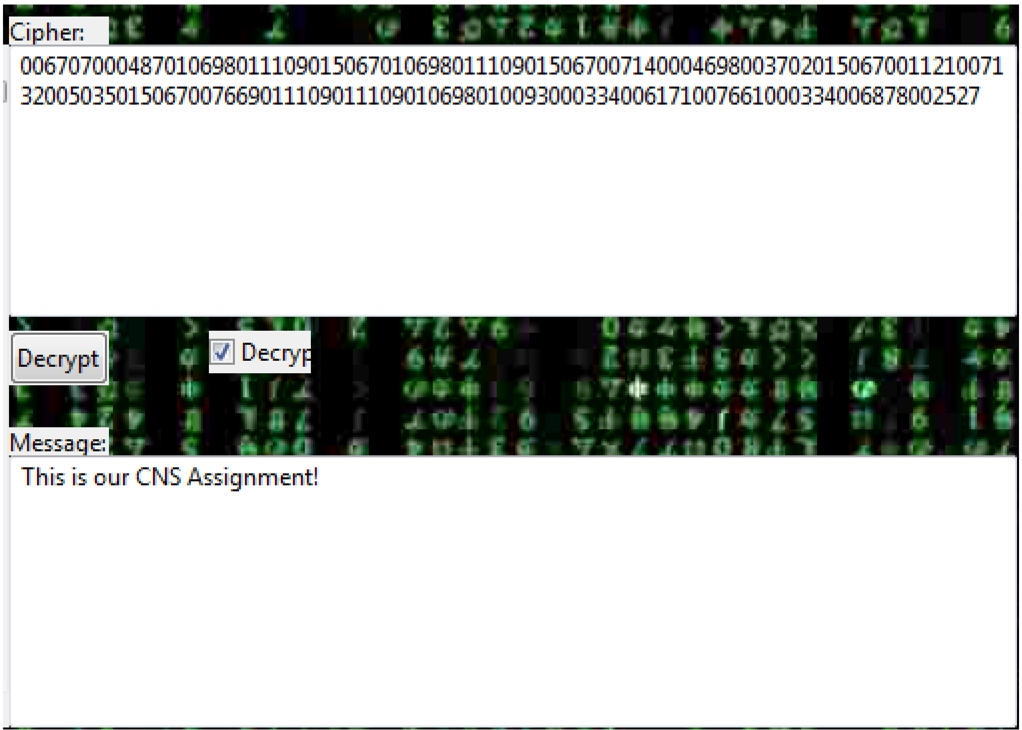
December 1, 2012

RV College of Engineering

Department of Computer Science



Soft Set Theory for Soft Computing

By:
Samir Sheriff [1RV09CS093]
Satvik N [1RV09CS095]

1 Introduction

Molodtsov initiated the concept of soft set as a new mathematical tool for dealing with uncertainties. Most of our traditional tools for formal modeling, reasoning and computing are crisp, deterministic and precise in character. However, there are many complicated problems in economics, engineering, environment, social science, medical science etc. that involve uncertainties. The Theory of Probability, Theory of Fuzzy Sets,Theory of Intuitionistic Fuzzy Sets,Theory of Vague Sets, Theory of Interval Mathematics and Theory of Rough Sets are considered as mathematical tools for dealing with uncertainties.

In 1999, Molodstov pointed out that these theories which are considered as mathematical tools for dealing with uncertainties, have certain limitations. He further pointed out that the reason for these limitations is, possibly, the inadequacy of the parameterization tool of the theory. The Soft Set Theory introduced by Molodstov is quite different from these theories in this context. The absence of any restrictions on the approximate description in Soft Set Theory makes this theory very convenient and easily applicable. Fuzzy set theory proposed by Professor L. A. Zadeh in 1965 is considered as a special case of the soft sets.

Current and soft mathematics can co-exist and can be used consistently to solve real world problems.

2 Definitions

Structured Subsets

According to the authors of this oh so marvelous paper, ideal membership functions for real world fuzzy sets are made from elastic material, because fuzzy sets should be able to tolerate certain amount of perturbation or stretching. Mathematically, each perturbation is a new function, that consists of a set of functions, each of which can be continuously stretched into another function. Such a set of functions is a highly structured subset of membership function space.

