

# Chemical Structure Comparison with String Matching

Fauzan Hilmi Ramadhian 13512003  
Program Studi Teknik Informatika  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung, Jl. Ganessa 10 Bandung 40132, Indonesia  
fauzan\_hilmi@students.itb.ac.id

**Chemical structure comparison is an important topic in chemical science. It is useful in modern approaches to predicting the properties of chemical compounds and designing chemicals with desired properties. Quite lot techniques have been found to do the comparison. One of the simple ways is by string matching. In this method, the chemical structure will be converted to one dimension form in SMILES notation. Then, the string matching algorithm will decide how the structures differ. In this paper, two string matching algorithms will be discussed upon their performances on comparing the chemical structures.**

***Index Terms*— String matching, chemical compound comparison, SMILES, Knut-Moris-Pratt, Boyer-Moore**

## I. INTRODUCTION

Chemistry is an interesting an important topic nowadays. Why? Because everything are made of chemicals. Cars, buildings, handphones, plants, even humans are made of chemical elements. Besides that, chemistry helps human to understand what is going on this world. Various events that happened everyday can be explained from its point of view. Another importance, and the most important, is that chemistry made human life a lot easier. Many processes like the made of foods, drinks, soaps, and medicines are done because of their chemical process.

One important subject in chemistry world is chemical structure comparison. It has many important applications. The main usage of structure comparison is to determine the molecules properties differences. Other applicative purposes are to determine the best components of a particular object and discover new drugs.

Many techniques have been invented to do the structure comparison. Some of them are quite complex. However, the simplest way is to check the structure image itself. Check whether one structure is same or not with the other one. In this simple technique, there are some ways to do so. A really simple way is to convert the structure image to a line of ASCII strings, then do the comparison with string matching algorithm.

Upon converting the structure, there is one standard

convention to represent molecule structures in ASCII string. It is SMILES (Simplified Molecular Input Line Entry Specification). All complex structure can be drawn precisely by this standard. SMILES will help the comparison too, since it has definite structure of strings.

For the SMILES string comparisons, a string matching algorithm is needed. There are several algorithms like so. Two simple and quite simple examples are Knuth-Morris-Pratt (KMP) and Boyer-Moore algorithm. Each of them has their own advantages and disadvantages.

In this paper, the SMILES structure comparison by KMP and Boyer-Moore algorithms will be discussed further. Each of them has its implementation example in Java. Later, their performances will be compared to determine which is better to do the molecule structure comparison.

## II. THEORY

### A. Chemical matters

Matters are all things that have mass and occupied mass. In chemistry, the matters are divided into smaller groups with distinct definitions and properties. In general, there are three matter types. They are atom, element, molecule, compound, and mixture.

Atom is the fundamental matter of all matters. Atom is the smallest particle of an element. It consists of a central nucleus containing protons and neutrons. The electrons revolve around the nucleus in imaginary paths called orbits or shells.

Element is a matter made up of a kind of atoms. Each element is distinguished by its atomic number, atomic weight, and mass number. Atomic number is the number of protons in the nucleus. Atomic weight is number of times an atom of that element is heavier than an atom of hydrogen. Mass number is the total number of protons and neutrons in the nucleus of an atom.

Molecule is a group of two or more atoms held together by chemical bonds. The atoms can be from the same or different elements. For example, two atoms of oxygen (O) combine to form a molecule of oxygen (O<sub>2</sub>) and one atom hydrogen (H) with two atoms of oxygen (O) form a molecule of water (H<sub>2</sub>O).

Chemical compound is a class of molecule. A compound is formed when the elements are chemically combined in a fixed proportion. For example, in H<sub>2</sub>O, the hydrogen and oxygen atoms are combined in 2:1 ratio.

Just like compound, chemical mixture is a class of molecule. The difference is that mixture is not chemically formed from its elements. Also, the elements may not be in a fixed ratio. The example for this is syrup and sand-water mixture.

