A Study on Complexity Theory in the Fuzzy Context

UGC MINOR RESEARCH PROJECT REPORT

F. No. MRP(S) - 1240/11-12/KLMG046/UGC-SWRO dated 02- Aug-12

Submitted to
University Grants Commission

By

Dr. Mathews M. George
Principal Investigator

Associate Professor in Mathematics
B.A.M College, Thuruthicad
Pathanamthitta, Kerala-689 597

2014

Table of Contents

Chapter 1

INTRODUCTION

In 1920s logicians began to investigate the concept of computability more seriously and buy a kind of miracle, as Gödel (1946) later expressed it, computability turned out to be a mathematically precise notion. The notion was first formulated by Turing (1936) and Post (1936), who arrived independently at a definition of computing machine, now called a Turing machine. Most of our traditional tools for formal modeling, reasoning, and computing are crisp, deterministic, and precise in character. By crisp we mean dichotomous, that is yes-or-no type rather than more-or-less type. The traditional set theory models the world as black or white and makes no provision for sets of grey. Life is not black and white always. This two valued logic has proved effective and successful in solving well defined problems.

L. A. Zadeh, an electrical engineer from USA, was forced to describe fuzziness mathematically for the first time in 1965 when he was studying the control problems of some complicated systems. The development of fuzzy set theory to fuzzy applications and technology during the first half of the 1990s has been very fast. Fuzzy logic applications in Computer science and information technology are numerous. Fuzzy set theory proposes a mathematical technique for dealing with imprecise concepts and problems that have many solutions. One of the advantages of fuzzy set theory is its extreme generality to accommodate the new developments to cope with existing the emerging problems and challenges. The strength of fuzzy set theory is its algorithmic computer promise.

How much information is sufficient for the computer to identify the scene from its two dimensional mapping and again reconstruct it if required? The type of Mathematics which is relevant in such studies is highly abstract and needs many new concepts based on computational geometry, topology, theory of algorithms and efficient methods of data processing. Fuzzy methods are designed to handle vague terms, and are most suited to finding optimal solutions to problems with vague answers. However, a class of problems exists which are typically complex in nature, and are often left to human beings to deal with. It is known that no polynomial algorithm could be found for the problems in NP or NP-hard problems like travelling salesman Problem, the Hamiltonian Cycle Problem. While a method for computing the solutions to NP-complete problems using a reasonable amount of time remains undiscovered, computer scientists and programmers still frequently encounter NP-complete problems.

Need for the Study

Fuzziness seems to be inherent in human thinking and our preferences are very often vague and so also our choices. The concept of fuzzy sets introduced by L.A. Zadeh in 1965 has been found more suitable for modeling activities involving vagueness. Consequently, it has found widespread applications in many fields of computer science. There are problems like NP- complete problems which are too complex in nature, for traditional approaches to deal with. The purpose of this study is to explore more such situations that occur in the fuzzy context.

Objectives of the Study

The study looks into three specific objectives. One, bring out the need for the study on complexity theory in the fuzzy context. Two, explore more on complexity that occurs in the fuzzy environment. Three, review some studies in NP-completeness. Also it looks into the relevance of computational complexity and NP-completeness, which can complement the earlier study done (1992), whose findings can be generalized in the fuzzy context.

Research methodology

Keeping in mind the above objectives and considering the nature of the topic and theme, this study is essentially the theory of computation in theoretical computer science and mathematics. Also it is descriptive and hence prescriptive in nature examining the complexity in fuzzy context. Content analysis approach was used to analyze different articles for making inferences. The work is based on documents and other information which was collected from journals, articles and the studies conducted by researchers.

This study is meant to be a preliminary attempt to explore more on computational complexity in the fuzzy environment. The first part of the work gives an introduction of the work stating the objectives and methodology. The second part is an overview of complexity and algorithm. The third part throws light on NP-completeness and NP- complete problems. The fourth part deals with fuzzy sets and fuzzy logic; followed by the flavor of results available for fuzzy algorithms in the fifth part. Finally, the conclusion is discussed in the sixth part.

CHAPTER 2

COMPLEXITY AND ALGORITHMS

In this chapter, we present the theory of computability, complexity and polynomial time algorithm. The classes P and NP are stated and their essential features are pointed out.

Complexity theory is generally regarded as an active area of theoretical computer science with results that have implications for the development and use of algorithms. The theory of computer science can be broadly divided into *three* parts:

1. Theory of formal languages
2. Theory of automata and
3. Theory of computability and computational complexity.

As we know a language is a set of strings from some alphabet (finite or infinite) and a grammar is a method of specifying a language. The study of grammars constitutes an important subarea of computer science called *formal languages*.

Automata theory is the study of abstract 'mathematical' machines or systems called automata and the computational problems that can be solved using these machines. *Automata theory* is also closely related to formal language theory, as the automata are often classified by the class of formal languages they are able to recognize.

Now we discuss here the third part in details: *Theory of computability* and *computational complexity*. Let us begin the discussion by recalling algorithms.

