Chemical Fingerprints Analysis Using Several Hash Functions and SIMD Fast Mersenne Twister Random Algorithm

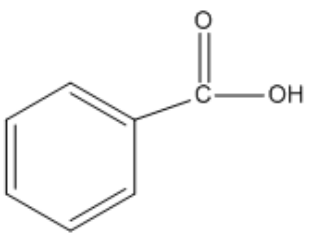
Vincentius Martin 13510017
Informatics Engineering
School of Electrical Engineering an Informatics
Bandung Institute of Technology, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*vincentiusmartin@students.itb.ac.id*

***Abstract*—Molecule consists of some complex structures. In particular, if we want to determine which molecules consist similar structure we have to search atom by atom which is very slow. Chemical fingerprint can be a solution to deal with this problem. It can be used to find molecule that has similar structure using bit string. To map structure and bit string, there are some ways that should be done like using hash function to determine which position that the specified structure should occupy. Of course every hash function has its own characteristic as well as its efficiency to be used in fingerprint area. It is important because fingerprint has to compute its process fast to determine similarity, so effective hash function is needed and can it be obtained through experiment. This paper is focused in determining the most suitable hash function to be used in fingerprint applications. SIMD Fast Mersenne Twister algorithm which is a very fast random algorithm will be used to map the structures' hash result to the specified position of bit in the bit string.**

***Index Terms*—Chemical Fingerprints, Hash Function, Molecule, SIMD Fast Mersenne Twister.**

I. INTRODUCTION

Find molecule that has similar pattern with another molecul needs carefulness, so every same pattern can be detected. This process is called structural search and it needs quite a lot efforts to be able to do that. It is because search in atom by atom can result on too many false verification. By using fingerprints, this problem can be made easier. This fingerprints application take quite a lot parts in chemoinformatics study. Many chemoinformatics system for molecules represented by fingerprints. It yields in the result of less storage space, also faster time in searching and matching process. But, the disadvantage is : it will be lossy,because some information may be lost during compression.



Picture 1.1. Molecule example : benzoic acid

Chemical fingerprints or called chemical hashed fingerprints (because it uses hash function) is bit string that contains information of the structure. The presence and absence of a pattern in the molecule is marked with '1' and '0' respectively. This pattern can also represent the number of occurences of some features like functional groups, chains, substructures, etc. Also, this fingerprints method can be used with another method to determine the similarity of a molecule with another. Other techniques examples are like data mining or pattern matching.

To be able to map every structure into a bit position, hash function is needed. Hash function is a function that accept input in form of string or bytes in any length and transform it into a fixed string which is much smaller than the original input. In chemical fingerprints, hash function is an important existence because it is the main function that will be used to map every structure into the position in the pattern.

One other thing that is important is the way every pattern mapped. In this paper, SIMD Fast Mersenne Twister random algorithm will be used. It is a random algorithm introduced in 2006 by Mutsuo Saito and Makoto Matsumoto as the transformation of Mersenne Twister Algorithm that is proposed by Makoto Matsumoto and Takuji Nishimura. The reason why this algorithm is used is because it is really fast and has good equidistribution property although it is not as good as WELL ("Well Equidistributed Long-period Linear").

II. SOME THEORIES

2.1. SIMD Fast Mersenne Twister

Before explaining how to generate fingerprints, first, the brief introduction about how the random algorithm works should be done. SIMD-oriented Fast Mersenne Twister (SFMT) uses Linear Feedback Shift Register to generate 128-bit pseudorandom integer. This algorithm is first implemented in C language. As what is contained in its name, this algorithm is designed using Single Instruction Multiple Data (SIMD) technique to achieve parallelism in its process.

Linear Feedback Shift Register (LFSR) is a method of generating a sequence of elements x0, x1, x2, ... etc. The recursion method is used in this algorithm to generate LFSR. In the implementation, the computation itself is

using an array of integers with specified size. At first, the array will be initialized with some values. Also, some bitwise processes will be used to fill the initializtion value for the array. After that, in order to produce the array's elements, a linear function will be used. It is a high-speed linear function because it does not use any multiplication process.