From here, the term "chemical structure" or "structure" is referred to chemical element or molecule.

### B SMILES

SMILES (Simplified Molecular Input Line Entry Specification) is a chemical nomenclature to represent element and molecule structures in one dimension. One dimension here means that the structures are written in a line notation using ASCII strings. Usually, chemists use a kind of software to convert the 2 or 3 dimension molecule structure to SMILES. Conversely, the SMILES can be converted to 2 or 3 dimension structure.

Original SMILES was invented by David Weininger at the USEPA Mid-Continent Ecology in Duluth, USA in the 1980s. Gilman Veith and Rose Russo (USEPA) and Albert Leo with Corwin Hansch (Pomona College) were acknowledged for their parts in the early development for supporting the work. Also Arthur Weininger (Pomona; Daylight CIS) were acknowledged for assistance in programming the system. The Environmental Protection Agency funded the initial project to develop SMILES. SMILES has since been modified and extended by other parties, most notably by Daylight Chemical Information Systems.

The main advantage of SMILES is its simplicity. It is very easy to read and write. It also can be memory-efficiently stored in data drive. Another advantage of SMILES is that it is easy to analyzed, as well as in structure comparison which is discussed on this paper.

There are some rules which must be obeyed upon converting between the 2/3 structure and SMILES. Here are them.

### Atoms

Atoms are represented by the standard abbreviation of the chemical elements, in square brackets. Brackets can be omitted for the "organic subset" of B, C, N, O, P, S, F, Cl, Br, and I. All other elements must be enclosed in brackets. If the brackets are omitted, the proper number of implicit hydrogen atoms is assumed.

An atom holding one or more electrical charges is enclosed in brackets, followed by the symbol H if it is bonded to one or more atoms of hydrogen, followed by the number of hydrogen atoms (as usual one is omitted), then by the sign '+' for a positive charge or by '-' for a negative charge. The number of charges is specified after the sign (except if there is one only); however, it is also possible write the sign as many times as the ion has charges. The symbol '\*' ("star") is treated by SMILES as a

valid atomic symbol meaning "unspecified atomic number" and is represented as an atom of atomic number zero.

Here are some examples of SMILES representation of element / molecule structures.

**Table 1.1** Atom representations on SMILES <sup>1</sup>

Depictions	SMILES	Remark
S	[S]	Defaults inside brackets: mass unspecified, charge 0, Hcount 0.
Au	[Au]	Second character of 2-character symbols is lower case.
CH <sub>4</sub>	C	Normal valence of carbon is 4
PH <sub>3</sub>	P	Lowest normal valence of phosphorous is 3.
HO <sup>-</sup>	[OH-] or [OH-1]	If charge value is missing, 1 is assumed, i.e., '+' is equivalent to '+1' and '-' is equivalent to '-1'
Fe <sup>+2</sup>	[Fe+2] or [Fe++]	Charge sign may be repeated or have a signed value, e.g., '+' is equivalent to '+2'.
<sup>235</sup> U	[235U]	A leading integer represents a specified atomic mass.
*+2	[*+2]	An atom of unknown atomic number with a +2 formal charge.

### Bonds

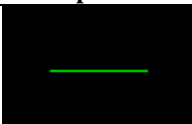
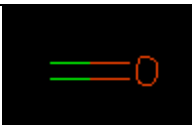

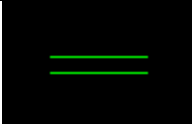
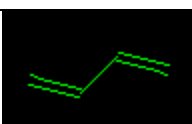
In SMILES, Single, double, triple, and aromatic bonds are represented by the symbols '-', '=', '#', and ':', respectively. Adjacent atoms without an intervening bond symbol connected by a valence-dictated bond (typically a single or aromatic bond). '-' (single) and ':' (aromatic) bond symbols may always be omitted on input.

Bonds between aliphatic atoms are assumed to be single unless specified otherwise and are implied by adjacency in the string. Ring closure labels are used to indicate connectivity between non-adjacent atoms in the SMILES string.