The origin of the word ‘algorithm’ dates back to Euclid who obtained a step-by-step procedure to find the gcd of two integers. During the time of Leibnitz (1646 – 1716) and perhaps earlier, there were attempts to provide algorithmic proofs for problems involving numerical computation. David Hilbert’s (1862-1943) aim was to device a formal mathematical system in which all problems can be precisely formulated in terms of statements which are either true or false. If Hilbert’s aim was fulfilled then any problem which was well defined could be solved by executing the algorithm. In 1931 Kurt Gödel discovered that no algorithm of the type desired by Hilbert exists. Gödel’s work is the famous *incompleteness theorem.*

Gödel’s incompleteness theorem states that there is no algorithm whose input can be any statement involving integers and whose output tells whether or not the statement is true. This was endorsed by A. Church, S. Kleene, E. Post and many

others of his time. Gödel defined an algorithm as a sequence of rules for forming complicated mathematical functions out of simpler mathematical functions. Church used a formalism called the lambda calculus, while Turing used a hypothetical machine called a *Turing machine*. The Church Turing thesis states that we can give a reasonably good definition for the word algorithm. In modern terminology, we can define an algorithm as a set of instructions which can be executed on a computer.

2.1 The Halting Problem

There are many problems which are non-computable. That is, there is no algorithm for solving such a problem which is a very strong statement. Are there problems of more direct to computer science which is not computable? The answer is no. The following describe a simple problem of that type, no matter how much time is provided. The problem to determine whether an arbitrary program halts or not, is called a *halting problem*. Its solution is an algorithm which, given an arbitrary program P and its input data D, can tell us whether or not P would eventually halt when executed with input data D. The halting problem is non-computable. There is no algorithm which can solve the halting problem.

Consider for example an algorithm for testing the truth of *Fermat's Last Theorem* which states that there are no positive integers x, y, and z such that $x^n + y^n = z^n$ when $n > 2$. An algorithm for testing the theorem is

Module Fermat (n)
Tests Fermat's last theorem on input n
repeat for x = 1, 2, 3… (For ever)
repeat for y= 1, 2, 3 …x
repeat for z = 2, 3, 4… x +y
If $x^n + y^n = z^n$
then output x, y, z and n and halt.

The algorithm exhaustively tests all positive integers x, y, and z and halts only if it finds x, y, and z such that $x^n + y^n = z^n$. If algorithm halts for some particular input $n > 2$ then Fermat's last theorem is disproved.

The algorithm will halt when the input n = 1, since when x = 1, y = 1 and z = 2, we have $x^1 + y^1 = z^1$. Similarly when the input n = 2, algorithm will halt, since $x^2 + y^2 = z^2$ when x = 3, y = 4 and z = 5. When the input n = 3, it is not easy to see whether

the algorithm will halt or not. What happens when n is very large? At present nobody knows.
The above example illustrates that in some cases the halting problem is easy to solve and in other cases it is very difficult to solve.

2.2 Computability

The theory of computing is one of the important areas in applied mathematics. The theory of computation (TOC) is the study of the inherent capabilities and limitations of computers: not just the computers of today, but any computers that could ever be built. By its nature, the subject is close to mathematics, with progress made by conjectures, theorems, and proofs. What sets TOC apart, however, is its goal of understanding computation -- not only as a tool but as a fundamental phenomenon in its own right.

TOC can be considered as the creation of models of all kinds in the field of computer science. Therefore mathematics and logic are used. In the last century it became an independent academic discipline and was separated from mathematics. Computability theory deals primarily with the question of the extent to which a problem is solvable on a computer. The statement that the halting problem cannot be solved by a Turing machine is one of the most important results in computability theory, as it is an example of a concrete problem that is both easy to formulate and impossible to solve using a Turing machine. Much of computability theory builds on the halting problem result.

TOC is closely related to the branch of mathematical logic called recursion theory, which removes the restriction of studying only models of computation which are reducible to the Turing model. Many mathematicians and computational theorists who study recursion theory will refer to it as computability theory. Some pioneers of the theory of computation were Alonzo Church, Alan Turing, Stephen Kleene, John von Neumann and Claude Shannon *et al*.

The world of computation can be subdivided into three classes:

- Polynomial problems (P)
- Exponential problems (E)
- Intractable (non-computable) problems (I)

Complexity theory is a branch of the theory of computation in theoretical computer science and mathematics that focuses on classifying computational problems according to their inherent difficulty, and relating those classes to each other. In this context, a computational problem is understood to be a task that is in principle amenable to being solved by a computer, which basically means that the problem can be stated by a set of mathematical instructions. Informally, a computational problem consists of problem instances and solutions to these problem instances.

For example:
Instance: A positive integer n.
Question: Is n prime?

It is the computer science field called complexity theory which asks and attempts to answer questions about the amount of use of computational resources *time, memory and hardware*.

Time is simply the period from start to finish of the execution of an algorithm

Memory is the amount of storage required by the algorithm

Hardware is the amount of physical mechanism required for the execution of the algorithm.

The complexity of a problem is just the complexity of the best algorithm which solves that problem. We consider *time* and *memory* as resources and the complexity class is formed by placing a bound (a function of the input value) on the amount of a particular resource that an algorithm may use. That is, a function is in a given class if there is an algorithm for computing the function, in which the amount of resource does not exceed the bound.

Complexity theory considers not only whether a problem can be solved at all on a computer, but also how 'efficiently' the problem can be solved. Two major aspects are considered: *time complexity* and *space complexity*, which are respectively how many steps does it take to perform a computation, and how much memory is required to perform that computation.

In order to analyze how much time and space a given algorithm requires, Computer scientists express the time or space required to solve the problem as a function of the size of the input problem. For example, finding a particular number in a long list of numbers becomes harder as the list of numbers grows larger. If we say there are *n* numbers in the list, then if the list is not sorted or indexed in any way we

may have to look at every number in order to find the number we're seeking. We thus say that in order to solve this problem, the computer needs to perform a number of steps that grows linearly in the size of the problem. *Complexity theory* studies various computer resources. The study of computability leads to an understanding of which problems admit algorithmic solution and which do not.