SFMT is MT-like pseudo-random number generator that makes full use of SIMD. The period of this algorithm is 2^{19937}-1 that's why sometimes this algorithm is called SFTM19937. Compared to MT algorithm, if we look into its performance, SFMT has better equi-distribution property and faster than MT, even without using SIMD instructions. SFMT itself is a LFSR generator that based on a recursion from its linear function.

Recursion in SFMT consists of some process. Parameter g which is the result of the recursion is computed using formula :

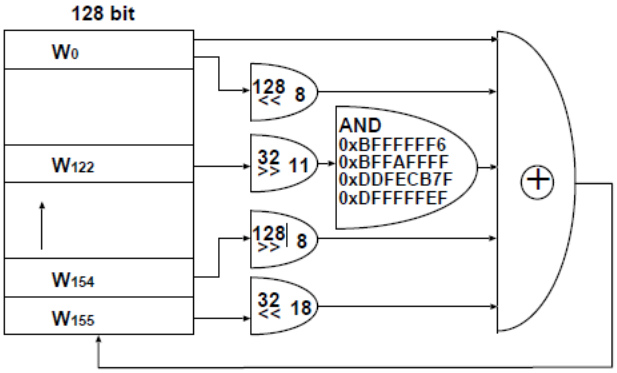
$$g(w_0, \ldots, w_{N-1}) = w_0 A + w_M B + w_{N-2} C + w_{N-1} D$$

Where w0,wM,... are 128 bit integers and wA, wB, wC, wD are computed using SIMD bit operations. After the operation, the copy of w[N-2] and w[N-1] will be stored in two 128-bit regiters(r1 and r2) which are available in CPU. The benefit from using these two registers is to use pipelining method effectively. While fetching the value of w[0] and w[M] from memory, w[N-2] and w[N-1] can be computed because their values are kept in the registers.

The period of SFMT is a multiple of 2^{19937}-1, with good equidistribution properties. The recursion degree can be computed using N/128 because each period is using 128 bit. So, because N is 19937, the recursion degree is 19937/128 = 156 combined with linear transformation A,B,C,D that are computed as follows :

- w*A* := (w << (128) 8) + w.
- w*B* := (w >> (32) 11)&(BFFFFFF6 BFFAFFFF DDFECB7F DFFFFFEF).
- w*C* := (w >>(128) 8).
- w*D* := (w <<(32) 18).

Operations above consist of some shifting operations, the numbers in brackets are the n bit integers. They are shifting operations of n-bit integer by m times. Below is the description if SFMT19937 generator with a period of 2^{19937}-1. The '+' operator which is used in wA computation is XOR operator in this paper. In wB, '&' operator is AND operation with 128-bit integer in hexadecimal form.



Picture 2.1. Circuit-like description of SFMT19937[12]

SFMT which is implemented in this paper is written in Java, so it can be said as SFMT-like random generator. The main difference is it does not use real SIMD like what is written in C. Parameters r1 and r2 are implemented using variables. But like what is said before, this algorithm is still fast even without using real SIMD.

2.2. Chemical Fingerprints

Molecule is represented by fingerprints in this model. The main part of the algorithm is to compress molecule including each feature into fingerprints which is modeled as byte array. There are some advantages of this method, which are :

- Every similar pattern will have similar bits set to '1' and similar bits are given chance to clash with certain probability.
- Fingerprints' size can be defined by user without knowing the pre-defined of the pattern.

But, even with the advantages that have been specified before, there are still some cons with this method. These cons are about effective hash and random algorithms, also the perfect size of fingerprints that will minimize the bit clashes and waste of the bit spaces. This problem mostly about finding the effective way of generating fingerprints and about choosing the effective algorithms.

There are some terms that are used in fingerprints field. They are :

-Fingerprint length

It defines the number of bit ('1' and '0') in the bit string that is generated.

-Maximum pattern length (depth)

Maximum length of atoms, computed in the molecule's linear path. It is important because the length of the cyclic molecules is limited to a fixed ring size.

-The position of bits that must be set to the pattern

Some bits in the pattern will be set to '1' which denotes the bit position.

-Darkness of the fingerprint

The comparison of '1' and '0' in the percentage of '1' appearance. Darker pattern is more choosen.