<sup>1</sup> Table was taken from <http://www.daylight.com/meetings/summerschool98/course/dave/smiles-atoms.html> on 17 May 2014, 9:26 PM

Here are some notable examples.

**Table 1.2.** Chemical bonds representations e in SMILES<sup>2</sup>

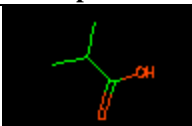
Depiction	SMILES	Remark
	CC or C-C or [CH3]-[CH3]	Adjacent aliphatic atoms are assumed to be bonded by a single bond: the single bond symbol '-' is not needed on input.
	C=O or O=C	Double bonds are represented by an equals sign. Note that the order of input doesn't matter (SMILES may start with any atom).
	C#N or N#C	Triple bonds are represented by a hash sign.
	C=C or cc	Ethene is normally written C=C, but the default bond between non-aromatic sp <sup>2</sup> atoms may be a double bond.
	C=CC=C or cccc	It's not always double bond. (Butadiene is normally written C=CC=C.)

### Branching

Branches are described with parentheses. Substituted rings can be written with the branching point in the ring. SMILES specifies no predefined limit to how deep branching may be nested. Most implementations define such a limit, typically between 10 and 50.


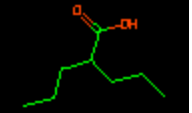
Here are few examples.

**Table 1.3.** Branching representations e on SMILES<sup>3</sup>

Depiction	SMILES	Remark
	CC(C)C(=O)O	If needed, bond symbols should appear inside branches.

<sup>2</sup> Table was taken from <http://www.daylight.com/meetings/summerschool98/course/dave/smiles-bonds.html> on 17 May 2014, 9:42 PM

<sup>3</sup> Table was taken from <http://www.daylight.com/meetings/summerschool98/course/dave/smiles-branching.html> on 17 May 2014, 9:57 PM

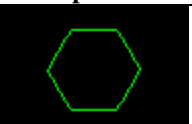

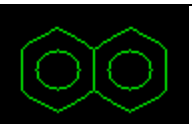
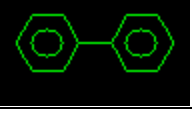
	FC(F)F or C(F)(F)F	Branches may be stacked. One can start with any atom equally well.
	CCCC(C(=O)O) CCC	Branches may be nested.

### Rings

Ring closure bonds are specified by inserting matching digits to the specifications of the joined atoms, with the bond symbol preceding the digit, if needed. The ability to re-use ring closure digits makes it possible to specify structures with more than 10 rings. Structures with more than 10 ring closures may be specified by prefacing a two digit number with percent(%) sign. For example, C2%13%24 is a carbon atom with ring closures 2, 13, and 24.

Here are more notable examples.

**Table 1.4.** Ring representation in SMILES<sup>4</sup>

Depiction	SMILES	Remark
	C1CCCCC1	If unspecified, the default bond order for the ring closure is the same as with any other bond.
	C1=CCCCC1 C=1CCCCC1 C1CCCCC=1 C=1CCCCC=1	The order of ring closure bonds may be specified as long as they don't conflict, e.g, C=1CCCCC-1 is <i>not</i> right
	c12c(ccc1)ccc2 same as c1cc2ccccc2cc1	Atoms can have more than one ring closure.
	c1ccccc1c2ccccc2 same as c1ccccc1c1ccccc1	A ring closure digit may be reused if desired.

### C. Chemical structure comparison

Upon comparing two molecules, say molecule 1 for the first and molecule 2 for the second, there are 4 outcome possibilities. The first result is *equal*. It happens when molecule 1 really has no difference with molecule 2. The second possibility is *substructure*. It means that molecule 2 contains molecule 1. The third kind is the reciprocal of the last one, which is *superstructure* that means molecule 1 contains molecule 2. Finally, the last possible result is different. It is when anything happens other than the first three.