The amount of any resource used by an algorithm may vary with size of the input data. We are interested only in finding the most '*efficient*' algorithm for solving a problem. The time requirements of an algorithm are conveniently expressed in terms of a single variable, the size of a problem. That is, the amount of input data needed to describe the instance.

To simplify this problem, computer scientists have adopted Big O notation, which allows functions to be compared in a way that ensures that particular aspects of a machine's construction do not be considered, but rather only the asymptotic behavior as problems become large. So in our previous example we might say that the problem requires *O* (*n*) steps to solve.

2.3 Big O notation

There are different types of mathematical notations which are very useful for this type of analysis. One of these is O-notation.

Let f and g be functions $\mathbf{N} \rightarrow \mathbf{R}$. We say that f (n) is O (g(n)) if there is a positive constant c such that
$|f(n)| \leq c|g(n)|$ for all n in **N** with possibly some exceptions.

Example: Let $f(n) = 15n^3$ and $g(n) = n^3$. When $n_0 = 1$, and c = 15, we see that f = O (g). It is also true that g = O (f)

Suppose we are determining the computing time, f (n), of some algorithm. The variable n might be the number of inputs and outputs, their sum or the magnitude of one of them.

The *efficiency* of a given algorithm is the number f(n) of operations required by the algorithm for an instance of size n.

Suppose we have calculated that the number of operations involved in a certain algorithm is $5n^3 + 20n^2 + 3n + 13$. Since $n \leq n^3$ and $n^2 \leq n^3$ we have the inequality $5n^3 + 20n^2 + 3n + 13 \leq (5 + 20 + 3 + 13)\ n^3$ where the right hand side is a constant multiple of n^3. Thus the efficiency of the algorithm is O (n^3) in this case.

As a generalization we have the following theorem.

Theorem 2.1

If $P(n) = a_m n^m + \ldots + a_1 n + a_0$ is a polynomial of degree m then $P(n) = O(n^m)$

The theorem says that if we can describe the frequency of execution of a statement in an algorithm by a polynomial such as P (n), then that statement's computing time is $O(n^m)$.

The theorem says that if we can describe the frequency of execution of a statement in an algorithm by a polynomial such as P (n), then that statement's computing time is $O(n^m)$. The amount of resource used as a function of n, such as, n, $5n^2 + 9n$ or $5n \log n + n + 13$. As n grows larger, some term in the function may dominate the other terms. For example, if the execution time is $5n^2 + 9n$, then as n grows larger $5n^2$ grows very much larger than 9n, and therefore 7n becomes increasingly less significant. The dominating term is $5n^2$ is called *asymptotic* behavior of the algorithm.

Algorithm, whose asymptotic behavior is 2^n or more generally c^n for some constant c, is called *exponential algorithm*. The algorithm whose asymptotic behavior is n, n^2 or more generally n^k for some constant k, are called *polynomial algorithm*.

Examples:

- $n^2 + 13n + 7$ is $O(n^2)$
- $2^n + 2n^5 + 13n^4$ is $O(2^n)$

2.4 Polynomial time Algorithm

In producing a complete analysis of the computing time of the algorithm we have two phases:

(i) A *priori* analysis

(ii) A *posteriori* analysis

In a *priori* analysis we obtain a function which bounds the algorithm's computing time. In a *posteriori* analysis we collect actual statistics about the algorithm's consumption time and space while it is executing.

The amount of resource used as a function of n and as n grows larger; some term in the function may dominate the other terms. The dominating term is called *asymptotic* behavior of the algorithm. Algorithm whose asymptotic behavior is 2^n or more generally c^n for some constant c, are called *exponential* algorithms. The algorithm whose asymptotic behavior is n, n^2 or more generally n^k for some constant k, are called *polynomial* algorithm.

Definition 2.1 The *time complexity function* of an algorithm expresses its time requirements by giving, for each possible input length, the largest amount of time needed by the algorithm to solve a problem. An algorithm is said to have time complexity O(f(n)) if the number of steps needed to process data of size n is at most c f(n), where f(n)is some function of n and c is a constant.

Definition 2.2 A *polynomial time algorithm* is defined to be one whose time complexity function is O (p (n)) for some polynomial function p, where n is the input length. An algorithm whose time complexity function cannot be so bounded is called an exponential *time algorithm*. Polynomial time algorithm is generally regarded as being much more desirable than exponential time algorithm. That is, a problem has not been 'well solved' until a polynomial time algorithm is known for it. A problem is so hard that no polynomial time algorithm can possibly solve it.

Definition 2.3 Any problem for which the answer is either zero or one is called a *decision problem*. An algorithm for a decision problem is termed a *decision algorithm*. Any problem that involves the identification of an optimal solution (either maximum or minimum) value of a given cost function is known as an *optimization problem*. An *optimization algorithm* is used to solve an optimization problem.

Definition 2.4 A decision problem is in P if there is a known polynomial-time algorithm to get that answer. A decision problem is in NP if there is a known polynomial-time algorithm for a non-deterministic machine to get the answer.

Result 2.1 Problems known to be in P are trivially in NP - the nondeterministic machine just never troubles itself to fork another process, and acts just like a deterministic one. There are problems that are known to be neither in P nor NP; a simple example is to enumerate all the bit vectors of length n because that takes 2^n steps.

