韓國컴퓨터情報學會 論文誌
第15卷 第1號, 2010. 1.

2010-15-1-4-5

패턴매칭을 이용한 유사도 비교 분석

고 방 원*, 김 영 철**

A Similarity Valuating System using The Pattern Matching

Bang-Won Ko*, Young-Chul Kim**

요 약

본 논문에서는 서로 다른 두 개의 문서에 등장하는 패턴 매칭을 이용하여 유사도를 평가하는 시스템을 제안한다. 기존의 문서들의 유사도를 평가하는 방법에는 지문법과 같은 통계적 방법을 주로 이용하였다. 하지만 이 방법은 관련이 없는 두 문서에서 우연히 유사한 단어가 많이 등장 할 때 유사성이 높게 나오는 정확성의 문제점이 있다. 이러한 문제점은 단순히 두 문서의 통계적인 수치를 비교하기 때문에 발생한다. 하지만 본 논문에서 제시하는 패턴을 이용한 방법은 일치하는 패턴을 검색하여 유사성을 판별하기 때문에 이러한 문제를 해결하였다. 하지만 패턴을 검색하는 시간이 오래 걸리는 단점이 있는데 이를 개선하는 알고리즘 또한 본문에서 소개한다.

Abstract

This research suggests that valuate similarities by using the matches of patterns which is appeared on different two documents. Statistical ways such as fingerprint method are mainly used for evaluate similarities of existing documents. However, this method has a problem of accuracy for the high similarity which is occurred when many similar words are appeared from two irrelevant documents. These issues are caused by simple comparing of statistical parameters of two documents. But the method using patterns suggested on this research solved those problems because it judges similarity by searching same patterns. This method has a defect, however, that takes long time to search patterns, but this research introduce the algorithms complement this defect.

▸ Keyword : 유사도 평가(simiarity valuating), 패턴 매칭(pattern matching)

• 제1저자 : 고방원
• 투고일 : 2009. 05. 28, 심사일 : 2009. 06. 22, 게재확정일 : 2010. 01. 26.
* 숭실대학교 정보과학대학 컴퓨터학부 석박사통합과정 ** 유한대학 전자상거래과 교수
※ 본 논문은 2009년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임"(KRF-2007- 331-D00435)"

Ⅰ. 서 론

표절이란 단어가 익숙한 용어임에는 틀림없지만 우리나라 학술계에서 문제가 되기 시작한 것은 비교적 최근이라고 할 수 있다[1]. 이는 그동안 음악과 미술 등 예능계에 집중해있던 표절 문제가 학술계에도 만연해 있다는 것을 인식한 계기가 되었다. 표절은 가장 손쉽게 창작물을 만들어 낼 수 있는 방법이기 때문에 그 유혹은 항상 존재해왔으며, 또한 연구의 정직성과 창작성을 떨어뜨려 연구의 의욕을 떨어뜨리는 문제가 있다. 특히 인터넷이 발달한 요즘 표절과 도용의 문제가 더 자주 일어나고 있다. 특히 이러한 가운데 수많은 문서들을 사람이 확인해서 표절여부를 판단하는 것은 불가능하며, 검수자의 개인의 판단에 따라서 평가가 틀려지기 때문에 객관적이지 못한 문제가 발생할 수 있다. 따라서 컴퓨터를 이용해서 객관적 기준으로 표절 여부를 판단할 수 있는 시스템이 필요하다. 이미 외국에서는 표절의 심각성을 인지하고 이를 방지하기 위해 지난 20년간 다양한 학문적 논의와 연구가 진행 되었다[2,3]. 대표적인 표절 검사방법에는 크게 지문법[4]과 구조분석 방법[5]이 있다. 먼저 지문법은 구조적 형태가 없는 일반 텍스트 문서에서 단어들의 빈도와 위치 등의 통계적인 값에 가중치 주어 계산 하는 방법이다. 이와는 달리 구조분석 방법은 프로그래밍 언어와 같이 특정 구조를 지닌 형태의 문서 즉 프로그램의 유사성을 측정하는데 사용되었다. 이러한 특징 때문에 자연어로 구성된 텍스트 문서들을 비교할 때는 지문법을 이용해 왔다. 하지만 지문법을 이용하면 단지 통계적 수치를 이용하기 때문에 우연히 동일한 단어들이 많이 등장했을 때 유사도의 정확성이 떨어지는 문제점이 있다. 따라서 본 논문에서는 지문법 대신 구조적 분석 방법의 하나인 단어들의 패턴을 이용한 매칭 방법을 사용하여 유사도를 측정한다. 패턴 매칭 방법을 이용하면 우연히 동일한 단어가 많이 등장하더라도 패턴이 다르므로 유사성의 정확성을 보장할 수 있는 장점이 있다. 하지만 이러한 패턴을 찾기 위해서는 상당한 시간이 걸리는 단점이 발생하는데 이 시간을 단축하기 위한 개선된 알고리즘도 제시한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구에 대해 알아보고 3장에서는 시스템의 전반적인 구조와 개선된 알고리즘을 언급하였다. 그리고 4장에서는 실험 및 평가 부분을 기술하였으면 5장에서는 결론을 기술하였다.

