韓國IT서비스學會誌
第11卷 第4號
2012年 12月, pp.263-275
http://dx.doi.org/10.9716/KITS.2012.11.4.263

토큰 크기 및 출현 빈도에 기반한 웹 페이지 유사도*

이은주** · 정우성***

Web Page Similarity based on Size and Frequency of Tokens*

Eun-Joo Lee**, Woo-Sung Jung***

■ Abstract ■

It is becoming hard to maintain web applications because of high complexity and duplication of web pages. However, most of research about code clone is focusing on code hunks, and their target is limited to a specific language. Thus, we propose GSIM, a language-independent statistical approach to detect similar pages based on scarcity and frequency of customized tokens. The tokens, which can be obtained from pages splitted by a set of given separators, are defined as atomic elements for calculating similarity between two pages. In this paper, the domain definition for web applications and algorithms for collecting tokens, making matrics, calculating similarity are given. We also conducted experiments on open source codes for evaluation, with our GSIM tool. The results show the applicability of the proposed method and the effects of parameters such as threshold, toughness, length of tokens, on their quality and performance.

Keyword : Web Application, Page Clone, Page Similarity, Token Frequency

논문투고일 : 2012년 07월 27일 논문수정완료일 : 2012년 09월 14일 논문게재확정일 : 2012년 10월 03일

* 이 논문은 2012학년도 충북대학교 학술연구지원사업의 연구비 지원, 그리고 2012학년도 경북대학교 학술연구비에 의하여 연구되었음.

** 경북대학교 IT대학 컴퓨터학부

*** 충북대학교 전자정보대학 컴퓨터공학과, 교신저자

1. 서 론

웹 어플리케이션(Web application, 이하 WA로 표기)은 초기에는 단순한 HTML 태그 위주의 페이지로 구성되어 있었지만, 오늘날에는 구조화 되지 않고 CSS, 스크립트 등이 섞여 있는 복잡한 형태를 가지게 되어 WA의 페이지를 전통적인 모듈 관점에서 볼 필요성이 증가하고 있다. 뿐만 아니라, 체계적이지 못한 프로세스나 빠른 개발 주기로 인하여[16] 복사하기와 붙여넣기(copy and paste)로 대변되는 중복 코드들이 여러 페이지에 산재하고[2, 14, 21], 효과적인 코드의 관리를 위해서 단순한 의미적 유사함 뿐 아니라 대상이 되는 두 페이지를 구성하는 코드들 간의 유사성을 찾아내는 것이 중요한 문제가 되고 있다. 실제로 중복된 웹 페이지들이 범람함으로써 유지보수 및 테스트 비용 등이 증가하고 있고, 이러한 중복은 항해 패턴(navigational pattern)이나 기능에서도 중복을 초래하여 체계적인 유지보수를 방해하고 있다[16].

그런데 이전의 웹 페이지 유사도는 검색엔진을 통해 쿼리 페이지와 유사한 페이지를 찾는 용도[13]로서 의미적 관점에서의 유사도를 뜻하는 경우가 대부분이었다. 따라서 웹 페이지의 유지보수적 관점에서의 유사도를 구하기 위하여 코드 클론(code clone)과 연계시켜 볼 필요가 있다. 코드 클론이란 코드 복제나 기존 코드에 약간의 수정을 가함으로서 생긴 코드 내부의 중복된 부분으로서[3, 9], 숙련된 프로그래머라면 명확한 의도를 가지고 코드 클론을 사용하기도 하지만[8] 일반적으로 이후 유지보수 단계에서 코드 사이의 일관성을 저해시켜 코드의 품질을 떨어뜨리고 프로그램의 성능을 저하시키는 원인이 된다. WA은 다른 도메인에서의 SW보다 클론의 비율이 높은데[8], 특히 WA 개발시 기본적 구조를 카피하고 일부만 변형시켜 개발하는 사례가 잦기 때문이다[2, 14]. 따라서 WA에서의 클론 식별은 해당 WA의 진화, 유지보수, 테스트 노력 등을 감소시키는데 큰 역할을 한다[14].

기존의 전통적 소프트웨어를 대상으로 한 클론 검출 연구는 다수 존재한다[7, 11, 12]. 하지만, 기존 접근법을 WA에 바로 적용하기에는 어려운 점이 많다[14]. 왜냐하면 WA를 구성하는 페이지들이 구조는 같으나 담고 있는 데이터가 다를 수 있고[14], WA의 페이지 자체가 구조적이지 않으므로 추상화시켜 비교를 하기에는 어려움이 존재하기 때문이다. 즉, 웹 애플리케이션과 같은 경우는 HTML, CSS, JavaScript, VBScript와 같은 클라이언트 코드와 JSP, ASP, PHP와 같은 서버 코드가 뒤섞여 있기 때문에 정확한 파싱이 어렵거나, 또는 가능하더라도 파싱 비용이 많이 든다. CCFinder의 경우도 text 문서에 대해 일반적인 방법으로 유사도를 비교할 수는 있지만 php와 같은 웹 애플리케이션에 대한 별도의 클론 검출 방법은 제공하고 있진 않으며, 페이지 단위의 비교가 아닌 코드 조각의 클론 검출만을 제공하고 있다. 또한, 기존의 연구들은 대부분 정확한 파싱을 요구하거나 비교 단위를 제공해주어야 하는데, 유사한 정도에 대한 정확도(precision) 혹은 회수율(recall)에 대한 기준을 조절하여 검색하기가 쉽지 않다. 즉, 경우에 따라서는 100% 일치하는 구조의 코드를 찾기도 해야 하지만, 50% 정도 일치하는 코드를 찾아야할 경우도 있다.

