프로그램 유사도 평가를 이용한 유사 프로그램의 그룹 짓기

(Grouping of Similar Programs using Program Similarity Evaluation)

유 재 우 † 김 영 철 ††
(Chae-Woo Yoo) (Young-Chul Kim)

요 약 프로그램 과제물과 같은 많은 프로그램을 모두 일일이 비교하는 것은 비용이 많이 든다. 더군다나 검수자가 과제물을 검사한다든가, 점수를 부여하고자 한다면 더욱 많은 시간이 요구된다. 물론 검수자가 많은 시간을 두고 평가해도 객관성이 떨어질 수도 있다. 이러한 문제점은 프로그램 과제물에 대해서 유사한 프로그램으로 서로 묶어 놓는다면 쉽게 해결할 수 있다. 즉, 유사한 프로그램으로 서로 묶어놓고 검사한다면 쉽게 검사나 평가가 가능하다. 본 논문에서는 많은 프로그램에 대해서 유사성이 높은 프로그램으로 그룹 짓기(grouping)를 수행하는 알고리즘을 제시하고 구현한다. 그룹 짓기 알고리즘은 [9]에서 제시한 프로그램 유사도 평가 알고리즘을 이용하여 유사도를 측정한 후, 유사성이 높은 프로그램을 그룹 짓기를 수행한다. 이 그룹 짓기 알고리즘을 이용하면 n개의 프로그램에 대해서 최대 n(n-1)/2 번에서 최소 (n-1)번까지 비교 횟수를 줄일 수가 있다. 본 논문의 실험 및 평가 부분에서는 실제로 모 대학의 과제물 10개를 추출하여 유사성을 기준으로 실험 평가한 결과를 보여준다.

키워드 : 프로그램 유사도, 그룹 짓기, 프로그램 평가

Abstract Comparing many programs like programming assignments one by one requires many costs. Moreover, if the checker would evaluate or grade assignments, much more time will be required. Even through the checker invest much time, fairness is not always guaranteed. These problems can be solved easily by grouping similar programs. So, programs after grouping can be easily evaluated and graded.

In this paper, we propose and implement algorithm performing grouping by similarity on many programs. The grouping algorithm evaluates similarity using algorithm proposed in [9], and performs a grouping following high similarity order. By using this grouping algorithm, the number of comparison among N programs can be reduced from N-1 times to N(N-1)/2 times. In the part of experiment and evaluation of this paper, we actually showed evaluation result by similarity using randomly 10 programming assignments at the university.

Key words : program similarity, grouping, program evaluation

1. 서 론

학생들이 제출한 프로그램 과제물의 복제 여부를 판단하는 일은 결코 쉬운 일이 아니다. 특히, 인터넷이 활성화되면서 학생들이 많은 참고문헌을 토대로 프로그램 과제물을 제출하기 때문에 제출한 과제물이 유사한 경우도 상당히 많다. 이러한 사유로 제출된 과제물을 정확히 비교, 평가하기란 상당히 어려운 일이며, 검수자가 똑같은 프로그램을 같은 기준으로 평가한다는 것도 사실 어렵다[1-3]. 학생들이 제출한 과제물에 대해서 복제 여부를 검사하는 것 또한 어렵다. 이는 모든 다른 과제물에 대한 비교 평가를 해야하기 때문이다. 따라서 과제물의 양이 많아지면 비교 평가에 소요되는 시간은 기하급수적으로 늘어난다. 또한 전체 복제가 아니라 특정한 패턴이나 일부 복제의 경우는 과제물 복제 여부를 알아내기란 사실상 불가능하다. 더군다나 표절이나 특정 패턴의 과제물을 알아내는 일은 더더욱 어렵고 많은 시간

· 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음
† 종신회원 : 컴퓨터학부 교수
cwyoo@comp.ssu.ac.kr
†† 비 회 원 : 컴퓨터학과
yckim@amin.ssu.ac.kr
논문접수 : 2003년 9월 22일
심사완료 : 2003년 12월 17일

을 요구한다[4,5].

이러한 문제점은 학생들이 제출한 프로그램 과제물을 유사한 그룹으로 묶어놓음으로써 해결할 수 있다. 즉, 프로그램 간에 유사성이 높은 집단으로 그룹을 지어 묶어놓으면 쉽게 복제 여부와 유사성, 점수 산정을 할 수 있다. 이처럼 프로그램 간에 유사성이 높은 그룹을 묶는 것을 그룹 짓기(grouping)이라고 한다[6-8]. 그룹 짓기는 유사한 집단의 프로그램이나 데이타를 분류해서 하나의 그룹을 만드는 것으로 동일한 그룹에 있는 데이타들은 서로 유사한 집단으로 분류되며, 서로 다른 그룹에 속한 데이타들은 상이하다는 특징을 가지고 있다.