CHAPTER 3

NP-COMPLETENESS

The concept of NP-completeness was introduced in 1971 by Stephen Cook in a paper entitled the complexity of theorem-proving procedures on pages 151–158 of the Proceedings of the 3rd Annual ACM Symposium on Theory of Computing. Since Cook's original results, thousands of other problems have been shown to be NP-complete by reductions from other problems previously shown to be NP-complete In this section we discuss the Theory of NP-Hard and NP-Complete problems.

3.1 Nondeterministic Algorithms

Algorithms such that the result of every operation is uniquely defined are called deterministic algorithms. A machine capable of executing a deterministic algorithm is called a deterministic machine.

Algorithms such that the result of every operation is not uniquely defined are called non deterministic algorithms. A machine capable of executing a non-deterministic algorithm is called a non-deterministic machine.

Non-deterministic fuzzy machine, as defined, do not exist in practice.

Definition 3.2

A decision problem L is *NP-complete* if:

- L $\in$ NP, and
- Every problem in NP is reducible to L in polynomial time.

L can be shown to be in NP by demonstrating that a candidate solution to L can be verified in polynomial time.

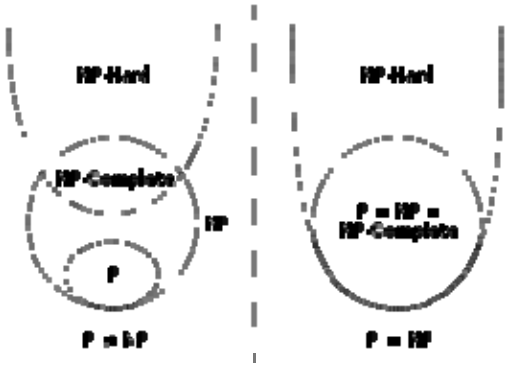
Note that a problem satisfying condition 2 is said to be NP-hard, whether or not it satisfies condition 1.

A problem is NP-Complete if it has the property that it can be solved in polynomial time if and only if all other NP-Complete problems can also be solved in polynomial time. If an NP-Hard problem can be solved in polynomial time, then all NP-Complete problems can be solved in polynomial time. All NP-Complete problems are NP-Hard, but not all NP-Hard problems are NP-Complete.

3.3 Euler Diagram

NP-complete is a subset of NP, the set of all decision problems whose solutions can be verified in polynomial time; *NP* may be equivalently defined as the set of decision problems that can be solved in polynomial time on a nondeterministic Turing machine. A problem *p* in NP is also in NP complete if and only if every other problem in NP can be transformed into *p* in polynomial time. NP-complete can also be used as an adjective: problems in the class NP-complete are known as NP-complete problems.

Euler diagram for P, NP, NP-complete, and NP-hard set of problems



3.4 NP-complete problems

A decision problem *L* is *NP-complete* if it is in the set of NP problems so that any given solution to the decision problem can be verified in polynomial time, and also in the set of NP-hard problems so that any NP problem can be converted into *L* by a transformation of the inputs in polynomial time.

Although any given solution to such a problem can be verified quickly, there is no known efficient way to locate a solution in the first place; indeed, the most notable characteristic of NP-complete problems is that no fast solution to them is known. That is, the time required to solve the problem using any currently known algorithm increases very quickly as the size of the problem grows. As a consequence, determining whether or not it is possible to solve these problems quickly, called the P versus NP problem, is one of the principal unsolved problems in computer science today.

NP-complete problems are studied because the ability to quickly verify solutions to a problem (NP) seems to correlate with the ability to quickly solve that problem (P). It is not known whether every problem in NP can be quickly solved - this is called the P = NP problem. But if any single problem in NP-complete can be solved quickly, then every problem in NP can also be quickly solved, because the definition of an NP-complete problem states that every problem in NP must be quickly reducible to every problem in NP-complete (that is, it can be reduced in polynomial time). Because of this, it is often said that the NP-complete problems are harder or more difficult than NP problems in general.

To prove that an NP problem L is in fact an NP-complete problem it is sufficient to show that an already known NP-complete problem reduces to L.A consequence of this definition is that if we had a polynomial time algorithm for L, we could solve all problems in NP in polynomial time. Some NP-complete problems, indicating the reductions typically used to prove their NP-completeness

The easiest way to prove that some new problem is NP-complete is first to prove that it is in NP, and then to reduce some known NP-complete problem to it. Therefore, it is useful to know a variety of NP-complete problems.

In a computer science conference, there was a fierce debate among the computer scientists about whether NP-complete problems could be solved in polynomial time on a deterministic Turing machine. John Hop croft brought everyone at the conference to a consensus that the question of whether NP-complete problems are solvable in polynomial time should be put off to be solved at some later date, since nobody had any formal proofs for their claims one way or the other. This is known as the question of whether P = NP.

Nobody has yet been able to determine conclusively whether NP-complete problems are in fact solvable in polynomial time, making this one of the great unsolved problems of mathematics. The Clay Mathematics Institute is offering a US $1 million reward to anyone who has a formal proof that P = NP or that P ≠ NP. It remains as an unsolved problem.