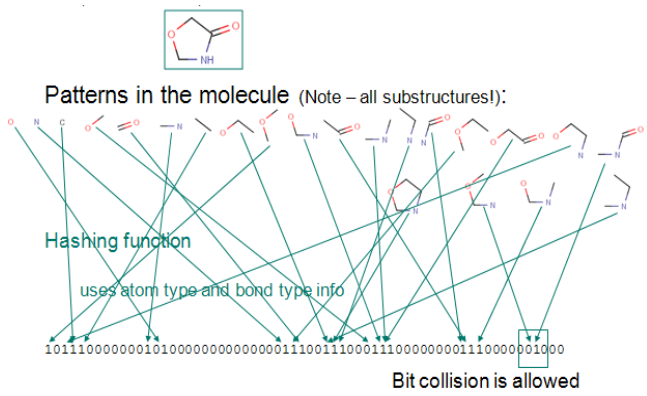
Also, there are unqiue pattern count (UPC) that counts the number of unique pattern for every depth. The list of unique pattern from every depth will be inserted into one list and used in the process. So the process will use all unique patterns from every depth (0..*).

Some steps are needed to be done to generate fingerprint from molecule. We have to keep track of generated pattern type to keep the qualitiy of the fingerprint that is about to be generated. It must be noticed that the important thing in generate a fingerprint is to determine the depth that will be used. Depth parameter is used to determine the maximum pattern length from the molecule that is inputted. It also indicate how accurate the fingerprint represents the molecule. By increasing the depth, we can get more precise fingerprint or it can be said that the fingerprint is darker. But, it also increase the chance of bit clashes, maybe a little or some

bit clashes are okay, but if too many bit clashes, it will be a problem because it will reduce the fingerprint's accuracy.

The algorithm to generate fingerprint is to split over molecule into structures in specified depth. In this method, the molecule is represented with a String of molecule. After that, by using hash and random function, we can generate its fingerprint. The detail algorithm and the ilustration for the process is shown below :

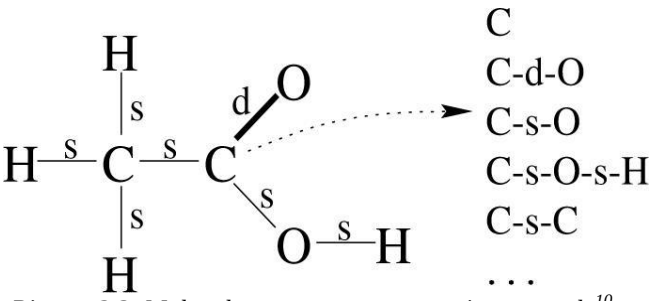
```
function generateFingerprint (input String : molecule,
                              integer : size) →String
{function to generate fingerprint bit from string of
molecule}
DECLARATION
  depth,i,seed,position_idx : integer
  molecule, fingerprint ← String
  molecule_pattern: array [0..*] of String
ALGORITHM
  molecule_pattern ← generatePattern(molecule,depth)
  i ← 0
  fingerprint ← ""
  {fingerprint initialization with '0' for all elements}
  while (i<1000) do
    fingerprint ← fingerprint + "0"
  endwhile
  foreach element in molecule_pattern do
    seed ← hashFunction(element)
    position_idx ← |SFMT19937(seed)| mod size
    fingerprint[position_idx] ← "1"
  endfor
  →fingerprint
```



Picture 2.2. Process example[1]

First, the algorithm generates every pattern from depth 0 to n. The pattern is in the form of the element (C, N, O, H, etc) and the number of edges (1,2,3,...) incident to that vertex or it can be said as the number of atoms bonded to the corresponding atom. Also, the type of bond (single as s, double as d, etc) is also written. For example, propane will be written as C1sC2sC1 (depth 3) and ethene as C1dC1 (depth 2). With this pattern, every molecule pattern can be written for depth 0 to depth n, with n is the maximum depth.

Also, the pattern can be represented as depth-first graph representation. The labels on the vertrices are the atom symbols and the edges are labeled with the type of covalent bond between atoms with the naming rule like before. This pattern is good to represent the bond types that a molecule has.

