity*, 13(1-2):1–46, December 2004.

[8] Richard A. Demillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, June 1978.

[9] Richard Zippel. Probabilistic algorithms for sparse polynomials. In G. Goos, J. Hartmanis, P. Brinch Hansen, D. Gries, C. Moler, G. Seegmüller, J. Stoer, N. Wirth, and Edward W. Ng, editors, *Symbolic and Algebraic Computation*, volume 72, pages 216–226. Springer Berlin Heidelberg, Berlin, Heidelberg, 1979. Series Title: Lecture Notes in Computer Science.

[10] J. T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *Journal of the ACM*, 27(4):701–717, October 1980.

[11] Manindra Agrawal and Somenath Biswas. Primality and identity testing via Chinese remaindering. *Journal of the ACM*, 50(4):429–443, July 2003.

[12] Gaston H. Gonnet. Determining equivalence of expressions in random polynomial time. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing - STOC '84*, pages 334–341, Not Known, 1984. ACM Press.