3.5 The graph isomorphism problem

It is the graph theory problem of determining whether a graph isomorphism exists between two graphs. Two graphs are isomorphic if one can be transformed into the other simply by renaming vertices. Consider these two problems:

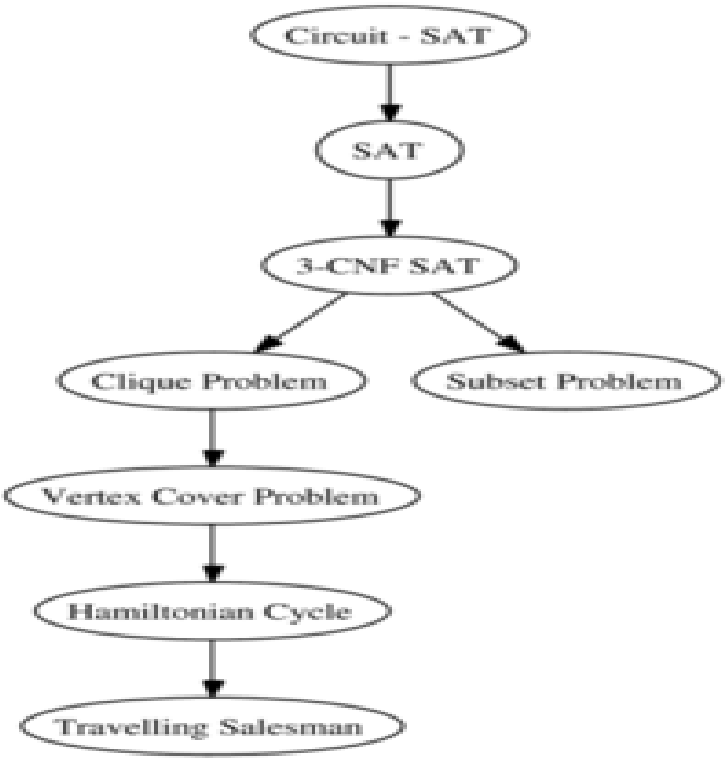
- Graph Isomorphism: Is graph G_1 isomorphic to graph G_2?
- Sub graph Isomorphism: Is graph G_1 isomorphic to a sub graph of graph G_2?

The Sub graph Isomorphism problem is NP-complete. The graph isomorphism problem is suspected to be neither in P nor NP-complete, though it is in NP. This is an example of a problem that is thought to be hard, but isn't thought to be NP-complete.

3.5 Solving NP-complete problems

NP-complete problems are often addressed by using algorithms. At present, all known algorithms for NP-complete problems require time that is super polynomial in the input size, and it is unknown whether there are any faster algorithms. The following techniques can be applied to solve computational problems in general, and they often give rise to substantially faster algorithms:

The easiest way to prove that some new problem is NP-complete is first to prove that it is in NP, and then to reduce some known NP-complete problem to it. Therefore, it is useful to know a variety of NP-complete problems. The diagram below contains some well-known problems that are NP-complete when expressed as decision problems.



To the right is a diagram of some of the problems and the reductions typically used to prove their NP-completeness? In this diagram, an arrow from one problem to another indicates the direction of the reduction. Note that this diagram is misleading

as a description of the mathematical relationship between these problems, as there exists a polynomial-time reduction between any two NP-complete problems; but it indicates where demonstrating this polynomial-time reduction has been easiest.

3.7 Completeness under different types of reduction

In the definition of NP-complete given above, the term *reduction* was used in the technical meaning of a polynomial-time many-one reduction.

Another type of reduction is polynomial-time Turing reduction. A problem p is polynomial-time Turing-reducible to a problem q if, given a subroutine that solves q in polynomial time, one could write a program that calls this subroutine and solves p in polynomial time. This contrasts with many-one reducibility, which has the restriction that the program can only call the subroutine once, and the return value of the subroutine must be the return value of the program.

Another type of reduction that is also often used to define NP-completeness is the logarithmic-space many-one reduction which is a many-one reduction that can be computed with only a logarithmic amount of space. Since every computation that can be done in logarithmic space can also be done in polynomial time it follows that if there is a logarithmic-space many-one reduction then there is also a polynomial-time many-one reduction.

CHAPTER 4

FUZZY SETS AND FUZZY LOGIC

L.A. Zadeh advanced the fuzzy theory in 1965. The theory proposes a mathematical technique for dealing with imprecise concepts and problems that have many possible solutions. The traditional set theory models our life as either black or white and thus makes no provision for a third option, grey. This age old binary logic has proved effective and successful in solving well defined problems. Naturally, many patterns of thinking familiar from classical logic are not applicable in the many-valued cases.

Consider the statements involving concepts, such as *low, medium, high* may not be a matter of true or false, but rather are graded, where the grade may be taken from a specified range and indicates the membership degree to which elements of the universe belong to the set. This idea can be formalized by generalizing characteristic functions in such a way that the values assigned to the elements of the universe fall within a specified range, which is not necessarily {0, 1}. Such a function is called membership function, the set so defined is called a fuzzy set, concepts such as *low, medium, high* are called fuzzy concepts, and statements involving fuzzy concepts are called fuzzy statements.

In its half century of existence since the seminal paper of L.A. Zadeh, fuzzy logic has absolved the stages of wide theoretical research, hardware-supported fuzzy systems and industrial applications. Fuzzy logic provides an effective means of dealing with approximate and inexact nature of the real world. As in the case of crisp computability, alternative to fuzzy algorithms, fuzzy Turing machines may be considered.

Let X be a non - empty set and I = [0, 1]

Definition 4.1 A fuzzy sub set A of X is defined to be a mapping $A : X \to [0,1]$ (or $\mu_A : X \to [0,1]$). The class A is characterized by the function and has the following representation,

$A = \{(x, \mu_A(x)) : x \in X\}$.

An ordinary set $A \subseteq X$ is identified with its characteristic function $\chi_A : X \to \{0, 1\}$.