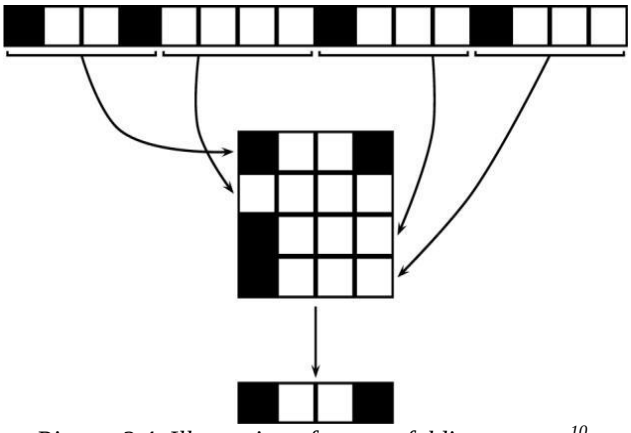


Picture 2.3. Molecule patterns representation example[10]

2.3. Modulo Compression of Chemical Fingerprints

The fingerprints itself can be folded into a shorter bit length. To shorten fingerprint, the modulo algorithm is used to make the fingerprint with length N to be a new fingerprint with fixed length M where M must be shorter than N. The other thing that must be noticed is the value of M must fulfill : N = M * C, where C is a constant.

For a given molecule, in the new compressed fingerprint that has been generated, the bit in position j of the new fingerprint is set to '1' if and only if there is at least one bit that has value of '1' in the position of $i \equiv j \bmod M$. In current systems, usually the compressed fingerprint size is set to 2^9 = 512 or 2^{10} = 1024. It must be considered that the compression will be effective only when all bits in the pattern are treated equally, it comes from the specific ordering of bits in the pattern. In practice, a random function or a hashing function (or both) can be used to generate the new compressed fingerprint's pattern. Of course, the good hashing or random function is preferred. The diagram below show a very basic simple compression of a pattern with 16 bit length to its compression with 4 bit length.



Picture 2.4. Illustration of pattern folding process[10]

In the illustration above, a pattern with length 16 (1001000010001000) is folded into compressed pattern with length 4 (1001). It can be seen that every position in

k position is checked. When at least one bit pattern in that position contain the bit '1' or black square in the illustration above. The position zero and three in modulo four of the pattern contain at least one bit '1' so in the new pattern, the position zero and three is filled with bit '1'.

Compression for the chemical fingerprint is really easy to implement. The algorithm itself is really simple with the basic way and it still simple when hash or random function is used. But, it has some drawbacks for the implementation. First, compression is a one way technique just like hash function. Once it converted into its compression, there is no may to return it into its original pattern because the position of the '1' bit in the original pattern is unknown if we just have the compressed pattern. Second, the possible optimal compression rate for the pattern is unknown so we don't know whether the compressed pattern may be close to optimal or not. The smaller the size of compression pattern, most likely the pattern will be more inaccurate. To prevent this, we have to find better compression rate. It is important because when the compression result is far away from optimal, many informations are lost because the nature of compression itself is to abolish some informations. But it can be minimized when the compression rate is high.

III. EXPERIMENTS

To be able to get accurate experiment results, this chemical fingerprint method is implemented using a Java program. For the SFMT algorithm, as what has been stated before, it doesn't use SIMD like the real SFMT which is implemented in C language but it still SFMT algorithm and it still fast. Also, java can do fast computation so it wil not be a problem for using the combination of hash function and SFMT.

The implementation is done using three hash functions which are MD5, SHA-1, and SHA-512. The main goal of this experiment is to decide which hash function is faster and has less bit collision. Also, some analysis for the algorithm performance will be done for some depths. The analysis will be about the darkness for some depths which are used in this experiment. Other goal from this experiment is to choose the best attribute for molecule fingerprints.

3.1. Benzoic acid

Benzene is the most common aromatic compounds that contain the benzene ring. There are six vertex of carbon atoms and each one is bonded to one H or one other atom or group, also adjacent with two carbon atoms. It is represented with a hexagon form. One example of benzene is benzoic acid. Benzoic acid (C_6H_5COOH) also known as carboxybenzene is a weak acid compound that is widely used as food preservative. The molecule ilustration can be seen in the first page.