Soft Sets

A fuzzy set, which is also called, in different parts of the world, a soft set, is an *abstract structured set* of membership function space. Such a structured subset may be an equivalence class, a neighborhood, or fuzzified structures in the membership function space. Soft sets are intended to capture and to defuse the conflicts among existing fuzzy theories. So a fuzzy set could be defined by

1. A collection of membership functions
2. that should be able to transform among themselves

Neighbourhood Systems

It is an abstraction of near or negligible distances in geometry. A neighborhood system is an association that assigns each datum a list of data that may or may not contain the datum. It is natural and implementable, and used for approximate retrieval in databases and approximate reasoning in knowledge bases

The central notion here is neighborhood systems. It is an abstraction of near or negligible distances in geometry. A neighborhood system is an association that assigns each datum a list of data that may or may not contain the datum. It is natural and implementable. So current and soft mathematics can co-exist and can be used consistently to solve real world problems.

Neighborhood systems express the semantics of nearby spaces. Let p be an object (or datum) in the universe or space X.

- A neighborhood, denoted by N(p), or simply N, of p is a non-empty subset of X, which may or may not contain the object p.
- Any subset that contains a neighborhood is a neighborhood.
- A neighborhood system of object p, denoted by NS(p), is a non-empty maximal family of neighborhoods of p.
- A neighborhood system of X, denoted by NS(X) is the collection of NS (p) for all objects p in X.
- If a neighborhood system NS(X) that satisfies certain axioms, then X is a topological space. In general, X is a Frechet (V) space.
- From this view, rough set theory is a special case of the neighborhood system theory.

Soft sets, defined using Neighbourhood Systems

A real world fuzzy set is defined abstractly by a neighborhood systems of membership function space. Neighborhood systems translate the real world problem into a mathematical problem. So our ultimate goal is to axiomatize fuzzy sets through such neighborhood systems.

Membership Function Space

Let U be the universe of discourse, and let FX : U $\rightarrow$ M be a map, where M is, in general, a membership space.

FX is called a membership function; FX(x) is called the grade or degree of membership of x $\in$ U.

If M is the set of two elements {0, 1}, then FX is the characteristic function of a classical crispy set. If M is a unit interval [0, 1], then FX is the membership function of a classical fuzzy set.

Let be a collection of fuzzy sets on U. Each fuzzy set is defined by a neighborhood (may be a singleton) in the membership function space;

The developments of various generalized set theories form a beginning of a "soft mathematics and may provide a foundation for soft computing. This paper is one of our attempt to provide a solid set theory for soft computing.

How Public-key Cryptosystems Work

The distinguishing technique used in public-key cryptography is the use of asymmetric key algorithms, where the key used to encrypt a message is not the same as the key used to decrypt it. Each user has a pair of cryptographic keys - a public encryption key and a private decryption key. The publicly available encrypting-key is distributed, while the private decrypting-key is kept secret. Messages are encrypted with the recipient's public key, and can be decrypted only with the corresponding private key. The keys are related mathematically, but the parameters are chosen so that determining the private key from the public key is either impossible or prohibitively expensive.

Schmidt-Samoa Public-key Cryptosystem

The Schmidt-Samoa cryptosystem is an asymmetric cryptographic technique, whose security, like Rabin and RSA depends on the difficulty of integer factorization.

- Key generation
  - Choose two large distinct primes p and q and compute $N = p^2 \times q$
  - Compute $d = N - 1$ mod $lcm(p-1, q-1)$
  - Now N is the public key and d is the private key.
- **Encryption** - To encrypt a message m we compute the cipher text as $c = m^N modN$
- **Decryption** To decrypt a cipher text c we compute the plaintext as $m = c^d mod(p \times q)$ which like for Rabin and RSA can be computed with the Chinese remainder theorem.
- **Security** - The algorithm, like Rabin, is based on the difficulty of factoring the modulus N, which is a distinct advantage over RSA. That is, it can be shown that if there exists an algorithm that can decrypt arbitrary messages, then this algorithm can be used to factor N.

3 Previous Cryptosystems

RSA Cryptosystem

RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described it in 1977.

- **Key Generation**
  - Let N = $p \times q$ be a product of two prime numbers
  - Compute $\varphi(n) = (p1)(q1)$, where φ is Euler's totient function.
  - Choose an integer e such that $1 \leq e \leq \varphi(n)$ and greatest common divisor of (e, $\varphi(n)$) = 1; i.e., e and $\varphi(n)$ are co-prime.
  - Determine d as: d = e^{-1} (mod $\varphi(n)$), d is the multiplicative inverse of e mod $\varphi(n)$.
- **Encryption**: Let M be a message, and c the ciphertext. Then, $c = m^e (modn)$
- **Decryption**: $m = c^d (modn)$ By construction, $d^e = 1$ mod $\varphi(n)$. The public key consists of the modulus n and the public (or encryption) exponent e. The private key consists of the modulus n and the private (or decryption) exponent d which must be kept secret.