웹 도메인에서의 클론 검출 연구들도 일부 존재한다[1, 3, 9, 14-17]. 특히, 스크립트 코드에서 함수의 클론을 검출하는 기법이 제안되었으나[3, 9], 정작 비교를 위한 구체적인 테크닉이 부족하다. Lucca 등은 ASP(Active Server Pages) 동적 웹 페이지에서 클론 페이지(cloned page)를 식별하는 연구를 수행하였고[14], De Lucia 등은 구조, 컨텐츠, 스크립트 코드 레벨에서 클론 페이지를 검출하는 연구를 수행하였다[15]. 이러한 연구들은 모두 LD(Levenshtein Distance)[10]를 이용하여 LD가 낮을수록 클론에 가깝다고 정의한다. 이는 이전의 클론 검출 기법과 크게 다르지 않으며 성능 문제가 있을 수 있다. 또한 구조화되기 어려운 웹 페이지의 경우 적절하지 않고 코드 삽입이나 삭제에 취약하다는 단점이 있다. 이러한 약점을 보완하기 위해 ED(Edit

Distance)를 사용하는 경우와 유사한 결과를 내면서 성능은 더 좋은 용어 빈도(term frequency)를 이용할 수 있지만[14], 프로젝트에 제한적인 용어의 특성을 반영하기 어렵다는 한계를 가진다. 하지만, 대부분의 유지보수는 하나의 프로젝트를 대상으로 수행되므로, 예를 들어 개발자들이 주로 쓰는 변수명과 같이, 특정 프로젝트에 제한적인(project-specific) 정보를 이용하는 것이 유용할 수 있다. 또한 토큰의 길이도 유사 페이지 검출을 위한 고려사항이 될 수 있다. 예를 들어 어떤 두 웹 페이지에서 특이한 파라미터명인 “abcxyzaaa”가 각각 두 번 나온 것과, “if”라는 키워드가 두 번 나왔다고 가정할 경우, 경험적으로 봤을 때 전자의 경우가 두 페이지 사이의 관련성이 후자보다 더 높다고 말할 수 있을 것이다. 그러므로 이러한 개념을 도입하여 기존 연구들의 단점을 보완하고, 보다 융통성있는 유사도 척도를 정의할 수 있다.

본 논문에서는 웹 어플리케이션 내의 페이지를 대상으로 하는 클론 기반의 페이지 단위의 유사도 GSIM을 제안하였고, 검출 결과의 수준을 조정할 수 있도록 하였다. 이를 위하여 코드 클론의 크기 및 전체 페이지에 대한 클론 비율 등을 기준으로 페이지 단위의 비교를 위한 새로운 방법을 제공하였다. 또한 다양한 옵션을 제공함으로써 사용자로 하여금 정확도(precision)나 회수율(recall) 조정이 가능하도록 하였다. 본 논문에서는 웹 페이지를 컨텐츠가 아닌 웹 애플리케이션으로 보고 있다. 웹 컨텐츠의 의미적 유사도는 키워드의 빈도를 기반으로 휴리스틱을 이용하여 측정할 수 있으나, 실제로 리팩토링의 대상은 의미적 유사성 보다는 구조적, 문법적으로 유사한 페이지를 대상으로 삼는 경우가 일반적이다. 의미적으로 유사하더라도 전혀 다른 구조 또는 뷰를 가지는 경우라면 페이지 자체의 코드는 분명 다르다고 볼 수 있고, 서로 다른 구현에 해당하기 때문이다. 또한, 웹 애플리케이션이 포함한 스크립트를 엄격한 의미 분석을 위해 정적 분석을 하게 되는 경우에는 구조적으로 유사하여 리팩토링이 가능하더라도 너무 엄격한 기준으로 인해 회수율(recall)이 떨어질 수 있다. 결국, CSS나 HTML, 스크립트 등을 기반으로 구성된 웹 페이지는 의미 비교보다는 유사한 구조나 토큰들의 분포가 높은 페이지들을 대상으로 리팩토링을 실시하는 것이 더 실용적이며 합리적이다.

본 논문의 구성은 다음과 같다. 제 2장에서는 관련연구를 기술하고, 제 3장에서는 GSIM을 구하기 위한 모델 및 GSIM의 알고리즘을 정의하고 제 4장에서 실제 WA에 적용한 사례를 보이며 제 5장에서 결론을 맺는다.

2. 관련 연구

코드 클론과 관련하여 CCFinder[7]와 같이 코드 클론을 검출하거나 Ekwa Duala-Ekoko의 Clone Tracker[4]와 같이 클론을 추적하기 위한 연구가 있다. 뿐만 아니라, Higo. 등은 Aries[5]에서 코드 클론 제거를 위해 메소드 추출(extract method), 메소드 올리기(pull up method), 수퍼 클래스 추출(extract super class) 등의 리팩토링 전략을 제공함으로써 개발자를 지원하기 위한 연구를 하였다. 정상급 클론 검출기로 평가받는[8] CCFinder[7] 는 프로그램을 특정 룰에 따라 토큰으로 변형한 다음 토큰 대 토큰으로 비교한다. 웹 어플리케이션은 ‘well-formed’가 아니므로 토큰의 변경룰을 정하기 어렵고, 현재 웹 도메인에 대한 CCFinder는 별도로 존재하지 않는다. 모든 코드 클론 연구들은 유사한 코드 조각을 찾기 위해 각기 다른 코드 패턴 비교 전략을 구현하고 있다. CCFinder의 경우는 코드 파싱을 통해 직접 검출하며, CloneTracker는 SimScan이라는 검출 도구를 사용하고 있다. CP-Miner[11, 12] 역시 토큰 기반의 클론 검출 기법으로, 데이터마이닝 기법을 사용하여 복제된 코드를 검출한다. DECKARD[6]의 경우는 트리(tree)에 기반하여 각 코드가 가지는 주요 키워드(keyword)들의 개수를 벡터로 구성하고, 이들 간의 거리를 구함으로써 유사도를 측정한다. 하지만, 기존의 연구들은 대부분 정확한 파싱을 요구하거나 비

교 단위를 제공해주어야 하며, 유사한 정도에 대한 회수율이나 정확도에 대한 기준을 조절하여 검색하기가 쉽지 않다. 하지만, 경우에 따라서는 완전 일치하는 구조의 코드를 찾기도 해야 하지만, 절반 정도 일치하는 코드를 찾아야할 경우도 발생한다. 그러므로 페이지 단위의 유사도 비교 측정을 위해서는 코드 클론의 크기 및 전체 페이지에 대한 클론 비율 등을 기준으로 페이지 단위의 비교를 위한 새로운 방법이 제공되어야만 한다.

