ISSN 2278 - 3091

International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE), Vol.2 , No.5, Pages : 22-30 (2013)
Special Issue of ICACET 2013 - Held during October 14-15, 2013, Kuala Lumpur, Malaysia

The P Vs NP Problem: A Method for Solving NP-Complete Problems by use of Graph Embodiment on a Quantum Computation Paradigm using a Relational Database Query

Shadrack Mwanik[1], Andrew Mwaura Kahonge [2], Evans K. Miriti [3]
[1] University of Nairobi, Kenya, Shadrack_mwaniki@yahoo.com
[2] University of Nairobi, Kenya, andrew.mwaura@uonbi.ac.ke
[3] University of Nairobi, Kenya, eamiriti@uonbi.ac.ke

Abstract: The relationship between the complexity class **P and NP** is one of the most fascinating and unresolved question in theoretical Computer Science. The classical computational paradigm, hedged on Turing thesis may be by itself the limiting factor. To investigate this relationship, a method for solving a query problem on a relational database on the classical and the quantum computational paradigm has been developed. The method solves queries or database problems through the use of graphs. The results indicate that the solution to the P vs NP cannot be resolved under the current computation paradigms and may requires a new computation paradigm to resolve it.

Key words: P, NP, NP-Complete, NP-Hard, Turing Computational Paradigm, Quantum Computational Paradigm.

INTRODUCTION

The relationship between the complexity class **P and NP** [1] is one of the most fascinating and unresolved question in theoretical computer science. Since 1971 when Stephen Cook formally outlined the open question [2] millions of man hours have been spent by computer scientists attempting to solve the problem [3][4].

It is our submission that today, more than ever before, computer scientists have encountered many computational problems that cannot be computed in polynomial time. In an attempt to solve these problems, optimizations, reductions and estimations have been used to get as close to the expected result as possible.

The fact that there exists a problem that cannot be effectively computed on the modern computational paradigms is a major hindrance to the development of computer science field. Until the P vs NP problem is resolved, the goal of achieving an intelligent computer that would pass the Turing test will not be achieved

The current computational paradigms have their theoretical foundations on the Turing Thesis [5]. The Turing Thesis envisaged a universal Turing machine that could simulate other Turing machines. Turing himself conceded to the fact that there existed some problems that could be computed on the Turing machines but their results could all the same be verified by a Turing machines. The famous non-halting problems embody the existence of problems that modern computers cannot effectively compute. [6]

Computer technology is today a major facet of human life. Huge data and information is stored in relational databases. Retrieval of these data and information is via the Structured Query Language (SQL) queries and other technologies. As data grows, the speed and accuracy of the database query results becomes a challenge. A solution to the above question of the relationship between P and NP class problems would guarantee a formal proof that the results of the query as a solution to the query problem. Saliently, when a user issues a database query, they have an idea of the expected result.

This implies that majority of the times; the user verifies that the answer is the true result of the query. Database query problems are both in P and NP complexity class problems with some traversing across the complexity class spectrum.

Given the importance of relational databases, a way to optimize the actual query results to the expected user results (certificate) offers more insight into the relationship between the computational complexity class problems

The existing Turing computational paradigm optimization algorithms were used to get the maximum clique of the database-query association graph.

The current computational paradigms as hedged on the Turing thesis may be themselves the limitation. To explore this aspect, a method of resolving the maximum clique problem using a quantum computer was investigated. This entailed use of graph embodiment on quantum computer

The Proposition:

if a method exists that transforms the query problem and the results space into a labeled-directed graph, the results of the query being derived as a solution to the maximum clique of the graph; and the same can be derived by graph embodiment into a quantum computer, then there is a high chance that P = NP. In contrasts, then the P vs NP problem cannot be resolved under the current computation paradigm and axioms. New computational and mathematical axioms would be required to solve the problem.

THE CURRENT STATUS OF P VS NP Problem

Computer science landscape has dramatically changed in the nearly four decades since Steve Cook presented his seminal NP-completeness paper "The Complexity of Theorem-Proving Procedures"[1] in May 1971. The cost of computing has dramatically decreased, not to mention the power of the Internet. Computation has become a standard tool in just about every academic field. Many fields in biology, chemistry, physics, economics and others are devoted to large-scale computational modeling, simulations, and problem solving.

As we solve larger and more complex problems with greater computational power and cleverer algorithms, the problems we cannot tackle begin to stand out. The theory of NP-completeness helps us understand these limitations and the P versus NP problem begins to loom large not just as an interesting theoretical question in computer science, but as a basic principle that permeates all the sciences.

In this section, we look at how people have tried to solve the P versus NP problem as well as how this question has shaped so much of the research in computer science and beyond. We look at how the NP-complete problems have been handled and the theory that has developed from those approaches. We show how a new type of "interactive proof systems" led to limitations of approximation algorithms and consider whether quantum computing can solve NP-complete problems.

This section describes the P versus NP problem and the major directions in computer science inspired by this question over the past several decades.

Defining the P versus NP Problem

Suppose we have a large group of students that we need to pair up to work on projects. We know which students are compatible with each other and we want to put them in compatible groups of two. We could search all possible pairings but even for 40 students we would have more than 300 billion trillion possible pairings.