Rabinś Cryptosystem

In 1979, Michael Rabin suggested a variant of RSA with public-key exponent 2, which he showed to be as secure as factoring.

- **Key Generation**
  - Choose two large distinct primes p and q.
  - Let n=pq. Then n is the public key. The primes p and q are the private key

- **Encryption**: For the encryption, only the public key n is used. The process follows - Let P = {0,...,n-1} be the plaintext space (consisting of numbers) and $m \in P$ be the plaintext. Now the ciphertext is determined by $c = m^2$(mod n).

  c is the quadratic remainder of the square of the plaintext, modulo the key-number n.

- **Decryption**: To decode the ciphertext, the private keys are necessary. The process follows: If c and r are known, the plaintext is then $m \in \{0,...,n-1\}$ with $m^2 = c$ (mod r). For a composite r (that is, like the Rabin algorithm's) there is no efficient method known for the finding of m. If, however r is prime (as are p and q in the Rabin algorithm), the Chinese remainder theorem can be applied to solve for m.

  Thus the square roots $m_p = \sqrt{c}$ mod p and $m_q = \sqrt{c}$ mod q must be calculated

4 Implementation

```
package com.jinkchak;

import java.security.InvalidAlgorithmParameterException;

import org.eclipse.swt.widgets.Display;

public class Schmidt_Samoa_Encryptor {
        private int p, q;
        private int public_key, private_key;

        private static final int BLOCK_SIZE = 6;   //For splitting a String of text into blocks

        /**
         * This constructor initializes the following variables:
         *   p - with a default value of 23
         *   q - with a default value of 31
         * After that, it calls a method that computes the private and public keys
         *
         */
        public Schmidt_Samoa_Encryptor()
        {
                reInitialize(23, 31);
        }

        /**
         * Re-initializes the system with the new values for p and q, and then
         * computes the new values of the public and private keys.
         * @param p A large prime number
         * @param q A large prime number that is distinct from q
         *
         */
        public void reInitialize(int p, int q)
        {
                this.p = p;
                this.q = q;
                public_key = computeN();
                try {
                        private_key = modular_Equation_Solver(public_key, 1, lcm(p-1, q-1));
```

```
        } catch (InvalidAlgorithmParameterException e) {
            e.printStackTrace();
        }

    }

    /**
     * Computes the lcm of two integers
     * @param a An integer
     * @param b An integer
     * @return LCM of a and b
     */
    private int lcm(int a, int b)
    {
        return (a*b)/gcd(a,b);
    }

    /**
     * Computes the GCD of two integers
     * @param a An integer
     * @param b An integer
     * @return GCD of a and b
     */
    private int gcd(int a, int b) {
        if (b==0)
            return a;
        return gcd(b,a%b);
    }

    /**This method contains an implementation of the extended Euclidean algorithm.
     * The extended Euclidean algorithm is an extension to the Euclidean algorithm.
     * Besides finding the greatest common divisor of two integers, as the Euclidean algorithm does,
     * it also finds integers x and y (one of which is typically negative) that satisfy Bzout's identity:
     *    ax + by = gcd(a, b)
     * @param a An integer
     * @param b An integer
     * @return An integer array z consisting of three element:
     *    z[0] = gcd(a, b)
     *    z[1] = x
     *    z[2] = y
     */
    private int[] extendedEuclidsAlgo(int a, int b)
    {
        int []result = new int[3];
        if(b==0)
        {
            result[0] = a;// index 0 is x
            result[1] = 1;// index 1 is y
            result[2] = 0;// index 2 is d ... ax+by = d
            //System.out.println(result[0]+" "+result[1]+" "+result[2]);
            return result;
        }

        int []result_temp = extendedEuclidsAlgo(b, a%b);
```

```java
        int []final_result = {result_temp[0],result_temp[2],result_temp[1]-(a/b)*result_temp[2]};
        //System.out.println(final_result[0]+" "+final_result[1]+" "+final_result[2]);
        return final_result;
    }

    /**
     * This method implements the modular exponentiation algorithm as defined in the CLRS text book.
     * It finds out the result of (a^b) mod n, even when b is very very large