웹 페이지를 대상으로 수행되었던 여러 유사도 연구들이 존재한다[3, 14, 15 20]. Lucca 등은 ASP에서 cloned pages를 식별하는 연구를 수행하였고[14], Levenshtein은 태그의 시퀀스를 문자열(string)로 변환한 다음 LD[10]를 사용하여, 구조적 수준에서의 클론 페이지를 식별하고 있다. 이때 LD가 0일 경우는 클론 페이지 쌍에 해당한다. [15. 20]에서도 LD를 사용하여 유사한 페이지를 판단하고 있다. 단, [14]에서는 문자열 길이에 따른 성능의 문제로 인하여 태그의 빈도(frequency)를 이용하여 유사한 페이지를 찾아내는 기법도 함께 제안하고 있다. 빈도를 이용할 경우, 정확도는 약간 떨어지지만 계산 비용을 떨어뜨릴 수 있다. 그러나 웹 페이지가 'well-formed'인 경우는 드물고, HTML 태그 뿐 아니라 스크립트 코드나 CSS 등이 혼재되어 있으므로, 태그정보로 스트링을 만들어 비교하기에는 어려움이 있으며, 그 비용 역시 무시할 수 없다. Calefato 등은 스크립트 코드에서의 함수 클론을 검출하는 기법을 제안하고 있으나[3], 클론에 해당하는 함수의 이름은 같다고 가정을 하는 등의 제한이 있으며, 정작 비교를 위한 구체적인 기법은 제안하고 있지 않다. 또한 스크립트 내부의 함수에 대한 것만을 대상으로 하므로 엄밀히 말하여 웹에 초점을 맞추었다고 보기 어렵다. 대부분의 클론 페이지를 찾는 연구에서는 페이지를 문자열로 만들어서 ED(edit distance)를 비교하는 접근법을 취하고 있으나 전기한 바와 같이 웹 페이지를 파싱하여 구조화 시키는 것이 쉽지 않으며, 태그 뿐 아니라 토큰 기반으로 데이터를 추출할 경우 문자열의 길이가 길어져 계산 비용이 높아질 것이다. 또한 태그의 빈도를 이용하는 경우에도, 태그의 특수성이나 상대적 빈도를 고려하지 않으므로 개선의 여지가 존재한다.

정리하자면, 기존 클론 검출 연구의 다수는 웹이 아닌 일반 프로그래밍 언어에 적합한 경우가 많고, 기존 웹 기반 유사도 연구의 대부분 컨텐츠의 유사성에 초점을 맞추고 있어서, 본 연구의 초점인 WA의 유지보수 관련하여 유사한 페이지를 찾고자 하는 관점이 다르다.

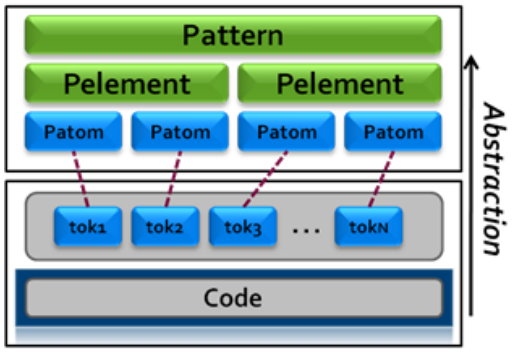
3. GSIM 정의

3.1 개요

GSIM(General SIMilarity)은 대상 언어에 따라 다른 종류의 구분자(separator)를 적용하여 추출된 토큰을 활용하여 유사도를 비교한다. 비교에 사용되는 토큰들을 Patom(atomic pattern)이라고 부르며, 유사도 측정의 최소 단위가 된다. 특히, GSIM은 기존의 다른 유사도 척도와는 달리 Patom들의 분포를 고려하여 희소성이 높은 Patom을 공통적으로 가지는 페이지들의 쌍들에 대해서는 그렇지 않은 Patom을 가지는 경우보다 더 높은 유사도 점수를 부여하고, 흔한 Patom들에 대해서는 무시할 수 있다. 또한, Patom이 의미를 가지지 못할 정도로 짧은 길이인 경우도 무시할 수 있도록 하였다. 즉, 전체 코드를 비교하지 않고 유사성을 확인하는데 필요한 특정 Patom들만 비교함으로써 페이지의 유사도를 측정하는 방법이다. 이러한 접근은 유사도 비교의 비용을 줄이면서도, 품질은 크게 떨어뜨리지 않는다.

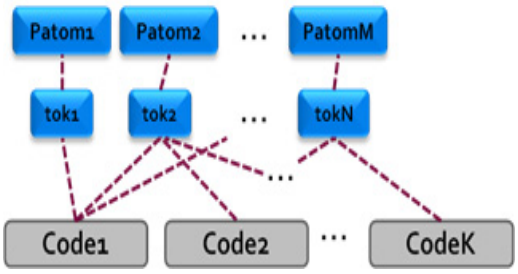
[그림 1]은 Code로부터 추출된 토큰들 중에서 비교에 참여하는 것들만 Patom으로 매핑되는 추상화(Abstraction)의 개요를 보여준다. 비교에 참여하는 Patom들의 집합을 Pelement라고 부르며, 본 논문에서는 동일한 Patom들에 대해서만 실험을 실시하였다. 이 경우는 결국 패턴 비교라는 것

은 서로 공통적인 Patom들을 얼마나 유사한 크기만큼 공유하는가로 결정짓게 된다.



