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

PRESENTED BY: SHADRACK MWANIKI

Registration No: **P56/8849/2005**

Submitted as partial fulfillment of the requirement for the award of Masters of Science in Computer Science, School of Computing and Informatics, University of Nairobi

APRIL 2013

DECLARATION

I Declare that this project, as presented in this report, is my original work and has not been presented for any other University award.

Signed……Date……..

Shadrack Mwaniki

Registration No: **P56/8849/2005**

I declare that the work reported in this research was carried out by the candidate under my supervision and has been submitted with my approval as the University supervisor.

Signed…….Date……

Mr. A. Mwaura

Lecturer

DEDICATION

This research project is dedicated to the dearest figures in my life:

- My father, Paul Kabunye who taught me that the best kind of knowledge to have is that which is learned for its own sake.
- My mother, Terasia Wanjuki who taught me that even the largest task can be accomplished if it is done one step at a time.
- My wife, Margaret Wanja whose support I could not have done without.
- Lastly my son Jayden Kabunye and daughter Tracy Wanjuki who despite being too young to understand, were always on my side cheering me on.

ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to my supervisor Mr. Andrew Mwaura who gave me the golden opportunity to do this wonderful project on the P vs NP problem which also helped me in doing a lot of research and i came to know about so many new things

Secondly i recognize the lecturers at the School of Informatics and Computing for the invaluable knowledge they transferred to me. It has made me what I am today

Lastly but not least, i thank my family and friends who sacrificed their time and resources to see me through the three years of course work and this research paper.

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 new quantum computational paradigm has been developed. The method solves queries or database problems through the use of graphs.

The relational database is converted into a directed labeled graph representing the set of the query solution space. The query problem is converted into a directed labeled graph representing the sub-set database solution space set. An association graph of the database and the query graphs is generated. The maximum clique (a known NP-Complete problem) of the association graph yields the solution to the query problem.

Comparison of the maximum clique algorithm on both classical and quantum computation paradigms has yielded additional knowledge on the relationship between P vs NP problem. The results have indicated a substantial improvement on the algorithm efficiency on quantum computation but not sufficient enough to resolve the problem.

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.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

ABBREVIATIONS

AGD	-	Automated Graph Drawing
CERN	-	Conseil Européen pour la Recherche Nucléaire (the European Organization for Nuclear Research)
E-R	-	Entity Relation
LHC	-	Large Hydron Collider
MC	-	Maximum Clique
NP	-	Non-Deterministic Polynomial Time (Class of problems that cannot be computed in Polynomial Time)
P	-	Polynomial Time (Class of problems that can be computed in Polynomial Time)
QUBIT	-	Quantum Bits
SQL	-	Structured Query Language
TSP	-	Travelling Salesman Problem
UQC	-	Universal Quantum computer
UTM	-	Universal Turing Machine
VSLI	-	Very Large Scale Integration

CHAPTER ONE: INTRODUCTION

1.1 PROBLEM STATEMENT

The relationship between the complexity class **P** and **NP** (Cook, 1971) is one of the most fascinating and unresolved question in theoretical computer science. Since 1971 when Stephen Cook formally outlined the open question (The P vs NP Problem, 2000), millions of man hours have been spent by computer scientists attempting to solve the problem (Lance 2000; William, 2002).

It is my 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. Many computer science fields such as artificial intelligence have not been able to achieve the goal of a computer passing the Turing test due to the existence of these NP problems. 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 (Turing, 1939). 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 not be computed on the Turing machines but their results could all the same be verified by a Turing machine. The famous non-halting problems embody the existence of problems that modern computers cannot effectively compute. (Alan 1936)

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 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

In this project, have offered a method of optimizing a relational database query problem by:

- Developing an algorithm that converts a database into a labeled directed graph on a worst case scenario representation of the query problem expected results space i.e Database Graph
- Developing an algorithm that converts a database query into a directed labeled graph i.e Query graph
- Developing an algorithm that generates an association graph from the database graph and the query graph
- Generated the query result by use of optimization using a known NP-complete problem (Finding Maximum clique on a directed labeled graph). In this case, the maximum clique of the association graph is the results of the query.
- Investigated the impact to the solution space of the query problem by use of the quantum computational paradigm

The existing Turing computational paradigm optimization algorithms were used 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 method has generated additional knowledge on the relationship between the P and NP complexity class problems and has opened further research on how many of the existing NP

problems can be optimized and or solved via graph theoretic embodiment into quantum computation paradigm.

In addition, the project has offered more insight in the field of computational complexity, graph theoretic, quantum computation and relation database theory. The project will assist computer science research students in theoretical computer science research areas. The results that have been derived will hopefully give research scientist a different approach into the P vs NP problem.

A consequence of the this project was: 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.

1.2 OVERALL OBJECTIVE

To find a method that transforms the query problem and the relational database solution space into labeled-directed graphs, the results of the query being derived as a solution to the maximum clique of the graph; and derive 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.

1.3 SPECIFIC OBJECTIVES

In this project, we present a method of solving a relational database problem by:

1) Developing an algorithm that converts a database into a labeled directed graph on a worst case scenario representation of the query problem expected results space

2) Developing an algorithm that converts a database query into a directed labeled graph

3) Developing an algorithm that generates an association graph from the database graph and the query graph

4) Adapting the Carraghan-Pardalos Maximum Clique algorithm (R. Carraghan and P. Pardalos, 1990) to find the "Maximum Clique" of the association graph. In this case, the maximum clique of the association graph is the solution to the query.

5) Adapt the Alan Bojic, (Bojic, 2012) Quantum Algorithm for finding the maximum Clique on a graph to find the maximum Clique of the association graph on quantum computation paradigm.

6) Carry out comparative analysis of the algorithms performances of 4 and 5 above.

1.4 PROBLEM JUSTIFICATION

As the world becomes a global village, more data and information will be generated that requires storage spaces. The data must be processed and retrieved. This research project offers a method to optimize database query processing.

A review of the projects undertaken by computer science students at the institute reveals there is very limited interest in computational complexity and computational theory. This research project opens more research areas on these fields.

Quantum computing is a relatively new paradigm. The project provides away of employing the computational power of quantum computers in resolving real life problems.

The project has explored the relationship between P vs NP computational complexity class problems. The new proposed approach that aids in resolving the problem is a highly welcome additional in the field of computer science.

CHAPTER TWO: LITERATURE REVIEW

2.1 TURING MACHINES

A Turing machine is a theoretical computing system, described in 1936 by Alan Turing (Turing, 1936). A Turing machine that can efficiently simulate any other Turing machine is called a Universal Turing Machine (UTM). The Church-Turing thesis states that any practical computing model has either the equivalent or a subset of the capabilities of a UTM.

The Church-Turing thesis (formerly commonly known simply as Church's thesis) says that any real-world computation can be translated into an equivalent computation involving a Turing machine. In Church's original formulation (Church 1935, 1936), the thesis says that real-world calculation can be done using the lambda calculus, which is equivalent to using general recursive functions (Turing, 1939).

The Church-Turing thesis encompasses more kinds of computations than those originally envisioned, such as those involving cellular automata, combinators, register machines, and substitution systems. It also applies to other kinds of computations found in theoretical computer science such as quantum computing and probabilistic computing.

There are conflicting points of view about the Church-Turing thesis. One says that it can be proven, and the other says that it serves as a definition for computation. There has never been a proof, but the evidence for its validity comes from the fact that every realistic model of computation, yet discovered, has been shown to be equivalent. If there were a device which could answer questions beyond those that a Turing machine can answer, then it would be called an oracle.

Some computational models are more efficient, in terms of computation time and memory, for different tasks. For example, it is suspected that quantum computers can perform many common tasks with lower time complexity, compared to modern computers, in the sense that for large enough versions of these problems, a quantum computer would solve the problem faster than an ordinary computer. In contrast, there exist questions, such as the halting problem, which an ordinary computer cannot answer, and according to the Church-Turing thesis, no other computational device can answer such a question.

The Church-Turing thesis has been extended to a proposition about the processes in the natural world by Stephen Wolfram in his principle of computational equivalence (Wolfram

2002), which also claims that there are only a small number of intermediate levels of computing power before a system is universal and that most natural systems are universal.

2.2 COMPUTATIONAL COMPLEXITY THEORY

In computer science, computational complexity theory is the branch of the theory of computation that studies the resources, or cost, of the computation required to solve a given computational problem (Hopcroft, Motwani, and Ullman, 2007). This cost is usually measured in terms of abstract parameters such as time and space, called computational resources. Time represents the number of steps required to solve a problem and space represents the quantity of information storage required or how much memory is required (Arora, Boaz 2009)

Computational complexity theory classifies computational problems into complexity classes. The number of complexity classes is ever changing, as new ones are defined and existing ones merge through the contributions of computer scientists. The complexity classes of decision problems include:

1. P—The complexity class containing decision problems that can be solved by a deterministic UTM using a polynomial amount of computation time;

2. NP ("Non-deterministic Polynomial time")—The set of decision problems solvable in polynomial time on a non-deterministic UTM. Equivalently, it is the set of problems that can be "verified" by a deterministic UTM in polynomial time;

3. NP-hard (Nondeterministic Polynomial-time hard)—A problem H is in the class NP-hard if and only if there is an NP-complete problem L that is polynomial time Turing-reducible to H. That is to say, L can be solved in polynomial time by an oracle machine with an oracle for H;

4. NP-complete—A decision problem C is NP-complete if it is complete for NP, meaning that:

   (a) it is in NP and

   (b) it is NP-hard,

   i.e., every other problem in NP is reducible to it. "Reducible" means that for every problem L, there is a polynomial-time many-one reduction, a deterministic

algorithm which transforms instances 1εL into instances cεC, such that the answer to c is YES if and only if the answer to 1 is YES. To prove that an NP problem A is in fact an NP-complete problem it is sufficient to show that an already known NP-complete problem reduces to A.

