Quantum Computation and Its Effects in Database Systems

Chapter · August 2020

DOI: 10.1007/978-3-030-54623-6_2

Quantum Computation and Its Effects in Database Systems[⋆]

Szabolcs Jóczik and Attila Kiss

Faculty of Informatics, Eötvös Loránd University, 1117 Budapest, Hungary
joczikszabi@inf.elte.hu, kiss@inf.elte.hu
https://www.elte.hu/en/faculties/informatics

Abstract. Classically, searching an unsorted database requires a linear search, which is $\mathcal{O}(n)$ in time. Using Grover's quantum search algorithm, it is possible to do it in $\mathcal{O}(\sqrt{n})$ time which is a quadratic speedup compared to its classical counterpart. The aim of this reasearch is to exploit this speedup and find other applications in different algorithms commonly used in database systems.

Keywords: Quantum computing · Grover's algorithm · Database operations.

1 Introduction

Today's computers—both in theory (Turing machines) and practice (PCs, laptops, tablets, smartphones,...)—are based on classical physics. They are limited by locality therefore. A quantum system can be in a superposition of many different states at the same time, and can exhibit interference effects during the course of its evolution. Moreover, spatially separated quantum systems may be entangled with each other and operations may have "non-local" effects because of this. Quantum computation is the field that investigates the computational power and other properties of computers based on quantum-mechanical principles. An important objective is to find quantum algorithms that are significantly faster than any classical algorithm solving the same problem [1]. Section 3 delivers a basic introduction to Grover's search algorithm and provides insight into how it can be used in the later chapters for database set operations. Section 4-7 proposes new algorithms that could be alternativly used in database systems. More specifically we propose a quantum algorithm for each of the following database set operations (Intersection, Set difference, Union, Projection). The majority of these algorithms are built upon the intersection operation using Grover's search algorithm. The validation is evaluated on both classical and quantum computers using IBM-Q in Section 8. Lastly in Section 9 we summarise the results we gathered using the proposed algorithms and make notes on some possible future improvements.

[⋆] The project has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

2 Related Works

Applications of existing quantum algorithms in different fields of researches have been a center of attention for many years now. Researchers have been working on developing new algorithms and finding interesting applications of them that could replace their classical counterparts in the future. There are many examples like Shor's algorithm [2] in the field of prime factorization or Grover's quantum search algorithm [3] that has applications in cryptography [5], collision problems [6] and many more [7] [8].

The motivation in this research paper was to find new applications of Grover's algorithm in database systems for basic set operations. Set operations such as intersection, difference, union, projection are fundemental building blocks in database queries that require fast and efficient algorithms. In this paper we are proposing four quantum algorithms for each of the previously mentioned operations that could possibly help in working with queries consisting of multiple operations. First we develop an algorithm for the intersection operation, then we use that algorithm to design the algorithms for set difference, union and projection. A similar work has been done before by Pang, C. Y., Zhou, R. G., Ding, C. B., and Hu, B. Q. [4]. They presented a quantum algorithm for the intersection operation with a running time of $\mathcal{O}(\sqrt{|A| \times |B| \times |C|}$, where $C = A \cap B$. The algorithm they developed is a combination of Grover's algorithm, classical memory and classical iterative computation that could be used as a starting point to develop algorithms for the other set operations as well but they were not presented in the paper. In another related work by Salman, T., and Baram, Y. [9] a similar approach was used in developing a quantum algorithm for the intersection operation.

3 Unstructured Database Search with Grover's Algorithm

Using Grover's search algorithm it is possible to find a specific item within a randomly ordered database of n items using $\mathcal{O}(\sqrt{n})$ operations (with probability $> \frac{1}{2}$). By contrast, a classical computer would require $\mathcal{O}(n)$ operations to achieve this, therefore, Grover's algorithm provides a quadratic speedup over an optimal classical algorithm and a good starting point to find applications in unstructured database systems.

3.1 Problem definition

Suppose there's an unstructured database of n items that we map to the set $\{0, 1, \ldots, n-1\}$. Among these items there is one item with a unique property ω that we wish to locate. A common way to encode such a list is in terms of a function $f : \{0,1\}^N \longrightarrow \{0,1\}$, where $n = 2^N$ defined as

$$f(x) = \begin{cases} 1 & \text{if } x = \omega \\ 0 & \text{if } x \neq \omega, \end{cases}$$

To use a quantum computer for this problem, we encode the function into a unitary matrix called an oracle. First we choose a binary encoding of the items $x, \omega \in \{0,1\}^N$, thus we can represent each item using N qubits on a quantum computer. We then define the oracle matrix U_ω to act on any of the simple, standard basis states $|x\rangle$ by $U_\omega |x\rangle = (-1)^{f(x)} |x\rangle$.