Ⅱ. 관련 연구

표절과 관련하여 유사성을 검사하는 방법에는 크게 두 가지로 연구가 되었다. 첫 번째는 일반 자연어로 된 문서들의 유사도를 검사하기 위해서 통계적인 수치를 이용하는 지문법이 있다. 두 번째는 구조적인 특징을 가지고 있는 프로그램 소스의 유사도를 측정하기 위해서 사용되는 구조적 분석 방법이다.

우선 지문법은 단어나 키워드 혹은 특수문자, 공백, 토큰 등의 통계적 수치를 이용하여 유사도를 비교하는 방법이다. 지문법의 장점은 문서의 길이에 영향을 받지 않고 수행속도가 빠르다는 장점이 있다. 하지만 문서 전체의 전체 유사도만을 알 수 있다는 단점이 있다. 또한 연관성이 적은 두 문서에서 우연히 동일한 단어나 키워드가 나왔을 때 유사도가 높게 나오는 단점이 있다. 이러한 지문법을 이용하는 시스템에는 Plagiarism.org[5], EVE[6], Turnitin[7], CopyCatch Gold[8] 등이 있다. 이러한 지문법에서 사용되는 유사도 측정 방법에는 거리를 이용하는 방법과 상관계수를 사용하는 방법이 있다. 우선 거리를 이용한 방법은 문서에 있는 단어나 키워드, 공백 등의 특징들을 좌표로 나타내어 두 문서간의 좌표 거리를 계산하는 방법으로 거리가 가까울수록 두 문서는 유사성은 높은 것이고 거리가 멀수록 유사성이 낮은 것이다. 이러한 이유로 비유사성 척도(dissimilarity measure)라고 불리어 진다. 거리를 이용한 방법에는 유클리디언거리(Euclidean distance), 민코프스키거리(Minkowski distance), 맨하탄 거리(Manhattan distance)등이 있다. 두 번째 방법은 문서와 질의어를 벡터로 표현하고 여기에 가중치를 부여한 다음 상관계수를 이용하여 유사도를 측정하는 방법이다. 상관 계수에는 코사인, 자카드, 피어슨 상관 계수 등이 있다.

이에 반해 구조적 분석 방법은 문서의 구조적 특징을 비교 분석함으로서 유사도를 평가하는 방법이다. 구조적으로 분석하기 때문에 부분유사성을 판별할 수 있는 장점이 있지만 모든 구조를 분석해야 하기 때문에 시간이 오래 걸리는 단점이 있다. 초기에는 프로그램 소스를 단순 텍스로 하여 비교하였다. 이러한 시스템 중 Ringer는 전체 라인들을 각각 비교하였는데 수많은 소스 라인들을 좀 더 효율적으로 찾기 위하여 해시를 이용하였다[9]. 또한 Johnon은 지문법에 기반 한 스트링 매칭 방법을 효율적으로 사용하였다[10, 11]. 이러한 초기 방법들은 단순히 변수명만 변경이 되어도 찾기 힘든 어려움이 있었다. 두 번째 방법은 토큰을 비교 하는 방법이 있다. Baker는 단순 스트링 비교 대신에 라인단위로 비교하는 방법을 제시하였다. 라인 마다 토큰들의 순서를 트리에 부가적으로 추가하여 비교용하였다. 이 방법에서 상수나, 변수 명의 실제 값들은 functor이라 불리는 함수에 의해 추가되었다[12]. 세 번째 방법은 메트릭을 비교 방법이다. 이 방법은 직접 코드를 비교하는 대신에 코드 조각들을 위한 다른 메트릭 벡터를 비교하는 방법이다[13]. 이 메트릭 벡터들을 위한 거리(예, 유클리디언 거리)는 유사코드를 찾는 힌트가 될 수 있다. 네 번째 방법은 AST(abstract syntax tree)를 비교 하는 방법이다.

Baxter는 프로그램의 AST의 서브트리를 해시를 이용하여 서브트리 간 비교를 하였다[14]. 이와 비슷한 방법은 이전에 Yang[15]이 같은 파일의 두 가지 버전사이에서 다른 점을 찾기 위한 동적 프로그래밍을 이용하여 제안했었다. 또한 Koschke 는 AST 노드들을 프리오더 방법으로 순회하면서 순차적인 구조를 갖는 추가 트리를 생성하고 이렇게 생성된 트리를 이용하여 비교하는 방법을 사용하였다[16]. 이러한 구조적 분석 방법을 이용한 시스템에는 YAP[17], MOSS[18], JPlag[19] 등이 있다.