The experiment is used benzoic acid with the parameters:

- Fingerprint length : 32
- Maximum depth : 3

Because benzoic acid is a simple molecule, so small fingerprint length $2^5 = 32$ is enough. Long fingerprint length will result in small darkness result (wasting more spaces). Also, there will be some unique patterns for every depth. The data for every depth is shown below :

Table 3.1. List of data for every depth (Benzoic acid)

Depth 0 :	
UPC	3
Pattern List	"O","C","H"
Total UPC	3
Depth 1 :	
UPC	6
Pattern List	"C-d-C","C-s-C","C-s-H","C-d-O","C-s-O","O-s-H"
Total UPC	9
Depth 2 :	
UPC	7
Pattern List	"C-d-C-s-C","C-s-C-d-C","C-s-C-s-C","C-d-C-s-C","C-s-C-d-O","C-s-C-s-O","C-s-O-s-H"
Total UPC	17
Depth 3 :	
UPC	7
Pattern List	"C-s-C-d-C-s-C","C-d-C-s-C-d-C","C-d-C-s-C-d-O","C-d-C-s-C-s-O","C-s-C-s-C-d-O","C-s-C-s-C-s-O","C-s-C-s-O-s-H"
Total UPC	23

The experiment is using two kind of ways of generating fingerprint. First is allow bit clashes to analyse how many bit clash with used hash function. Second is do not allow bit clash to maximize the darkness value. Results from the experiment are shown below :

Table 3.2. First experiment result with length 32 (Benzoic acid)

MD5	
Generated Pattern	00000110000001010000001001000101000 01110001100000110100000010000
Bitclashes	6
Darkness	26.56%
SHA-1	
Generated Pattern	11011000010000100010000100000000000 01110000010000001101000000100
Bitclashes	7
Darkness	25.0 %
SHA-512	
Generated Pattern	00000011001100001011001100010010000 00010010010001000000100001000
Bitclashes	6
Darkness	26.56%

Table 3.3. Time result for first experiment(Benzoic acid)

No.	MD-5	SHA-1	SHA-512
1	45 ms	18 ms	33 ms
2	41 ms	17 ms	46 ms
3	47 ms	17 ms	32 ms

From the first experiment's result above, MD5 and SHA-512 give better result for the darkness point. The bit clashes are fewer than SHA-1. But it just a little fewer, in the time experiment, SHA-1 performs much better that the other functions. The SHA-1 hash function generate fingerprint very fast (much faster than the other functions) and more stable in the time factor.

Table 3.4. Second experiment result with length 32 (Benzoic acid)

Darkness	35.93%
MD5	
Generated Pattern	00001111000001010000001001001101100 01110001100000110100010010001
Bitclashes	19
SHA-1	
Generated Pattern	11111000011001100010000100001000000 01110000010001101101000000101
Bitclashes	10
SHA-512	
Generated Pattern	01010011001101001011001101010010000 00010010010001010000100001010
Bitclashes	6

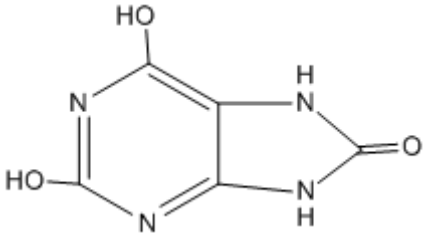
Table 3.5. Time result for second experiment(Benzoic acid)

No.	MD-5	SHA-1	SHA-512
1	43 ms	19 ms	52 ms
2	47 ms	18 ms	32 ms
3	56 ms	19 ms	37 ms

From the second experiment, the darkness result is better than the first experiment. This result is same for all hash function because it will occupy same number of positions in the fingerprint with the number of molecule patterns that a molecule contained. When the bit clash happened, the SFMT algorithm will generate next random number from the given seed. This process will be repeated until the position given is not clash with any bit positions. If the seed given is not good enough, more iteration will be done to find the right value. SHA-512 is good to be used in this experiment, there are only six bit clashes, this value is same with first experiment. It means the next value from the SFMT function is always generate unique position number.