<sup>4</sup> Table was taken from <http://www.daylight.com/meetings/summerschool98/course/dave/smiles-rings.html> on 17 May 2014, 10:07 PM

#### D. String Matching Algorithms

String matching problem is an important problem in computer science. The problem states that given a string text  $T$ , decides whether another string pattern  $P$  is found within  $T$  or not. In this paper, the problem extended to decide if  $T$  contains  $P$ , find out is  $T=P$  or not. And if  $P$  is not in  $T$ , decide is  $T$  in  $P$  or not. These extensions are adaptations so that the string matching algorithm can handle all chemical structure comparison results, whether it is similar, substructure, superstructure, or different.

There are many algorithms that can solve this problem. However, only two that will be discussed on this paper. They are Knuth-Morris-Pratt(KMP) and Boyer-Moore algorithms.

#### Knuth-Morris-Pratt (KMP) Algorithm

KMP algorithm looks for the pattern in the text in a left-to-right order. It is like the traditional (brute-force) algorithm, only it is more efficient by employing observation when a mismatch happens, the word embodies sufficient information to determine where the next match could begin. Thus, it is skipping wasteful examination of previously matched characters.

KMP uses a function called *border function* or *failure function* to determine how far the “jump” must be done to be efficient. The function executed before the comparison process to find matches of prefixes with the pattern itself. Here is a pseudo-code of the function.

#### KMP BORDER FUNCTION (P)

**Input:** Pattern with  $m$  characters

**Output:** Border function  $f$  for  $P[i..j]$

```
 $i \leftarrow 1$ 
 $j \leftarrow 0$ 
 $f(0) \leftarrow 0$ 
while  $i < m$  do
  if  $P[j] = P[i]$ 
     $f(i) \leftarrow j + 1$ 
     $i \leftarrow i + 1$ 
     $j \leftarrow j + 1$ 
  else if
     $j \leftarrow f(j - 1)$ 
  else
     $f(i) \leftarrow 0$ 
     $i \leftarrow i + 1$ 
```

The border function maps  $j$  to the length of the longest prefix of  $P$  that is a suffix of  $P[1..j]$ , encodes repeated substrings inside the pattern itself.

After counting the failure function, the main algorithm begins. Let's say that Text =  $T$  and Pattern =  $P$ . In each comparison, if a mismatch occurred at  $P[j]$ , then  $j = f(j)+1$ . Then, shift  $P$  to the right as long as  $j$  and do the comparison again. For clearer view, here is the pseudo-code.

#### KNUTH-MORRIS-PRATT (T, P)

**Input:** Strings  $T[0..n]$  and  $P[0..m]$

**Output:** Starting index of substring of  $T$  matching  $P$

```
 $f \leftarrow$  compute failure function of Pattern  $P$ 
 $i \leftarrow 0$ 
 $j \leftarrow 0$ 
while  $i < \text{length}[T]$  do
  if  $j \leftarrow m - 1$  then
    return  $i - m + 1$  // we have a match
     $i \leftarrow i + 1$ 
     $j \leftarrow j + 1$ 
  else if  $j > 0$ 
     $j \leftarrow f(j - 1)$ 
  else
     $i \leftarrow i + 1$ 
return -1 //no match
```

#### Boyer-Moore Algorithm

This algorithm works by scanning the characters from right to left beginning with the rightmost character. During the testing of a possible placement of pattern  $P$  against text  $T$ , a mismatch of text character  $T[i]=c$  with the corresponding pattern character  $P[j]$  is handled as follows: if  $c$  is not contained anywhere in  $P$ , then shift  $P$  completely past  $T[i]$ . Otherwise, shift  $P$  until an occurrence of  $c$  in  $P$  gets aligned with  $T[i]$ .