We see that if x is an unmarked item, the oracle leaves the state unaffected. However, when we apply the oracle to the basis state $|\omega\rangle$, it maps $U_\omega |\omega\rangle = -|\omega\rangle$.

3.2 Algorithm

The initialization is carried out by applying a Hadamard transform on the system to achieve a uniform superposition of all states as follows

$$|0\rangle^{\otimes N} |1\rangle \longrightarrow \left(\frac{1}{\sqrt{2^N}} \sum_{x=0}^{2^N-1} |x\rangle \right) \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

The second step is the Grover Iteration, which is repeated $\lfloor \frac{\pi}{4}\sqrt{n} \rfloor$ times to achieve optimal probability. The teration consists of the following two steps:

1.) Apply the oracle operator U_ω.
2.) Apply the Grover diffusion operator

$$G = 2|\psi\rangle\langle\psi| - I$$

In the last step a classical measurement is performed to determine the result. Since all proposed algorithms are inherited from Grover's algorithm, the optimal number of iterations in all cases equal to $\lfloor \frac{\pi}{4}\sqrt{n} \rfloor$.

We'll be also using the following function implicitly throughout the rest of the paper. Let $f : A \longrightarrow [0, \ldots, n-1]$ be an indexing function that maps the elements of A to the set $\{0, \ldots, n-1\}$. Now we can define a natural bijection between the subsets $X \subset A$ and the set of bitstrings with length n as follows

$$\begin{aligned} \mathcal{P}(A) &\longrightarrow \{0,1\}^n \\ X &\longmapsto [h(f^{-1}(0)), h(f^{-1}(1)), \ldots, h(f^{-1}(n-1))], \end{aligned}$$

where

$$h(x) = \begin{cases} 1 & x \in X \\ 0 & x \notin X. \end{cases}$$

4 Set Operation: Intersection

The intersection operator takes the results of two queries and returns only those records that appear in both result sets. A straightforward classical algorithm would be to go through each element of one of the sets and check wheter it is in the second set or not. This algorithm would give us a $\mathcal{O}(nm)$ efficiency.

In this paper we assume that the given sets are unstructured and using sorting algorithms or hash tables is not allowed or computationally not worth it. With these assumptions and the help of Grover's algorithm, it is possible to find $A \cap B$ in $\mathcal{O}(\sqrt{2^N})$ computational time using the following proposed quantum.

4.1 Quantum Algorithm

Let A and B be two sets and let's assume that $n = |A| \leq |B|$. The intersection of A and B must be in the power set of A, thus $A \cap B \subset \mathcal{P}(A)$, where $|\mathcal{P}(A)| = 2^n$. Therefore we can use excatly n bits to represent all of the subsets of A, where in this case every element of A is represented by one qubit.

Now we can make use of Grover's search algorithm to search through the subsets $X \subset A$ and find $A \cap B$ with an appropriate Oracle function O_f, such as

$$O_f(X) = \begin{cases} 1 & X = A \cap B \\ 0 & X \neq A \cap B, \end{cases}$$

4.2 Implementation

As an example let $A = \{1, 2, 3\}$, $B = \{2, 3, 4\}$ and f defined as follows

$$f(1) = 0 \quad f(2) = 1 \quad f(3) = 2$$

We'll be using this definition of f for the rest of the paper. In this case the bitstring that represents $A \cap B$ is 011. The oracle function O_f needs to flag the value 011 and leave all other values unaltered. The implementation of the full algorithm including the oracle function can be seen on Fig. 1.

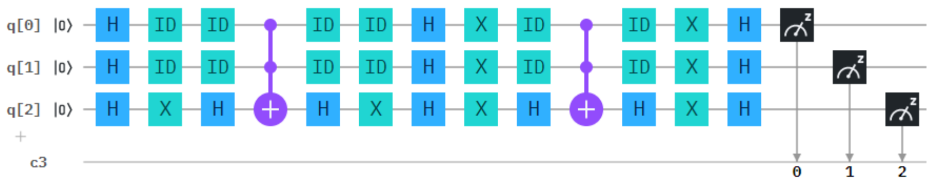

Fig. 1: Finding the intersection of A and B

5 Set Operation: Difference

The next set operation that we look at is the set difference which takes the results of the two queries and returns only those records of entries that appear only in the first query result. Although set difference operation by itself is required by only a few number of queries (i.e. `except`), it will be useful for the implementation of the set union operation in the next section. Similarly to the intersection operation, a straightforward classical algorithm would be to go through each element of the first query and check wheter it is in the second set or not. If an element is not in the second query, then it is in $A \backslash B$, otherwise it's not. This algorithm would give us a $\mathcal{O}(nm)$ computational time, which is also the worst case.