보통 지문법은 일반 자연어로 구성된 문서들의 유사도를 평가하는데 사용되고 구조적 분석 방법은 구조적 특징을 지니는 인공언어, 즉 프로그램 간의 유사도를 평가하는데 주로 사용된다.

III. 패턴 매칭을 이용한 유사도 평가

본 논문에서 제안하는 유사도 시스템의 구조는 그림 1과 같이 크게 3가지 과정으로 이루어져 있다. 우선 문서에서 색인어로 사용될 수 있는 명사들만을 추출 하는 어휘 분석과정이 있다. 분석 과정이 끝난 문서 A와 B는 명사들만을 가지는 순차구조의 리스트를 가지게 된다. 이러한 명사들을 색인으로 변환하기 위한 색인 과정을 거치는데 이때 비교를 위해서 공통색인 테이블이 필요하다. 그리고 마지막으로 변환된 색인의 연속적인 집합인 시퀀스(sequence)를 이용하여 본 논문에서 제시하는 유사도 평가 알고리즘을 이용하여 유사도를 평가한다.

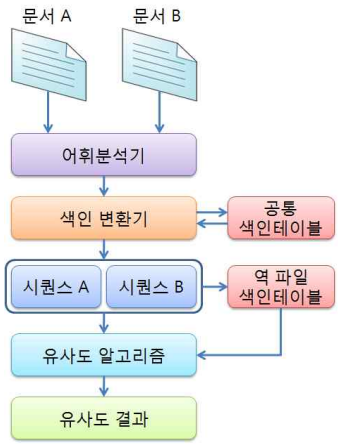


그림 1. 유사도 평가 시스템의 구조
Fig.1. Architecture of Similarity Evaluation System

1. 어휘 분석기

어휘 분석기에서는 명사만을 추출하고 불필요한 단어들은 제거하는 역할을 한다. 우선 색인의 후보군이 되는 명사만을 추출하기 위해서 자동 색인 방법을 사용하였다. 자동 색인은 “색인어로 선택될 가능성이 있는 모든 용어를 추출하여 후보 색인어를 생성하는 과정”과 “후보 색인어로부터 불용어를 제거하고 색인어를 선별하는 과정” 으로 이루어진다[20]. 즉 모든 단어가 색인어로 선택되는 것이 아니고 조사, 형용사, 동사 등의 용어들은 불용어로 간주하고 색인어 후보에서 제외시키는 것이다. 이렇게 명사들만을 색인어로 사용하는 이유는 실제로 대부분의 의미전달은 명사를 통해 전달되기 때문이다. 또한 이렇게 명사만을 색인어로 사용했을 경우 총 단어수가 40~50% 제거되어 색인어 수를 줄일 수 있다[21].

2. 색인 변환기

어휘 분석 과정이 끝나면 두 문서의 비교를 위해서 추출된 명사들을 색인으로 변환하는 과정이 필요하다. 이때 두 문서 간에 공통적인 단어에 대해 같은 색인이 필요한데 이를 위해서 색인 테이블을 구성한다. 색인 테이블은 해시테이블로 구성되어 있으며 구성하는 방법은 알고리즘 1과 같다.

표 1. 공통 테이블 구성 알고리즘
Table 1. Common Table Algorithm

```
HashMap<String, int>
MakeCommonTable(StringArray A)
{
        HashMap<String, int> map = new
HashMap<String, int>();
        int index = 0;
        int lengthA = A.size;
        for(int i   = 0; i < lengthA; i++) {
                key = A[i];
                if(!map.find(key)) {
                        map.set(key,
index);
                        index++;
                }
        }
        return map;
}
```

색인 테이블 구성이 끝나면 이를 이용하여 명사들이 순차적으로 나열된 문서 A, B를 색인의 나열인 시퀀스 A, B로 변환하여 준다. 변환하는 과정에서 역파일 색인 테이블을 만들

어주는데 이는 유사도 알고리즘에서 비교횟수를 줄이기 위한 것이다. 역파일 색인 테이블 역시 해시 테이블로 구성되어 있으며 키 값은 인덱스 번호이고 값은 시퀀스에서 실제 인덱스가 어디서 등장하는지 위치를 가지고 있는 배열이다. 역 파일 색인테이블 구성 방법은 표 2와 같다.

표 2에 positionList는 위치정보를 가지고 있는 리스트이다. 우선 find 함수로 키값이 테이블에 있는지 찾고 있으면 킷값 즉 위치정보를 가지고 있는 리스트에 현재 위치 i값을 추가해 주고 없으면 위치정보 리스트를 새로 만들어 위치 값을 할당한다. 색인 변환기와 테이블의 전체 모습은 그림 2와 같다.