본 시스템에서 제시한 그룹 짓기 알고리즘의 전체 시스템 모델은 다음 그림 1과 같다.

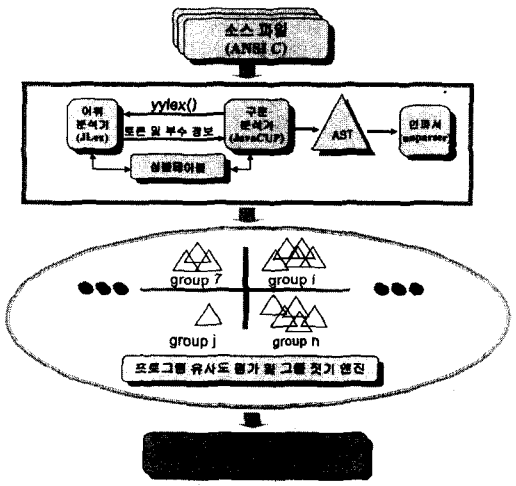

그림 1 그룹 짓기 알고리즘의 전체 시스템 모델

그림 1에서 입력된 프로그램들은 순서대로 어휘분석과 구문분석을 거친 후 구문트리(AST)를 생성한다. 생성된 구문트리는 언파서에 의해서 노드 스트링으로 변환되어 프로그램 유사도 평가 엔진에 의해서 다른 프로그램과 유사도를 평가한다. 본 논문에서 이용된 프로그램 유사도 평가 알고리즘은 [9]에서 제시한 프로그램 유사도 평가 알고리즘을 이용하였다. 유사도를 평가한 후, 그룹 짓기를 수행하며, 특정 그룹과 유사성이 높으면 해당 프로그램을 그룹에 포함시키며, 높지 않다면 다른 그룹의 프로그램과 비교한다. 만일 비교하고자 하는 프로그램이 모든 그룹과 유사성이 높지 않다면 다른 그룹을 생성한다. 이 과정을 모두 거치면 최종적으로 유사도 그룹 행렬을 마이크로소프트 엑셀(excel) 파일 형태로 출력함으로써 그룹 짓기가 끝나게 된다.

본 논문은 다음과 같이 구성되었다. 제2장에서는 관련 연구에 대해서 기술하였으며, 제3장에서는 프로그램 유사도 평가 알고리즘에 대해서 기술하였으며, 제4장에서는 그룹 짓기 알고리즘에 대해서 설명하였다. 또한 제 5장에서는 실험 및 평가를 기술하였다. 마지막으로 제6장에서는 결론을 기술하였다.

2. 관련 연구

그룹 짓기와 관련된 개념은 데이타 마이닝, 문헌 검색, 생명공학, 데이타베이스, 이미지 처리, 병렬처리 등의 많은 분야에서 군집화(clustering)라는 개념으로 연구되어 왔다[10-12]. 이러한 분야들은 형태는 다양하지만 개념적으로 모두 유사하다. 일반적으로 군집화 개념은 인공지능 분야의 데이타 마이닝 기법에서 비롯되었다. 이 분야에서 연구되고 있는 군집화 알고리즘은 크게 4가지 즉, 분할 알고리즘(Patitioning Algorithm), 계층적 알고리즘(Hierarchical Algorithm), 결합 기반 알고리즘(Density-based Algorithm), 모델 기반 알고리즘(Model-based Algorithm) 형태로 나누어 볼 수 있다[10]. 분할 알고리즘 방법은 n개의 개체를 k개의 군집으로 분할하여 그룹으로 만드는 알고리즘으로 K-means와 k-medoids 알고리즘이 이에 속한다[11]. k-means 알고리즘은 '거리(distance)'라는 개념을 이용하여 가깝게 위치한 점들을 찾아 군집으로 묶어주는 기법으로 차원의 제약이 전혀 없다. k-means에서 K는 이 사전에 정의된 군집의 수를 의미한다. 계층적 알고리즘은 군집간 연결방법에 따라 연결법(linkage method) 군집, 워드 군집(Ward's method) 및 분리적 방법인 DIANA 군집 방법이 있다. 계층적 알고리즘의 개략적인 의사코드는 다음과 같다.