Generally, Boyer-Moore algorithm is based on two techniques. They are the looking-glass and character jump technique. The looking-glass technique means that the algorithm searches  $P$  in  $T$  by moving backwards through  $P$ , starting at its end. Then, the character jump technique means that when a mismatch occurs at  $T[i]=c$  and so  $T[i] \neq P[j]$ . In that case, there are 3 possibilities.

- If  $P$  contains  $c$  somewhere, try to shift  $P$  right to align the last occurrence of  $c$  in  $P$  with  $T[i]$ .
- If  $P$  contains  $c$  somewhere, but a shift right to the last occurrence of  $c$  is not possible, shift  $P$  right by one character to  $T[i+1]$ .
- If above cases are not happened, shift  $P$  to align  $P[1]$  with  $T[i+1]$ .

Just like KMP, Boyer-Moore needs compute a pre-processed function called *last occurrence function*. This function takes a character  $c$  from the alphabet then specifies how far may shift the pattern  $P$  if  $c$  is found in the text that does not match the pattern. Here is the general function.

$\text{last}(c) = i$ , index of the last occurrence  $c$  in  $P$   
-1, if  $c$  is not in  $P$

After counting the function, the main algorithm begins. Here is the pseudo-code.



## A. KMP Algorithm Implementation

### Algorithm

```

public static int kmpMatch(String text,String pattern)
{
    if (pattern.length()>text.length())
    {
        String temp = pattern;
        pattern = text;
        text = temp;
    }
    int n = text.length();
    int m = pattern.length();
    int fail[] = computeFail(pattern);

    int i=0;
    int j=0;
    while(i <n){
        if(pattern.charAt(j) ==
text.charAt(i)){
            if(j == m-1){
                return i-m+1;
            }
            i++;
            j++;
        }else if(j>0)
        {
            j = fail[j-1];
        }else{
            i++;
        }
    }

    return -1;
}

//Border function
private static int[] computeFail(String pattern)
{
    int[] fail = new int[pattern.length()];
    int k, q;

    fail[0] = -1;
    q = 1;
    k = -1;

    for (q=1;q<pattern.length(); q++)
    {
        while ((k >= 0) &&
(pattern.charAt(q) != pattern.charAt(k+1)))
        {
            k = fail[k];
        }

        if (pattern.charAt(q) ==
pattern.charAt(k+1))
        {
            k = k + 1;
        }

        fail[q] = k;
    }

    //Increment all elements
    int i;
    for (i=0; i<pattern.length(); i++)
    {
        fail[i]++;
    }
    return fail;
}

```

### Results

**Fig 3.1.** Result with molecule 1 = hemoglobin and molecule 2 = hemoglobin

```

D:\Kuliah\IF\java>java stringmatching
Enter molecule 1 SMILES:
O=C(N[C@@H](C(=O)O)CC(C)C)[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)
[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)
[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](N)C)CO)CC(C)C)CC(=O)O)CCCCN)Cei1ccc
C)C)CO)[C@@H](O)C)C(C)C
Enter molecule 2 SMILES:
O=C(N[C@@H](C(=O)O)CC(C)C)[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)
[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)
[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](N)C)CO)CC(C)C)CC(=O)O)CCCCN)Cei1ccc
C)C)CO)[C@@H](O)C)C(C)C
Result : Molecule 1 is EQUAL with Molecule 2
Execution time : 119586 nanoseconds

```

**Fig 3.2.** Result with molecule 1 = benzene and molecule 2 = hemoglobin

```

D:\Kuliah\IF\java>java stringmatching
Enter molecule 1 SMILES:
c1ccccc1
Enter molecule 2 SMILES:
O=C(N[C@@H](C(=O)O)CC(C)C)[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)
[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)
[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](N)C)CO)CC(C)C)CC(=O)O)CCCCN)Cei1ccc
C)C)CO)[C@@H](O)C)C(C)C
Result : Molecule 1 is SUBSTRUCTURE of Molecule 2
Execution time : 57562 nanoseconds