3.2. More Complex Molecule

After using simple molecule like benzoic acid, how about if another more complex molecule is used? The result is necessary to determine the accuracy of the experiment because it may be end in different result from first experiment. More used molecules is needed so the real result can be obtained (increase the accuracy). Also, chemical fingerprint that is about to be generated might use longer time. The molecule that is used in this experiment is shown below :



Picture 3.1. Uric acid

This molecule is an uric acid, a heterocylic compound that consists of carbon, nitrogen, oxygen, and hydrogen. It comes from the degradation of certain elements in the body. The cause elements can be occured because of foods.

Although the molecule used is more complex than the first one, it still simple enough and the fingerprint length $2^5 = 32$ still can be used. The used parameters are like the experiment using benzoic acid. Data for every depth in second experiment is shown below :

Table 3.6. List of data for every depth (Uric acid)

Depth 0 :	
UPC	4
Pattern List	"O","C","H","N"
Total UPC	4
Depth 1 :	
UPC	8
Pattern List	"O-s-H","O-s-C","C-s-N","C-d-N","C-s-C","C-d-C","N-s-H","C-d-O",
Total UPC	12
Depth 2 :	
UPC	14
Pattern List	"H-s-O-s-C","O-s-C-s-N","O-s-C-d-N","C-d-N-s-C","C-s-N-d-C","O-s-C-d-C","C-d-C-s-C","C-d-C-s-N","N-d-C-s-N","N-d-C-s-C","C-s-C-s-N","N-s-C-s-N","H-s-N-s-C","N-s-C-d-O"
Total UPC	26
Depth 3 :	
UPC	18
Pattern List	"H-s-O-s-C-d-N","H-s-O-s-C-s-N","O-s-C-d-N-s-C","O-s-C-s-N-d-C","C-d-N-s-C-s-O","N-s-C-s-O-s-H","N-s-C-d-C-s-C","C-s-N-d-C-s-C","C-d-C-s-C-d-N","C-d-C-s-N-s-H","N-d-C-s-N-s-H","N-s-C-s-C-s-N","N-d-C-s-C-s-N","C-d-C-s-C-s-N","C-d-C-s-N-s-C","C-s-C-s-N-s-C","N-d-C-s-N-s-C","C-s-N-s-C-d-O"
Total UPC	44

Just like the first experiment, the same way will be done. By using two ways in the experiment, there will be two kind of results with bit clashes or not. The results from the experiment are shown below :

Table 3.7. First experiment result with length 32 (Uric acid)

MD5	
Generated Pattern	10110110101001110100000110000111001 01111001011010011100001010011
Bitclashes	12
Darkness	50%
SHA-1	
Generated Pattern	11010110011000111010100100000110101 01011100010000101011000010110
Bitclashes	15
Darkness	43.31 %
SHA-512	
Generated Pattern	10110110001100010010100010010010100 10110010110100101101111111110
Bitclashes	11
Darkness	51.56%

Table 3.8. Time result for first experiment(Uric acid)

No.	MD-5	SHA-1	SHA-512
1	54 ms	31 ms	26 ms
2	67 ms	33 ms	29 ms
3	51 ms	25 ms	25 ms

The given results are different from the benzoic acid experiment. In this process, SHA-1 shows lesser bit clashes than other two algorithms. For the speed, SHA-512 is the best for this experiment, but only a little better than the time that is needed by SHA-1.

Table 3.9. Second experiment result with length 32 (Uric acid)

Darkness	68.75 %
MD5	
Generated Pattern	10111111101101111110001110100111011 01111011011010011111001110011
Bitclashes	27
SHA-1	
Generated Pattern	11111111111101111110101101101110101 01011101010001111011000110110
Bitclashes	42
SHA-512	
Generated Pattern	10110111001101011110101011010110100 11110010111110111101111111110
Bitclashes	31

Table 3.10. Time result for second experiment(Uric acid)

No.	MD-5	SHA-1	SHA-512
1	54 ms	23 ms	25 ms
2	55 ms	27 ms	23 ms
3	52 ms	25 ms	26 ms

In this experiment, MD5 shows less bit clashes. The result is really different from benzoic acid experiment so the number of bit clashes cannot be used as a parameter to determine effective algorithm. From given time result, it seems that SHA-512 shows a little better time result.