Decision problems have binary outcomes. Problems in NP are computation problems for which there exists polynomial time verification. That is, it takes no more than polynomial time (class P) in the size of the problem to verify a potential solution. It may take more than polynomial time, however, to find a potential solution. NP-hard problems are at least as hard as any problem in NP.

Optimization problems are problems for which one or more objective functions are minimized or maximized over a set of variables, sometimes subject to a set of constraints. For example, the Traveling Salesman Problem ("TSP") is an optimization problem where an objective function representing, for example, distance or cost, must be optimized to find an itinerary, which is encoded in a set of variables representing the optimized solution to the problem . For example, given a list of locations, the problem may consist of finding the shortest route that visits all locations exactly once. Other examples of optimization problems include Maximum Independent Set, integer programming, constraint optimization, factoring, prediction modeling, and k-SAT. These problems are abstractions of many real-world optimization problems , such as operations research, financial portfolio selection, scheduling, supply management, circuit design, and travel route optimization. Many large-scale decision-based optimization problems are NP-hard (A high - Level Look at Optimization, 2000).

Simulation problems typically deal with the simulation of one system by another system, usually over a period of time. For example, computer simulations can be made of business processes , ecological habitats, protein folding, molecular ground states, quantum systems, and the like. Such problems often include many different entities with complex inter-relationships and behavioral rules. In Feynman it was suggested that a quantum system could be used to simulate some physical systems more efficiently than a UTM.

Many optimization and simulation problems are not solvable using UTMs. Because of this limitation, there is need in the art for computational devices capable of solving computational problems beyond the scope of UTMs. In the field of protein folding, for example, grid computing systems and supercomputers have been used to try to simulate large protein

systems (Shirts et al., 2000). The NEOS solver is an online network solver for optimization problems, where a user submits an optimization problem , selects an algorithm to solve it, and then a central server directs the problem to a computer in the network capable of running the selected algorithm (Dolan et al, 2002). Other digital computer-based systems and methods for solving optimization problems can be found (Fourer et al., 2001). All these methods are limited, however, by the fact they utilize digital computers, which are UTMs, and accordingly, are subject to the limits of classical computing that impose unfavorable scaling between problem size and solution time.

2.3 THE P vs NP PROBLEM

The relationship between the complexity classes P and NP is an unsolved question in theoretical computer science. It is considered to be the most important problem in the field – the Clay Mathematics Institute has offered a $1 million prize for the first correct proof (The P vs NP Problem, 2002).

In essence, the question P = NP? asks: if 'yes'-answers to a 'yes'-or-'no'-question can be verified "quickly" (in polynomial time), can the answers themselves also be computed quickly?

Consider, for instance, the subset-sum problem, an example of a problem which is "easy" to verify, but whose answer is believed (but not proven) to be "difficult" to compute. Given a set of integers, does some nonempty subset of them sum to 0? For instance, does a subset of the set {−2, −3, 15, 14, 7, −10} add up to 0? The answer "yes, because {−2, −3, −10, 15} add up to zero", can be quickly verified with a few additions. However, finding such a subset in the first place could take much longer. The information needed to verify a positive answer is also called a certificate. Given the right certificates, "yes" answers to our problem can be verified in polynomial time, so this problem is in NP.

An answer to the P = NP question would determine whether problems like the subset-sum problem are as "easy" to compute as to verify. If it turned out P does not equal NP, it would mean that some NP problems are substantially "harder" to compute than to verify.

The relation between the complexity classes P and NP is studied in computational complexity theory, the part of the theory of computation dealing with the resources required during

computation to solve a given problem. The most common resources are time (how many steps it takes to solve a problem) and space (how much memory it takes to solve a problem).

In such analysis, a model of the computer for which time must be analyzed is required. Typically, such models assume that the computer is deterministic (given the computer's present state and any inputs, there is only one possible action that the computer might take) and sequential (it performs actions one after the other). As of 2009, these assumptions are satisfied by all practical computers yet devised.

In this theory, the class P consists of all those decision problems that can be solved on a deterministic sequential machine in an amount of time that is polynomial in the size of the input; the class NP consists of all those decision problems whose positive solutions can be verified in polynomial time given the right information, or equivalently, whose solution can be found in polynomial time on a non-deterministic machine. Arguably, the biggest open question in theoretical computer science concerns the relationship between those two classes:

Is P equal to NP?

In a 2002 poll of 100 researchers, 61 believed the answer is no, 9 believed the answer is yes, 22 were unsure, and 8 believed the question may be independent of the currently accepted axioms, and so impossible to prove or disprove (Gasarch, 2002)

2.4 RELATIONAL DATABASES

Many entities employ relational databases to store information. The information may be related to almost any aspect of business, government or individuals. For example, the information may be related to human resources, transportation, order placement or picking, warehousing, distribution, budgeting, oil exploration, surveying, polling, images, geographic maps, network topologies, identification, and/or security.

A relational database stores a set of “relations” or “relationships.” A relation is a two-dimensional table. The columns of the table are called attributes and the rows of the table store instances or “tuples” of the relation. A tuple has one element for each attribute of the relation. The schema of the relation consists of the name of the relation and the names and data types of all attributes. Typically, many such relations are stored in the database with any given relation having perhaps millions of tuples.

Searching databases typically employs the preparation of one or more queries. One common way of formatting queries is through Structured Query Language (SQL). QL-99 is the most recent standard, however many database vendors offer slightly different dialects or extensions of this standard. The basic query mechanism in SQL is the statement: SELECT L FROM R WHERE C, in which L identifies a list of columns in the relation(s) R, C is a condition that evaluates to TRUE, FALSE or UNKNOWN. Typically, only tuples that evaluate to TRUE are returned. Other query languages are also known, for example DATALOG, which may be particularly useful for recursive queries. Traditional querying or searching of databases presents a number of problems . Boolean matching is particularly onerous and unforgiving. Hence, searchers must specify a query that will locate the desired piece of information, without locating too much undesired information. Overly constrained queries will have no exact answer. Queries with insufficient constraints will have too many answers to be useful . Thus, the searcher must correctly constrain the query, with a suitable number of correctly selected constraints.

A common approach only identifies information that exactly corresponds to the constraints of the query. Another approach allows for inexact matches to some constraints. For example, the use of a wildcard character may be employed to locate variations of words being used in a query, for instance to capture both singular and plural occurrences of the word in the database . These approaches prove unsatisfactory in many situations. For example, it is common to have to reformulate a query numerous times before the results include the desired information, without including too much information. Often, it is not possible to form a satisfactory query that captures the desired information without capturing too much undesired information.

A further approach employs “fuzzy” matching capability. Such an approach typically replaces Boolean constraints with fuzzy counterparts. Thus, all tuples in the fuzzy or probabilistic database are directly or indirectly annotated with a number giving the probability or fuzziness level of the particular fact or piece of information. This approach disadvantageously requires the storage of a substantial amount of additional information.

These problems limit the usefulness of databases . This may particularly be a problem where queries are directed toward extracting useful knowledge from a large database of facts or other information.

2.5 QUANTUM COMPUTATION

A quantum computer is any physical system that harnesses one or more quantum effects to perform a computation. A quantum computer that can efficiently simulate any other quantum computer is called a Universal Quantum Computer (UQC).

In 1981 Richard P. Feynman proposed that quantum computers could be used to solve certain computational problems more efficiently than a UTM and therefore invalidate the Church-Turing thesis. Feynman (1982) noted that a quantum computer could be used to simulate certain other quantum systems, allowing exponentially faster calculation of certain properties of the simulated quantum system than is possible using a UTM.

There are several general approaches to the design and operation of quantum computers. One such approach is the “circuit model” of quantum computation. In this approach, qubits are acted upon by sequences of logical gates that are the compiled representation of an algorithm. Circuit model quantum computers have several serious barriers to practical implementation. In the circuit model, it is required that qubits remain coherent over time periods much longer than the single-gate time. This requirement arises because circuit model quantum computers require operations that are collectively called quantum error correction in order to operate. Quantum error correction cannot be performed without the circuit model quantum computer's qubits being capable of maintaining quantum coherence over time periods on the order of 1,000 times the single-gate time. Much research has been focused on developing qubits with coherence sufficient to form the basic information units of circuit model quantum computers (Shor, 2001). The art is still hampered by an inability to increase the coherence of qubits to acceptable levels for designing and operating practical circuit model quantum computers.