단계 1. 각 개체를 하나의 군집으로 간주하며, k = n을 초기화한다.

단계 2. 현재의 군집 결과에 있는 모든 군집 간의 쌍에 대해여 군집간 유사성 척도인 $D(C_i, C_j)$, $1 \le i \ne j \le k$를 산출한다. 이 중 최소가 되는 군집 i와 군집 j를 묶어 하나의 군집으로 만든 후 군집 결과를 수정한다. 또한 k = k-1를 수행한다.

단계 3. k=2일 때까지 단계 2를 반복하며, k=1일 때 멈춘다.

여기서 C_i는 군집 i의 개체 집합을 말하며, D(u, v)는 개체 u와 개체 v의 거리(또는 비유사성 척도)를 말한다. 이처럼 계층적 군집 방법은 각 개체를 하나의 군집으로 시작하여 군집들을 묶어가는 과정을 반복하여 결국 모든 개체가 하나의 군집이 되도록 하는 것이다.

군집화 개념은 많은 분야에서 다양한 형태로 연구되고 있지만 궁극적으로는 유사한 데이타 개체들을 군집시키고 유사성이 높지 않은 그룹을 분리하여 쉽게 관리

하고 분석하기 위한 것이었다. 그러나 본 논문에서 제시하는 그룹 짓기(grouping) 알고리즘은 군집화 개념과 본질적으로 다른 차이점을 가지고 있다. 즉, 군집화는 유사한 집단으로 분류하기 위하여 먼저 모든 데이타나 프로그램을 일일이 비교한 후에 집단을 형성한다. 그러나 본 논문에서 제시하는 그룹 짓기는 많은 횟수의 비교를 줄이기 위하여 두 프로그램의 유사성을 검사하면서 그룹을 만들어간다. 따라서 모든 대상 프로그램을 일일이 비교하지 않아도 유사성이 높은 그룹으로 분류할 수 있다.

프로그램 유사성 그룹 짓기는 국내에서도 연구되었다. [8]에서는 프로그램 복제 검사에서 유사성을 기반으로 많은 프로그램을 검사하는 연구가 있었다. 이 연구에서는 전역 유사성과 지역 유사성을 기반으로 그룹 짓기를 수행하였다. 예를 들면 전역 유사성 g에 의해 정의된 그룹 G는 "$\forall a \in G, \forall b \in G$. Similarity(a, b) $\geq g$"로 정의되어 수행되었다. 예를 들면, 다음 표 1과 같은 5개의 프로그램에 대한 각각의 유사성이 있다고 가정하자.

표 1 프로그램 P1, P2, P3, P4, P5의 유사도

프로그램	P2	P3	P4	P5
P1	0.80	0.93	0.62	0.65
P2		0.92	0.67	0.5
P3			0.43	0.8
P4				0.85

만일 전역 유사성이 0.8일 경우, [8]에서 제시한 그룹 짓기는 다음 그림 2와 같이 수행된다. 즉, 그림에서처럼 표 1의 프로그램들은 {{P1, P2, P3}, {P4, P5}}와 같은 2 그룹으로 만들어진다.

그러나 이렇게 전역 유사성을 이용하여 그룹을 지으면 문제가 생길 수도 있다. 예를 들면, 그림 3에서와 같이 P5는 P1, P2에 대해서 전역 유사성 이하의 값을 가진다. 그러나 P3에 대해서는 전역 유사성을 초과하는