[그림 1] 코드로부터 패턴까지의 매핑

우선 GSIM은 확보된 Patom들을 해시 테이블에 순서대로 삽입하면서 해당 Patom들의 개수를 센다. 이 과정에서 해당 Patom으로부터 근원지인 코드에 대한 포인터 정보 또한 저장하게 된다. 그러므로, 특정 patom이 어느 파일(file)에 속해있는지, 또는 해당 파일에는 어떤 Patom들을 가지고 있는지를 양방향으로 빠르게 확인할 수 있다. 이러한 1차 수집 과정은 전체 소스를 처음부터 끝까지 스캔하면서 이루어진다. 결국 GSIM은 이러한 수집 과정을 거쳐, 코드를 통해 Patom들이 복잡하게 얽혀 있는 Patom-Net([그림 2])에 기반하여 유사도를 측정하게 되는 것이다. 이 Patom-Net을 분석하여 희소성이 높은 Patom을 높은 비율로 함께 포함하고 있는 코드들의 쌍이 유사도가 높다고 판단하게 되는 것이다.



[그림 2] 토큰과 코드의 1 : N 관계(Patom-Net)

최종적인 유사 비교는 페이지 단위로 이루어지기 때문에 2개 페이지간의 공통 Patom들의 비율을 조사하면 '유사함' 또는 '포함' 등의 판단을 내릴 수 있게 된다. C1, C2는 각각 비교 코드라고 하고, Patom(C1), Patom(C2)는 각각 페이지들이 가지는 Patom들의 Bag(중복을 지원하는 집합) 이라고 하자. 이 때, X가 Bag일 때 |X|는 X의 크기를 나타내며, 본 연구에서는 X의 모든 요소들의 길이를 더한 값으로 정의하였다. 즉, 다음 식 (1)과 같다.

$$|Patom(C)| = \sum_{i=1}^{n} |p_i| \quad (1)$$

여기서

$Patom(C) = \{p_1, p_2, \cdots, p_n\}$,

$|p_i|$: p_i의 길이

Patom의 길이를 고려하지 않고, 개수만으로 비교할 수도 있다. 이 경우에는 |Patom(C)| = n이 된다. 본 연구에서는 Patom의 길이를 반영하였는데, 그 이유는 동일한 Patom 조각을 공유하더라도 길이가 긴 경우가 그렇지 않은 경우보다 확률적으로 일치할 확률이 더 어렵다고 보기 때문이다. Patom들의 희소성에 따라 추가되는 유사도 점수는 달라지겠지만, GSIM은 기본적으로 다음 식 (2)에 따라 페이지 쌍에 대한 유사도를 측정하게 된다.

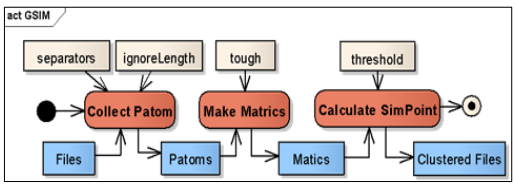
$$GSIM(P_1, P_2) = \frac{|Patom(P_1) \cap Patom(P_2)|}{|Patom(P_2)|} \quad (2)$$

그러므로 만약 GSIM(P1, P2)와 GSIM(P2, P1)이 모두 특정 임계치보다 높다면(예를 들어 70% 이상) 유사하다고 판단할 수 있다. 하지만, 만약 GSIM(P1, P2)가 임계치보다 상당히 높은데 비해, GSIM(P2, P1)이 임계치에 비해 낮다면 P2가 P1에 포함되어 있는 경우를 의심해볼 수 있다.

3.2 알고리즘

[그림 3]은 GSIM을 통해 유사도 측정을 하기

위한 전반적 프로세스를 보여준다. 전체적으로 Patom을 수집하고, 유사도 계산을 위한 메트릭을 생성한 후에, 최종적으로 유사도를 계산하게 된다. 구분자(separators)는 타겟에 따라 달라지며 토큰 추출을 위한 기준을 제공한다. ignoreLength, tough, threshold는 사용자로부터 입력받는 설정값으로 실험에서 사용한 구체적인 값들은 이후에 나오는 <표 1>에 설명하였다. outMatrics와 mType은 출력과 관련한 설정이기 때문에 [그림 3]에서는 생략하였다. 옵션에 대한 설명은 아래와 같다.



[그림 3] GSIM의 전반적 프로세스

- Threshold(0-1) : 유사도를 판정하기 위한 기준치. 즉, 2개의 페이지에 대한 GSIM 계산값이 이 threshold값 이상이면 유사하다고 판단하게 된다. 그러므로 0에 가까울수록 recall은 높아지지만, precision은 떨어지게 되고, 1에 가까울수록 그 반대가 된다.
- Tough(0-1) : 유사도 계산에 참여하는 Patom들의 희소성 판단 기준치. 예를 들어, tough가 0.1이면, 전체 파일의 10% 이내에서만 공통적으로 나타나는 patom 조각만을 이용하게 된다. 즉, 0에 가까울수록 유사도 비교에 사용하는 Patom의 수가 줄게 되어 계산 비용은 줄어들지만, 그만큼 일부만 비교하여 판단하게 되고, 1에 가까울수록 많은 Patom들을 참여시키게 되기 때문에 계산 비용이 증가하게 된다.
- ignoreLength(0 이상) : 비교에서 제외시킬 patom들의 길이이다. 만약, 3일 경우 3 이하의 patom들은 무시되기 때문에 유사도 비교에 사용하지 않는다.
- outMatrics : 유사도 결과 저장을 위한 경로
- mType(fp/pf/ff중 하나) : 결과물의 형식을 정의한 것으로 f는 file, p는 patom을 나타낸다. 즉, fp,pf는 file과 patom간의 관계 메트릭스를 보여줌으로써 patom들의 분포를 눈으로 확인할 수 있게 해준다. 유사한 페이지일 경우는 이 분포가 유사하게 나타나는 경향을 보인다. Ff는 파일과 파일간의 유사도 메트릭스를 출력한다.