Another approach to quantum computation, involves using the natural physical evolution of a system of coupled quantum systems as a computational system. This approach does not make critical use of quantum gates and circuits. Instead, starting from a known initial Hamiltonian, it relies upon the guided physical evolution of a system of coupled quantum systems wherein the problem to be solved has been encoded in the terms of the system's Hamiltonian, so that the final state of the system of coupled quantum systems contains information relating to the answer to the problem to be solved. This approach does not require long qubit coherence

times. Examples of this type of approach include adiabatic quantum computation, cluster-state quantum computation, one-way quantum computation, quantum annealing and classical annealing (Farhi, E. et al, 2000).

As mentioned previously, qubits can be used as fundamental units of information for a quantum computer. As with bits in UTMs, qubits can refer to at least two distinct quantities; a qubit can refer to the actual physical device in which information is stored, and it can also refer to the unit of information itself, abstracted away from its physical device.

Qubits generalize the concept of a classical digital bit. A classical information storage device can encode two discrete states, typically labeled “0” and “1”. Physically these two discrete states are represented by two different and distinguishable physical states of the classical information storage device, such as direction or magnitude of magnetic field, current, or voltage, where the quantity encoding the bit state behaves according to the laws of classical physics. A qubit also contains two discrete physical states, which can also be labeled “0” and “1”. Physically these two discrete states are represented by two different and distinguishable physical states of the quantum information storage device, such as direction or magnitude of magnetic field, current, or voltage, where the quantity encoding the bit state behaves according to the laws of quantum physics. If the physical quantity that stores these states behaves quantum mechanically, the device can additionally be placed in a superposition of 0 and 1. That is, the qubit can exist in both a “0” and “1” state at the same time, and so can perform a computation on both states simultaneously. In general, N qubits can be in a superposition of 2 N states. Quantum algorithms make use of the superposition property to speed up some computations.

In standard notation, the basis states of a qubit are referred to as the |0 and |1 states. During quantum computation, the state of a qubit, in general, is a superposition of basis states so that the qubit has a nonzero probability of occupying the |0 basis state and a simultaneous nonzero probability of occupying the |1 basis state. Mathematically, a superposition of basis states means that the overall state of the qubit, which is denoted |Ψ, has the form |Ψ =a| +b|1, where a and b are coefficients corresponding to the probabilities |a| 2 and |b| 2 , respectively. The coefficients a and b each have real and imaginary components, which allows the phase of the qubit to be characterized. The quantum nature of a qubit is largely derived from its ability to exist in a coherent superposition of basis states and for the state of the qubit to have a phase.

A qubit will retain this ability to exist as a coherent superposition of basis states when the qubit is sufficiently isolated from sources of decoherence.

To complete a computation using a qubit, the state of the qubit is measured (i.e., read out). Typically, when a measurement of the qubit is performed, the quantum nature of the qubit is temporarily lost and the superposition of basis states collapses to either the |0 basis state or the |1 basis state and thus regaining its similarity to a conventional bit. The actual state of the qubit after it has collapsed depends on the probabilities |a| 2 and |b| 2 immediately prior to the readout operation.

2.6 GRAPH EMBEDDING

Graphs are an effective way of representing relationships among entities, and are commonly used in areas such as economics, mathematics, natural sciences and social sciences. While some graphs are simply used as a visual aid, others can be used to represent a problem to be solved. In fact, mapping a problem into graph format can sometimes help solve the problem. Instances of such problems include stock portfolio selection, microwave tower placement, delivery route optimization and other large-scale problems. Quantum computers can be used to solve such problems by way of translation of the original problem to a form that the quantum computer can solve. One method of doing this is through graph embedding, where a graph composed of a set of vertices and a set of edges that connect various vertices, representing a problem to be solved, is mapped into the qubit structure of a quantum processor and then solved.

Graph embedding is defining a particular drawing of a graph by mapping every node, or vertex, to a point on a plane and every edge to a straight or curved line that connects two nodes. This drawing is not unique, as there can be many permutations of the same graph. The number of ways a graph can be embedded depends on the characteristics and rules of the grid system upon which they are drawn. For example, one grid system can be a two-dimensional lattice. The edges may, for example, be constrained to be in two mutually orthogonal directions (e.g., up-down or left-right). Such a grid system has a connectivity of 4, meaning that each node can have at maximum four edges connected to it, the edges going only in the directions mentioned above. A similar grid system wherein edges can also extend diagonally (e.g., at 45°) and where they can cross is of connectivity 8. One form of graph embedding

involves taking a graph drawn on one grid system and drawing an equivalent graph on another grid system.

Graphs that can be embedded can be broken into two types: planar and non-planar. Planar graphs are graphs that can be embedded on a two-dimensional plane such that no two edges intersect. A non-planar graph is a graph where at least two edges intersect. Some forms of graph embedding involve embedding a planar graph into a non-planar graph or attempting to make a non-planar graph as planar as possible, i.e., by reducing the number of intersections. However, some non-planar graphs cannot be embedded into a planar graph. The most famous examples of such graphs are the graphs K5 and K(3, 3). More information on non-planar graphs and their embeddings can be found in Boyer et al., (2004).

One way of characterizing graph embeddings is their "efficiency", that is, the amount of resources (e.g., vertices and edges), area, and path lengths that an embedding uses . An efficient graph embedding uses fewer resources, has less area, has lower average path lengths, or any combination thereof between vertices than an inefficient graph embedding. Since the same graph can be embedded in more than one way, it is preferred to find the most efficient embedding possible. Finding the most efficient graph is also known as graph optimization.

For small graphs, finding the most efficient graph embedding might be quite easy and might be performed on a conventional processor in a reasonable amount of time. However, when the graph has a substantial number of vertices and edges, finding the most efficient graph embedding becomes a complex computational task. Several techniques have been developed to find an efficient graph embedding, such as that developed by Gutwenger et al., (2002). The Automated Graph Drawing (AGD) software program described in that publication is capable of mapping and compacting graphs using a variety of different techniques. However, all these techniques rely on the planarization of the original graph, which means the original graph is redrawn to have as few, if any, crossings as possible. This comes at the expense of having longer edge lengths and greater surface area, since non-planar graphs are generally more compact.

Other forms of graph embedding are discussed in Mutzel, (2002). Mutzel describes many different methodologies for graph embedding and optimization, but again all concentrate on making the graph as planar as possible. Part of the reason Mutzel desires planarity is that it is

aesthetically better. However, in instances where aesthetics is not an important aspect of graph optimization, the techniques outlined by Mutzel would not produce the most efficient graph embedding.

Graph embedding is also used in the field of very large scale integration (VLSI) chip design. Given an electronic circuit with many different elements that need to be wired together in a limited space and with specific design rules to be followed, finding an efficient wiring scheme is a form of graph embedding. Examples of applying graph embedding techniques to VLSI design can be found in Shields et al., (2001) and Heckmann et al., (1991). However, techniques for efficient wiring in chips are limited by the design rules and thus are usually not extendable to general forms of graph embedding.

A technique of graph embedding into a lattice of qubits is described in Knysh et al., (2005). Knysh describes a technique of mapping NP-complete problems into a lattice of qubits and performing adiabatic quantum computation to solve the problem . However, Knysh uses constant couplings between qubits and only nearest neighbor couplings, both of which reduce the flexibility and efficiency of the embedding and subsequent computation.

CHAPTER THREE: METHODOLOGY

3.1 INTRODUCTION

This chapter describes 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.

3.2 HIGH LEVEL DESCRIPTION

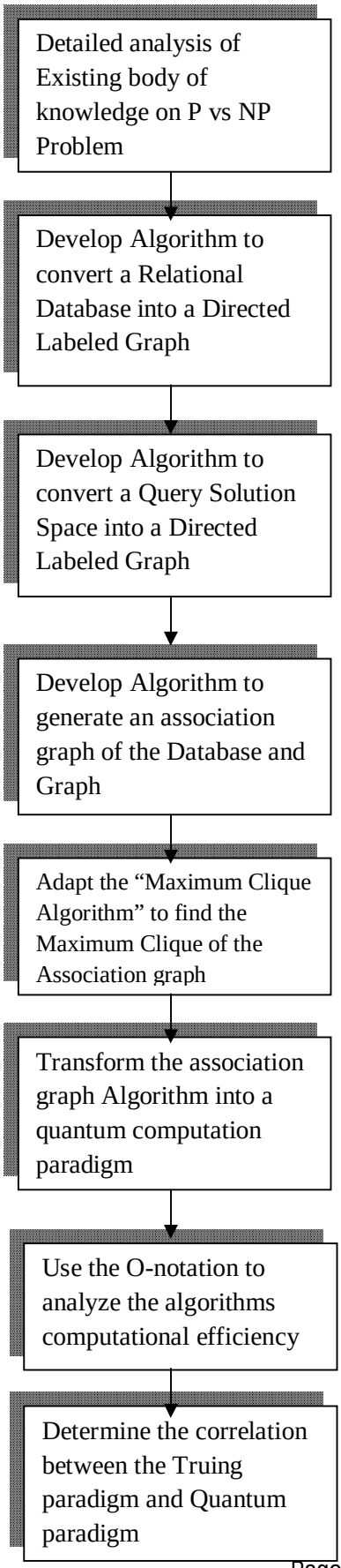

Figure 1 – Summary of methodology

3.3 ANALYSIS, DEVELOPMENT & 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 development of the algorithms to transform the relational database and Query into the 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 simple student's information management database was developed as the basis for developing and testing the algorithms. The database consisted of Students, Courses and Registrations records. Appendix A, Figure 2 lists the E-R diagram of the students' database used.