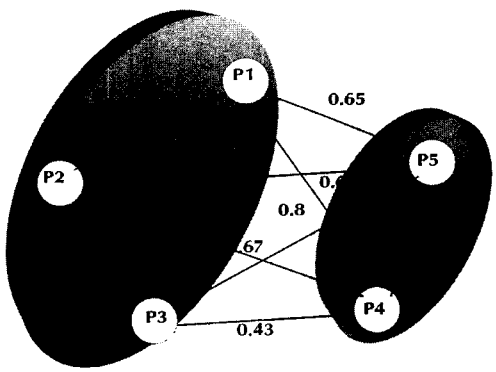

그림 2 전역 그룹의 예

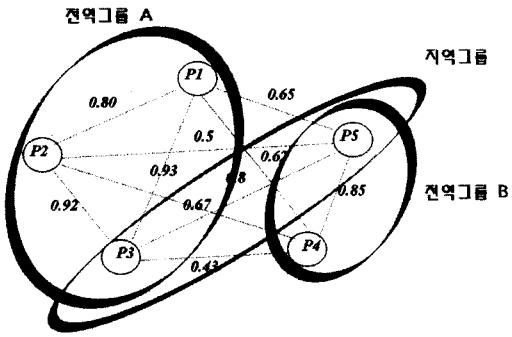

그림 3 지역 그룹의 예

값을 가지므로 포함되지가 않는다. 간단히 말해서 P5는 P1, P2 프로그램과는 다르나 P3과 유사하다는 점이다. 따라서 전역 그룹을 짓는 경우에는 P5와 같이 소외되는 프로그램도 있다. 이러한 경우를 대비해서 [8]에서는 지역 유사성을 주어 보완하고 있다. 즉 지역 유사성 l은 "$\forall a \in G, \exists b \in G$. Similarity(a, b) $\geq l$"로 정의하고 있다.

3. 유사도 평가와 그룹 짓기

본 논문에서는 프로그램 사이에 유사도가 높은 과제물에 대해서 그룹 짓기를 수행한다. 따라서 그룹 짓기를 수행하기 이전에 먼저 서로 다른 두 개의 프로그램에 대해서 유사도를 평가해야만 한다. 본 논문에서는 [9]에서 제시한 프로그램 유사도 측정 알고리즘을 이용한다.

3.1 유사도 평가

[9]에서 제시한 프로그램 유사도 측정 알고리즘은 크게 2단계로 이루어져 유사도를 평가하며, 개략적인 알고리즘은 다음과 같다.

알고리즘 1. 두 노드 스트링(A, B)에서 매칭 스트링 찾기

```
NodeString MatchString(NodeString A, NodeString B) {
  long int i, j, k;      /* 각각 A, B에서 일치되는 스트링
  의 인덱스 */
  long int matchsize, maxmatch; /* 각각 일치되는 스트
  링의 크기 */
  matchstring = ""; /* 가장 길게 일치되는 스트링 */

  for Ai in A {
    for Bj in B {
      matchsize = 0; /* 일치되는 스트링의 크기 */
      while(Ai+matchsize == Bj+matchsize && Ai+matchsize &&
      Bj+matchsize)
          matchsize++;
      if (matchsize > maxmatch) {
          /* 찾은 스트링의 개수가 이전에 찾은 스트링
            의 개수보다 클 때 수행 */
```

```
            matchstring = match(Ai, Bj, matchsize);
            /* 가장 큰 스트링을 matchstring에 할당 */
            maxmatch = matchsize;
        }
    } end for
} end for
Ai = Bj = "NULL"; /* 잘못 검사되는 것을 방지하기 위한
플래그 삽입 */
for k = 1 to matchsize -1 {
  Deleting(Ai+k);
  Deleting(Bj+k);
} end for

return matchstring; /* 찾은 스트링을 반환 */
}
```

알고리즘 1은 노드 스트링 A에서 각각의 서브스트링 Ai에 대해서 동일한 Bj를 찾는다. 동일한 서브 노드 스트링을 찾으면 가능한 최대 크기의 토큰 스트링을 찾기 위해서 계속 비교된다. 두 토큰(Ai+matchsize와 Bj+matchsize)이 다르다면, 찾은 서브스트링을 추가할 것인지 검사한다. 만일 matchsize가 maxmatch 보다 크다면, matchstring에 새롭게 추가된다. 추가된 스트링은 중복 검사를 방지하기 위하여 노드스트링 A, B에서 삭제된다.

알고리즘 2. 트리 유사도 평가

```
double Sim(NodeString A, NodeString B, long int minlength) {
  String matchstring, totalmatchstring; /* 일치된 스트링 */
  int maxmatch = 0;   /* 일치된 스트링의 개수 초기화 */
  long int matchlength = 0;  /* 일치된 스트링의 전체 개수
  초기화 */
  Set(totalmatchstring) = {}; /* 일치되는 전체 스트링 집합*/

  /* 일치되는 스트링을 찾을 때까지 알고리즘 1 반복 */
  do {
    matchstring = ""; /* 일치되는 스트링 */
    matchstring = MatchString(A, B); /* 알고리즘 1 호출 */
    Set(totalmatchstring) = Set(totalmatchstring) + matchstring;
  } while (maxmatch > minlength);

  /* 일치되는 스트링의 총 개수 계산*/
  for each matchstring in Set(totalmatchstring)
    matchlength = matchlength + Length(matchstring);
  end for
  return (2 * matchlength / (Length(A) + Length(B))); /* 유사도 값 계산
  및 반환 */
}
```

위의 프로그램 유사도 평가 알고리즘 설명하기 위하여 다음과 같은 노드 스트링 P1과 P2가 있다고 가정하자(단, 다음의 숫자는 노드를 나타내는 숫자이다).