In 1965, Jack Edmonds[7] gave an efficient algorithm to solve this matching problem and suggested a formal definition of "efficient computation" (runs in time a fixed polynomial of the input size). The class of problems with efficient solutions would later become known as ***P "for Polynomial Time."***

But many related problems do not seem to have such an efficient algorithm. What if we wanted to make groups of three students with each pair of students in each group compatible (Partition into Triangles)? What if we wanted to find a large group of students all of whom are compatible with each other (Clique)? What if we wanted to sit students around a large round table with no incompatible students sitting next to each other (Hamiltonian Cycle)? What if we put the students into three groups so that each student is in the same group with only his or her compatibles (3-Coloring)?

All these problems have similar characteristic: Given a potential solution, for example, a seating chart for the round table, we can validate that solution efficiently. The collection of problems that have efficiently verifiable solutions is known as ***NP "for Nondeterministic Polynomial-Time"***

So P = NP means that for every problem that has an efficiently verifiable solution, we can find that solution efficiently as well.

We call the very hardest NP problems (which include Partition into Triangles, Clique, Hamiltonian Cycle and 3-Coloring) "NP-complete," that is, given an efficient algorithm for one of them, we can find an efficient algorithm for all of them and in fact any problem in NP. Steve Cook, Leonid Levin, and Richard Karp[8][9][10] developed the initial theory of NP-completeness that generated multiple ACM Turing Awards.

In the 1970s, theoretical computer scientists showed hundreds more problems NP-complete (see Garey and Johnson [13]). An efficient solution to any NP-complete problem would imply P = NP and an efficient solution to every NP-complete problem.

Most computer scientists quickly came to believe P ≠ NP and trying to prove it quickly became the single most important question in all of theoretical computer science and one of the most important in all of mathematics. Soon the P versus NP problem became an important computational issue in nearly every scientific discipline.

As computers grew cheaper and more powerful, computation started playing a major role in nearly every academic field, especially the sciences. The more scientists can do with computers, the more they realize some problems seem computationally difficult.

In 2000, the Clay Math Institute named the P versus NP problem as one of the seven most important open questions in mathematics and has offered a million-dollar prize for a proof that determines whether or not P = NP.

The Implications of P = NP

To understand the importance of the P versus NP problem let us imagine a world where P = NP. Technically we could have P = NP, but not have practical algorithms for most NP-complete problems. But suppose in fact we do have very quick algorithms for all these problems.

Many focus on the negative, that if P = NP then public-key cryptography becomes impossible. True, but w*hat we would* ***gain from P = NP will make the whole Internet look like a footnote in history.***

Since all the NP-complete optimization problems become easy, everything will be much more efficient. Transportation of all forms will be scheduled optimally to move people and goods around quicker and cheaper. Manufacturers can improve their production to increase speed and create less waste. And we've just scratched the surface.

Learning becomes easy by using the principle of Occam's razor—we simply find the smallest program consistent with the data. Near perfect vision recognition, language comprehension and translation and all other learning tasks become trivial. We will also have much better predictions of weather and earthquakes and other natural phenomenon.

P = NP would also have big implications in mathematics. One could find short, fully logical proofs for theorems but these proofs are usually extremely long. But we can use the Occam razor principle to recognize and verify mathematical proofs as typically written in journals. We can then find proofs of theorems that have reasonable length proofs say in under 100 pages.

Complexity theorists generally believe P ≠ NP and such a beautiful world cannot exist.

Approaches Used to Show that P ≠ NP

Here, we present a number of ways that have been tried and failed to prove P ≠ NP. The survey of Fortnow and Homer[12] gives a fuller historical overview of these techniques.

Diagonalization

Can we just construct an NP language L specifically designed so that every single polynomial-time algorithm fails to compute L properly on some input? This approach, known as diagonalization, goes back to the 19th century.

In 1874, Georg Cantor[13] showed the real numbers are uncountable using a technique known as diagonalization. Given a countable list of reals, Cantor showed how to create a new real number not on that list.

Alan Turing, in his seminal paper on computation [14], used a similar technique to show that the Halting problem is not computable. In the 1960s complexity theorists used diagonalization to show that given more time or memory one can solve more problems. Why not use diagonalization to separate NP from P?

Diagonalization requires simulation and we don't know how a fixed NP machine can simulate an arbitrary P machine. Also a diagonalization proof would likely relativize, that is, work even if all machines involved have access to the same additional information. Baker, Gill and Solovay[15] showed no relativizable proof can settle the P versus NP problem in either direction.

Complexity theorists have used diagonalization techniques to show some NP-complete problems like Boolean formula satisfiability cannot have algorithms that use both a small amount of time and memory,[16] but this is a long way from P ≠ NP.

Circuit Complexity

To show P ≠ NP it is sufficient to show some NP-complete problem cannot be solved by relatively small circuits of AND, OR, and NOT gates (the number of gates bounded by a fixed polynomial in the input size).

In 1984, Furst, Saxe, and Sipser [17] showed that small circuits cannot solve the parity function if the circuits have a fixed number of layers of gates. In 1985, Razborov [18] showed the NP-complete problem of finding a large clique does not have small circuits if one only allows AND and OR gates (no NOT gates). If one extends Razborov's result to general circuits one will have proved P ≠ NP.