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. Appendix A, Figure 3 lists the sample data in the three tables of the students' database used. An adjacency matrix as a means of representing the directed labeled graphs was used during the algorithm development and testing

The Development of 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" Alan Bojic (Alan, 2012) 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.

3.4 STATUS OF P VS NP PROBLEM

The detailed review of the existing body of knowledge relating to the NP and P problems will include but not limited to:

i. Computational complexity theory
ii. Computability theory
iii. The P vs NP problem
iv. Graph theory
v. Maximum Clique Problem
vi. Quantum computational theory
vii. Graph embedding
viii. Algorithm analysis and Design

3.5 CONVERT A RELATIONAL DATABASE INTO A DIRECTED LABELED GRAPH

A Directed labeled graph G_b, may be constructed from a relation database as follows:

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

ii. The 1-arity relations (the relations consisting of a single attribute) can be represented by a loop connecting an element u ε V to itself and labeled with the name of the 1-arity relation. Similarly, a 2-arity relation involving a pair of elements u, is indicated as an edge (u, u') labeled by the name of the relation.

iii. Since any given element u or pair of elements u, u' may appear in multiple tables there may be multiple loops of edges in the graph (though these edges will all have different labels).

iv. A table element may appear in multiple positions in one or more tables. In principle, a single graph node could be used to represent this element and the element would have multiple edges or loops emanating from the node. This reuse would save space in the representation of the graph.

v. Most databases do not have only 1 -arity and 2-arity relations. Commonly, relations have more than two attributes, and often have keys which are compound (the table requires the specification of more than a single attribute to uniquely specify a row). A relation with more than 2 attributes, it is converted to a 2-arity relations follows:

   a) For a table with *a* attributes, any set of *a-1* attributes which uniquely identify a row may be aggregated into a single compound attribute. Then, the 2-arity table consisting of two attributes is formed: firstly the single compounds attribute and secondly the remaining attribute which was not included in the definition of the compound attribute.

   b) Another way to view the conversion of non-2-arity relations to 2-arity relations is the following; A table with more than two attributes is a hyperedge in the hypergraph representing the database. To convert the hypergraph to a graph a new node is introduced for each hyperedge. This new node is then connected to all the other nodes in the hyperedge. In this way the node corresponding to the key is a unique node which identifies the hyperedge.

   c) A row of a relation with arity A (with A>2) is converted into a graph of A+1 nodes. Nodes 1 through A represent the information stored in each of the A attributes. Node A+1 represents the key for the row (since the key uniquely identifies the row). The key node is connected to each of the attribute nodes. The edge between the key and the ith attribute is labeled Ra_i, where R is the relation name and a_i is the ith attribute name.

The pseudo algorithm to convert a relational database into a directed Graph is presented in section 4.2

3.6 CONVERT A QUERY INTO A DIRECTED LABELED GRAPH

A query labeled directed graph (G_q), being a representation of the database solution space subset may be constructed as follows:

i. An analogy to the database procedure above may be used to represent queries. Any conjunctive query may be represented by the conjunction of a number of single relation queries, for example:

the SQL statement;

SELECT a.,b.Student_Name, b.Sudent_Tel,c.Course_Name,c.Course_Lecturer*
FROM
Registration a, Student b, Course c
WHERE
a.Reg_Student_No = b.Student_No AND a.Reg_Course_Code = c.Course_Code AND c.Course_Lecturer = 'Mr. Mwaura'.

This SQL statement query involves three tables and returns the details of students who have registered in a Course taught by "Mr. Maura". This query may be processed exactly as in Method 4.3 above where now each table includes only a single row (unless the same table is referred to multiple times in the query), and the row may involve variables. The only difference is that nodes associated with variables are annotated to include this information. This procedure may be extended to allow for negation of predicates. For queries with multiple bodies—disjunctive queries (i.e., representing OR operations)—the query for each body is formed in the same manner as described above in reference to conjunctive queries.

ii. It may be desirable to minimize the sizes of the graphs produced by converting databases and queries to graphs. In particular, the complexity of an SQL query may be a function of the number of nodes in both the database and query graphs, in which case it may be desirable to extract only the portion (or portions) of the database that is relevant to answering the query.

iii. The database may be pruned in the two ways alluded to above: firstly only tables associated with the query are included, and secondly all such tables are projected down to eliminate rows which do not match grounded (i.e., bound versus free or unbound) variables. This sets the database domain for all variables

Appendix A figure 4 lists the SQL query used as the test control for the development of the graph algorithms and the expected resulted as generated from the student database. These results are finally compared with the maximum clique algorithm results.

The pseudo algorithm to convert a query, by pruning the relevant representation of the database solution space into a directed Graph is presented in section 4.3

3.7 GENERATE AN ASSOCIATION GRAPH OF THE DATABASE AND QUERY

The association graph $G_{a,}$

(1). $G_a=(V_a,E_a,)$

may be constructed from a database graph G_b and a query graph G_q as follows:

i. Vertices of the association graph (a-vertices) are possible pairs of vertices from V_b and V_q having the same label, i.e.,

(2). $V_a=\{(v_{b,}v_q)|v_b \varepsilon\ v_q\ = \mu_q(v_{b,}v_q\ )\}$

As described above, the edge/loop label mappings return sets of labels. Equality between label sets,

(3). $\mu(v_b) = \mu(v_q)$,

means that the sets are identical, that is, they have the same number of elements and the same elements

ii. Edges of the association graph (a-edges) may be defined to be present/absent as follows. If

(4). $(v_{b,}v_q\ )\varepsilon V$,

i.e. v_b and v_q have compatible label sets, and

(5). $(v_{b,}\ v'_{q)}\ \varepsilon\ V_a$

then these two a-vertices may be connected.

The pseudo algorithm to generate the association graph is presented in section 4.4

3.8 MAXIMUM CLIQUE OF THE ASSOCIATION GRAPH

The association graph above is constructed in order to allow identification of labeled graphs common to both G_b and G_q.

i. The goal is to identify the largest common labeled pseudograph by finding a largest (maximum size) clique in the association edges. A maximal clique is a set of fully connected nodes in a graph such that an additional node cannot be added to the set with the set still remaining fully connected. The set of nodes making up a maximal

clique is not necessarily a maximum clique, that is, the largest set of mutually connected nodes.

ii. The maximum or max clique (MC) problem in a graph

(6). $G=(V, E)$

having vertex set V and edge set

(7). $E \in VxV$

or equivalently the largest subset of nodes of V all pairs of which are connected by an edge, also known as the largest clique. MC is related to the maximum independent set problem (MIS) through the complement graph, in other words, the MC of G is equal to the MIS of co-G where

(8). $co\text{-}G=(V,E^c)$ and

(9). $E^c=(Vx\ V)\backslash E$

So E^c is the set of anti-edges of G. MC is an NP complete problem.

iii. The maximum clique in the association graph identifies a set of a-vertices in V_a. Recalling that the a-vertices consist of pairs of vertices from G_b and G_q these flagged a-vertices define the values for the variables in the query by matching the vertex with the paired v_q vertex with the paired v_b. If the clique consists of all vertices of the association graph then the query can be matched exactly (in which case the query could also have been solved with a standard SQL query).

iv. Of most interest is the case where the largest common pseudograph is a subset of the query. In this case there is no exact match so any standard SQL query would return an empty set of results. However, the largest common sub-graph identifies the next best response to the query which requires the least number of relaxations to satisfy the query.

v. By employing the present systems, methods and articles, a query or database search may be reduced to the determination of cliques in the association graph formed from the query and the relevant portion of the database. As discussed below, a number of

algorithms exist that may be used to solve the (max) clique problem, including both classical and quantum approaches.

vi. Maximum clique is a well studied problem and there are many complete and incomplete methods. The typical association graphs that result from queries or database searches are large enough that complete methods are impractical and so focus is placed on incomplete search methods. Such methods will return large cliques, but not necessarily the largest clique. A number of such methods exist including Reactive Local Search, Deep Adaptive Greedy Search, k-opt algorithm, Edge-AC+LS and Dynamic Local Search among others.