Example 4.2 Let X = {a, b, c, d}. Define $\mu : X \to [0, 1]$ by $\mu(a) = 0.1$, $\mu(b) = 1$, $\mu(c) = 0.4$, $\mu(d) = 0.6$. Then the class A = {(a, 0.1), (b, 1), (c, 0.4), (d, 0.6} is a fuzzy subset of X.

The fuzzy sets $\underline{0}$ and $\underline{1}$ are given by $\underline{0}(x) = 0$ and $\underline{1}(x) = 1$ for all $x \in X$.

Let I(X) be the family of all the fuzzy sets in X called fuzzy space and the power set P(X) be the class of fuzzy sets whose membership functions have all their values in {0,1}. X is called the carrier domain of each fuzzy subset in it, and I is called the value domain of each fuzzy subset of X.

Definition 4.3 Let μ and σ be any two fuzzy sets in a set X. Then μ is said to be contained in σ, denoted by $\mu \leq \sigma$, if $\mu(x) \leq \sigma(x)$ for all $x \in X$.

If $\mu(x) = \sigma(x)$ for all $x \in X$, μ and σ are said to be equal and we write $\mu = \sigma$.

Definition 4.4 A fuzzy set in X is called a fuzzy point if and only if it takes the value 0 for all $y \in X$ except one, say, $x \in X$. If its value at x is λ for $\lambda \in (0, 1]$, we denote this point by x_λ where the point x is called its support.

Definition 4.5 The fuzzy point x_λ is said to be contained in a fuzzy setμ, denoted by $x_\lambda \in \mu$ if and only if $\lambda \leq \mu(x)$.

In I(X) the following additive operations can be introduced:

Definition 4.6

i. The sum of two fuzzy sets μ and σ in a set X ,denoted by $\mu \oplus \sigma$,is a fuzzy set in X defined by $(\mu \oplus \sigma)(x) = \min(1, \mu(x) + \sigma(x))$ for all $x \in X$
ii. The difference of two fuzzy sets μ and σ in X ,denoted by $\mu \Theta \sigma$ is a fuzzy set in X defined by $(\mu \Theta \sigma)(x) = \max(0, \mu(x) - \sigma(x))$ for all $x \in X$
iii. The conjunction of two fuzzy sets μ and σ ,denoted by $\mu \& \sigma$, is a fuzzy set in X defined by $(\mu \& \sigma)(x) = \max(0, \mu(x) + \sigma(x) - 1)$ for all $x \in X$
iv. The product $\mu . \sigma$ is a fuzzy set defined by
$(\mu.\sigma)(x) = \mu(x).\sigma(x)$ for all $x \in X$.

In his classical paper, Zadeh have introduced the union, the intersection, the complementary and the inclusion as follows:

- The union $\mu \cup \sigma$ is the fuzzy set in X defined by $(\mu \cup \sigma)(x) = \max(\mu(x), \sigma(x))$ for all $x \in X$

- The intersection $\mu \cap \sigma$ is the fuzzy set defined by $(\mu \cap \sigma)(x) = \min(\mu(x), \sigma(x))$ for all $x \in X$ The complement of a fuzzy set μ is in a set X, denoted by μ^c is the fuzzy set of X defined by $\mu^c(x) = 1 - \mu(x)$ for all $x \in X$.
- The inclusion of fuzzy sets is given by $\mu \subseteq \sigma$ iff $\mu(x) \leq \sigma(x)$ for all $x \in X$.

4.7 *Fuzzy Logic*

The theory of fuzzy logic is based on the notion of relative graded membership, as inspired by the processes of human perception and cognition. Fuzzy logic provides an effective means of dealing with the approximate and inexact nature of the real world. Fuzzy logic can deal with information arising from computational perception and cognition, that is, uncertain, imprecise, vague, partially true, or without sharp boundaries.

Fuzzy logic allows for the inclusion of vague human assessments in computing problems. Also, it provides an effective means for conflict resolution of multiple criteria and better assessment of options. Fuzzy logic allows for the inclusion of vague human assessments in computing problems. Also, it provides an effective means for conflict resolution of multiple criteria and better assessment of options. New computing methods based on fuzzy logic can be used in the development of intelligent systems for decision making, identification, pattern recognition, optimization, and control.

The most tangible applications of fuzzy logic control have already appeared in many commercial appliances. New computing methods based on fuzzy logic can be used in the development of intelligent systems for decision making, identification, pattern recognition, optimization, and control. The most tangible applications of fuzzy logic control have already appeared in many commercial appliances. Fuzzy logic applications in Computer science and information technology are numerous.

Chapter 5

FUZZY ALGORITHMS

In this chapter we introduce the two classes fuzzy P and fuzzy NP.

The concept of fuzzy set introduced by Zadeh motivated a lot of mathematical activity on the generalization of the notion of a fuzzy set. A fuzzification part transforms crisp data into fuzzy sets. The concept of fuzzy algorithm may be viewed as a generalization, through the process of fuzzification, of the conventional (non fuzzy) concept of an algorithm. A fuzzy algorithm may contain fuzzy statements containing names of fuzzy sets. Just as the notion of a non fuzzy algorithm defined precisely in the context of countable sets by placing it in one- one correspondence with a Turing machine, so can the notion of a fuzzy algorithm be given a precise, although restricted, meaning by placing it in one-one correspondence with a Turing machine.

Different formulations of fuzzy algorithms are developed. Turing algorithm and Markov normal algorithm are the most important among them. It is shown that the two formulations are equivalent in a certain sense.