Razborov later showed his techniques would fail miserably if one allows NOT gates.[19] Razborov and Rudich [20] develop a notion of "natural" proofs and give evidence that our limited techniques in circuit complexity cannot be pushed much further.

Proof Complexity

Consider the set of Tautologies, the Boolean formulas *f* of variables over ANDs, ORs, and NOTs such that every setting of the variables to True and False makes *f true*, for example the formula

(1) (x AND y) OR (NOT x) OR (NOT y)

A literal is a variable or its negation, such as x or NOT x. A formula, like the one here, is in Disjunctive Normal Form (DNF) if it is the OR of ANDs of one or more literals.

If a formula *f* is not a tautology, we can give an easy proof of that fact by exhibiting an assignment of the variables that makes *f* false. But if *f* were indeed a tautology, we don't expect short proofs. If one could prove there are no short proofs of tautology that would imply P ≠ NP.

Resolution is a standard approach to proving tautologies of DNFs by finding two clauses of the form (*v*1 AND x) and (*v*2 AND NOT x) and adding the clause (*v*1 AND *v*2). A formula is a tautology exactly when one can produce an empty clause in this manner.

In 1985, Wolfgang Haken [21] showed that tautologies that encode the pigeonhole principle (n + 1 pigeons in n holes means some hole has more than one pigeon) do not have short resolution proofs.

Since then complexity theorists have shown similar weaknesses in a number of other proof systems including cutting planes, algebraic proof systems based on polynomials, and restricted versions of proofs using the Frege axioms, the basic axioms one learns in an introductory logic course.

But to prove P ≠ NP we would need to show that tautologies cannot have short proofs in an arbitrary proof system. Even a breakthrough result showing tautologies don't have short general Frege proofs would not suffice in separating NP from P.

Dealing with NP Hardness

So you have an NP-complete problem you just have to solve. If, as we believe, P ≠ NP you won't find a general algorithm that will correctly and accurately solve your problem all the time. But sometimes you need to solve the problem anyway. We describe some of the tools used to solve NP-complete problems and how computational complexity theory studies these approaches. Typically one needs to combine several of these approaches when tackling NP-complete problems in the real world.

Brute Force

Computers have gotten faster, much faster since NP-completeness was first developed. Brute force search through all possibilities is now possible for some small problem instances. With some clever algorithms we can even solve some moderate size problems with ease.

The NP-complete traveling salesperson problem asks for the smallest distance tour through a set of specified cities. Using extensions of the cutting-plane method we can now solve, in practice, traveling salespeople problems with more than 10,000 cities [22]

Consider the 3SAT problem, solving Boolean formula satisfiability where formulas are in the form of the AND of several clauses where each clause is the OR of three literal variables or negations of variables). 3SAT remains NP-complete but the best algorithms can in practice solve SAT problems on about 100 variables. We have similar results for other variations of satisfiability and many other NP-complete problems.

But for satisfiability on general formulae and on many other NP-complete problems we do not know algorithms better than essentially searching all the possibilities. In addition, all these algorithms have exponential growth in their running times, so even a small increase in the problem size can kill what was an efficient algorithm. Brute force alone will not solve NP-complete problems no matter how clever we are.

Parameterized Complexity

Consider the Vertex Cover problem; find a set of k "central people" such that for every compatible pair of people, at least one of them is central. For small k we can determine whether a central set of people exists efficiently no matter the total number n of people we are considering. For the Clique problem even for small k the problem can still be difficult.

Downey and Fellows [3] developed a theory of parameterized complexity that gives a fine-grained analysis of the complexity of NP-complete problems based on their parameter size.

Approximation.

We cannot hope to solve NP-complete optimization problems exactly but often we can get a good approximate answer. Consider the traveling salesperson problem again with distances between cities given as the crow flies (Euclidean distance). This problem remains NP-complete but Arora[8] gives an efficient algorithm that gets very close to the best possible route.

Consider the MAX-CUT problem of dividing people into two groups to maximize the number of incompatibles between the groups. Goemans and Williamson[4] use semi-definite programming to give a division of people only a .878567 factor of the best possible.

Heuristics and Average-Case Complexity

The study of NP-completeness focuses on how algorithms perform on the worst possible inputs. However the specific problems that arise in practice may be much easier to solve. Many computer scientists employ various heuristics to solve NP-complete problems that arise from the specific problems in their fields.

While we create heuristics for many of the NP-complete problems, Boolean formula Satisfiability (SAT) receives more attention than any other. Boolean formulas, especially those in conjunctive normal form (CNF), the AND of ORs of variables and their negations, have a very simple description and yet are general enough to apply to a large number of practical scenarios particularly in software verification and artificial intelligence. Most natural NP-complete problems have simple efficient reductions to the satisfiability of Boolean formulas. In competition these SAT solvers can often settle satisfiability of formulas of one million variables.[23]

Leonid Levin [24] developed a theory of efficient algorithms over a specific distribution and formulated a distributional version of the P versus NP problem.

Some problems, like versions of the shortest vector problem in a lattice or computing the permanent of a matrix, are hard on average exactly when they are hard on worst-case inputs, but neither of these problems is believed to be NP-complete. Whether similar worst-to-average reductions hold for NP-complete sets is an important open problem.

Average-case complexity plays an important role in many areas of computer science, particularly cryptography, as discussed later.

