Senior Capstone Project

Benchmarking Speedups in Factoring through Quantum Annealing

João Marcos Vensi Basso, Faizan Muhammad

Department of Computer Science

School of Engineering

Tufts University

Spring 2020

IQC Institute for Quantum Computing

Tufts UNIVERSITY

Table of Contents

Introduction

The development of fully scalable, fault-tolerant Quantum Computers is still an ongoing research issue. However, it might be possible to achieve certain algorithmic speedups over Classical Computers using Quantum Annealing[1] in conjunction with Classical Computers.

This Senior Capstone Project explores the feasibility of such an approach in the domain of factoring problems[2]. A factoring problem is posed to the classical computer. The classical problem uses certain algorithmic techniques to rephrase the factoring problem as a boolean satisfiability problem[3]. This is finally converted to the QUBO form that the D-Wave Quantum Annealers API supports[4]. The solution provided by the Quantum Annealer is then translated back and used to find the solution to the factoring problem posed initially.

The algorithmic techniques we will be implementing have a theoretical prediction of quantifiable speedups over classical approaches. As a result, the final outcome of our capstone project would be the benchmark statistics confirming or failing to confirm that prediction.

[1] "Introduction to Quantum Annealing — D-Wave System ...." https://docs.dwavesys.com/docs/latest/c_gs_2.html. Accessed 9 Nov. 2019.
[2] "Integer factorization - Wikipedia." https://en.wikipedia.org/wiki/Integer_factorization. Accessed 9 Nov. 2019.
[3] "Boolean satisfiability problem - Wikipedia." https://en.wikipedia.org/wiki/Boolean_satisfiability_problem. Accessed 9 Nov. 2019.
[4] "Reformulating a Problem — D-Wave System Documentation ...." https://docs.dwavesys.com/docs/latest/c_handbook_3.html. Accessed 9 Nov. 2019.

Sponsor Expectations

Our sponsors from IQC have provided us with some initial code that contains circuits for some basic operations such as NAND, XOR, MUX. It is expected that we would be building upon that code with higher level operations such as GCD leading to Elliptic Curve Methods leading to the final overall circuit.

Alongside the development of such code, it is also expected that we will carry out benchmarks to confirm or fail to confirm the predictions suggested by the algorithm.

Finally, it is expected that the code and results would be presented and made available in appropriate forms; Github repositories, charts and diagrams, academic papers.

During the year, we would receive continuous guidance and support from our sponsors in any areas that we find challenging and the sponsors would also serve to review our progress throughout the year and at the end.

Technologies Used

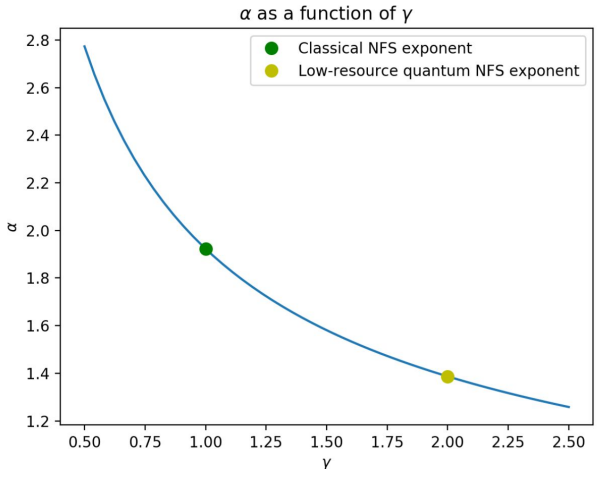
- **Haskell:** Development of Circuits to translate input problems to Boolean Satisfiability Problems
- **Python:** Translate Boolean Satisfiability Problems to QUBO form and interface with the D-Wave API to send queries and receive results
- **D-Wave Quantum Annealer:** Solving the translated factoring problem