표 2. 역파일 색인 테이블 구성 알고리즘
Table 2. Reverse Index Table Algorithm

```
HashMap<int, ArrayList>
MakeInvertedFileTable(ArrayList<int>
sequence)
{
    HashMap<int, ArrayList> map =
                              new
HashMap<int,ArrayList>();
    int lengthA = indexArray.size;
    for(int i          = 0; i < lengthA; i++) {
        key = A[i];
        if(map.find(key)) {
            ArrayList<int>
            positionList = map.get(key);
            positionList .add(i);
            map.set(key, positionList);
        }
        else {
            ArrayList<int> positionList =
                              new
ArrayList<int>();
          positionList .add(i);
          map.set(key, positionList);
        }
    }
    return map;
}
```

그림 2에서 공통 색인 테이블의 구성과 색인 변환 과정이 끝난 후 역파일 색인 테이블이 만들어 짐을 알 수 있다. 이때 역파일 색인 테이블의 0, 1, 2 값은 공통 색인 테이블의 값인 0, 1, 2와 같다. 즉 색인 값을 역파일 색인 테이블에서는 키로 사용한다. 또한 0, 32, 57은 시퀀스에서 0 색인이 등장하는 위치이다.

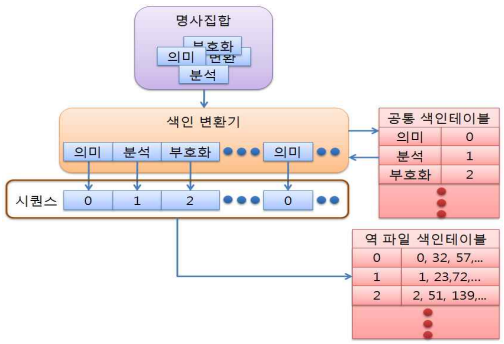


그림 2. 테이블을 이용한 시퀀스 생성과정
Fig.2. Sequence Creation Process using Table

3. 유사도 알고리즘

본 논문에서 제시하는 유사도 알고리즘은 기존의 [22]에서 제시했던 유사도 측정 알고리즘을 개선한 방법이다. [22]에서 제시했던 알고리즘에는 두 가지 문제점이 있는데 첫 번째는 매칭알고리즘의 수행시간이다.

표 3. 개선된 시퀀스 찾기 알고리즘
Table 3. New Sequence Search Algorithm

```
String MatchSequence(Sequence A, Sequence B)
{
      int i, j;
      int matchSize, maxMatchSize;
      Match longSubSequence;
      for each unmark(A_i) in A {
            search(B_j) in B {
            matchSize = 0;
            while((unmark(A_{i+matchsize}) &&
            unmark(B_{j+matchsize}))

                  matchSize++;

            if(matchSize > maxMatchSize) {

                        longSubSequence =
match(A_i, B_j, matchSize);

                        maxMatchSize = matchSize;
            }
      }
      MarkSubSequence(A, B, i, j, matchSize);
      return longSubSequence;
}
```

[22]에서는 일치하는 패턴을 찾기 위해서 N x N의 매우 큰 시간적 비용을 들여야만 했다. 특히 이러한 방법은 문서의 크기가 커지면 커질수록 그 비용은 기하급수적으로 늘어나게 되는 단점을 가지고 있다. 두 번째 문제는 두 문서의 크기가 달

라졌을 때 유사도가 0에 가깝게 나오는 단점이 있다. 우선 수행시간을 개선한 알고리즘은 표 3과 같다.

알고리즘 3에서 matchSize가 의미하는 것은 두 개의 시퀀스 A, B에서 일치하는 색인의 개수이다. maxMatchSize는 이전에 검색된 가장 긴 서브 시퀀스의 개수를 의미한다. 따라서 matchSize가 maxMatchSize보다 클 경우 이 둘을 교체해야한다. longSubSequence는 가장 긴 서브시퀀스의 정보를 가지고 있는 객체이다. 이 객체는 각각의 시퀀스 A, B의 몇 번째 위치부터 서로 일치하는 서브시퀀스를 가지고 있는지를 표시한다. 그리고 unmark(A_i) 함수는 현제 시퀀스 A에서 I번째에 해당하는 색인이 이전에 서브시퀀스로 마킹이 되었는지를 검사하는 함수이다. search 함수는 이전에 만들었던 역 파일 색인테이블을 이용하여 비교횟수를 줄여주는 함수이며 동작 과정은 그림3과 같다