Files는 유사도 분석 대상인 코드들의 집합이며, Patoms는 이로부터 얻어진 Patom들과 code 등과의 관계 정보까지 포함하는 Patom 객체들의 전체 집합을 의미한다. Matrics는 Files의 크기를 N이라고 할때 N×N이며 각 셀에는 공통에 해당하는 Patom들로의 포인터 집합들을 담게 되며 '열 < 행'인 부분에 대해서만 값을 가진다. 최종적으로는 Files들이 유사도 정도에 따라서 유사하다고 판단되는 것끼리 묶이게 된다. 아래는 그림에 사용된 해당 개체들에 대한 도메인 정의이다.

```
File = Code
Patoms = PatomObjects+FileObjects
         +InterReferences
Matrics = File×File→Patoms
Clustered Files = set of code sets
```

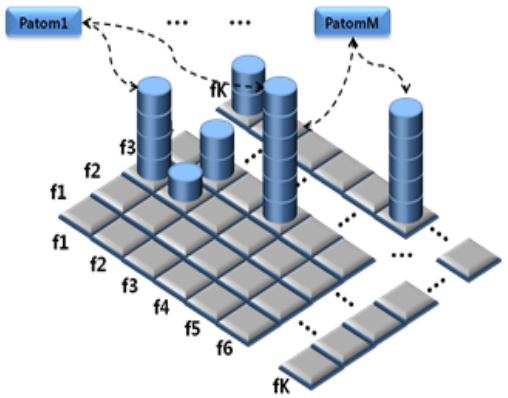
다음은 각각 패턴 수집(Collect Patom), Matrics 형성(Make_Matrics), 유사도 계산(Calculate_Sim) 과정을 pseudo code로 나타낸 것이다.

```
Collect_Patom(separators, ignoreLength)
  For each f in Files
    tokens = Split(f,separators);
    For each t in tokens
      If (t.Length > ignoreLength)
        CreatePatomObjects(t);
        CreateFileObjects(f);
        MakeInterReferences(t,f);
      End If
    End For
  End For
```

Collect_Patom 단계에서는 주어진 separators에 기반하여 모든 File들에 대해 각각의 코드를 쪼개고, 여기서 얻어진 token들 중에서 ingnoreLength보다 큰 것만을 추출하여 PatomOjbect와 FileObject를 생성한 후에 상호 Reference를 통해 연결시켜서 Patom-Net을 구축하게 된다. 이 정보는 다음 단계인 Make_Matrics에서 사용하게 된다.

```
Make_Matrics(tough)
  For each ptm in Patoms.PatomObjects
    If (ptm.Count <= 1 or ptm.FileCount <= 1)
        continue;
    If (ptm.FileCount > Files.Count×tough)
        continue;
    For each f1 in Files which include ptm
      For each f2 in Files which include ptm
        Matrics[f1,f2].Add(ptm);
      End For
    End For
  End For
```

Make_Matrics 단계에는 앞서 얻어진 모든 Patom들에 대해 개수가 2개 이상 존재하며, 2개 이상의 코드에서 공통적으로 나타나는 patom들에 대해서 전체 File 개수 대비 tough 비율 이하로 존재하는 희귀한 패턴에 대해서만 Matrics를 생성한다.



[그림 4] File×File→Patoms 관계 메트릭스

Matric은 [그림 4]와 같이 File×File→Patoms의 도메인을 가지기 때문에 임의의 2개의 화일에 대해 서로 공통적인 Patom들을 모두 알 수 있도록 해준다. [그림 4]에서는 f1과 f3의 경우 공통으로 나타나는 Patom의 개수는 4개이며, 그 중의 하나가 Patom1 임을 의미한다. 이처럼 Matics는 해당 파일들간에 공유되는 Patom 정보들을 모두 가지고 있다.

```
Calculate_Sim(threshold)
  For each f1 in Files where Matrics[f1, *] exists
    For each f2 in Files where Matrics[*, f2] exists
      Sim1 = |Matrics[f1, f2].Patoms|/|f1.Patoms|;
      Sim2 = |Matrics[f1, f2].Patoms|/|f2.Patoms|;
      If (Sim1 > threshold and Sim2 > threshold)
        Cluster(f1, f2);
      End If
    End For
  End For
```

최종적으로 유사도를 계산하게 된다. Matrics에서 공통 Patom을 가지는 file들의 쌍에 대해서만 계산을 하며, 각각의 Patom 크기 대비 공통 Patom의 크기의 비율로 유사도를 계산한다. 주어진 임계치(threshold) 이상을 만족하는 것에 대해서는 2개의 file을 클러스터링(clustering)한다.

3.3 특징

GSIM은 웹 페이지 단위로 주로 작업이 이루어지는 웹 애플리케이션의 경우 효과적인 방법이라고 생각되며, 분석적인 방법이 아닌 코드를 표현이나 코드 내용들을 바라보았을 때 남는 인상에 기초한 비교이기 때문에 뉴럴 네트웍(neural network) 등을 이용하여 유사도에 대한 기준을 학습시키기에도 적합한 형태의 데이터 구조라고 생각한다. 다만, 코드로부터 얻어진 패턴 조각들로부터 비교를 하는 것인 만큼 해당 애플리케이션에서 사용하는 특정 함수나 변수의 이름의 영향을 많이 받게 된다. 이는 GSIM이 서로 다른 웹 애플리케이션 사이에서의 유사도 측정에는 적합하지 않지만, 특정

웹 애플리케이션 내에서 존재하는 유사 페이지는 더 효과적으로 찾아낼 수도 있음을 의미한다.