Also, to understand the relation between fingerprint length and the bit clashes of the pattern, the anaysis is done for it. The analysis is done using fingeprints in various size. Also, based on the experiments before, the used hash function is SHA-512 because it fast and show less bit clashes. The diagram below is the anaylsis the relation result between pattern size and bit clashes probability :

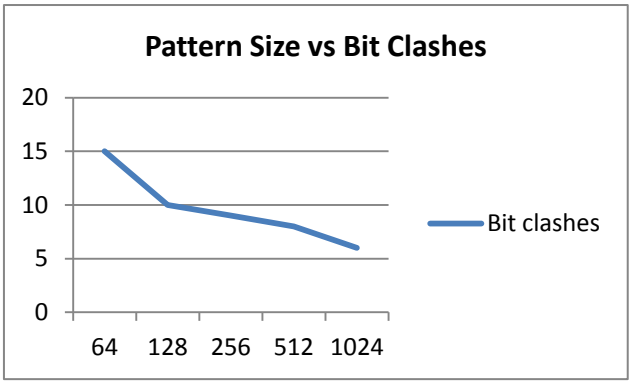


Diagram 3.1. Pattern size vs bit clashes

From the diagram above, the bit clashes is getting less by the increasing of pattern size. Although the reduction of the number of bit clashes is not really large for every size, but it can be concluded that by the increase of the pattern size the bit clashes will be less likely to happened. Also, the reduction is different for every depth, but it significant enough for pattern size 64 to 128. It is about delta = 5 from pattern size 64 to pattern size 128.

Also, this molecule will be used to generate a pattern with size of 1024 to generate its modullo compression of chemical fingerprints with 64 bit length. First, the fingerprint pattern with length 1024 is obtained using SHA-1 hash function. The length of the pattern must be large enough to increase the accuracy of its compression. Because if it is not large enough, the compression can return all '1's in its value for many molecules. The test is using every hash function to generate compression pattern. Below is shown the compression and time result for each hash function :

Table 3.11. Compression result*(Uric acid)*

MD5			
Compression Pattern	11101001010001101000111110000110010 00000001101100001111000110010		
Time	53 ms	53 ms	57 ms
SHA-1			
Compression Pattern	00100011111000110001000100000001111 01010110010001001001011010010		
Time	21 ms	28 ms	22 ms
SHA-512			
Compression Pattern	01010000100100000001100001010100100 00110000010011010010011111110		
Time	27 ms	29 ms	28 ms

From the compression result above, the accurate compression might be gotten from 64-bit compression that is used. It is because the distribution of '0' and '1' is more likely to spread equally. Fastest function is SHA-1, it can be seen from table above. The difference between SHA-1 and SHA-512 time is not much and the two of the functions can compute the result effectively.

IV. Analysis

From the experiment above, some results can be analysed. In the experiment using benzoic acid molecule patterns, the number of bit clashes between function is similar. SHA-1 get the most clashes but it just only in delta one with other functions. But, the advantage of SHA-1 is it faster in this experiment compared to other two functions. For the second experiment with benzoic acid, SHA-512 is less likely to have bit clashes in its result. But it may be just a coincidence because of the seeds quality that enter the SFMT19937 function. So to determine this, another experiment using different molecule is needed. In the second experiment for benzoic acid, SHA-1 has the least time and most stable process time. SHA-1 is good to be used based on this first experiemnt because it has fast process time and the process time is also stable for given molecule.

For the experiment with uric acid, SHA-1 has the least clashes compared to other two functions. The best time result is SHA-512. But, compared to SHA-1, the difference is not much and these two functions are stable in their value. In the second experiment, the bit clashes is more for SHA-1.

Based on these two experiments using two different molecules, the bit clashes factor is not too important. It is because the bit clashes factor is really depend on the seed. When the seed quality is good, there will be less bit clashes and when the quality is bad, there will be more bit clashes. Sometimes, a hash function can produce good seed and sometimes, it can produce bad seed too and it cannot be determined. So, the most important factor here is process time. From the time result, the most stable and fast function is SHA-1 although the difference with SHA-512 is not much. SHA-1 can make the computation process fast for this process.