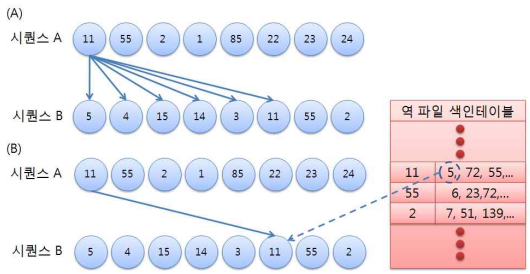


그림 3. 역파일 색인을 이용한 색인 위치 찾기
Fig.3. Index Position Search using Reverse Index

그림 3의 A는 기본적으로 시퀀스 A와 B를 모두 비교하면서 일치하는 패턴을 찾는다. 이렇게 부분 패턴을 찾는 알고리즘을 적용할 경우 $O(N^2)$의 시간 복잡도를 가지게 된다. 이는 상당한 시간이 소요되므로 매우 비효율적이다. 하지만 본 논문에서 제시하는 방법은 역파일 색인테이블에 색인의 위치가 저장되어 있으므로 한번에 찾을 수 있는 장점이 있다. 그림 3의 B에서 색인 11은 시퀀스 B의 다섯 번째에 위치하므로 다섯 번째부터 비교한다. 이 방법을 이용할 경우 모두 비교를 하지 않고 $O(N+M)$ 이므로 $O(N)$의 시간 복잡도를 갖는다. 그리고 마지막으로 가장 긴 서브시퀀스를 MarkSubSequence 함수를 통해서 각각의 시퀀스 A, B에 표시해준다.

두 번째 문제점인 두 문서간의 크기가 달라졌을 때의 문제점이다. [22]에서 제시 했던 유사도 측정 알고리즘은 두 문서의 크기 차이가 많이 날수록 유사도 값이 0에 가까워지는 단점이 있다. 이러한 단점을 보완한 알고리즘은 표 4와 같다.

표 4. 개선된 유사도 알고리즘
Table 4. New Similarity Algorithm

```
double sim(Sequence A, Sequence B, int
minMatchLength) {
  int totalMatchSize;
  double persentA, persentB;
  Match longestMatch;
  ArrayList<Match> totalMatch = new
ArrayList<Match>();
  do {
    longestMatch = MatchSequence(A, B);
    if(longestMatch.matchSize >=
minMatchLength)
      totalMatch.add(longestMatch);
  } while(longestMatch.matchSize >=
minMatchLength);
  while(totalMatch hasNext) {
    longestMatch = totalMatch.next;
    totalMatchSize+ = Length(longestMatch);
  }
  persentA = totalMatchSize / Length(A);
  persentB = totalMatchSize / Length(B);
  if( | persentA - persentB | < 0.1)
    return (2 * totalMatchSize / (Length(A) + Length(B)));
  else {
    if(persentA > persentB)
      return persentA;
    else
      return persentB;
  }
}
```

표 4에서 minMatchLength는 최소 연속적으로 일치해야하는 색인의 개수이다. 즉 minMatchLength가 6으로 설정되면 연속적으로 6개가 일치하는 색인이 나와야 서브시퀀스로 인정한다는 뜻이다. MatchSequence에서 가장 긴 시퀀스를 리턴하는데 이 값이 minMatchLength보다 크거나 같을 때 까지만 반복하여 서브시퀀스를 찾는다. totalMatch는 MatchSequence에서 찾은 모든 서브시퀀스들을 가지고 있는 변수이다. 그리고 totalMatchSize는 모든 서브시퀀스들의 개수를 의미하며 이는 두 시퀀스에서 일치하는 색인의 개수이다. 마지막으로 persentA와 persentB는 각각 A, B 문서에 대한 유사도를 나타낸다. 즉 문서 A와 B의 각각에 대한 유사도를 나타내며 두 유사도의 값이 0.1 미만의 오차일 경우 식 1과 같이 다이스 상관 계수를 이용하여 유사도 값을 계산한다. 여기서 0.1은 평균 절대 오차(mean absolute e1과o과)값으로 예상하는 값과 실제 측정값의 오차를 나타낸 것이다. 0.1을 넘어 갔을 경우는 유사도 값이 높은 것을 우선으로 하였다. 이는 다이스 상관 계수의 단점을 보완한 것으로 다이스 상관계수는 두 문서의 크기가 다를 때 0에 가까운 값이 나오게 되는 단점을 가지고 있다. 하지만 본 논문에서 제시하는 알고리즘은 이러한 단점을 보완하여 두 문서의 크기와 상관없이 유사도 값을 계산한다.

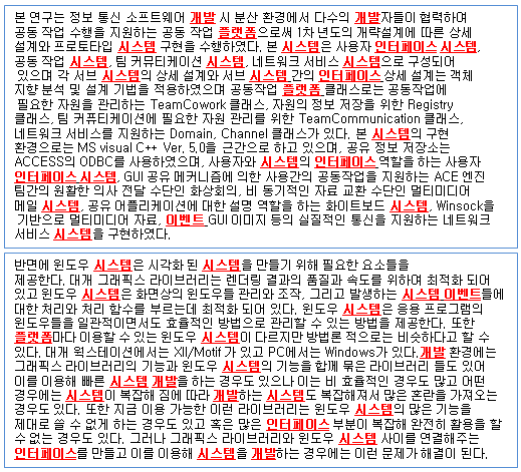
Ⅳ. 실험 및 평가

본 시스템은 Java로 구현 되어 있어 Window나 UNIX 계열의 운영체제에서 모두 사용할 수 있다. 실험 대상으로 사용된 문서들은 일반 자연어로 구성된 문서들이며 크게 두 가지 방법으로 실험을 한다. 첫 번째는 기존의 유사도 측정방법과의 비교이며, 두 번째는 [22]에서 제시한 유사도 알고리즘과 본 논문에서 제시한 개선된 알고리즘과의 속도 비교이다.