Could Quantum Computers Solve NP-Complete Problems?

While we have randomized and nonrandomized efficient algorithms for determining whether a number is prime, these algorithms usually don't give us the factors of a composite number. Much of modern cryptography relies on the fact that factoring or similar problems do not have efficient algorithms.

In the mid-1990s, Peter Shor [25] showed how to factor numbers using a hypothetical quantum computer. He also developed a similar quantum algorithm to solve the discrete logarithm problem. The hardness of discrete logarithm on classical computers is also used as a basis for many cryptographic protocols. Nevertheless, we don't expect that factoring or finding discrete logarithms are NP-complete. While we don't think we have efficient algorithms to solve factoring or discrete logarithm, we also don't believe we can reduce NP-complete problems like Clique to the factoring or discrete logarithm problems.

So could quantum computers one day solve NP-complete problems? So far, no indications yet.

Only Physists can address the problem as to whether these machines can actually be built at a large enough scale to solve factoring problems larger than we can with current technology (about 200 digits). After billions of dollars of funding of quantum computing research we still have a long way to go.

Lov Grover [26] did find a quantum algorithm that works on general NP problems but that algorithm only achieves a quadratic speed-up.

ISSN 2278 - 3091

International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE), Vol.2 , No.5, Pages : 22-30 (2013)
Special Issue of ICACET 2013 - Held during October 14-15, 2013, Kuala Lumpur, Malaysia

METHODOLOGY

In this section we describe the methodology employed to gain more understanding on the relation between NP and P problems and hopefully answer the question; is P = NP?.

The goal was to determine if a method exists that transforms a database query problem and the relational database solution space (a problem in P) into labeled-directed graphs, the results of the query being derived as a solution to the maximum clique of the graph; and deriving the same by graph embodiment into a quantum computer.

If the above method exists, then there is a high chance that P = NP. Otherwise, the P vs NP problem cannot be resolved under the current computation paradigm and axioms i.e Turing and Quantum. New computational and mathematical axioms would be required to solve the problem.

High Level Description

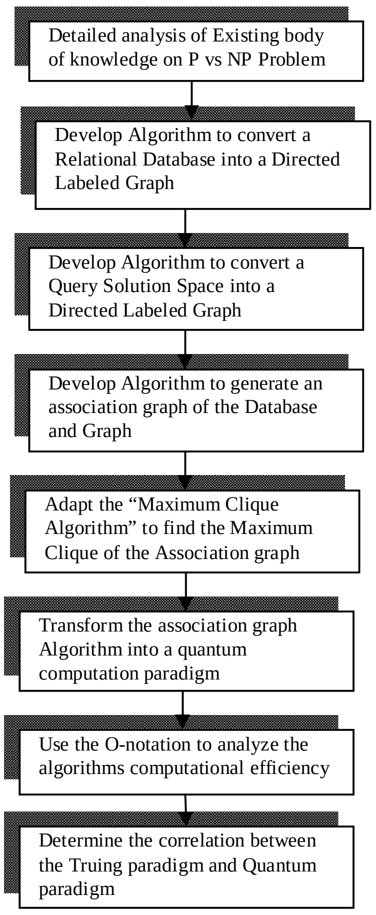
Detailed analysis of Existing body of knowledge on P vs NP Problem
Develop Algorithm to convert a Relational Database into a Directed Labeled Graph
Develop Algorithm to convert a Query Solution Space into a Directed Labeled Graph
Develop Algorithm to generate an association graph of the Database and Graph
Adapt the "Maximum Clique Algorithm" to find the Maximum Clique of the Association graph
Transform the association graph Algorithm into a quantum computation paradigm
Use the O-notation to analyze the algorithms computational efficiency
Determine the correlation between the Truing paradigm and Quantum paradigm

Fig 1: High Level Description of the Methodology.

Analysis, Development and Testing

The Detailed analysis of Existing body of knowledge on P vs NP Problem Involved Contents analysis of literatures and publications relating to the P vs NP problem. The summary is presented in section 2 above.

The development of the algorithms to transform the relational database and Query into a graph representation was informed by the analysis of the current systems and methods of representing a relational database as a binary tree formed the basis of the algorithm development

A student's information management database was developed as the basis for developing and testing the algorithms.

The Development of the algorithms was carried out largely on experimental basis and try and error methodologies.

Tests were carried out using the students' database to refine the algorithms until the final algorithms presented here were achieved.

The Algorithm to Find the Maximum Clique of the Association graph is based on the standard maximum clique algorithm. Modifications are made to the algorithm to ensure the algorithm terminates in case of no clique.

The O-notation was used to analyze the performance of the algorithms and compute the time complexity efficiency of the algorithms

The "Quantum Algorithm for finding the maximum Clique on a graph" Allan Bojic was used to analyse the effect of the for the maximum Clique on quantum computation paradigm

Finally, the results of the of the maximum clique on classical computer and Quantum computation paradigms were analysed to establish the relationship between P and NP on both paradigms.