Using the proposed quantum algorithm with the help of Grover's algorithm, we can reach an $\mathcal{O}(\sqrt{2^n})$ efficiency on a quantum computer, assuming that no sorting algorithms were used.

5.1 Quantum Algorithm

It is possible to obtain $A \backslash B$ using the intersection algorithm by simplyfing the problem as $A \backslash B = A \backslash (A \cap B)$, where $A \backslash (A \cap B)$ can be viewed as the complement set of $(A \cap B)$ relative to A. Therefore the elements of $A \backslash B$ are going to be represented by those bits excatly that have zero values in the bitstring representation of $A \cap B$. Thus by flipping each of the bits' values, which can be achieved by applying X gates as the last step, we can get $A \backslash B$ from $A \cap B$.

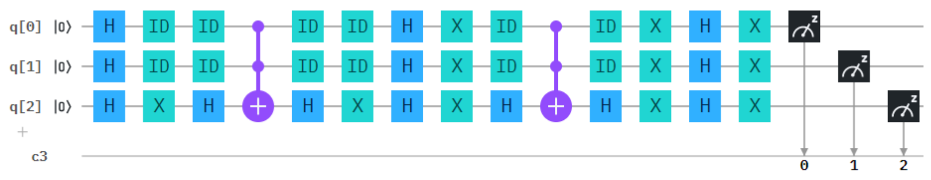

Fig. 2: Finding the difference of A and B

6 Set Operation: Union

6.1 Quantum Algorithm

First we need to find a partion of $A \cup B$, thus decomposing it into a union of distinct sets

$$A \cup B = (A \backslash B) \cup B,$$

then concatenating the two distinct sets afterwards. Since concatenation can be done efficiently on a classical computer, the speed up comes from calculating $A \backslash B$, which we already proposed an algorithm for. The elements of $A \backslash B$ and B are distinct and their union forms $A \cup B$, therefore it is convenient to use the sets $A \backslash B$ and B to construct $A \cup B$.

6.2 Implementation

The implementation is the same as for the set difference with an additional concatenation since $A \cup B$ is constructed using partition sets $(A \backslash B) \cup B$.

7 Set Operation: Projection

Database projection can be viewed as a function that takes a relation and a list of attributes of that relation as input and returns a relation containing only the specified list of attributes, thus the duplicate instances are removed. Therefore we need to deal with the following two independent problems.

7.1 Problem 1: Converting Multiset To Set

Since projection does not return duplicate attributes, we need to convert our resulting multiset into a set. Let's assume that the resulting multiset is

$$A = \{a_1^{m(a_1)}, a_2^{m(a_2)}, \dots, a_n^{m(a_n)}\}.$$

We can further assume that the elements of A can be mapped to the set

$$X = \{0, 1, \dots, n-1\} \text{ for some } n \in \mathbb{N}$$

Finding the distinct elements of A is equivalent to finding $X \cap A$.

7.2 Problem 2: Projection

The second problem is to acquire the elements of the specific column that we are looking for. First we use a binary coding of the table column name with length N, where $n = 2^N$. In this case n represents the number of columns. Let B_i be the i.th column and A_i the corresponding set of attributes that belong to that column, where $|A_i| = m = 2^M$.

Table 1: General Database Table

B_0	B_1	B_2	$\cdots$	B_n
a_{00}	a_{01}	a_{02}	$\cdots$	a_{0n}
a_{10}	a_{11}	a_{12}	$\cdots$	a_{1n}
a_{20}	a_{21}	a_{22}	$\cdots$	a_{2n}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
a_{m0}	a_{m1}	a_{m2}	$\cdots$	a_{mn}

The idea is to to encode each element in the following state $|a_{ik}|b_i\rangle$ so we can use an oracle function to find excatly those particular elements that have the specific column encoded in its state. In order to do this we need to transform the elements into a superposition and use Grover's search algorithm for multiple entries as follows

$$\frac{1}{\sqrt{2^{M+N}}} \sum_i \sum_k |a_{i_k}\rangle |b_i\rangle ,$$

where we assume that the number of records equal to 2^{M+N}. Next we define the oracle function to flag all elements that belong to B_i, the column that we are looking for

$$f(|x\rangle |b_j\rangle) = \begin{cases} 1 & \text{if } j = i \\ 0 & \text{if } j \neq i. \end{cases}$$

In the example below, there are four columns in the database system encoded as $00, 01, 10$ and 11 where each column has four elements, thus $N = M = 2$ and the total number of records in the table is 16.