기존 연구는 CSS, Script 코드를 제외하고 HTML 태그와 속성 및 속성값들을 이용하여 웹 페이지들의 유사도를 계산하였다[14]. 크게 두 가지로 분류되는데, 하나는 각 태그 및 속성에 유일한 식별자를 부여하고 전체 페이지를 string으로 표현한 후 LD를 구하는 방법이고, 다른 하나는 식별자 별 빈도수를 이용하여 벡터로 표현한 후 비교 대상인 두 벡터 사이의 거리를 이용하는 방법이다. 기존 두 방법은 값이 0 이상을 가지나 GSIM은 0부터 1사이의 값을 가지므로 직접적인 값 비교는 어렵기 때문에, 설명을 통하여 그 차이를 보인다. 기존 방법들은 페이지 구조를 기반으로 접근한 것으로 웹 애플리케이션의 구현이나 스타일 정보를 반영하는데 한계가 있다. 예를 들어, js 파일이나 css 파일은 유사 비교를 할 수 없지만, GSIM은 다른 확장자들에 대해서도 교차 비교가 가능하다. 또한, 확장자가 php 또는 html이라고 하더라도 태그 외의 스타일, 스크립트 정보가 많은 비중을 차지하고 있는 페이지의 경우는 태그 구성이 유사하더라도 실제로는 다른 페이지로 봐야 하는데, 기존 연구는 태그에 기반하기 때문에 이를 유사하다고 판단하게 된다. 반대로, 태그 구성에서 큰 차이가 나더라도 실제 내부의 스크립트나 스타일 코드가 유사하고 이들의 비중이 높다면 유사하다고 봐야하는데, 태그에 기반한 벡터 비교로는 이러한 구분이 어렵다. 아래 exam1.html과 exam2.css는 GSIM으로는 높은 유사도를 보이지만, 기존 연구로는 유사도 검출이 어려운 예이다.

```
exam1.html
<html>
…
<style>
a : link {color : #FF0000;}
a : visited {color : #00FF00;}
a : hover {color : #0000FF;}
a : active {color : #AA00FF;}
</style>
…
<body bgcolor = “#AAAAAA”>
    <div style = “font-style : italic”>
    …
    </div>
… (추가적인 짧은 웹 소스 코드) …
</body>
…
</html>
```

```
exam2.css
a : link {color : #FF0000;}
a : visited {color : #00FF00;}
a : hover {color : #0000FF;}
a : active {color : #AA00FF;}
body {background-color : #AAAAAA;}
.exam {font-style : italic;color : #0055FF;}
```

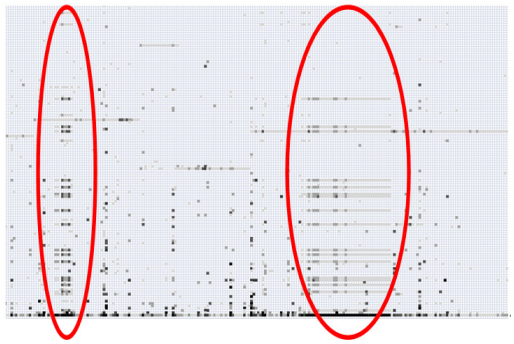
4. 실험 결과

4.1 도구 구현

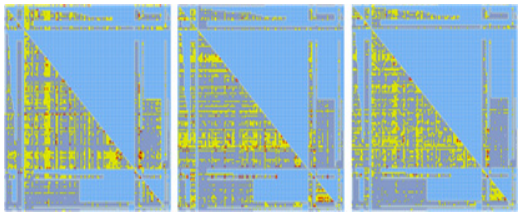
실험을 위해 GSIM 도구를 구현하였다. 코맨드라인(Command line)으로 실행가능하며, 앞서 표에서 설명하였던 tough, threshold, ingoreLength, outMatrics, mType의 5개의 옵션을 지원한다. 검색할 화일들이 모여 있는 폴더와 확장자는 필수입력 사항이다.

[그림 5]는 mType = pf로 출력한 결과 중 일부이다. x축은 각각의 patom들을 나타내고, y축은 file을 나타낸다. 그러므로 그림에서 표시된 것처럼 horizontal line이 유사한 묶음으로 나타나는 것은 해당 file들이 유사한 patom들의 묶음을 가지고 있다는 의미가 된다. 특히, 색이 짙을 수록 patom의 개수가 많다는 의미인데, 색의 분포까지 비슷하다는 것은 해당 patom들로 구성된 코드 조각을 공유한다는 의미가 된다. 하지만, 전체 patom들 중에서 유사한 patom들의 비율을 알 수는 없기 때문에, file들간의 유사도를 확인하기 위해서는 mType = ff인 [그림 6]를 보는 것이 더 낫다. 하지만, 어차피 클러스터링된 파일(clustered file)들의 리스트는 [그림 7]과 같이 표준 출력(standard output)으

로 화면에 표시되기 때문에, 이를 참고해도 된다. [그림 6]에서는 회색영역은 유사 patom이 없는 경우이고, 노란색은 공통 patom이 존재하는 경우이고, 붉은 색이 threshold를 넘긴 경우이다. 첫 번째, 두 번째 것은 각각의 화일에 대해서만 비율을 비교한 것이고, 3번째는 2개 모두 threshold를 넘길 경우에 대해서만 붉은색으로 표시된 결과이다. 1, 2번째 붉은색이지만, 3번째 사라진 쌍들은 포함관계를 의심해 볼 수 있으며, Threshold의 조절에 따라 붉은색의 비중이 바뀌게 된다.



[그림 5] mType = bf인 경우, GSIM 메트릭스 일부



[그림 6] mType = ff인 경우, GSIM 메트릭스 (임계치 이상은 붉은색)

GSIM은 tough, threshold, ignoreLength 등의 다양한 옵션을 제공함으로써 precision과 recall을 정밀하게 조절할 수 있다. 하지만, 상황에 맞는 최적화된 설정값을 찾기 위한 노력이 필요하다. 우선은 조건을 높게 하여, 유사도가 높은 것부터 시작하고 조건을 완화하면서 더 낮은 유사 페이지들을 찾아보는 것이 좋다. 특정 수준 이하의 유사 페이지들이 등장하게 되면 작업을 중단한다.