RESULTS

After development and testing of the algorithms, the analysis was done with the aim to determine if the performance of the algorithms. The performances under Turing and Quantum computational paradigms are compared and evaluated

Conversion of a Database into a Directed Labeled Graph

1.1.1 THE ALGORITHM

The algorithm requires that:

The database b, has a set of relations R; Where

(2). $R = r_i$;for i=1, ..,|R|

The resultant graph G_b, is a labelled directed graph where

(3). $G_b=(V_b, E_b, \mu_b)$

```
1.  Procedure GENERATEGRAPH(R);
2.    Vb← []; Eb←[]; μb←[]              //Initialize the graph
3.     For ri ε R do                                //loop over all the
    relations in the database
4.       TableName:=RELNAME(ri);
5.       KeyAttrib:=GETKEY(ri);
6.       For t ε TUPLES(ri) do
7.         n←t[KeyAttrib];
8.         Vb←ADDNODE(n,Vb);
9.         If ISCOMPOUND(KeyAttrib) and  (PARTY(ri)) > 2 then;  //Add
    compound key as a  Node
10.          For i := 1 to |KeyAttrib| Do
11.                    a:=n[i];
```

```
12.                 Vb=ADDNODE(a,Vb);
13.                 Eb:=ADDEDGE(<n,a>,Eb);
14.                 µb(n,a):=TableName.KeyAttrib[i];
15.         End for
16.       End if
17.               NonKeyAttrib←KeyAttrib;
18.               If ISEMPTY(NonKeyAttrib) then     //No non-key
19.   attributes so add a loop
20.                 E←ADDEDGE(n,n);
21.                  µb(n,n):=TableName;
22.               Else
23.               For i ε 1, [NonKeyAttrib] do //Add edge between key
      and non-key attribute
24.                 m←t[NonKeyAttrib[i]];
25.                 Vb←ADDNODE(m,Vb);
26.                 Eb:=ADDEDGE(<n,m>,Eb);
27.                 µb(n,m):=TableName.NonKeyAttrib[i];
28.               End for
29.             End if
30.       End for
31.     End for
32.   End Procedure
```

1.1.2 DESCRIPTION

A Labeled directed graph is generated from the database as follows:

If all the relations in the database are at most 2-arity (meaning they have only one or two attributes) and the elements of all tables came from a set U, then the set U is identified with the vertex V, of the graph. This means there is a node in the graph for each element occurring in a tuple and a column of every table in the database

The method processes all relations, and for each relation adds the appropriate nodes to the vertex list and the appropriately names edges to the edge list and name mappings. The table name is obtained and used as the basis for naming edges. The KeyAttrib list records the attributes which are keys for the table. At line 6 begins a loop over all the rows in the table. t labels any particular row. For this row, the values corresponding to the keys are extracted and added to the node list.

The ADDNODE function adds the node to list of vertices V. If the node is already present mV it is added again. If the key to the table is compound (there is more than 1 attribute in the key), then nodes are also added for the values of all key attributes in t. An edge is added from each of these attribute nodes to the node representing the compound key for the row. Each node is labeled by TableName.keyAttrib[i] where KeyAttrib[i] is the attribute name for ith attribute node.

Next, edges are added between the key node and the values in the non-key attributes. If there are no non-key attributes (line 18) then a loop is added on the key node. The loop is labeled by the table's name. If there are non-key columns in the table, an edge is added from the compound node to each of these non-key nodes. Each edge is labeled by TableName.nonKeyAttrib[i] where NonKeyAttrib[i] is the attribute name for the ith non key attribute node.

1.1.3 ANALYSIS

The running time for the algorithm to convert the relational database into a directed labeled graph is

(4). $\Theta(n^3)$.

Since n is bound to the total rations in the database U, the running time is therefore

(5). $\Theta(|R|^3)$.

As the number of relations R in a database increases, the running time of the algorithm increases by power of 3.

Conversion of a Query Q, Into a Directed Labelled Graph.

1.1.4 THE ALGORITHM

The algorithm involves the pruning of the database d into smaller database (Subset of d) being the representation of the solution space relevant to the Query,

(6). q={Q1, Q2, ...Q|q| }

The algorithm requires that:

- The database has a set of relations R Where
  (7). $R = R_i$;for i=1, ..,|R|
- A query q,
  (8). q= {Q1,Q2,... Q|q| }
- Minimal database R' Where
  (9). $R' = r_i$;for i=1, ..,|q|
- The resultant graph G_q, is a labelled directed graph G_q
  (10). $G_q=(V_q, E_q, \mu_q)$

```
1.  Procedure PRUNEDATABASEGRAPH(R, q);
2.   For i ε 1,|q| do
3.      [Attribs, Vals] ← GROUND(Qi );
4.      ri ←GETTABLE(Qi);
5.      ri ←SELECTANDPROJECT(ri,Qi,Attribs,Vals);
6.     End for
7.   End Procedure
```

1.1.5 DESCRIPTION

For each table (Q_i) entering into the query the set of attributes (attribs) which are grounded is determined, and as is the value (vals) to which each attribute is constrained. In line 4 the table referenced by Q_i retrieved from the database.

In SELECTANDPROJECT a row is eliminated from the table by selecting down to include only the rows which have the assigned values for the attributes and which are then projected out to include only those columns of Q_i involving variables.