The solution to the above maximum clique problem is an NP problem. The Maximum Clique problem algorithm for the association graph is presented in section 4.5

3.9 MAXIMUM CLIQUE ON A QUANTUM COMPUTATION PARADIGM

Quantum computing offers an alternative for solving maximum clique problems. The clique problem is embodied directly in physical hardware and natural physical evolution processes can be harnessed to determine cliques.

i. Each clique in the association graph gives a relaxation of the query. Larger cliques give better scoring and closer to ideal relaxations. The present systems, methods and articles employ the discovery of many preferably large cliques. Most of the classical heuristics and quantum algorithms are stochastic and naturally return random solutions. By repeatedly running such algorithms a number of distinct solutions may be obtained.

ii. To decrease the likelihood that previously discovered answers are returned many of these algorithms can be modified so that a tunable amount of randomness affects the results. The higher the randomness setting the smaller the cliques but the more likely it is that the clique will be different from any previously discovered clique. Given this tunable randomness setting, the randomness may be slowly increased so that the better cliques are the first to be discovered.

iii. Rather than finding large cliques in the association graph it may be useful to directly search for large common subgraphs between the query and database labeled

pseudo-graphs. There are algorithms which address this problem, but typically they have been developed only for simple graphs. In some cases however these algorithms are directly extensible to labeled pseudographs.

3.10 ALGORITHM ANALYSIS USING O-NOTATION

An algorithm can be said to exhibit a growth rate on the order of a mathematical function if beyond a certain input size n, the function f(n) times a positive constant provides an upper bound or limit for the run-time of that algorithm. Big O notation is a convenient way to express the worst-case scenario for a given algorithm, although it can also be used to express the average-case — for example, the worst-case scenario for quicksort is

(10). $\Theta(n^2)$, but the average-case run-time is

(11). $\Theta(n \log n)$ (Sedgewick, 1978)

The above algorithms will be analysed for their efficiency using the Big Θ notation

3.11 COMPARISON OF TURING AND QUANTUM COMPUTATIONAL PARADIGM

From the algorithms analysis, it will be possible to correlate the behavior of the both P and NP problems under different computational paradigms. The analysis shall give the indications of the effect of transforming P problems (Database Conversion to Directed Graph, Query representation, Association graph generation) into an NP problem (Maximum clique problem)

The behavior of the maximum clique problem on the quantum computational paradigm shall either collaborate or invalidate the behavior of the NP under the new computation paradigm.

3.12 SUMMARY

It is envisaged that a method will be found 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 derive the same by graph embodiment into a quantum computer.

CHAPTER FOUR: RESULTS

4.1 CURRENT STATUS OF P vs NP

A detailed literature review on the status of P vs NP problem has been described in section 2. Suffice to say, the energy and interest geared towards resolving this open problem would warrant a research paper on the "Current Status of the P vs NP" Problem.

As a sequel to this section, i undertake to present an academic paper on the same. This paper will however not be part of this research project.

In summary, 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 what we will 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.

Learning becomes easy by using the principle of Occam's razor (Occam's, 2002), 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 which are usually extremely long. We can also 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 fewer than 100 pages.

A person who proves P = NP would walk home from the Clay Institute not with $1 million check but with seven (actually six since the Poincaré Conjecture appears solved).

However, most complexity theorists generally believe $P \neq NP$ and such a beautiful world cannot exist.

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 (Shor, 1997) 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? This is part of the question this research project attempts to investigate.

4.2 CONVERT A DATABASE INTO A DIRECTED LABELLED GRAPH

4.2.1 ALGORITHM

The algorithm requires that:

- The database b, has a set of relations R; Where
  (12). $R = r_i$;for i=1, ..,|R|
- The resultant graph G_b, is a labelled directed graph where
  (13). $G_b=(V_b, E_b, \mu_b)$

```
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
                                                                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 attributes so add a loop
19.           E←ADDEDGE(n,n);
20.           μb(n,n):=TableName;
21.         Else
22.         For i ε 1,  [NonKeyAttrib] do //Add edge between key and non-key attribute
23.           m←t[NonKeyAttrib[i]];
24.           Vb←ADDNODE(m,Vb);
25.           Eb:=ADDEDGE(<n,m>,Eb);
26.           μb(n,m):=TableName.NonKeyAttrib[i];
27.         End for
28.       End if
29.     End for
30.  End for
31. End Procedure
```

4.2.2 DESCRIPTION

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

i. 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

ii. 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.

iii. 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.

iv. 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 i^{th} non key attribute node.

The results of the algorithm as simulated using the student database are presented in Appendix B, table 1– Adjacency matrix representation of the database graph.

4.2.3 ANALYSIS

i. First inner loops, and let n = KeyAttribs or n = NonKeyAttribs
Line 22- 1 operation (initialization) + 2(n + 1) operations (Tests) + n operations = 2n + 2
Line 23-26: 4 arithmetic operation executed n times = 4n operations

Total number of operations for the inner loop operations (lines 22 - 26) = (2n + 2) + 4n = 6n + 2

Thus, the running time of the inner loop is Θ (n), or *linear* time since control passes to either of the loop based on the if condition.

ii. Second the outer loop:
Line 6-1 operation (initialization) + (n + 1) operations (test) + n operations (increment) = 2n + 2 operations

Line 7-8-2 arithmetic operation executed n times = 2n operations
Body of Second outer loop =Total number of operations for inner loop executed n times
(14). (3n +2n+ 2) * n = 3n^2 + 2n^2 +2n

Total number of operations

(15). 2n + 2 (line 1) + 3n^2 + 2n^2 +2n (body of Second outer loop) = 5n^2 + 2n

Thus, the running time of the nested loops is Θ (n^2).

iii. Outer loop:
Line 1- 1 operation (initialization) + (n + 1) operations (test) + n operations (increment) = 2n + 2 operations
Line 2-3: 2 arithmetic operation executed n times = 2n operations
Body of outer loop =Total number of operations for Second inner loop and inner loop executed n times
(16). (3n +2n+ 2) * n *n = 3n^3 + 2n^3 +2n operations

Total number of operations

(17). 2n + 2 (line 1) + 3n^3 + 2n^3 +2n (Second outer loop) = 3n^3 + 4n + 2

Thus, the running time of the nested loops is Θ (n^3).

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

(18). **Θ(n^3).**

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

(19). ***Θ(|R|^3).***

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

4.3 CONVERT A QUERY Q, INTO A DIRECTED LABELLED GRAPH.

4.3.1 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,

(20). $q=\{Q1, Q2, ...Q_{|q|}\}$

The algorithm requires that:

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

```
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
```

4.3.2 DESCRIPTION

i. 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.

ii. 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.

The results of the algorithm as simulated using the student database are presented in Appendix B, table 2 – Adjacency matrix representation of the query graph.

4.3.3 ANALYSIS

i. The function SELECTAND PROJECT, let Ri = n and qi =n
Line 5

(25). n*n+ 2(n) = n^2 + 2n operations

Thus, the running time of the function is

(26). O(n^2)

ii. Outer loop:

Line 1-1 operation (initialization) + (n + 1) operations (test) + n operations (increment) = 2n + 2 operations

Line 3-4 - 2 arithmetic operation executed n times = 2n operations
Outer loop =Total number of operations for SELECTAND PROJECT executed n times =

(27). (3n +2n+ 2) * n = 3n^2 + 4n^2 +2n operations

Total number of operations =

(28). n^2 + 2n + 3n^2 + 2n^2 +2n = 5n^2 + 4n

Thus, the running time of the nested loops is O(n^2).

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).***

4.4 GENERATE ASSOCIATION GRAPH (G_a) FROM THE PRUNED QUERY (G_q) AND DATABASE (G_b)

4.4.1 ALGORITHM

The algorithm requires that:

- A set of pruned relations R; Where
  (29). $R = r_i$;for i=1, ..,r|R|
- A pruned query q'; Where q'
  (30). $q' = Q'_i$ for i = 1,..,|q|
- The association graph G_a
  (31). $G_a=(V_a, E_a)$
  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.     [Vq^i,Eq^i, μq^i]←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,,Vq^i, μq^i,TableName)
14.        E'a:= ASSOCIATEEDGES(V'a,Et,Eq^i, μq^i,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
```

4.4.2 DESCRIPTION

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

i. 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.

ii. 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.

iii. 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 V_a, of a-vertices duplicates are not permitted (line 15).

iv. 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 μ_q^i 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 E_a.

The results of the algorithm as simulated using the student database are presented in Appendix B, table 3 – Adjacency matrix representation of the association graph.

4.4.3 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 ***Θ*(n^3).**

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

4.5 MAXIMUM CLIQUE OF THE ASSOCIATION GRAPH G_a.

4.5.1 ALGORITHM

The algorithm requires that:

- Association Graph G_a,
  (32). $G_a = (V_a, E_a)$,
- 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)
```

CLIQUE Recursive Subroutine

```
1.  procedure CLIQUE(Ga U, size)
2.     if U = 0 then
3.     if size > max then
4.        max ← size
5.        return
6.      while |U| > 0 do
7.           if size + |U| <= max then   //Pruning 4
8.                return
9.           Select any vertex u from U
10.          U←U \{u}
11.          N'(u) := {w|w ε N(u) ^ d(w) >= max  // Pruning 5
12.          CLIQUE(Ga,U ∩N'(u), size + 1)
```

4.5.2 DESCRIPTION

i. 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.

ii. Throughout the algorithm, the variable *max* stores the size of the maximum clique found thus far. Initially it is set to be equal to the lower bound l*b* provided as an input parameter, and it gives the maximum clique size when the algorithm terminates.

iii. To obtain the largest clique containing a vertex vi, it is sufficient to consider only the neighbors of vi. The main routine MAXCLIQUE thus generates for each vertex vi ε V a set U c N(vi) (neighbors of vi that survive pruning) and calls the subroutine CLIQUE on U.

iv. 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 (R. Carraghan and P. Pardalos, 1990)

v. We use size to maintain the size of the clique found at any point through the recursion. Since we start with a clique of just one vertex, the value of size is set to be one initially when the subroutine CLIQUE is called (Line 10).

vi. 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.

vii. While forming the neighbor list *U* for a vertex v*i*, 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 $v_j \varepsilon N(v_i)$ that have degree less than the current value of *max*, since any such vertex could not form a clique of size larger than *max*.

viii. 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.

ix. 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.