1. 우연히 동일한 단어가 많이 등장했을 때의 유사도

컴퓨터 분야에서 자주 쓰이는 용어들이 있는데, "시스템", "분석", "설계" 등의 단어들은 매우 흔하게 사용하는 용어들이다. 이러한 용어들은 컴퓨터 관련 기사나 논문에서 빈번하게 발생한다. 즉 단어의 빈도수를 주로 사용하는 지문법에서는 서로 다른 문서라도 문서임에도 불구하고 유사도 값이 높게나오는 단점이 있다. 그림 4는 전혀 다른 두 논문에서 우연히 일치하는 단어가 많이 등장하는 경우의 예를 보여준다.

그림 4에서 밑줄 친 부분이 중복되는 단어들이며, 이러한 단어들은 컴퓨터 분야에서 흔하게 사용될 수 있다. 그림 4와 같은 경우는 매우 흔하게 일어나는데, 이러한 경우에 유사도가 높게 나오는 것은 정확성의 문제가 있다. 따라서 본 실험에서는 그림 4와 같은 경우의 유사도 측정 실험을 하였다.


본 연구는 정보 통신 소프트웨어 **개발** 시 분산 환경에서 다수의 **개발**자들이 협력하여 공동 작업 수행을 지원하는 공동 작업 **플랫폼**으로써 1차 년도의 개략설계에 따른 상세 설계와 프로토타입 **시스템** 구현을 수행하였다. 본 **시스템**은 사용자 **인터페이스 시스템**, 공동 작업 **시스템**, 팀 커뮤티케이션 **시스템**, 네트워크 서비스 **시스템**으로 구성되어 있으며 각 서브 **시스템**의 상세 설계와 서브 **시스템**간의 **인터페이스** 상세 설계는 객체 지향 분석 및 설계 기법을 적용하였으며 공동작업 **플랫폼**클래스로는 공동작업에 필요한 자원을 관리하는 TeamCowork 클래스, 자원의 정보 저장을 위한 Registry 클래스, 팀 커뮤티케이션에 필요한 자원 관리를 위한 TeamCommunication 클래스, 네트워크 서비스를 지원하는 Domain, Channel 클래스가 있다. 본 **시스템**의 구현 환경으로는 MS visual C++ Ver. 5.0을 근간으로 하고 있으며, 공유 정보 저장소는 ACCESS의 ODBC를 사용하였으며, 사용자와 **시스템**의 **인터페이스** 역할을 하는 사용자 **인터페이스 시스템**, GUI 공유 메커니즘에 의한 사용간의 공동작업을 지원하는 ACE 엔진 팀간의 원활한 의사 전달 수단인 화상회의, 비 동기적인 자료 교환 수단인 멀티미디어 메일 **시스템**, 공유 어플리케이션에 대한 설명 역할을 하는 화이트보드 **시스템**, Winsock를 기반으로 멀티미디어 자료, **이벤트** GUI 이미지 등의 실질적인 통신을 지원하는 네트워크 서비스 **시스템**을 구현하였다.

반면에 윈도우 **시스템**은 시각화 된 **시스템**을 만들기 위해 필요한 요소들을 제공한다. 대개 그래픽스 라이브러리는 렌더링 결과의 품질과 속도를 위하여 최적화 되어 있고 윈도우 **시스템**은 화면상의 윈도우를 관리와 조작, 그리고 발생하는 **시스템 이벤트**들에 대한 처리와 처리 함수를 부르는데 최적화 되어 있다. 윈도우 **시스템**은 응용 프로그램의 윈도우들을 일관적이면서도 효율적인 방법으로 관리할 수 있는 방법을 제공한다. 또한 **플랫폼**마다 이용할 수 있는 윈도우 **시스템**이 다르지만 방법론 적으로는 비슷하다고 할 수 있다. 대개 워스테이션에서는 X11/Motif 가 있고 PC에서는 Windows가 있다. **개발** 환경에는 그래픽스 라이브러리의 기능과 윈도우 **시스템**의 기능을 합쳐 묶은 라이브러리 들도 있어 이를 이용해 빠른 **시스템 개발**을 하는 경우도 있으나 이는 비 효율적인 경우도 많고 어떤 경우에는 **시스템**이 복잡해 짐에 따라 **개발**하는 **시스템**도 복잡해져서 많은 혼란을 가져오는 경우도 있다. 또한 지금 이용 가능한 이런 라이브러리는 윈도우 **시스템**의 많은 기능을 제대로 쓸 수 없게 하는 경우도 있고 혹은 많은 **인터페이스** 부분이 복잡해 완전히 활용을 할 수 없는 경우도 있다. 그러나 그래픽스 라이브러리와 윈도우 **시스템** 사이를 연결해주는 **인터페이스**를 만들고 이를 이용해 **시스템**을 **개발**하는 경우에는 이런 문제가 해결이 된다.