P1nodestring : 23 **34 25 54** 44 45 49 **81 83 84 22 55 44 33 90** 68

P2nodestring : **34 25 54** 46 47 **81 83 84 22 55 44 33 90** 93 92 95 34 35

[알고리즘 1]에서는 두 노드 스트링의 일치된 부분을 검사한다. 즉, 예에서 {{34, 25, 54}, {81, 83, 84, 22, 55, 44, 33, 90}}을 찾아낸다. 또한 찾은 부분 스트링을 집합에 추가한 후, P1, P2 노드 스트링에서 이를 제거한다. 왜냐하면, 찾은 스트링은 다음에 일치하는 스트링과 다시 비교하는 것을 피하기 위함이다. [알고리즘 2]에서는 찾은 스트링을 이용하여 유사도를 측정한다. 즉, 노드 스트링 P1, P2는 다음과 같은 유사도를 갖는다.

$$sim(P1, P2) = \frac{2*MatchLength}{Length(P1)+Length(P2)} = \frac{2*(3+8)}{16+18}$$

= 0.647 이다.

3.2 그룹 짓기

본 논문에서는 많은 양의 프로그램을 검사하고 유사한 그룹으로 묶기 위해 다음과 같은 그룹 짓기 알고리즘을 이용하였다.

알고리즘 3. 그룹짓기

```
file *P;   /* 비교할 프로그램 입력 */
int  g;   /* 전역 유사성 값 입력 */
boolean addgroup = false; /* 그룹에 추가되었는지에 대한 flag */
add P to G(1);   /* 첫 번째 비교 프로그램은 무조건 그룹에 추
가 */

i =1;  /* 그룹 개수 나타내는 계수 */
Set G(1) = ∅              /* 그룹 초기화 */
while( not eof) {
  input P;   /* 비교 대상 프로그램 입력 */
  for each i in G(i)   /* 모든 그룹에 대해서 수행 */
    if Sim(P, G(i)) > g then   /* 그룹에 포함되는 경우 */ {
      add P to G(i); /* 그룹에 추가 */
      addgroup = true;
    }
  end for

  /* 그룹에 추가가 안 되었으면 그룹 생성 */
  if (not addgroup) then {
    i = i + 1;  /* 그룹 카운터 추가 */
    Set G(i) = ∅   /* 그룹 초기화 */
    add P to G(i);   /* 모두다 포함이 안 될 경우 */
  }
}
```

위의 그룹 짓기 알고리즘은 비교할 대상 프로그램이 입력되면 이미 그룹화되어 있는 프로그램과 비교를 수행한다. 비교한 유사성이 전역유사성보다 낮으면 다른 그룹과 비교하며, 만일 전역유사성보다 프로그램 유사도가 높다면 프로그램을 해당 그룹에 포함시킨다. 또한 모

든 그룹의 프로그램보다 유사성이 낮다면 새로운 그룹으로 포함시킨다. 이 과정은 프로그램 입력이 끝날때까지 반복적으로 수행된다.

본 알고리즘을 설명하기 위하여 위에서 언급한 표 1과 같은 5개의 프로그램이 있다고 가정하자. [8]의 예에서처럼 전역 유사성이 0.8로 주어졌다면 다음 그림 4와 같이 그룹 짓기가 순차적으로 이루어진다.

만일 그룹 짓기를 하지 않고 프로그램의 비교가 이루어진다면 (5*4)/2 번 즉, 10번의 비교가 이루어져야 한다. 그러나 그룹 짓기를 수행하면 다음 그림 4와 같이 단 6번을 비교하면 모든 프로그램의 유사도를 측정할 수 있다. 이처럼 그룹 짓기 알고리즘을 이용하면, 최소 n-1번 ~ n(n-1)/2 번까지 비교가 수행된다. 쉽게 말하자면, 복제된 프로그램이 많으면 많을수록 그룹 수가 적어지므로 n개의 프로그램을 모두 비교할 수 있는 횟수는 크게 줄어든다. 반면, 복제되지 않은 프로그램이 많으면 많을수록 그룹 수가 증가하여 비교할 횟수가 많아진다. 또한 모든 프로그램 비교 횟수는 전역 유사성 g 값에 비례한다. 즉, 전역 유사성 g값이 크면 클수록 그룹이 많아지므로 비교횟수가 많아지며, 작을수록 그룹수가 적어지므로 모든 프로그램의 비교횟수는 적어진다.