Algorithm Overview

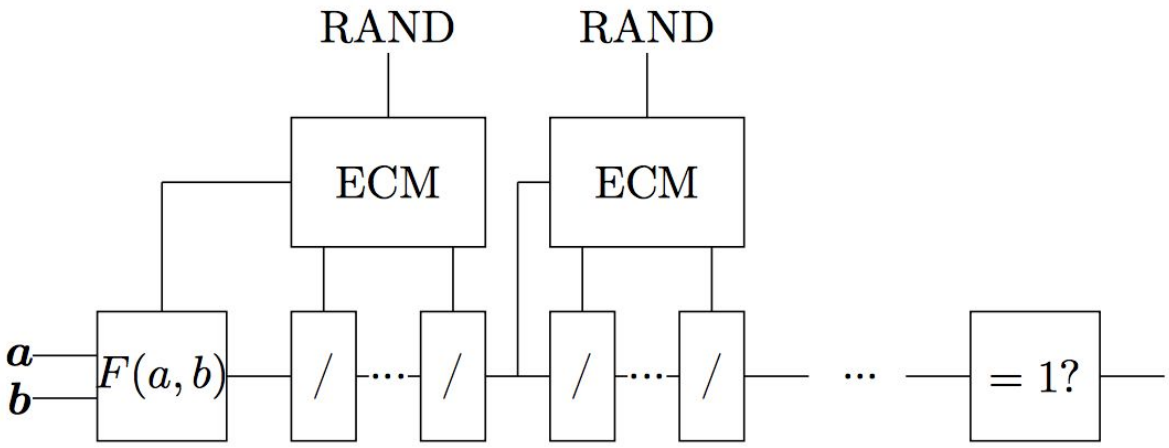
This work is based on the preprint in [1]. The paper develops circuit-NFS, an algorithm for factoring that is a modification to the low-resource quantum factoring algorithm in [2]. The low-resource algorithm modifies the Number Field Sieve (NFS) [3] by using Grover's search [4] to find smooth numbers, which are those with small prime factors.

The low-resource algorithm runs in time $L[⅓,\ 1.387]$, where $L[a,\ b] = e^{b(\log n)^a\ (\log\log n)^{1-a}}$, while the classical NFS runs in $L[⅓,\ 1.923]$. Circuit-NFS encodes the problem of finding smooth numbers as a boolean circuit and solves it with a quantum SAT solver such as a quantum annealer. If the quantum SAT solver achieves a γ speedup, circuit-NFS runs in $L[⅓, (32(\gamma+1)/9\gamma^2)^{1/3} + o(1)]$.

Note that this implies that, in the worst case of no quantum speedup, this is as fast as the classical algorithm. In the best-case scenario where the quantum SAT solver achieves a quadratic speedup, this algorithm is as fast as the low-resource one, while not requiring a fault-tolerant quantum computer. The following plot shows the scaling of α, where the algorithm runs in time $L[1/3,\ \alpha + o(1)]$.



Circuit-NFS implements the following circuit:



A boolean solution of this circuit is equivalent to a full factorization of the smooth number. A theorem in [1] shows that the scaling of this ECM circuit is not a problem since it has size upper-bounded by $L[1/6, (2\beta/3)^{1/2}]$ and probability of success $1 - o(1)$. The use of ECM is what makes the whole algorithm work, since the Elliptic Curve Method is specialized to find small factors and has a time complexity dependent on those primes, unlike other factoring algorithms.

After all the smooth numbers are found, it is just a matter of linear algebra steps to find a factor of the number being factored.

Software Architecture

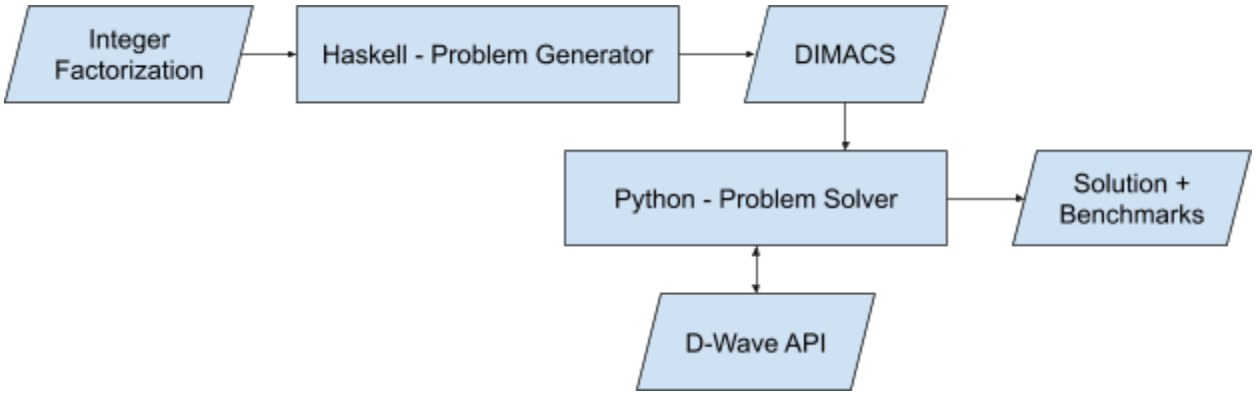
The system is divided into two parts:

- Haskell-based SAT problem generator
- Python-based SAT problem solver and benchmarker

The problem generator takes in the integer that needs to be factorized and using the algorithms outlined previously phrases this problem as a boolean satisfiability problem. The code for this part is an extension of some of the code already developed by IQC. The challenge here is designing the circuits that can perform the required operations and then representing them in Haskell. The problem generator produces a DIMACS format file.