The results of the algorithm as simulated using the student database are presented in Appendix B, table 4 – Adjacency matrix representation of the maximum Clique graph. table 5 shows the results of maximum clique translation into the equivalent database representation while table 6 shows the results of maximum clique being the same as that of the initial relational database query used for the development, testing and simulation of the .

4.5.3 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

(33). 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

To militate against the exponential growth, the pruning sections introduced ensure that the algorithm terminates if the given size of the clique is not found.

Though the algorithm returns the size of the maximum clique, getting the clique subgraphs is a non-trivial task as it would requires extraction of the sub-graph forming the maximum clique of the returned size.

4.6 MAXIMUM CLIQUE ON QUANTUM COMPUTATION.

A Quantum Algorithm for finding the maximum Clique on a graph is given by Allan Bojic (Allan, 2012), 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 4.5 above

CHAPTER FIVE: DISCUSSION

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

i. A method does exist that can transform a relation database into a directed labeled graph. The running time complexity of the algorithm is ***Θ*(n^3).** 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

ii. 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 ***Θ*(n^2).** 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

iii. A method exists that generate an association graph of the database and query graph. The running time complexity of the algorithm is **Θ(n^3).** 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

iv. 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.

v. 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 ***Θ* (n^2/3).**

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 mechanicals.

CHAPTER SIX: 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 current flow as medium of realizing the implementation.

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

CONJUCTURE

Supposing we had a new element Oracle Ω in the periodic table, with *N* states. It suffices to say that it would be possible reduce the computational time complex of many NP-Complete problems with the 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

The following areas, arising from the research project are proposed for further research:

i. 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.

ii. The maximum clique problem on quantum computation is based on the Alan Jobic algorithm (Jobic, 2012) which is an improvement of the Grower's database search algorithm (Grower, 1996) Further research is required on the effect of using Shor's fourier transformation quantum computation algorithm (Matthew, 2005).

iii. CERN, has announced the possibility of the Boson particle (CMS collaboration; Khachatryan, Sirunyan, Tumasyan, Adam, Aguilo, Bergauer, Dragicevic, et al. 2012) identified by the Large Hydron Collider (LHC) experiment as being the elusive Higgs Boson. Further research on what value the Boson can add.

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

REFERENCES

Alan, Turing, 1936, *On computable numbers with an application to the entscheidnungsproblem*, Proc. London Math. Soc. 42 230–265.

Arora, Sanjeev and Barak, Boaz , 2009, *Computational Complexity: A Modern Approach,* Cambridge

Boyer et al., 2004. , *Journal of Graph Algorithms and Applications 8*, pp. 241-273.

Clay Math Institute of Mathematics, 2000, *Official Problem Description*, Available from http://www.claymath.org/millennium/P_vs_NP/pvsnp.pdf [15th August 2012]

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

CMS collaboration; Khachatryan, V.; Sirunyan, A.M.; Tumasyan, A.; Adam, W.; Aguilo, E.; Bergauer, T.; Dragicevic, M. et al. 2012, *Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC*, Physics Letters B 716 (1): 30–61. arXiv:1207.7235. doi:10.1016/j.physletb.2012.08.021.

Dolan et al, 2002 , *SIAM News* Vol. 35, p. 6.

Farhi, E. et al, 2000, *Quantum Adiabatic Evolution Algorithms versus Stimulated Annealing,* arXiv.org:quant-ph/0201031 pp 1-24.

Feynman, R, P, 1982, *Simulating Physics with Computers, International Journal of Theoretical Physics*, Vol. 21 pp. 467-488.

Fourer et al., 2001 , *Interfaces 31*, pp. 130-150.

Gutwenger et al., 2002, *Lecture Notes in Computer Science 2269*, pp. 307-323

Heckmann et al., 1991, *Proc. 17 th Int. Workshop on Graph-Theoretic Concepts in Comp. Sci.,* pp. 25-35

High - Level Look at Optimization: Past, Present, and Future, 2000, Available from http://wenku.baidu.com/view/1ee80bfc04a1b0717fd5dd02.html, [16 January 2013]

Hopcroft, J.E., Motwani, R. and Ullman, J.D., 2007, *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley, Boston/San Francisco/New York (page 368)

Jobic, Alan, 2012, *A Quantum Algorithm for finding the maximum Clique on a undirected graph,* Jios, Vol 36 No. 2.

Knysh et al., 2005, G*raph Embedding into a Lattice of Qubits*, arXiv.org:quant-ph/0511131

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

Lov, K, Grover , 1996, *A fast quantum mechanical algorithm for database search*, Proceedings, 28th Annual ACM Symposium on the Theory of Computing (STOC), pages 212-219

Matthew, Hayward, 2005, *Quantum Computing and Shor's Algorithm*, imsa.edu

Mutzel, 2002 , *Handbook of Applied Optimization* , Oxford University Press, New York, pp. 967-977

Occam's, Razor, 2002, *Formal basis for a physical theory*, arxiv.org AN Soklakov - Foundations of Physics Letters– Springer

R. Carraghan and P. Pardalos, 1990, *An exact algorithm for the maximum clique problem*, Oper. Res. Lett. 9 pp.375–382.

Sedgewick, R. 1978, *Implementing Quicksort programs*, Comm. ACM 21 (10): 847–857

Shields et al., 2001 , *Parallel and Distributed Computing and Systems Conference*, Anaheim, California

Shirts et al., 2000, *Science 290*, pp. 1903-1904, and Allen et al, 2001 , *IBM Systems Journal* 40, p. 310

Shor, P. W. ,2001, *Introduction to Quantum Algorithms*, arXiv.org:quant-ph/0005003, pp. 1-27

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

Turing, M., 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 [05 October 2010]

William, I. Gasarch, June 2002, *The P=?NP* poll.

Wolfram, Stephen, 2002, *A New Kind of Science*, Wolfram Media, Inc.

APPENDIX A – THE STUDENTS TEST DATABASE

FIGURE 2 – THE STUDENT DATABASE E-R DIAGRAM

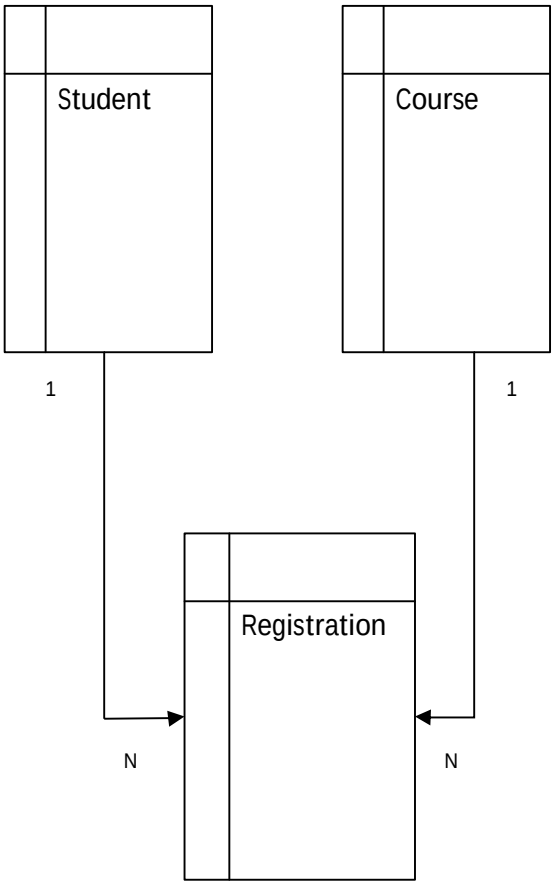

FIGURE 3 – THE SAMPLE DATA USED IN THE STUDENT DATABASE

A. STUDENTS DATA

Student_No	Student_Name	Student_Tel	Student_Address	Student_DOB
P56/1000/2005	Shadrack	0722112233	321 NBI	16/08/1973
P56/1001/2005	John	0722112234	322 NBI	17/08/1973
P56/1002/2005	Paul	0722112235	323 NBI	18/08/1973
P56/1003/2005	Mwangi	0722112236	324 NBI	19/08/1973
P56/1004/2005	Kamau	0722112237	325 NBI	20/08/1973

B. COURSES DATA

Course_Code	Course_Name	Course_Hrs	Course_Room	Course_Lecturer
C001	Introduction	30	Lecture Room 1	Mr. Mwaura
C002	Algorithms	30	Blue Room	Prof. Waema
C003	Complexity	30	Theatre 1	Mr. Orwa
C004	Automata Theory	30	Hall 3	Dr. Wanjiku
C005	Programming	30	Theatre 1	Mrs. Masinde