The result for pattern size vs bit clashes is really clear. It is because the result is really clear to be seen. As the pattern size increased, the chance for the bit clashes is less likely to happened. Also, the difference in delta value is big for an early value, in this experiment, it is from size 64 to 128 that has big delta value of bit clashes. The reduction is not many for the next value, only about one reduction for every value.

In the bit compression experiment, the fastest function is SHA-1 although still the difference between SHA-1 and SHA-512 is not much. The most important thing in the modulo compression is the bits in the pattern is spread equally. As the size of the original pattern gets larger, the resulted compression pattern will be more accurate. It indicates that compression function is indeed to be used for large size of the pattern. The distribution itself is good for the pattern with 64 bit size from 1024 bit size, but the good parameters to generate pattern is also has to be studied.

Also, SFMT19937 is a fast random function and it is good to be used in generating chemical fingerprints. It does not take much time, the time factor is more likely to be caused by the hash functions and other processes in generating chemical fingerprints. This random function is suitable to do something that needs random function and has to be fast.

V. Conclusion

Chemical fingerprints can be generated using the combination of hash function and random function. The good parameters also determine the optimal result for the fingerprint. Good hash function will have fast computation time process. From the experiment, it is gotten that the suitable hash function from any other hash functions that are tested is SHA-1 because of the stable and fast characteristic that it has. SFMT19937 is also good to be used in this process because of the fast computation that is done by SFMT19937. The random function is used in the last process of determining the right position for the bit pattern.

REFERENCES

[1] Asad. *Revisiting Molecular Hashed Fingerprints*. May, 19, 2013 (11.55PM) <http://chembioinfo.com/2011/10/30/revisiting-molecular-hashed-fingerprints/>

[2] Brady, James E; Frederick A.Senese, & Neil D.Jespersen.2009. *CHEMISTRY* 5th ed. Asia : John Wiley & Sons.

[3] Cahyana, Ucu; Dede Sukandar; Rahmat. 2007. *KIMIA* 3rd ed. Jakarta : Piranti Darma Kalokatama.

[4] ChemAxon. *Chemical Hashed Fingerprints.* May, 19, 2013 (11.50PM) < http://www.chemaxon.com/jchem/doc/user/fingerprint.html>

[5] ChemAxon. *Parameters for Generating Chemical Hashed Fingerprints.* May, 19, 2013 (11.00PM) <http://agave.health.unm.edu/iphace/iPHACEdir/jchem/doc/user/fingerprint.html>

[6] Dalke Scientific. *Molecular Fingerprints, Background*. May, 20, 2013 (7.00AM) <http://www.dalkescientific.com/writings/diary/archive/2008/06/26/fingerprint_background.html>

[7] Fieser, Louis F; Mary Fieser. 1963. *Pengantar Kimia Organik*. Bandung : DHIWANTARA.

[8] Matsumoto, Makoto. *Mersenne Twister Homepage*. May 17, 2013 (10.20PM) <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html>

[9] PubChem. *Benzoic Acid- Compound Summary*. May 19, 2013 (11.55PM) <http://pubchem.ncbi.nlm.nih.gov/summary/summary.cgi?cid=243/>

[10] PubMed Central. *Lossless Compression of Chemical Fingerprints Using Integer Entropy Codes Improves Storage and Retrieval*. May 19, 2013 (11.55PM) <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2536658/>

[11] Saito, Mutsuo; Makoto Matsumoto. *SFMT Homepage*. May,19, 2013 (11.45PM) < http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/ >

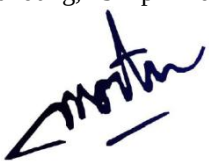
[12] Saito, Mutsuo; Makoto Matsumoto.2007; . *An Application of Finite Field : Design and Implementation of 128-bit Instruction-Based Fast Pseudorandom Number Generator*. Japan : Dept. of Math. Graduate School of Science Hiroshima University.

[13] UricAcid. *Uric Acid information-descriptions*. May 19, 2013 (11.45PM) <http://uricacid.eu/>

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 29 April 2010



Vincentius Martin 13510017