As an example a record is referenced as $|01\rangle\,|11\rangle = |0111\rangle$, where $|01\rangle$ is the encoded record and $|11\rangle$ is the corresponding column. The number of iterations needed for an optimal solution in this case is $\left\lfloor \frac{\pi}{4}\sqrt{\frac{16}{4}} \right\rfloor = 1$.

The circuit below implements Grover's algorithm for multiple solution that flags all elements in the 01 column.

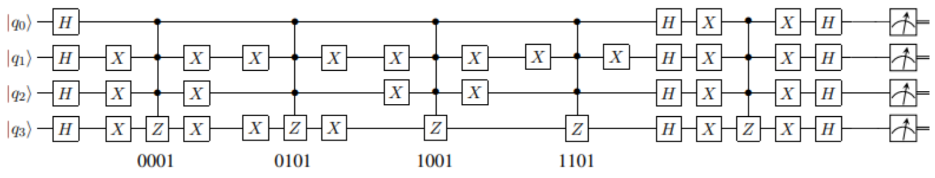

Fig. 3: Making a projection to the column 01 with multiple elements as a solution

8 Evaluation on IBM's Quantum Computers

The proposed algorithms were evaluated and tested on existing quantum computers where the computers' properties such as accuracy and computational time were compared using five publicly available processors. The evaluation was made on IBM's 5-qubit quantum computers where each computer has a different topology and error rate. For reference the algorithms were also tested on the simulator as well which assumes no error rate. The quantum circuits were created using Qiskit which is an open-source quantum computing Python framework.

8.1 Intersection [1]

As seen on Table 2 below, the deployed quantum computers are still working with relativly high error rates (the best result has less than 50% probability).

Table 2: Results of Intersection using IBM Quantum Computers.

Backend	Result	Computational Time	Total Runtime
ibmq_simulator	94 %	5ms	15.8s
ibmq_london	10 %	10.3s	54.1s
ibmq_burlington	31 %	9.6s	1m 23.5s
ibmq_vigo	38 %	8.1s	30m 45.4s
ibmqx2	41 %	7.4s	3m 36s
ibmq_oursense	44 %	8.3s	3m 3.1s

[1] `https://github.com/jodczikszabi/ADBIS2020/blob/master/Intersection.ipynb`

8.2 Set Difference [2]

In case of the set difference operation, we are negating the results from the previous run using X gates in the end. This seems to be increasing the resulting probabilities in almost every case (except on `ibmq_burlington`). An explanation for this could be that adding X gates to the circuit leads to an overall different transpiled circuit which has a lower error rate thus implying a better result.

Table 3: Results of Difference using IBM Quantum Computers.

Backend	Result	Computational Time	Total Runtime
ibmq_simulator	95 %	4ms	29.2s
ibmq_london	12 %	10.3s	38.5s
ibmq_burlington	25 %	10.5s	30.5s
ibmq_vigo	40 %	8.5s	28m 56.2s
ibmq_oursense	46 %	8.4s	12m 25.1s
ibmqx2	53 %	7.5s	2m 14.2s

8.3 Union

Our proposed algorithm for the set union consists of a set difference operation in order to obtain the pairwise disjunct sets $A\setminus B$ and B and the concatenation of these two sets. Therefore the results in this case are equivalent to the ones for the set difference algorithm above.

8.4 Projection [3]

Using the example from the previous section, there were a total of 16 records where 4 of them were marked and searched. Below you can find two tables where Table 4 describes the computational time as before and Table 5 shows the resulting probabilites for each marked item after the run.

Table 4: Results of Projection using IBM Quantum Computers.

Backend	Result	Computational Time	Total Runtime
ibmq_simulator	25 %	3ms	4.1s
ibmq_vigo	5 %	9.8s	43m 28.1s
ibmq_burlington	5 %	12.4s	1m 9.8s
ibmq_oursense	6 %	8.5s	1m 58.6s
ibmqx2	7 %	7.7s	15.4s
ibmq_london	6 %	10s	54s

[2] `https://github.com/joczikszabi/ADBIS2020/blob/master/Difference.ipynb`
[3] `https://github.com/joczikszabi/ADBIS2020/blob/master/Projection.ipynb`

Table 5: Probability results of each marked states

Backend	$\|0001\rangle$	$\|0101\rangle$	$\|0101\rangle$	$\|1101\rangle$
ibmq_simulator	25 %	24 %	25 %	25 %
ibmq_vigo	4 %	6 %	6 %	5 %
ibmq_burlington	6 %	5 %	6 %	6 %
ibmq_oursense	6 %	6 %	6 %	6 %
ibmqx2	7 %	5 %	7 %	7 %
ibmq_london	6 %	5 %	8 %	7 %

As seen above, using Grover's algorithm with multiple marked items leads to lower probabilities than the expected results. It is also notable that the results tend to be around the average on every used backends and none of the computers seem to have exceptionally high probabilities unlike in the previous evaluation.