1.1.6 ANALYSIS

The running time for the algorithm to convert the Query into a directed labeled graph is **O(n^2).**

Since n is bound to the query attributes in the query Q, the running time is therefore **O(|q|^2).**

Generation of Association Graph (G_a) From the Pruned Query (G_q) and Database (G_b)

1.1.7 THE ALGORITHM

The algorithm requires that:

- A set of pruned relations R; Where
  (11). $R = r_i$;for i=1, ..,r|R|
- A pruned query q'; Where q'
  (12). $q' = Q'_i$ for i = 1,..,|q|
- The association graph G_a
  (13). $G_a=(V_a, E_a)$

is formed from the query and the database graph

```
1.  Procedure FORMASSOCIATIONGRAPH(R,q');
2.    Va← []; Ea←[];        //Initialize the graph
3.    For Q'i ε q' do                  //Generate and store graphs for
    each query subgoal
4.       [Vqi,Eqi, µqi]←ROWGRAPH(Q'i);
5.    Endfor
```

```
6.    For ri ε R do
7.      TableName:=RELNAME(ri);
8.      AttribNames:=ATTRIBNAMES(ri);
9.      For t ε ROWS(ri) do
10.         Vt←ROWNODES(t)
11.         Et←ROWEDGES(t)
12.         For i ε1 to |q| Do
13.                 V'a:=ASSOCIATENODES(Va,Vt,,Vqi, µqi,TableName)
14.                 E'a:= ASSOCIATEEDGES(V'a,Et,Eqi, µqi,TableName,
      AttribNames)
15.                 Va←ROWNODES(Va,V'a)
16.                 Ea←ROWEDGES(Ea,E'a)
17.         End for
18.       End for
19.     End For
20.   End Procedure
```

1.1.8 DESCRIPTION

An Association graph is generated from the pruned database and the Query as follows:

The graph for the pruned query is constructed and stored (line 4). Then looping over all pruned relations and all rows in each relation is performed. For each row t the nodes V, and edges E, for the row are determined (lines 10 and 11). The nodes are simply the elements in the attributes of the table and an additional key node if required. The edges connect all attributes to the key node.

Once the graph for the row has been defined, it may then compared with the graphs for each of the |q| query subgoals. For each subgoal, the association graph vertices may be defined by pairing variable nodes from the query with compatible row nodes.

Compatible nodes have the same labeled loops which can be determined from the relation name (stored in TableName). Note that some of these a-vertices may have previously been generated. Even in such cases V'a includes all a-vertices generated by the tuple and the subgoal graphs. However, when these a-nodes are added to the set Va, of a-vertices duplicates are not permitted (line 15).

The ASSOCIATEEDGES routine then generates the a-edges that are generated between the a-vertices stored in V'a. Compatible edges can be identified by knowing µqi the relation name (TABLENAME) and the attribute names (AttribNames). Some of these edges may have previously been generated, but some will be new (for example those nodes connected to the key). ADDEDGES appends only the new edges to the a-edge set Ea.

1.1.9 ANALYSIS

The Construction of the Association Graph (G_a) from the pruned query and Database is similar to the conversion of the database into a directed graph only that this time around, the graph is created from the pruned database to reflect the query solution space.

The running time for the algorithm to generate the association is **$\Theta(n^3)$.**

Since n is bound to the number of vertices v in the database graph, the running time is therefore **$\Theta(|v|^3)$.**

Maximum Clique of the Association Graph G_a.

1.1.10 THE ALGORITHM

The algorithm requires that:

- Association Graph G_a, (14). $G_a = (V_a,E_a)$,
- Lower bound on clique l_b (default, 0).
- The result is the size of the maximum clique in the graph

```
1.    procedure MAXCLIQUE(Ga,lb)
2.      max ← lb
3.      for i ε 1 to n do
4.        if d(vi)  >= max then          // Pruning 1
5.          U ←0;
6.        for each vj ε N(vi) do
7.          if j > i then                // Pruning 2
8.            if d(vj) >= max then  // Pruning 3
9.              U ← U U{vj}
10.       CLIQUE(Ga , U, 1)
11.
12.   CLIQUE Recursive Subroutine
13.   procedure CLIQUE(Ga U, size)
14.     if U = 0 then
15.     if size > max then
16.       max ← size
17.       return
18.     while |U| > 0 do
19.       if size + |U| <= max then   //Pruning 4
20.         return
21.       Select any vertex u from U
22.       U←U \{u}
23.       N'(u) := {w|w ε N(u) ^ d(w) >= max  // Pruning 5
24.       CLIQUE(Ga,U ∩N'(u), size + 1)
```

1.1.11 DESCRIPTION

The maximum clique for the association graph is found by computing the largest clique containing each vertex and picking the largest among them. A key element of the algorithm is that during the search for the largest clique containing a given vertex, vertices that cannot form cliques larger than the current maximum clique are pruned, in a hierarchical fashion.

The subroutine CLIQUE goes through every relevant clique containing vi in a recursive fashion and returns the largest. The subroutine is similar to the Carraghan-Pardalos algorithm [19].

Our algorithm consists of several pruning steps. The pruning in Line 4 of MAXCLIQUE (Pruning 1) filters vertices having strictly fewer neighbors than the size of the maximum clique already computed. These vertices can be safely ignored, since even if a clique were to be found, its size would not be larger than max.