그림 4. 동일한 단어가 많은 경우
Fig.4. Case of Same Words

표 5. 동일한 단어가 많이 등장했을 때의 유사도 결과
Table 5. Similarity Result of Same Words Case

유사도 알고리즘	코사인	유클리드	제안방법
결과 값	0.4339	0.468	0.0

표 5의 결과를 보면 코사인과 유클리드 알고리즘은 각각 약 0.43, 0.46을 결과 값을 가지고 제안 방법은 0의 유사도가 나타났다. 기존의 유사도 알고리즘은 서로 다른 문서에 대한 판단을 할 수 없는 문제점이 나타났다. 이러한 결과가 나타난 것은 기존의 유사도 알고리즘은 모두 통계적인 수치를 이용하기 때문이다. 하지만 본 시스템에서 제안하는 방법은 패턴 정보를 이용하기 때문에 그림 5와 같이 단순히 동일한 단어의 우연한 등장에 따른 유사도 검출 오류를 회피할 수 있었다.

2. 역파일 색인을 사용했을 경우의 수행시간 비교

본 실험에서는 두 문서 간 크기가 다를 경우의 유사도를 측정한 값과 수행시간을 비교한다. 문서의 크기는 1KB씩 증가하도록 데이터를 구성하였고 수행시간을 좀 더 정확하게 측정하기 위해서 100회 수행하여 평균값을 기록하였으며 수행시간의 단위는 밀리세컨드이다.

표 6과 7의 결과를 보면 수행시간이 월등하게 줄어든 것을 알 수 있다. 특히 문서의 크기가 커졌을 기존의 알고리즘은 문서의 크기와 수행시간이 비례하는 것을 알 수 있다. 본 논문에서 제시한 개선된 알고리즘 또한 수행시간이 문서 크기에 비례함을 알 수 있지만 그 폭이 기존의 알고리즘보다 매우 적음을 알 수 있다. 이는 불필요한 비교 횟수를 줄임으로써 수행시간을 단축했기 때문이다. 또한 문서의 크기가 커지면 커질수록 수행시간의 차이가 매우 커졌는데 최고 약 50배 정도의 수행시간 차이를 보인 결과도 있었다.

표 6. 기존 알고리즘의 수행시간
Table 6. Execution Time of Traditional Algorithm

	1KB	2KB	3KB	4KB	5KB	6KB	7KB	8KB	9KB	10KB
1KB	0.16	0.94	1.15	1.56	2.18	2.34	2.81	2.97	3.58	4.05
2KB		0.16	2.18	2.65	3.74	4.06	4.68	5.93	5.99	6.70
3KB			0.47	4.99	5.46	5.77	6.55	7.49	9.67	9.83
4KB				0.31	7.95	8.12	9.67	10.30	11.70	13.42
5KB					0.47	10.92	12.48	14.36	16.22	18.87
6KB						0.62	14.03	15.13	18.57	19.81
7KB							0.63	19.19	21.84	25.27
8KB								0.62	22.62	26.05
9KB									0.94	29.64
10KB										1.25

표 7. 개선된 알고리즘의 수행시간
Table 7. Execution Time of Improved Algorithm

	1KB	2KB	3KB	4KB	5KB	6KB	7KB	8KB	9KB	10KB
1KB	0	0	0.15	0.15	0.16	0.16	0.16	0.15	0.16	0.16
2KB		0	0.16	0.15	0.16	0.15	0.16	0.15	0.16	0.16
3KB			0	0.15	0.16	0.15	0.15	0.17	0.17	0.17
4KB				0.16	0.17	0.17	0.15	0.17	0.18	0.18
5KB					0.16	0.63	0.63	0.63	0.64	0.76
6KB						0.16	0.47	0.47	0.48	0.48
7KB							0.15	0.47	0.47	0.48
8KB								0.16	0.63	0.63
9KB									0.16	0.65
10KB										0.16

V. 결론

일반적으로 프로그래밍 언어와는 달리 자연어로 이루어진 문서들은 통계적 분석 방법을 이용한 지문법을 사용하여 유사도를 측정해 왔다. 지문법은 통계적인 수치만을 이용해 빠르게 유사도를 측정할 수 있는 장점이 있지만 통계적인 수치를 이용함으로써의 서로 관련이 없는 문서에서 단순히 같은 단어가 많이 나오는 경우에도 유사도가 높게 나오는 정확성의 문제가 있다. 본 논문에서 제시한 알고리즘은 패턴을 이용한 방법으로 우연히 동일한 단어가 등장하더라도 연속적으로 색인의 패턴이 일치하지 않으면 동일한 것으로 보지 않기 때문에 정확성을 높였다. 하지만 패턴을 이용한 유사도 측정방법은 수해시간이 매우 큰 단점이 있는데 이를 해결하기 위해서 역파일 색인테이블을 이용한 방법을 제시하였고 실험을 통해서 이를 입증하였다. 본 논문에서 제시한 방법은 기존의 유사도 측정방법에 비해 정확성과 수행속도를 향상시켰기 때문에 Stand-alone 컴퓨터에서 뿐만 아니라 웹에서도 활용 가능할 것으로 기대된다.