4. 실 험

본 시스템은 Java 언어로 구현되었으며, jdk1.3.1을 이용하였다. 따라서 윈도우 체제나 UNIX 환경에서 모두 활용가능하며, 프로그램 트리를 구성하기 위하여 구문 분석과 어휘 분석이 가능한 JLex[13,14]와 Java CUP[15] 유틸리티를 사용하였다.

본 절에서는 S대 컴퓨터 학부 학생들이 제출한 과제물에 대해서 유사도 평가를 수행하고 분석한다. 과제물은 정렬(sort)에 관한 C 프로그램 예제이며, 분석을 위하여 10여 명의 학생들의 과제를 추출하여 수행하였다.

<실험 4-1. 전체 프로그램의 유사도 테이블 결과>

다음 표 2는 전체 프로그램을 모두 비교하여 유사도 테이블로 나타낸 결과이다. 표에서 -1 값은 비교하지 않은 경우를 나타내며, 나머지 값들은 유사도를 나타낸

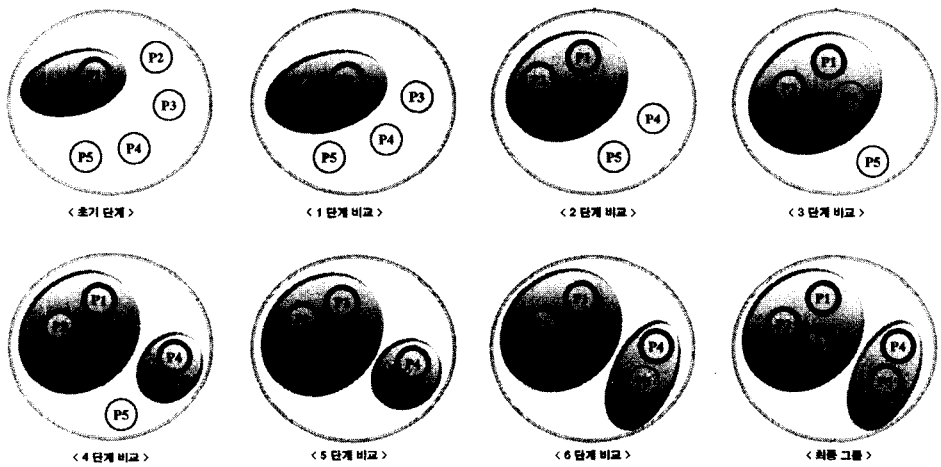

그림 4 표 1의 그룹 짓기의 예

표 2 전체 프로그램의 유사도 결과

	0.66690	0.85556	0.92987	0.89830	0.75231	0.82222	-1	0.84886	0.87282
		0.78691	0.70613	0.66399	0.5258	0.80889	-1	0.60466	0.64314
			0.89058	0.84	0.69112	0.91719	-1	0.76459	0.81351
				0.90946	0.76179	0.82220	-1	0.83590	0.86497
					0.81253	0.79964	-1	0.89170	0.90638
						0.65712	-1	0.85480	0.84421
							-1	0.73117	0.77869
								-1	-1
									0.91350

다. 따라서 "2486.c" 파일은 구문 오류가 있기 때문에 다른 프로그램과의 유사도 검사 대상에서 제외되었다.

표 2에서 알 수 있듯이 모든 프로그램들은 0.5~1 사이의 유사성을 나타내고 있다. 특히 이 프로그램 사이에는 완전 복제가 없었으며, 강한 유사는 많이 나타났다. 만일 이 프로그램을 앞서 논의했던 그룹 짓기를 이용하여 수행하면 다음 실험 4-2, 4-3과 같다. 다음 실험 4-2와 실험 4-3은 전역 유사성을 각각 0.8과 0.9로 설정하여 비교한 실험결과이다.

<실험 4-2. 전역 유사성 0.8로 설정하여 그룹 짓기를 수행하였을 경우>

다음 표 3은 전체 프로그램을 그룹 짓기를 수행하여 비교횟수를 줄이는 실험이다. 그룹 짓기를 수행한 결과 표 2보다 많은 비교횟수가 줄어들었음을 알 수 있다.