This file is then used as the input for the solver and benchmarker developed in Python by us. It transforms the SAT problem in QUBO form for the D-Wave API and uses it to solve it. In the process, it maintains statistics regarding the difficulty and performance of the task.

The diagram below represents the high-level architecture:



User Interface

The user interface for this project would consist entirely of command-line interface. We expect the users to be able to perform the following tasks:

- Download and compile the source code
    - Download code from Github or any other online repository
    - Compile the code following the instructions in the attached README
- Present Custom Integers for Factorization
    - Be able to run the code with text files containing correctly-formatted requested integers
- Present Custom Boolean SAT problems for Quantum Annealing
    - Be able to run the code with text files containing correctly-formatted requested boolean problem
- Obtain Benchmarks for the above tasks
    - Be able to view and save graphs and diagrams demonstrating the performance in the above tasks
- Edit source code for further exploration
    - Be able to modify and extend current source code in new ways to meet their own research requirements

Error Handling

We expect four kinds of critical errors to possibly arise for the user:

- Compile Errors: the code does not compile after following the instructions
- Runtime Errors: the code crashes with valid input
- Network/API Errors: the code is unable to properly interface with D-Wave API
- Usage Errors: the code is used improperly (invalid input)

We will be performing testing to counter the first three types of errors and providing usage documentation and appropriate error messages for the user to help them overcome the final type of error.

Benchmarks and Testing

We will be benchmarking the complexity of the input problem against each step of the transformation. In other words, we would compare the complexity of the factoring problem with the complexity of the boolean satisfiability problem, the QUBO translation and the time taken by the Quantum Annealer to solve it. Finally, we would be comparing the scaling rates of the problems using our technique with that of classical computers.

For testing, initially we would be checking our circuits using SAT solvers on Classical Computers. For example, we would create the circuit for solving a particular GCD problem, run it by a classical computer SAT solver algorithm, then check the results to see if it is as expected.

Once we have the whole system functional we would be able to test the factoring problems and their solutions provided by the Quantum Annealer by comparing to the solution generated by the Classical Computers.

Implementation Plan and Timeline

Week Ending	Goal
November 22nd	Protoype GCD Circuit
November 29th	Thanksgiving
December 6th	Protoype elliptic curve addition (The most challenging aspect of the project)
December 13th	Winter break
December 20th	Winter break
January 17th	Work on circuits for loops and/or recursion
January 24th	Prototype elliptic curve scalar multiplication
January 31st	Put full circuit together
February 7th	Buffer week to catch up on previous work if needed
February 14th	Debugging and Testing (Use classical set solvers instead of quantum annealers)
February 21st	Benchmarking scalability of input
February 28th	Develop Python scripts to interact with D-Wave API and request solutions
March 6th	Run circuits on the annealer
March 13th	Run circuits on the annealer (continued)
March 20th	Spring break
March 27th	Benchmarking factoring times
April 3rd	Generation of overall benchmark plots
April 10th	Start writing paper
April 17th	Buffer week to catch up on previous work if needed
April 24th	Work on paper
May 1st	Have paper ready for review

References

[1] Michele Mosca, João Marcos Vensi Basso, and Sebastian R. Verschoor. Speeding up factoring with quantum SAT solvers. CoRR, abs/1910.09592, 2019. URL: https://arxiv.org/abs/1910. 09592.

[2] Michele Mosca and Sebastian R. Verschoor. “Factoring semi-primes with (quantum) SAT- solvers”. In: CoRR abs/1902.01448 (2019). arXiv: 1902.01448.

[3] Joe P. Buhler, Hendrik W. Lenstra, and Carl Pomerance. “Factoring integers with the number field sieve”. In: The development of the number field sieve. Springer, 1993, pp. 50– 94. doi: 10.1007/BFb0091539.

[4] Lov K. Grover. “A Fast Quantum Mechanical Algorithm for Database Search”. In: Pro- ceedings of the 28th Annual ACM Symposium on the Theory of Computing. Ed. by Gary L. Miller. ACM, 1996, pp. 212–219. doi: 10.1145/237814.237866.