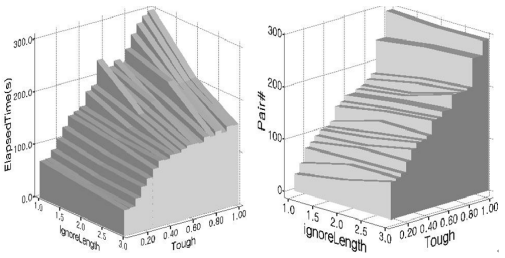
```
Path:       D:₩Downloads₩mediawiki-1.14.0.tar
Ext:        php
Tough:      0.5
Threshold:          0.5
Ignore Length:      2

D:₩Downloads₩mediawiki-1.14.0.tar₩mediawiki-1.14.0₩languages₩messages₩MessagesZh_hant.php
            46023/80842     =       0.569295663145395       ,       3417/4539       =
            0.562099029445633
D:₩Downloads₩mediawiki-1.14.0.tar₩mediawiki-1.14.0₩languages₩messages₩MessagesZh_tw.php
            46023/80842     =       0.784746022814466       ,       3417/4539       =
            0.752808988764045

D:₩Downloads₩mediawiki-1.14.0.tar₩mediawiki-1.14.0₩languages₩messages₩MessagesZh_hans.php
            45593/83029     =       0.549121391321105       ,       3003/6079       =
            0.473808772483433
D:₩Downloads₩mediawiki-1.14.0.tar₩mediawiki-1.14.0₩languages₩messages₩MessagesZh_hant.php
            45593/83029     =       0.563976645802924       ,       3003/6079       =
            0.493995722980753
...
Gathering Elapsed Time:     00:00:54.7910188
Making Elapsed Time:        00:01:44.3127660
Calculating Elapsed Time:   00:00:00.7995527
Total Elapsed Time:         00:02:39.9033375
```

[그림 7] 유사도가 높은 파일 쌍들의 목록 출력

4.2 실험

GSIM의 유용성을 보이기 위해 실험을 실시하였다. 실험 대상은 Mediawiki-1.14.0[19]으로 php로 작성된 오픈소스이다. 소스의 크기는 38.7MB이고, 총 1420개 파일로 구성되어 있다. 임계치를 0.5로 잡고 실험을 해본 결과 threshold와 tough의 옵션에 따라 1~5분 정도 소요되었다. 실험은 2G메모리의 Pentium core2duo에서 실시하였다. 앞서 보인 [그림 5]는 이 실험의 ff 메트릭 결과이다.

[그림 8]은 실험 옵션에 따른 성능 및 pair#의 개수를 보여준다. 임계치가 0.5로 고정되어 있기 때문에, tough가 감소하면서 pair#쌍이 줄어든 결과를 보였다. Tough가 줄어들게 되면, 비교기준 대상의 patom 전체 개수가 줄어들기 때문에 threshold를 같이 줄여주어야 한다. 여기서는 ignoreLength의 영향에 따른 pair#의 개수 변화가 거의 없다는 것에 주목해야한다. 이는 아주 짧은 patom 조각들은 유사도 측정에 거의 영향을 주지 않고 있다는 것을 의미한다. 하지만, 이러한 의미없는 patom 조각들을 비교 대상에서 제외하였을 경우에 실험 시간은 상당히 줄어들고 있음을 확인할 수 있다. 이는 tough 값이 큰 경우 더욱 뚜렷해진다. tough가 1일 경우 300초에서 150초 정도로 측정 시간은 거의 절반으로 줄어들었으나, pair# 개수는 거의 변함이 없음을 알 수 있다.



(a) 성능 (b) 쌍의 개수

[그림 8] 옵션에 따른 성능 및 쌍의 개수

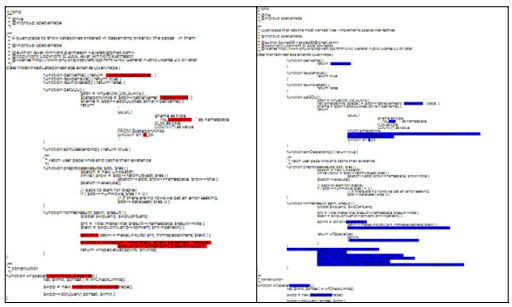
다음은 실험에서 얻어진 실제 페이지 소스의 일부를 정리한 것이다. Match는 각각의 비교 화일을 f1, f2라고 하고, |f1|, |f2|를 해당 화일의 크기(size), f1∩f2을 두 개 화일의 공통 코드라고 할 때, |f1∩f2|/(|f1|+|f2|)로 구한 것이다.

〈표 1〉 실험에서 사용한 파라미터 및 결과

#	parameters	results	match
test 1	threshold 0.8 tough 0.8 ignoreLength 1	/skins/MySkin.deps.php /skins/Check.deps.php /skins/Simple.deps.php	100%
		/skins/Modern.deps.php /skins/MonoBook.deps.php	100%
test 2	threshold 0.5 tough 0.1 ignoreLength 3	Test1 Results	100%
		/languagesfolder files	82.6%
test 3	threshold 0.5 tough 0.5 ignoreLength 1	/includes/specials/SepcialMostlinkcategories.php /includes/specials/SpecialWantedfiles.php	81.7%
		…	…

이에 비해 threshold 기준이 다소 높았던 실험 1의 경우는 검출된 페이지들이 100% 일치했으며, tough 값이 실험 3에 비해 낮았던 실험 2의 경우는 실험 1의 내용을 포함하고, 추가적으로 languages 폴더상의 파일들을 검출해냈다.

[그림 9]는 Test 3에서의 실제 코드사례이다. 붉은색과 푸른색으로 표시된 부분을 제외한 나머지 코드는 포멧 등만 틀릴 뿐 실제 내용은 100% 일치한다. 아래와 같은 코드는 중복된 부분이 많기 때문에 리팩토링의 후보가 될 수 있다. 이는 각각의 조각들을 클론으로 찾아서 살피기 보다는 페이지 단위에서 살펴보는 것이 바람직해 보이는 좋은 예이다.