Fuzzy algorithm may contain fuzzy instructions such as:

(a) "Set y approximately equal to 10 if x is approximately equal to 5"
(b) "If x is large, increase y by several units"
(c) "If x is large, increase y by several units; if x is small, decrease y by several units; otherwise keep y unchanged"

The sources of fuzziness in these instructions are fuzzy sets which are identified by their underlined names.

Fuzzy instruction which is a part of a fuzzy algorithm can be assigned a precise meaning by making use of the concept of the membership function of a fuzzy set. An algorithm is a *fuzzy algorithm* when its variables range over fuzzy sets, regardless of whether they are finite sets or continua.

Familiar examples of fuzzy algorithms drawn from everyday experience are cooking recipes, directions for repairing a TV, instructions on how to treat a disease, instructions for parking a car, etc. generally, such instructions are not dignified with the name 'algorithm'. From our point of view, however, they may be regarded as very crude forms of fuzzy algorithms.

A fuzzy instruction which is a part of a fuzzy algorithm can be assigned a precise meaning by making use of the concept of the membership function of fuzzy set. However, the assignment of a precise meaning to a fuzzy instruction does not in itself resolve the ambiguity of how it should be executed. Consider the simple example of an unconditional instruction 'move *several steps* forward'. Suppose that the membership function of A, the fuzzy set named '*several steps'*, is specified as follows:

$\mu_A(0) = \mu_A(1) = \mu_A(2) = \mu_A(3) = 0$; $\mu_A(4) = 0.8$; $\mu_A(5) = \mu_A(6) = \mu_A(7) = 1$; $\mu_A(8) = 0.7$; $\mu_A(x) = 0$ for $x \geq 9$. What would a human being do given $\mu_A(x)$ and instructed to move several steps forward, assuming that his actions are not influenced by any external factors such as expenditure of energy involving in taking n steps, etc ?

This question led us to consider the following modes of execution.

1. Probabilistic execution
2. Non deterministic execution with threshold

The classes fuzzy P and fuzzy NP

Definition 5.1 Fuzzy algorithms such that the result of every operation is uniquely defined are called deterministic fuzzy algorithms. A machine capable of executing a deterministic fuzzy algorithm is called a deterministic fuzzy machine.

Definition 5.2 Fuzzy algorithms such that the result of every operation is uniquely defined are called deterministic fuzzy algorithms. A machine capable of executing a nondeterministic fuzzy algorithm in this way is called a nondeterministic fuzzy machine.

Definition 5.3 The class *fuzzy P* is defined to be the class of all decision problems that can be solved in polynomial time by a non-deterministic fuzzy algorithm.

Definition 5.4

The class *fuzzy NP* is defined to be the class of all decision problems that can be solved in polynomial time by a non-deterministic fuzzy algorithm.

Fuzzy NP-complete is a subset of fuzzy NP, the set of all decision problems whose solutions can be verified in polynomial time; fuzzy *NP* may be equivalently defined as the set of decision problems that can be solved in polynomial time on a nondeterministic fuzzy Turing machine.

5.4 Fuzzy NP-completeness

A decision problem L is *fuzzy NP-complete* if:

1. L ∈ fuzzy NP, and
2. Every problem in fuzzy NP is reducible to L in polynomial time.

L can be shown to be in NP by showing that L can be verified in polynomial time.

Note that a problem satisfying condition 2 is said to be NP-hard, whether or not it satisfies condition 1.

There is often only a small difference between a problem in fuzzy P and a fuzzy NP-complete problem.

Result 4.5 A decision problem is in P if there is a known polynomial-time algorithm to get that answer. A decision problem is in NP if there is a known polynomial-time algorithm for a non-deterministic machine to get the answer.

Result 4.6 Problems known to be in P are trivially in NP — the nondeterministic machine just never troubles itself to fork another process, and acts just like a deterministic one. There are problems that are known to be neither in fuzzy P nor in fuzzy NP.

Result 4.7 If an NP-Hard problem can be solved in polynomial time, then all NP-Complete problems can be solved in polynomial time. All NP-Complete problems are NP-Hard, but not all NP-Hard problems are NP-Complete in fuzzy context.

There is often only a small difference between a problem in fuzzy P and a fuzzy NP-complete problem. The statement P ≠ NP is still an unsolved question.

CHAPTER 6

CONCLUSION

The introduction of the notions of polynomial time and NP properties signaled the birth of modern complexity theory. Notions and paradigms from this property have generated a large part of mathematics and its applications. The ideas of complexity theory can be applied in one of the most important areas of theoretical computer science, cryptography. There are problems like NP- Complete problems which are too complex in nature, for traditional approaches to deal with. This area has ample space for further development. Nobody has yet been able to determine conclusively whether NP-complete problems are in fact solvable in polynomial time, making this one of the great unsolved problems of mathematics.

Zadeh introduced the concept of a fuzzy set which motivated a lot of mathematical activity on the generalization of the notion of fuzzy set. During the last three decades, beginning in about 1985, fuzzy set theory began finding its way into a large number of applications. Basic for the study of fuzzy computability is the concept of fuzzy algorithm that was introduced by Zadeh in 1968, but not further developed. The type of Mathematics which is relevant in such studies is highly abstract and needs many new concepts based on computational geometry, topology, theory of algorithms and efficient methods of data processing.