C. REGISTRATION DATA

Reg_Student_No	Reg_Course_Code	Reg_Date	Reg_Semister
P56/1000/2005	C001	01/11/2012	Sem 1
P56/1000/2005	C002	01/11/2012	Sem 2
P56/1000/2005	C003	01/11/2012	Sem 3
P56/1003/2005	C003	01/11/2012	Sem 2
P56/1003/2005	C004	01/11/2012	Sem 1
P56/1003/2005	C005	01/12/2012	Sem 3
P56/1004/2005	C001	01/12/2012	Sem 2
P56/1004/2005	C005	01/12/2012	Sem 2
P56/1001/2005	C001	01/12/2012	Sem 1
P56/1001/2005	C002	01/12/2012	Sem 3

D. SQL STATEMENT AND EXPECTED RESULTS

```
Select a.*,b.Student_Name, b.Sudent_Tel,c.Course_Name,c.Course_Lecturer
FROM
Registration a, Student b, Course c
WHERE
a.Reg_Student_No = b.Student_No AND a.Reg_Course_Code = c.Course_Code
AND c.Course_Lecturer = 'Mr. Mwaura'
```

Student_No	Student_name	Student_Tel	Reg_date	Reg_Semister	Course_Name	Course_Lecturer
P56/1000/2005	Shadrack	0722112233	01/11/2012	Sem 1	Introduction	Mr. Mwaura
P56/1001/2005	John	0722112234	01/12/2012	Sem 2	Introduction	Mr. Mwaura
P56/1004/2005	Kamau	0722112237	01/12/2012	Sem 1	Introduction	Mr. Mwaura

APPENDIX B – THE ALGORITHMS TEST RESULTS

TABLE 1 – STUDENT DATABASE GRAPH ADJACENT MATRIX REPRESENTATION

TableName.nonKeyAttrib[i]		TableName.keyAttrib[i]									
		Student.Student_No					Course.Course_Code				
		P56/100/05	P56/101/05	P56/102/05	P56/103/05	P56/104/05	C001	C002	C003	C004	C005
Student.Student_name	Shadrack	1	0	0	0	0	0	0	0	0	0
	John	0	1	0	0	0	0	0	0	0	0
	Paul	0	0	1	0	0	0	0	0	0	0
	Mwangi	0	0	0	1	0	0	0	0	0	0
	Kamau	0	0	0	0	1	0	0	0	0	0
Student.Student_Tel	0722112233	1	0	0	0	0	0	0	0	0	0
	0722112234	0	1	0	0	0	0	0	0	0	0
	0722112235	0	0	1	0	0	0	0	0	0	0
	0722112236	0	0	0	1	0	0	0	0	0	0
	0722112237	0	0	0	0	1	0	0	0	0	0
Student.Student_Address	321 NBI	1	0	0	0	0	0	0	0	0	0
	322 NBI	0	1	0	0	0	0	0	0	0	0
	323 NBI	0	0	1	0	0	0	0	0	0	0
	324 NBI	0	0	0	1	0	0	0	0	0	0
	325 NBI	0	0	0	0	1	0	0	0	0	0
Student.Student_DOB	16/08/1973	1	0	0	0	0	0	0	0	0	0
	17/08/1973	0	1	0	0	0	0	0	0	0	0
	18/08/1973	0	0	1	0	0	0	0	0	0	0
	19/08/1973	0	0	0	1	0	0	0	0	0	0
	20/08/1973	0	0	0	0	1	0	0	0	0	0
Course.Course_name	Introduction	0	0	0	0	0	1	0	0	0	0
	Algorithms	0	0	0	0	0	0	1	0	0	0
	Complexity	0	0	0	0	0	0	0	1	0	0
	Automata	0	0	0	0	0	0	0	0	1	0
	Programming	0	0	0	0	0	0	0	0	0	1
Course.Course_Hrs	30	0	0	0	0	0	1	0	0	0	0
	30	0	0	0	0	0	0	1	0	0	0
	30	0	0	0	0	0	0	0	1	0	0
	30	0	0	0	0	0	0	0	0	1	0
	30	0	0	0	0	0	0	0	0	0	1
Course.Course_Room	Lecture Room 1	0	0	0	0	0	1	0	0	0	0
	Blue Room	0	0	0	0	0	0	1	0	0	0
	Theatre 1	0	0	0	0	0	0	0	1	0	0
	Hall 3	0	0	0	0	0	0	0	0	1	0
	Theatre 1	0	0	0	0	0	0	0	0	0	1
Course.Course_Lecturer	Mr. Mwaura	0	0	0	0	0	1	0	0	0	0
	Prof. Waema	0	0	0	0	0	0	1	0	0	0
	Mr. Orwa	0	0	0	0	0	0	0	1	0	0
	Dr. Wanjiku	0	0	0	0	0	0	0	0	1	0
	Mrs. Masinde	0	0	0	0	0	0	0	0	0	1
Registration.Reg_date	01/11/2012	1	0	0	0	0	1	0	0	0	0
	01/11/2012	1	0	0	0	0	0	1	0	0	0
	01/11/2012	1	0	0	0	0	0	0	1	0	0
	01/11/2012	0	0	0	1	0	0	0	1	0	0
	01/11/2012	0	0	0	1	0	0	0	0	1	0
	01/12/2012	0	0	0	1	0	0	0	0	0	1
	01/12/2012	0	0	0	0	1	1	0	0	0	0
	01/12/2012	0	0	0	0	1	0	0	0	0	1
	01/12/2012	0	1	0	0	0	1	0	0	0	0
	01/12/2012	0	1	0	0	0	0	1	0	0	0
Registration.Reg_Semister	Sem 1	1	0	0	0	0	1	0	0	0	0
	Sem 2	1	0	0	0	0	0	1	0	0	0
	Sem 3	1	0	0	0	0	0	0	1	0	0
	Sem 2	0	0	0	1	0	0	0	1	0	0
	Sem 1	0	0	0	1	0	0	0	0	1	0
	Sem 3	0	0	0	1	0	0	0	0	0	1
	Sem 2	0	0	0	0	1	1	0	0	0	0
	Sem 2	0	0	0	0	1	0	0	0	0	1
	Sem 1	0	1	0	0	0	1	0	0	0	0
	Sem 3	0	1	0	0	0	0	1	0	0	0

TABLE 2 – STUDENT QUERY GRAPH ADJACENT MATRIX REPRESENTATION

TableName.nonKeyAttrib[i]		TableName.keyAttrib[i]									
		Student.Student_No					Course.Course_Code				
		P56/100/05	P56/101/05	P56/102/05	P56/103/05	P56/104/05	C001	C002	C003	C004	C005
Student.Student_name	Shadrack	1	0	0	0	0	0	0	0	0	0
	John	0	1	0	0	0	0	0	0	0	0
	Paul	0	0	1	0	0	0	0	0	0	0
	Mwangi	0	0	0	1	0	0	0	0	0	0
	Kamau	0	0	0	0	1	0	0	0	0	0
Student.Student_Tel	0722112233	1	0	0	0	0	0	0	0	0	0
	0722112234	0	1	0	0	0	0	0	0	0	0
	0722112235	0	0	1	0	0	0	0	0	0	0
	0722112236	0	0	0	1	0	0	0	0	0	0
	0722112237	0	0	0	0	1	0	0	0	0	0
Course.Course_name	Introduction	0	0	0	0	0	1	0	0	0	0
	Algorithms	0	0	0	0	0	0	1	0	0	0
	Complexity	0	0	0	0	0	0	0	1	0	0
	Automata	0	0	0	0	0	0	0	0	1	0
	Programming	0	0	0	0	0	0	0	0	0	1
Course.Course_Lecturer	Mr. Mwaura	0	0	0	0	0	1	0	0	0	0
Registration.Reg_date	01/11/2012	1	0	0	0	0	1	0	0	0	0
	01/11/2012	1	0	0	0	0	0	1	0	0	0
	01/11/2012	1	0	0	0	0	0	0	1	0	0
	01/11/2012	0	0	0	1	0	0	0	1	0	0
	01/11/2012	0	0	0	1	0	0	0	0	1	0
	01/12/2012	0	0	0	1	0	0	0	0	0	1
	01/12/2012	0	0	0	0	1	1	0	0	0	0
	01/12/2012	0	0	0	0	1	0	0	0	0	1
	01/12/2012	0	1	0	0	0	1	0	0	0	0
	01/12/2012	0	1	0	0	0	0	1	0	0	0
Registration.Reg_Semister	Sem 1	1	0	0	0	0	1	0	0	0	0
	Sem 2	1	0	0	0	0	0	1	0	0	0
	Sem 3	1	0	0	0	0	0	0	1	0	0
	Sem 2	0	0	0	1	0	0	0	1	0	0
	Sem 1	0	0	0	1	0	0	0	0	1	0
	Sem 3	0	0	0	1	0	0	0	0	0	1
	Sem 2	0	0	0	0	1	1	0	0	0	0
	Sem 2	0	0	0	0	1	0	0	0	0	1
	Sem 1	0	1	0	0	0	1	0	0	0	0
	Sem 3	0	1	0	0	0	0	1	0	0	0