Group0 : 4160.c, 4035.c, 4039.c, 4078.c,
3469.c, 1234.c, 4560.c
Group1 : 4129.c
Group2 : 7180.c
Group3 : 2486.c

표 3 전역 유사성 0.8로 설정하여 그룹 짓기를 수행한 결과

Group	Group0	Group1	Group2	Group3
Group0		0.666908	0.752316	-1
Group1			0.52589	-1
Group2				-1
Group3				

표 3의 의미는 실제로 같은 유사도 그룹에 포함되어 있으나, 실제로 누가 어떤 프로그램을 복제했는지 분별하기 어렵다는 점이다. 앞서 언급했듯이 유사도 0.8은 실제로 중간 유사에 해당되는 경우로, 약간의 복제 가능성을 나타낸다. 따라서 실제 복제 판정을 내리기가 어려운 단계이다. 따라서 실험 4-3에서는 실제 강한 유사를 갖는 프로그램을 그룹 짓기 위하여 유사도 0.9를 이용하여 수행하였다.

<실험 4-3. 전역 유사성 0.9로 설정하여 그룹 짓기를 수행하였을 경우>

Group0 : 4160.c, 4039.c
Group1 : 4129.c
Group2 : 4035.c, 3469.c
Group3 : 4078.c, 4560.c
Group4 : 7180.c
Group5 : 2486.c
Group6 : 1234.c

표 4에서 눈여겨 볼 점은 전역 유사성 값을 0.9로 하였을 때 0.8로 한 결과보다 훨씬 많은 비교가 이루어졌다는 점이다. 이것은 당연한 결과이다. 왜냐하면 유사도 80% 이상인 프로그램들을 묶는 것보다 90% 이상인 프로그램들을 묶는 것이 더 세분화되었기 때문이다.

본 시스템의 신뢰성을 알아보기 위하여 전역 유사성 0.9에 대한 그룹 짓기 결과를 오프라인으로 검사해보았다. 즉 그룹 짓기 결과 그룹을 형성한 그룹에 대해서 조사한 결과 완전 복제는 없었으며, 비슷한 부분이 상당히 많았다. 이는 그룹이 형성된 프로그램은 실제 0.9 유사성을 갖는다는 것을 증명해주었다. 여기서 하나 유의할 점은 지역 그룹이다. 즉, 표 6-3에서 "group3"의 "4078.c"와 "4560.c"는 "group6"의 "1234.c"와 그룹이 형성되지 않았다. 즉, 표 6-1에서는 두 프로그램은 유사성이 0.9 이상이었지만 표 6-3에서는 그룹을 형성하지 못하였다. 따라서 본 시스템의 신뢰성을 더욱 높이기 위해서는 그룹을 형성하는 프로그램에 대해서 전체적인 비교가 필요하다. 이는 앞서 논의한 지역 유사성의 문제이며 그룹에 대해서 전체적으로 1:1 유사도 검사를 수행함으로써 해결할 수 있다.

그룹 짓기의 관건은 전역 유사성을 어떻게 주고 그룹 짓기를 할 것인가가 중요한 요인이다. 즉, 유사성 0.9를 준다는 의미는 강한 유사를 나타내므로 이 그룹에 포함되어 있는 프로그램들은 복제될 확률이 높다는 점을 의미한다. 물론 완전 복제 혹은 0.97이상의 값을 준다면, 거의 복제되었다는 의미로 받아들여지며[9], 유사성 0.8 이하면 두 프로그램의 복제 유무 판단에 많은 의문점이 생길 수 있다. 따라서 이러한 문제는 많은 경험을 토대로 검수자가 적절히 활용해야만 할 것이다.

표 4 전역 유사성 0.9로 설정하여 그룹 짓기를 수행한 결과

Group	Group0	Group1	Group2	Group3	Group4	Group5	Group6
Group0		0.666908	0.855566	0.898305	0.752316	-1	0.848864
Group1			0.786918	0.663991	0.52589	-1	0.604669
Group2				0.84	0.691122	-1	0.76459
Group3					0.812539	-1	0.891705
Group4						-1	0.854806
Group5							-1
Group6							

5. 결 론

본 논문에서는 학생들이 제출한 많은 프로그램에 대해서 유사성이 높은 프로그램으로 그룹 짓기(grouping)를 수행하는 알고리즘을 제시하고 구현하였다. 그룹 짓기에 이용된 유사도 측정 방법은 [9]에서 제시한 프로그램 유사도 평가 알고리즘을 이용하였으며, 그룹 짓기 알고리즘을 이용하면 n개의 프로그램에 대해서 최대 n(n-1)/2 번에서 최소 (n-1)번까지 비교 횟수를 줄일 수가 있다는 것을 실험에서 보여주었다. 본 논문의 실험 부분에서는 실제로 모 대학의 컴퓨터 학부 과제물 10개를 추출하여 실험하였으며, 쉽게 그룹핑할 수 있다는 것을 보여준다.