Today mathematics and computers go together which play a vital role in modern warfare. It is therefore needless to say that many new branches of mathematics such as fuzzy set theory again vital in a modern war-like situation where decisions cannot be taken either way have grown due to their application in defence. Even though fuzzy set theory is not a philosopher's stone to solve all problems that confront us today, it has considerable potential for practical as well as for mathematical applications.

The present study does not claim itself to be the sole and repository of fuzzy complexity theory. It throws light to a section we haven't looked and the results of the study will be useful for future studies. The *findings* of this study are new and have a significant impact on the new trends in fuzzy mathematics.

- Any problem in fuzzy P is also in fuzzy NP since if a problem is in fuzzy P then we can solve it in polynomial time without even being given a certificate. So, fuzzy P $\subseteq$ fuzzy NP.
- NP-Complete problems are in NP, and also are NP-Hard, but not all NP-Hard problems are in NP in the fuzzy context.

Hope that further research efforts will advance in this area and help to close still existing gaps. Future studies on the effect of fuzzy concepts as well as fuzzy set-based techniques will provide much better results. Research in the direction of fuzzy algorithms is urgently needed.

Many complexity classes are defined using the concept of a reduction. A reduction is a transformation of one problem into another problem. The most commonly used reduction is a polynomial-time reduction. This means that the reduction process takes polynomial time.

For example, the problem of squaring an integer can be reduced to the problem of multiplying two integers. This means an algorithm for multiplying two integers can be used to square an integer. Indeed, this can be done by giving the same input to both inputs of the multiplication algorithm. Thus we see that squaring is not more difficult than multiplication, since squaring can be reduced to multiplication.

Limitations of the study and Scope for Future Studies

In this study mainly utilized descriptive research method to address the study objectives. Several limitations of the study provide additional research opportunities. One of the major limitations is that the study is theoretical in nature and is based on the documents and other information from journals and articles. Also this study is mainly focused on complexity, only a few areas were considered for evaluation of the results. There is more scope for future research in the above mentioned areas.

Why are NP-complete problems interesting for future studies**?**

1. Although no polynomial time algorithm for an NP-complete problem has ever been found, nobody has ever proven that an efficient algorithm for one cannot exist. In other words, it is unknown whether or not a polynomial time algorithm exists for NP-complete problems.
2. If a polynomial time algorithm exists for any NP-complete problem then polynomial time algorithms exists for all of them.
3. Several NP- complete problems are similar, but not identical, no problems for which we do know of polynomial time algorithms.

Future research studies can focus on the following:

- Is the class of problems NP-Complete more difficult to solve than the class of problems NP-Hard in fuzzy context? In other words, NP-Complete problems are in NP, and also are NP-Hard, but not all NP-Hard problems are in NP... How to tell which one is more difficult to solve in fuzzy environment?
- How much information is sufficient for the computer to identify the scene from its two dimensional mapping and again reconstruct it if required?

Further research is needed to explore and deepen the understanding of the nature of complexity in fuzzy context. This study opens new avenues for further investigation for this new concept fuzzy NP- completeness.

Acknowledgement

This study is based on the work under minor research project titled '*A Study on Complexity theory in Fuzzy Context*' sponsored by the University Grants Commission. I express my sincere gratitude to UGC for funding the research work. Acknowledgments are due to Prof. Thomas Abraham, manager for his support to complete the project successfully.

REFERENCES

1. Ahamed M. Ibrahim(2004) Fuzzy Logic for Embedded Systems Applications Elsevier Science (USA)
2. D. Butnariu , *Additive Fuzzy Measures and Integrals I,* Journal of Mathematical analysis and applications 93,(1983) 436-452
3. Cook S.A. (1971) "The complexity of theorem proving procedures". *Proceedings, Third Annual ACM Symposium on the Theory of Computing, ACM, New York*. pp. 151–158.
4. Cormen T.H.; Leiserson, C.E., Rivest, R.L.; Stein, C. (2001). *Introduction to Algorithms* (2nd edn.). MIT Press and McGraw-Hill. Chapter 34: NP–Completeness, pp. 966–1021. ISBN 0-262-03293-7.
5. Garey M.R.; Johnson, D.S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman. ISBN 0-7167-1045-5. This book is a classic, developing the theory, and then cataloguing *many* NP-Complete problems.

6. James J. Buckley (2014) An Introduction to Fuzzy Logic and Fuzzy Sets, Springer International edition

7. Mathews M. George (1992), *Stephan Cook's theorem and NP-Completeness,* M. Phil dissertation, Calicut University, Kerala India

8. Mathews M. George (2008), *Fuzzy measures and Riesz representation theorem and related results in the fuzzy context,* Ph.D Thesis, Mahatma Gandhi University, Kerala India.

9. D. Stephen Dinagar, K. Palanivel(2013) *On Quadratic Membership Functions in solving assignment Problem under fuzzy environment*, The Journal of fuzzy Mathematics Vol. 21, N0. 1, (59-68) Los Angeles

10. Umberto Straccia (2014) Foundations of Fuzzy Logic and Semantic Web Languages, CRC Press A Champan & Hall Book

11. Yong Ho Kim,sang Chul Ahn,Wook Hyun Kwon (2000) *Computational complexity of general fuzzy logic control and its simplification for a loop controller,* Fuzzy Sets and Systems , 111 , 215-224, USA

12. L.A. Zadeh (1965) *Fuzzy sets. Information and Control* ,8, 338-353

13. L.A. Zadeh(1968) *Fuzzy Algorithms*, Information and Control,12, 91-102

14. H.Z. Zimmermann(2012) *Fuzzy Set Theory and Applications*, Spinger International, Fourth edition

…….