The implementation follows the one found in the work of Strömberg, P., & Blomkvist Karlsson [10].

9 Conclusion and Future Work

The evaluation of the proposed algorithms in this paper using different IBM-Q Experience platforms allows some statement on how suitable their settings are for the implementation. IBM-Q has five 5-qubit quantum computers which differ in the topology of the qubits and the error rate. Based on the acquired results, quantum computers are still working with relatively high errors and are not efficent for practical use.

Since the resulting probabilities are so low, it'd be challenging to effectively use the proposed algorithms as of now. Reducing the error rates is neccessary for the algorithms to properly work and have useful applications in the future. In fact, the presented work has to be seen as an attempt to build a theoretical foundation for future quantum-based database algorithms. Nevertheless improvements can be made for future experimentation. Possible improvements would include creating different circuit designs for the algorithms to reduce the error that is inhereted by the different circuits' hardware realization.

The presented set based operations are built upon the intersection operation which exploits the fact that the resulting set is a subset of the power set of the smaller input relation A. The approach is then to enumerate the 2^n sets of the power set and checking all these for containment in the second relation B. Since the number of elements that we need to go through in this case is 2^n, this results in $\mathcal{O}(\sqrt{2^n}) = O(2^{n/2})$ computational time. Therefore using the proposed algorithms in case only a single operation is needed is not suggested since the computational time could be improved using other existing quantum algorithms. But since all proposed operations are inherited from an implementation of set intersection using the well-known Grover's algorithm, this approach would allow for achieving a combined performance gain due to the quantum effects in case of evaluating a combination of multiple set operations as one.

Many different useful operations have been left out of this paper that could be implemented in the future to allow the use of the proposed algorithms for more practical purposes. These operations could include different types of join operations, cross-product and the composition of SQL operations. This implementation then would allow the use of SPJ queries which is crucial for any database systems. However there are many other operations that need to be implemented aswell in order for a database system to work properly such as transactions, concurence, logging, etc. [11]. Furthermore there are still discussions and significant concers regarding the applicability of Grover's algorithm for real-life databases wheter it could be practical for use or not [12].

References

1. De Wolf, R. Quantum computing: Lecture notes. arXiv preprint arXiv:1907.09415. (2019)
2. Shor, P. W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM review, 41(2), 303–332. (1999)
3. Grover, Lov K. A fast quantum mechanical algorithm for database search. Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, pp. 212–219. (1996)
4. Pang, C. Y., Zhou, R. G., Ding, C. B., & Hu, B. Q. Quantum search algorithm for set operation, Quantum information processing, 12(1), 481–492. (2013)
5. Sakhi, Z., Kabil, R., Tragha, A., & Bennai, M. Quantum cryptography based on Grover's algorithm. In Second International Conference on the Innovative Computing Technology, pp. 33–37. IEEE, INTECH (2012)
6. Brassard, G., Hoyer, P., & Tapp, A. Quantum algorithm for the collision problem, arXiv preprint quant-ph/9705002. (1997)
7. Lavor, C., Liberti, L., & Maculan, N. Grover's algorithm applied to the molecular distance geometry problem, In Proc. of VII Brazilian Congress of Neural Networks, Natal, Brazil. (2005)
8. Baritompa, W. P., Bulger, D. W., & Wood, G. R. Grover's quantum algorithm applied to global optimization, SIAM Journal on Optimization, 15(4), 1170–1184. (2005)
9. Salman, T., & Baram, Y. Quantum set intersection and its application to associative memory, Journal of Machine Learning Research, 13(Nov), 3177-3206. (2012)
10. Strömberg, P., & Blomkvist Karlsson, V. 4-qubit Grover's algorithm implemented for the ibmqx5 architecture, (2018)
11. Roy, S., Kot, L., & Koch, C. Quantum databases, In Proc. CIDR (No. CONF). (2013)
12. Viamontes, G. F., Markov, I. L., & Hayes, J. P. Is quantum search practical?. Computing in science & engineering, 7(3), 62-70. (2005)