While forming the neighbor list *U* for a vertex *vi*, we include only those of v_i's neighbors for which the largest clique containing them has not been found (Line 7, Pruning 2), to avoid recomputing previously found cliques. Furthermore, the pruning in Line 8 (Pruning 3) excludes vertices v_j *ε N(vi)* that have degree less than the current value of *max*, since any such vertex could not form a clique of size larger than *max*.

The pruning strategy in Line 7 of subroutine CLIQUE (Pruning 4) checks for the case where even if all vertices of *U* were added to get a clique, its size would not exceed that of the largest clique encountered so far in the search, max.

The pruning in Line 11 of CLIQUE (Pruning 5) reduces the number of comparisons needed to generate the intersection set in Line 12. Pruning 4 is used in most existing algorithms, whereas pruning steps 1, 2, 3 and 5 are new.

ISSN 2278 - 3091
International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE), Vol.2 , No.5, Pages : 22-30 (2013)
Special Issue of ICACET 2013 - Held during October 14-15, 2013, Kuala Lumpur, Malaysia

1.1.12 ANALYSIS

First inner while loop, and let n = Size

The while loop loops for log(i) times, so inside one iteration of the for i loop, c * log(i) operations are done. In total

(15). c*log(1) + c*log(2) + c*log(3) + ... + c*log(n)

The two nested for loops and the recursive Clique procedure call would introduce the exponential growth on the n of order 2, as the number of vertices grows

The running time for this algorithm to find the maximum clique is *Θ*(**2^n).**

Since n is bound to the total vertices in the database U, the running time is therefore *Θ*(**2^|u|).**

This represents an exponential growth of order 2 as the number of vertices increases.

Clearly, the modified maximum clique algorithm still exhibits exponential tendencies making a perfect NP-Complete problem

Maximum Clique on Quantum Computation

A Quantum Algorithm for finding the maximum Clique on a graph is given by Allan Bojic [27] , A Quantum Algorithm for finding the maximum Clique on a graph.

Bojic algorithm gives the complexity of the worst case scenario as $O(|V|\sqrt{2^{|V|}}$). If |v| = n, the running time is therefore $O(n\sqrt{2^n}\,)$

This is a tremendous improvement compared to the *Θ*(2^n) of the classical implementation of the maximum clique described in section 4.4 above

DISCUSSION OF THE RESULTS

From the results of algorithm analysis in section 4, we have been able to show that:

A method does exist that can transform a relation database into a directed labeled graph. The running time complexity of the algorithm is **<u>*Θ*(n^3).</u>** As n increase, the performance degrades rapidly. However in normal database implementation, n is tightly bound by the number tables and relations therein. The running time is there limited to the number of relations R. This yields a running time complexity Θ(|n|^3) for n=R

A method exists that can transform the query into a query database graph being a representation of the database solution space. The running time complexity of the algorithm is **<u>*Θ*(n^2).</u>** As n increase, the performance degrades rapidly. However in normal database implementation, n is tightly bound by the number variables and table relations in the query. The running time is therefore limited to the number of relations q. This yields a running time complexity Θ (|n|^2) for n=q

A method exists that generate an association graph of the database and query graph. The running time complexity of the algorithm is **<u>Θ(n^3).</u>** As n increase, the performance degrades rapidly. However in normal database implementation, n is tightly bound by the number of tables and relations therein. The running time is there limited to the number of relations R. This yields a running time complexity Θ (|n|^3) for n=R

The maximum clique of the association graph yields the solution to the query. The algorithm running complexity is *Θ*(**2^n).** Since n is bound to the total vertices in the database U, the running time is therefore *Θ* (2^|u|). The algorithm has an exponential growth of order 2 on the number of vertices in the association graph. This reinforces the fact that maximum clique is a NP-Complete problem.

A method exists that transforms the maximum clique into a quantum computation paradigm. The time complexity of the algorithm is $O(|V|\sqrt{2^{|V|}}$)The algorithm has greatly improved on the quantum computation by 2-order square root.

The relationship between the generation of the query graph and the database graph is interesting. While running time complexity for the generation of the database graph is in order of 3 of the input size, the query is in order of 2 of the query input size i.e Number of relations within the query. Instead of generating the database graph and the query graph separately, the association graph could be generated directly by pruning the database query representation. By directly generating the association graph this way, the running time complexity can be significantly reduced to the order of 2/3 of the database table relation **<u>*Θ* (n^2/3).</u>**

Of greatest interest is the relationship between the maximum clique algorithm on a classical computation paradigm (Turing) and the quantum computational paradigm. On quantum computation, the performance improves by the root of the input. This improvement can be attributed to the extra state of the Qbits superimposition in the quantum mechanics.

CONCLUSION

From the hypothesis, it suffices to conclude that there is a high likelihood of the N =NP. However the relationship between the maximum clique algorithm on a classical computation paradigm (Turing) and the quantum computational paradigm has opened a new thought into the relationship between P and NP problems.

As mentioned earlier, the improvement of the maximum clique problem on quantum computation paradigm may be attributed to the extra bit, Qubit that can be superimposed to give 4 state (0,1,0', 1') as opposed to the normal classical computing paradigm of 2 states (1,0).

The limitation of the classical computation paradigm is by itself hedged on the current representation of the states by use of **electric current (Electrons) flow** as medium of realizing the implementation.