TABLE 3 – ASSOCIATION GRAPH ADJACENT MATRIX REPRESENTATION

TableName.nonKeyAttrib[i]		TableName.keyAttrib[i]									
		Student.Student_No					Course.Course_Code				
		P56/100/05	P56/101/05	P56/102/05	P56/103/05	P56/104/05	C001	C002	C003	C004	C005
Student.Student_name	Shadrack	1	0	0	0	0	0	0	0	0	0
	John	0	1	0	0	0	0	0	0	0	0
	Paul	0	0	1	0	0	0	0	0	0	0
	Mwangi	0	0	0	1	0	0	0	0	0	0
	Kamau	0	0	0	0	1	0	0	0	0	0
Student.Student_Tel	0722112233	1	0	0	0	0	0	0	0	0	0
	0722112234	0	1	0	0	0	0	0	0	0	0
	0722112235	0	0	1	0	0	0	0	0	0	0
	0722112236	0	0	0	1	0	0	0	0	0	0
	0722112237	0	0	0	0	1	0	0	0	0	0
Course.Course_name	Introduction	0	0	0	0	0	1	0	0	0	0
Course.Course_Lecturer	Mr. Mwaura	0	0	0	0	0	1	0	0	0	0
Registration.Reg_date	01/11/2012	1	0	0	0	0	1	0	0	0	0
	01/11/2012	1	0	0	0	0	0	1	0	0	0
	01/11/2012	1	0	0	0	0	0	0	1	0	0
	01/11/2012	0	0	0	1	0	0	0	1	0	0
	01/11/2012	0	0	0	1	0	0	0	0	1	0
	01/12/2012	0	0	0	1	0	0	0	0	0	1
	01/12/2012	0	0	0	0	1	1	0	0	0	0
	01/12/2012	0	0	0	0	1	0	0	0	0	1
	01/12/2012	0	1	0	0	0	1	0	0	0	0
	01/12/2012	0	1	0	0	0	0	1	0	0	0
Registration.Reg_Semister	Sem 1	1	0	0	0	0	1	0	0	0	0
	Sem 2	1	0	0	0	0	0	1	0	0	0
	Sem 3	1	0	0	0	0	0	0	1	0	0
	Sem 2	0	0	0	1	0	0	0	1	0	0
	Sem 1	0	0	0	1	0	0	0	0	1	0
	Sem 3	0	0	0	1	0	0	0	0	0	1
	Sem 2	0	0	0	0	1	1	0	0	0	0
	Sem 2	0	0	0	0	1	0	0	0	0	1
	Sem 1	0	1	0	0	0	1	0	0	0	0
	Sem 3	0	1	0	0	0	0	1	0	0	0

TABLE 4 – MAXIMUM CLIQUE ADJACENT MATRIX REPRESENTATION

TableName.nonKeyAttrib[i]		TableName.keyAttrib[i]									
		Student.Student_No					Course.Course_Code				
		P56/100/05	P56/101/05	P56/102/05	P56/103/05	P56/104/05	C001	C002	C003	C004	C005
Student.Student_name	Shadrack	1	0	0	0	0	0	0	0	0	0
	John	0	1	0	0	0	0	0	0	0	0
	Paul	0	0	1	0	0	0	0	0	0	0
	Mwangi	0	0	0	1	0	0	0	0	0	0
	Kamau	0	0	0	0	1	0	0	0	0	0
Student.Student_Tel	0722112233	1	0	0	0	0	0	0	0	0	0
	0722112234	0	1	0	0	0	0	0	0	0	0
	0722112235	0	0	1	0	0	0	0	0	0	0
	0722112236	0	0	0	1	0	0	0	0	0	0
	0722112237	0	0	0	0	1	0	0	0	0	0
Course.Course_name	Introduction	0	0	0	0	0	1	0	0	0	0
Course.Course_Lecturer	Mr. Mwaura	0	0	0	0	0	1	0	0	0	0
Registration.Reg_date	01/11/2012	1	0	0	0	0	1	0	0	0	0
	01/11/2012	1	0	0	0	0	0	1	0	0	0
	01/11/2012	1	0	0	0	0	0	0	1	0	0
	01/11/2012	0	0	0	1	0	0	0	1	0	0
	01/11/2012	0	0	0	1	0	0	0	0	1	0
	01/12/2012	0	0	0	1	0	0	0	0	0	1
	01/12/2012	0	0	0	0	1	1	0	0	0	0
	01/12/2012	0	0	0	0	1	0	0	0	0	1
	01/12/2012	0	1	0	0	0	1	0	0	0	0
	01/12/2012	0	1	0	0	0	0	1	0	0	0
Registration.Reg_Semister	Sem 1	1	0	0	0	0	1	0	0	0	0
	Sem 2	1	0	0	0	0	0	1	0	0	0
	Sem 3	1	0	0	0	0	0	0	1	0	0
	Sem 2	0	0	0	1	0	0	0	1	0	0
	Sem 1	0	0	0	1	0	0	0	0	1	0
	Sem 3	0	0	0	1	0	0	0	0	0	1
	Sem 2	0	0	0	0	1	1	0	0	0	0
	Sem 2	0	0	0	0	1	0	0	0	0	1
	Sem 1	0	1	0	0	0	1	0	0	0	0
	Sem 3	0	1	0	0	0	0	1	0	0	0
		4	**4**			**4**	**8**				

Maximum Clique = 20

TABLE 5 – MAXIMUM CLIQUE SOLUTION

A. THE CLIQUE TRIMMED

TableName.nonKeyAttrib[i]		TableName.keyAttrib[i]			
		Student.Student_No			Course.Course_Code
		P56/100/05	P56/101/05	P56/104/05	C001
Student.Student_name	Shadrack	1	0	0	0
	John	0	1	0	0
	Kamau	0	0	1	0
Student.Student_Tel	0722112233	1	0	0	0
	0722112234	0	1	0	0
	0722112237	0	0	1	0
Course.Course_name	Introduction	0	0	0	1
Course.Course_Lecturer	Mr. Mwaura	0	0	0	1
Registration.Reg_date	01/11/2012	1	0	0	1
	01/12/2012	0	0	1	1
	01/12/2012	0	1	0	1
Registration.Reg_Semister	Sem 1	1	0	0	1
	Sem 2	0	0	1	1
	Sem 1	0	1	0	1

B. THE CLIQUE TRANSLATED

COLUM	ROW
P56/100/05	Shadrack
P56/101/05	John
P56/104/05	Kamau
P56/100/05	0722112233
P56/101/05	0722112234
P56/104/05	0722112237
C001	Introduction
C001	Mr. Mwaura
P56/100/05	01/11/2012
C001	01/11/2012
P56/104/05	01/12/2012
P56/101/05	01/12/2012
P56/100/05	Sem 1
C001	Sem 1
P56/104/05	Sem 2
C001	Sem 2
P56/101/05	Sem 1
C001	Sem 1

C. THE CLIQUE QUERY RESULTS

Student.Student_No	Student.Student_name	Student.Student_Tel	Registration.Reg_date	Registration.Reg_Semister	Course.Course_Name	Course.Course_Lecturer
P56/1000/2005	Shadrack	0722112233	01/11/2012	Sem 1	Introduction	Mr. Mwaura
P56/1001/2005	John	0722112234	01/12/2012	Sem 2	Introduction	Mr. Mwaura
P56/1004/2005	Kamau	0722112237	01/12/2012	Sem 1	Introduction	Mr. Mwaura

TABLE 6 – FINAL RESULTS COMPARISON

A. CONVENTIONAL SQL STATEMENT AND EXPECTED RESULTS

```
Select a.*,b.Student_Name, b.Sudent_Tel,c.Course_Name,c.Course_Lecturer
FROM
Registration a, Student b, Course c
WHERE
a.Reg_Student_No = b.Student_No AND a.Reg_Course_Code = c.Course_Code
AND c.Course_Lecturer = 'Mr. Mwaura'
```

Student_No	Student_name	Student_Tel	Reg_date	Reg_Semister	Course_Name	Course_Lecturer
P56/1000/2005	Shadrack	0722112233	01/11/2012	Sem 1	Introduction	Mr. Mwaura
P56/1001/2005	John	0722112234	01/12/2012	Sem 2	Introduction	Mr. Mwaura
P56/1004/2005	Kamau	0722112237	01/12/2012	Sem 1	Introduction	Mr. Mwaura

B. GRAPH APPROACH BY USE OF MAXIMUM CLIQUE RESULTS

Student.Student_No	Student.Student_name	Student.Student_Tel	Registration.Reg_date	Registration.Reg_Semister	Course.Course_Name	Course.Course_Lecturer
P56/1000/2005	Shadrack	0722112233	01/11/2012	Sem 1	Introduction	Mr. Mwaura
P56/1001/2005	John	0722112234	01/12/2012	Sem 2	Introduction	Mr. Mwaura
P56/1004/2005	Kamau	0722112237	01/12/2012	Sem 1	Introduction	Mr. Mwaura