참고문헌

[1] 김수영, "*표절과 올바른 인용 방법*," 가정의학회지, 167-174쪽, 2008년

[2] P. J. Larkham, & Manns, "S. Plagiarism and its treatment in higher education," Journal of Further and Higher Education, 26(4), pp.339-349. 2002.

[3] D. L. McCabe, L. K. Trevino, & K. D. Butterfield, "Cheating in academic institutions: A decade of research," Ethics & Behavior, 11(3), pp.219-232. 2001.

[4] J. H. Jonson, "Identifying Redundancy in Source Code using Fingerprints," In proc. of CASCON 93, pp.171-183, 1993.

[5] *http://www.plagiarism.org/learning_center/what_is_plagiarism.html*

[6] *http://www.canexus.com/eve/index.shtml*

[7] *http://www.turnitin.com/*

[8] *http://www.copycatch.freeserve.co.uk/*

[9] S. Ducasse, M. Rieger, S. Demeyer, "A Language Independent Approach for Detecting Duplicated Code,"

International Conference on Software Maintenance. pp.109-118, 1999.

[10] J. H. Johnson, "Identifying redundancy in source code using fingerprints," Conference of the Centre for Advanced Studies on Collaborative research, IBM Press pp.171-183, 1993.

[11] J. H. Johnson, "Substring matching for clone detection and change tracking," International Conference on Software Maintenance, IEEE Computer Society Press pp.120-126, 1994.

[12] B. S. Baker, "On finding duplication and near-duplication in large software systems," Second Working Conference on Reverse Engineering, Los Alamitos, California, IEEE Computer Society Press pp. 86-95, 1995.

[13] K. A. Kontogiannis, R. Demori, E. Merlo, M. Galler, M. Bernstein," Pattern matching for clone and concept detection.", Automated Software Engineering Vol. 3, No. 1/2, pp.79-108, 1996.

[14] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, L. Bier, "Clone Detection Using Abstract Syntax Trees," International Conference on Software Maintenance, IEEE Computer Society Press pp.368-378, 1998.

[15] W. Yang, "Identifying syntactic differences between two programs," Software-Practice and Experience Vol. 21, No. 7, pp.739-755, 1991.

[16] R. Koschke, R. Falke, P. Frenzel, "Clone detection using abstract syntax suffix trees," Working Conference on Reverse Engineering, IEEE Computer Society Press, 2006.

[17] M. J. Wise, "Detection of Similarities in Student Programs: YAP'ing may be Preferable to Plague'ing," ACM SIGSCE Bulletin, In Proc. of 23rd SIGCSE Technical Symp., Vol. 24, No. 1, pp.268-271, March 1992.

[18] A. Aiken, "MOSS(Measure Of Software Similarity) Plagiarism detection system," Available at *http://www.cs.berkeley.edu/~moss/*, University of Berkeley, CA, Apr. 2000.

[19] L. Prechelt, G. Malpohl & M. Philppsen, "JPlag: Finding Plagiarism Among a Set of Programs," available at *http://www.ipd.ira.uka.de/EIR/* D-76128 Karlsruhe, Germany, Technical Report 2000-1, March 2000.

[20] 강승식, 권혁일, 김동렬, "한국어 자동 색인을 위한 형태소 분석 기능," 한국정보과학회, 학술발표논문집 제22권 제1호, 929-932쪽, 1995년 4월

[21] J. H. Jonson, "Identifying Redundancy in Source Code using Fingerprints," In Proc. of CASCON 93, pp.171-183, 1993.

[22] Y. C. Kim, S. K. Kim, S. H. Yeom, J. M. Choi &C. W. Yoo. "A Program-Plagiarism Checker using Abstract Syntax Tree," KISS(Korea Information Science Society), Vol. 30, No. 8, Aug. 2003.

저 자 소 개

고 방 원

2005 : 동아대학교 컴퓨터학부 공학사.
현 재 : 한국대학교 컴퓨터학부 석박사통합과정
관심분야 : 프로그래밍 언어, 컴파일러, XML, HCI

김 영 철

1990 : 한남대학교 전자계산학과 공학사.
1998 : 숭실대학교 컴퓨터공학과 공학석사.
2003 : 숭실대학교 전자계산학과 공학박사
현 재 : 유한대학교 전자상거래과 조교수
관심분야 : 프로그래밍 언어망관리, 컴파일러. XML, 컴퓨터 통신