Moreover, many of the quantum algorithms are likewise based on the classical implementation.

Conjecture

Supposing we had a new element Oracle Ω in the periodic table, that can yield N different states. It suffices to say that it would be possible reduce the computational time complex of many NP-Complete problems by root of N. This would be sufficient to solve many of the NP problems hopefully in polynomial time

Since the Oracle Ω element does not exist at the moment, then it suffices to say that the P vs NP problem cannot be resolved under the current computational paradigm. A new, Ω computational paradigm is required to resolve the problem.

Further work

Further research can be carried out to enable:-

The algorithms presented are independent of the implementation platform. Further work is required to implement the algorithms and run tests to derive actual computation complexity in both time and space.

The maximum clique problem on quantum computation is based on the Alan Jobic algorithm [2] which is an improvement of the Grower's database search algorithm. Further research is required on the effect of using Shor's fourier transformation quantum computation algorithm

CERN, has announced the possibility of the Boson particle identified by the Large Hydron Collider (LHC) experiment as being the elusive Higgs Boson. Further research on what value the Boson can add.

Further research on use of labeled directed graph embodiment as way to improve database query problems using the methods presented

ACKNOWLEDGMENT

Thanks to Prof. Rodriquez for the useful discussions and guidance on relevant journals on computational complexity.

REFERENCES

[1] Cook, Stephen,1971. *The complexity of theorem proving procedures*. Proceedings of the Third Annual ACM Symposium on Theory of Computing. pp. 151–158.

[2] Clay Math Institute of Mathematics, *Official Problem Description*, Retrieved from http://www.claymath.org/millennium/P_vs_NP/pvsnp.pdf)

[3] Downey, R. and Fellows, M. *Parameterized Complexity*. Springer, 1999.

[4] Goemans, M. and Williamson, D. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM 42*, 6 (1995), 1115–1145.

[5] M. Turing,1939, Systems of Logic Based on Ordinals (Ph.D. thesis). Princeton University. Retrieved from https://webspace.princeton.edu/users/jedwards/Turing Centennial 2012/Mudd Archive files/12285_AC100_Turing_1938.pdf

[6] A. Turing, 1936, On computable numbers with an application to the entscheidnungsproblem, Proc. London Math. Soc. 42 230–265.

[7] Edmonds, J. Paths, trees and owers. *Canadian Journal of Mathematics 17*, (1965), 449–467.

[8] Arora, S. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM 45*, 5 (Sept. 1998), 753–782.

[9] Karp, R. Reducibility among combinatorial problems. *Complexity of Computer Computations*. R. Miller and J. Thatcher, Eds. Plenum Press, 1972, 85–103.

[10] Levin, L. Universal'nyie perebornyie zadachi (Universal search problems: in Russian). *Problemy Peredachi Informatsii 9*, 3 (1973), 265–266. Corrected English translation.

[11] Garey, M. and Johnson, D. Computers and Intractability. A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, NY, 1979.

[12] Lance Fortnow, 2000, The status of the P versus NP problem, Communications of the ACM 52 no. 9, pp. 78–86.

[13] Cantor, G. Ueber eine Eigenschaft des Inbegriffes aller reellen algebraischen Zahlen. *Crelle's Journal 77* (1874), 258–262 – Translated by Google translator .

[14] Turing, A. On computable numbers, with an application to the Etscheidungs problem. *Proceedings of the London Mathematical Society 42* (1936), 230–265.

[15] Baker, T., Gill, J., and Solovay, R. Relativizations of the P = NP question. *SIAM Journal on Computing 4*, 4 (1975), 431–442.

[16] van Melkebeek, D. A survey of lower bounds for satisfiability and related problems. *Foundations and Trends in Theoretical Computer Science 2*, (2007), 197–303.

[17] Furst, M., Saxe, J., and Sipser, M. Parity, circuits and the polynomial-time hierarchy. *Mathematical Systems Theory 17* (1984), 13–27.

[18] Razborov, A. Lower bounds on the monotone complexity of some Boolean functions. *Soviet Mathematics-Doklady 31*, (1985) 485–493.

[19] Razborov, A. On the method of approximations. In *Proceedings of the 21st ACM Symposium on the Theory of Computing*. ACM, NY, 1989, 167–176.

[20] Razborov, A., and Rudich, S. Natural proofs. *Journal of Computer and System Sciences 55*, 1 (Aug. 1997), 24–35.

[21] Haken, A. The intractability of resolution. *Theoretical Computer Science, 39* (1985) 297–305.

[22] Applegate, D., Bixby, R., Chvátal, V., and Cook, W. On the solution of traveling salesman problems. Documenta Mathematica, Extra Volume ICM III (1998), 645–656.

[23] Agrawal, M., Kayal, N., and Saxena, N. PRIMEs. In *Annals of Mathematics 160*, 2 (2004) 781–793.

[24] Levin, L. Average case complete problems. *SIAM Journal on Computing 15*, (1986), 285–286.

[25] Shor. P. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing 26*, 5 (1997) 1484–1509.

[26] Grover, L. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th ACM Symposium on the Theory of Computing*. ACM, NY, 1996, 212–219.

[27] Alan Jobic, 2012, A Quantum Algorithm for finding the maximum Clique on a undirected graph – Jios, Vol 36 No.