     * @param a An integer that has to be raised to the power b
     * @param b An integer that denotes the power to which a has to be raised.
     * @param n An integer based on which all multiplication operations are performed (mod n)
     * @return An integer containing the result of ((a ^ b) mod n)
     */
    public int modularExponentiator(int a, int b, int n)
    {
        int c = 0;
        int d = 1;
        String binaryB = Integer.toBinaryString(b);

        for(int i = 0; i < binaryB.length(); i++)
        {
            c = 2 * c;
            d = (d * d) % n;
            if(binaryB.charAt(i) == '1')
            {
                c++;
                d = (d * a) % n;
            }
        }

        return d;
    }

    /**
     * Encrypts a message using the Schmidt-Samoa Algorithm. The message is split into blocks of size
     * BLOCK_SIZE and each block is encrypted to form a cipher string. If a given block is less than the
     * BLOCK_SIZE, then the toNLengthString() method is called to
     * convert the block to a string of size BLOCK_SIZE.
     * @param message A string of plaintext.
     * @return A string containing the cipher text
     */
    public String encrypt(String message)
    {
        int []cipher = new int[message.length()];
        String cipherString = "";
        for(int i = 0; i < message.length(); i++)
        {
            cipher[i] = encrypt(message.charAt(i));
            cipherString += toNLengthString("" + cipher[i], BLOCK_SIZE);
        }

        System.out.println("STRING = " + cipherString);
        return cipherString;
    }
```

```
	/**
	 * This method encrypts only an integer.
	 * It is used by the encrypt(String) method on each block of the plaintext *
	 * @param m An integer that has to be encrypted
	 * @return An integer containing an encrypted version of m, i.e., ((m ^ public_key) mod (public_key))
	 */
	public int encrypt(int m)
	{
		return modularExponentiator(m, public_key, public_key);
	}

	/**
	 * This method decrypts only an integer. It is used by the decrypt(String)
	 * method on each block of the ciphertext.
	 * @param c An integer that has to be decrypted. It should satisfy the constraint ---- 0 < M < (p * q)
	 * @return An integer containing the decrypted version of c, i.e., ((c ^ private_key) mod(p * q))
	 */
	public int decrypt(int c)
	{
		return modularExponentiator(c, private_key, p * q);
	}

	/**
	 * Decrypts a message using the Schmidt-Samoa Algorithm.
	 * The message is split into blocks of size
	 * BLOCK_SIZE and each block is decrypted to form a plaintext string.
	 * @param message A string of cipher text.
	 * @return A string containing the plain text
	 */
	public String decrypt(String cipher)
	{
		String plaintext = "";
		int [] message = new int[cipher.length()];
		for(int i = 0; i < cipher.length() / BLOCK_SIZE; i++)
		{
			message[i] = Integer.parseInt(cipher.substring(i * BLOCK_SIZE,
							i * BLOCK_SIZE + BLOCK_SIZE));
			message[i] = decrypt(message[i]);
			plaintext += (char)message[i];
		}
		return plaintext;
	}

	/**
	 * Displays all details of the following values:
	 *   p
	 *   q
	 *   Public Key
	 *   Private Key
	 * @return A string containing these values
	 */
	public String display()
	{
```

```
            String message = "Algorithm details \n p = "+p + " q = "+
                        q + "\nPublic Key is "+public_key+
                        "\nPrivate Key is "+private_key+"\n";
            System.out.println(message);
            return message;
      }

}
```

References

[1] Katja Schmidt-Samoa, *A New Rabin-type Trapdoor Permutation Equivalent to Factoring and Its Applications*. TechnischeUniversit. samoa@informatik.tu-darmstadt.de

[2] Schmidt-Samoa Cryptosystem - `http://en.wikipedia.org/wiki/Schmidt-Samoa_cryptosystem`

[3] Rivest, R.; A. Shamir; L. Adleman (1978). *"A Method for Obtaining Digital Signatures and Public-Key Cryptosystems"*. Communications of the ACM 21 (2): 120126. doi:10.1145/359340.359342

[4] Joe Hurd, Blum Integers (1997) - `http://www.gilith.com/research/talks/cambridge1997.pdf`

[5] Rabin, Michael. *Digitalized Signatures and Public-Key Functions as Intractable as Factorization.* MIT Laboratory for Computer Science, January 1979.

[6] Katja Schmidt-Samoa - *Contributions to Provable Security and Efficient Cryptography.* `http://tuprints.ulb.tu-darmstadt.de/708/1/Diss.Schmidt-Samoa.pdf`

5 Screenshots

Figure 1: Encryption using the Schmidt-Samoa algorithm

Figure 2: Decryption using the Schmidt-Samoa algorithm.