```

**Fig 3.3.** Result with molecule 1 = hemoglobin and molecule 2 = benzene

```

D:\Kuliah\IF\java>java stringmatching
Enter molecule 1 SMILES:
O=C(N[C@@H](C(=O)O)CC(C)C)[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)
[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)
[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](N)C)CO)CC(C)C)CC(=O)O)CCCCN)Cei1ccc
C)C)CO)[C@@H](O)C)C(C)C
Enter molecule 2 SMILES:
c1ccccc1
Result : Molecule 1 is SUPERSTRUCTURE of Molecule 2
Execution time : 95936 nanoseconds

```

**Fig 3.4.** Result with molecule 1 = hemoglobin and molecule 2 = aspirin

```

D:\Kuliah\IF\java>java stringmatching
Enter molecule 1 SMILES:
O=C(N[C@@H](C(=O)O)CC(C)C)[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)
[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](NC(=O)
[C@@H](NC(=O)[C@@H](NC(=O)[C@@H](N)C)CO)CC(C)C)CC(=O)O)CCCCN)Cei1ccc
C)C)CO)[C@@H](O)C)C(C)C
Enter molecule 2 SMILES:
O=C(OCC)C
Result : Molecule 1 is DIFFERENT with Molecule 2
Execution time : 64701 nanoseconds

```



As can be seen, KMP looks more efficient than Boyer-Moore, with 3 results are faster against 1. Why this could happen?

KMP is faster than Boyer-Moore when the input strings contain a lot of repeatedly characters. Because the characters in SMILES strings usually appear multiple times to represent the multiple appearances of the elements, the string matching in SMILES is faster with KMP algorithm rather than Boyer-Moore's.

It should be noted that even though KMP is better than Boyer-Moore, their performances only differ slightly. In this test, the largest execution time difference on a test case is about 80000 nanoseconds or about 0.008 milliseconds. This is really a small time difference for a quite large input string. Thus, for SMILES string matching, it can be safely said that KMP and Boyer-Moore performances are quite same.

## V. CONCLUSION

Molecule structure comparison is one important subject in chemistry science. It has many applications in humans' life. A simple way to do so is by comparing the molecules structure image. A simpler way is to convert the image to SMILES string, then compare it by simple string matching algorithm. Two examples of such algorithms are KMP and Boyer-Moore algorithms.

After a test, it is proven that KMP is more efficient than Boyer-Moore in SMILES strings comparison. However, because their performances only differ slightly, it can be assumed that their performances are same.

## VII. ACKNOWLEDGMENT

First of all, Author would say thank you to Almighty God because of His mercy and grace Author can finish this paper. Then, Author also wants to express his thanks to Mr. Rinaldi Munir, and Mrs. Masayu Leylia Khodra, whose give helpful advices and assistances. Finally, Author wants to say thank you to his parents and beloved friends who are always give Author strengths and spirits to pass the struggles during the writing of this paper.

## REFERENCES

- [1] Daylight Chemical Information Systems, Inc. "SMILES Tutorial," <http://www.daylight.com/meetings/summerschool98/course/dave/smiles-intro.html>, accessed on May 17, 2014 08:10 PM
- [2] Munir, Rinaldi, 2009. "Diktat Kuliah IF2211 Strategi Algoritma," Program Studi Teknik Informatika STEI ITB
- [3] Rashid bin Muhammad. "Knuth-Morris-Pratt Algorithm," <http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/StringMatch/kuthMP.htm>, accessed on May 18, 2014 08:37 PM

[4] Rashid bin Muhammad, "Boyer-Moore Algorithm," <http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/StringMatch/boyerMoore.htm>, accessed on May 18, 2014 08:45 PM

[5] Syvum Technologies Inc. "Chemistry : Atoms, Molecules, Elements and compounds," <http://www.syvum.com/cgi/online/serve.cgi/squizzes/chem/atomic2.html>, accessed on May 17, 2014 11:21 AM.

## 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, 19 Mei 2014



Fauzan Hilmi Ramadhian 13512003