본 연구 논문의 향후 연구과제로는 그룹 짓기에 이용되는 전역 유사성 값을 적절히 지정하는 방법과 지역 유사성을 갖는 프로그램을 더 정확히 선별해야하는 연구가 필요할 것이다. 또한 타 분야에서 연구되고 있는 군집화(clustering) 개념을 활용하여 보다 효율적인 그룹 짓기를 수행할 수 있는 방법을 연구해야 할 것이다. 본 논문에서 제시한 그룹 짓기를 활용하면, 많은 양의 프로그램 혹은 문서의 과제물의 검사에 유용하게 이용할 수 있으며, 소프트웨어 공학의 프로그램 사이의 유사성과 그룹 짓기 분야에 많은 영향을 줄 것으로 기대된다.

참 고 문 헌

[1] J. A. Faidhi & S. K. Robinson, "An Empirical Approach for Detecting Program Similarity within a University Programming Environment," Computers and Education 11(1), pp. 11~19, 1987.

[2] J. Carter, "Collaboration or Plagiarism: What Happens when Students Work Together?," In proc. ITiCSE, pp. 52~55, Cracow, Poland, 1999.

[3] S. Horwitz, "Identifying the Semantic and Textual Difference Between two Versions of a Program," In proc. of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pp. 234~245, Jun., 1990.

[4] K. K. Vercon & M. J. Wise, "Software for Detecting Suspected and Prevension of Plagiarism: Comparing Structure and Attribute-Counting Systems," In John Rosenberg, editor, proc. of 1st Australian Conference on computer Science Education, Sydney, ACM, Jul., 1996.

[5] M. Joy & M. Luck, "Plagiarism in Programming Assignments", IEEE Transaction in Education, 42(2), pp. 129~133. 1999.

[6] P. H. A. Sneath & R. Sokal, Numerical Taxonomy: The Principles and Practices of Numerical Classification. Freeman and Company. 1973.

[7] Z. Huang, "A fast clustering algorithm to cluster very large categorical data sets in data mining," In SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (SIGMOD-DMKD'97), Tucson, Arizona, May 1997.

[8] 장성순, 서선애, 이광근. "프로그램 유사성 검사기", 정보과학회 가을 학술대회, 28(2), pp. 334~336, October, 2001.

[9] 김영철, 김성근, 염세훈, 최종명, 유재우, "구문트리 비교를 통한 프로그램 유형 복제 검사", 한국 정보과학회 논문지, 30(8), Aug. 2003.

[10] J. Han and M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers, ISBN 1-55860-489-8, 2000.

[11] Y. M. Cheung, "K*-means-A Generalized k-means Clustering Algorithm with Unknown Cluster Number," available at http://www.comp.hkbu.edu.hk/~ymc/papers/conference/ideal2002.pdf, Hong Kong Baptist Univ., 2002.

[12] G. Salton & M. J. McGill, Introduction to Modern Information Retrieval. NY: McGraw-Hill Book Company. 1983.

[13] E. Berk, "Jlex:A Lexical Analyzer Generator for Java TM," available at http://www.cs.princeton.edy/~appel/modern/java/JLex/.

[14] J. Lin, JLex Tutorial, available at http://bmrc.berkeley.edu/courseware/ cs164/spring98/proj/jlex/tutorial.html.

[15] S. E. Hudson, "CUP Parser Generator for Java," available at http://www.cs.princeton.edy/~appel/modern/java/CUP/.

유 재 우

1976년 학사 숭실대학교 전자계산학과 1985년 박사 한국과학기술원 전산학과 1983년~현재 숭실대학교 컴퓨터학부 교수. 1986년~1987년, 1996년~1997년, 코넬대학교, 피츠버그대학교 객원교수 1999년~2000년 한국정보과학회 프로그래밍언어 연구회 위원장. 관심분야는 프로그래밍언어, 컴파일러, 인간과 컴퓨터 상호작용

김 영 철

1990년 한남대학교 전자계산학과 졸업 1996~현재 숭실대학교 컴퓨터학과 박사과정. 2002년 3월~현재 명지전문대학 겸임교수. 관심분야는 컴파일러, 망관리, 프로그래밍 언어