[그림 9] 실험 3에서 검출된 유사 페이지의 예

5. 결 론

본 연구에서 제안된 GSIM은 웹 페이지 구현에 사용한 언어를 엄격하게 파싱(parsing)하지 않기 때문에 웹 애플리케이션 개발 언어에 독립적으로 페이지를 비교하고, 유사도를 효율적으로 계산할 수 있다. 구체적으로, 정확한 파싱이 아니라 구분자에 기반하여 분해된 특정 길이 이상의 토큰들의 분포를 비교함으로써 유사도를 빠르게 비교할 수 있다. 이는 여러 언어들이 뒤섞여 있는 웹 페이지의 유사도 측정을 위해 별도의 비용을 사용하지 않음으로서 보다 효과적으로 분석할 수 있음을 의미한다. GSIM으로 검출된 중복 페이지는 동적 또는 정적 페이지 여부에 상관없이 리팩토링을 통해 하나의 구현으로 통합시킬 수 있다. 이는 여러 개의 유사 페이지들을 중복적으로 가지고 있는 경우보다 소스 코드의 일관성을 유지하기 위해 필요한 작업이다. 결국 GSIM은 다양한 언어가 뒤섞여 있는 페이지 분석 과정에서의 파싱의 어려움과 비용을 줄이고 보다 효율적인 방식으로 유사 페이지를 검출할 수 있으며, 이를 통해 페이지 코드의 복잡도를 줄이고, 코드의 품질 향상시켜 유지보수비용을 줄이는데 기여할 수 있다.

희소성이 낮아서 거의 대부분의 코드에 존재하는 patom 조각들의 경우는 단순 유사도 비교 단계에서는 무시되지만, 거의 모든 페이지 상에 존재하는 patom 조각들은 주로 특정 명령어에서 사용하는 제어 명령과 관련한 키워드일 가능성이 많다. 제어 명령의 구조는 코드 클론에서 보이는 중요한 유사성 중에 하나이기 때문에, 일반적으로 많이 등장하는 patom 조각들에 대해서는 DECAKRD에서 처럼 벡터를 만들어서 거리를 비교하거나 순서 및 개수등을 비교함으로써 보다 정교한 유사도 비교가 가능할 것이다. 즉, 희소성이 높은 것뿐 아니라 낮은 Patom을 활용하여 유사도의 품질을 개선할 가능성이 있다. 이는 높은 Patom만을 고려하여 유사 페이지를 검출한 본 연구의 범위를 벗어나지만, 유사도 측정 결과의 품질 향상을 위한 좋은 방향으로 향후 연구과제에 해당한다.

참 고 문 헌

[1] Aversano, L., G. Canfora, A. De Lucia, and P. Gallucci, “Web Site Reuse : Cloning and Adapting”, *Proc. of the 3rd Int’l Workshop on Web Site Evolution*, (2001), pp.107-111.

[2] Boldyreff, C. and R. Kewish, “Reverse Engineering to Achieve Maintainable WWW Sites”, *Proc. of 8th Working Conf. on Reverse Eng.*, (2001), pp.249-257.

[3] Calefato, F., F. Lanubile, and T. Mallardo, “Function Clone Detection in Web Applications : A Semiautomated Approach,” *Journal of Web Engineering*, Vol.3, No.1(2004), pp.3-21.

[4] Duala-Ekoko, E. and M. P. Robillard, “Clone Tracker : Tool Support for Code Clone Management,” *Proc. of Int’l Conf. on Software Eng.*, (2008), pp.843-846.

[5] Higo, Y., T. Kamiya, S. Kusumoto, and K. Inoue, “Aries : Refactoring Support Environment Based on Code Clone Analysis,” *Proc. of the 8th IASTED Int’l Conf. on Software Eng., and Applications*, (2004), pp. 222-229.

[6] Jiang, L., G. Misherghi, Z. Su, and S. Glondu, “DECKARD : Scalable and Accurate Tree-based Detection of Code Clones”, *Proc. of Int’l Conf. on Software Eng.*, (2007), pp.96-105.

[7] Kamiya, T., S. Kusumoto, and K. Inoue, “CCFinder : A multilinguistic token based code clone detection system for large scale source code”, *IEEE Trans. Software Engineering*, Vol.28, No.7(2002), pp.654-670.

[8] Kim, M., V. Sazawal, D. Notkin, and G. C. Murphy, “An Empirical Study of Code Clone Genealogies,” *Proc. of the Joint European Software Eng. Conf. and ACM SIG SOFT Symposium on the Foundataion of Software Eng.*, (2005), pp.187-196.

[9] Lanubile F. and T. Mallardo, “Finding Function Clones in Web Applications”, *Proc. of the 7th European Conf. on Software Maintenance and Reeng*, (2003), pp.379-386.

[10] Levenshtein, V. L., “Binary Codes Capable of Correcting Deletion, Insertions, and Reversals”, *Cynernetics and Control Theory*, Vol.10(1966), pp.290-299.

[11] Li, Z., S. Lu, S. Myagmar, and Y. Zhou, “CP-Miner : A Tool for Finding Copy-paste and Related Bugs in Operating System Code,” *Proc. of Operating System Design and Implementation*, (2004), pp.289-302.

[12] Li, Z., S. Lu, S. Myagmar, Y. Zhou, “CP-Miner : Finding copy-paste and related bugs in large-scale software code”, *IEEE Trans. on Software Eng.*, Vol.32, No.3(2006), pp.176-192.

[13] Lin, Z., M. Lyu, and I. King, "PageSim : A Novel Link-based Measure of Web Page Similarity", *Proc. of World Wide Web*, (2006), pp.1019-1020.

[14] Di Lucca, G. A., M. Di Penta, and A. R. Fasolino, "An Approach to Identify Duplicated Web Pages", *Proc. of the 26th Annual Int'l Computer Software and Applications Conference*, (2002), pp.481-486.

[15] De Lucia, A., G. Scanniello, and G. Tortora, "Identifying Clones in Dynamic Web Sites Using Similarity Thresholds", *Proc. of Int'l Conf. on Enterprise Information Systems*, (2004), pp.391-396.

[16] De Lucia, A., R. Francese, G. Scanniello, and G. Tortora, "Reengineering Web Applications Based on Cloned Pattern Analysis", *Proc. of the 12th Int'l Workshop on Program Comprehension*, (2004), pp.132-141.

[17] De Lucia, A., R. Francese, G. Scanniello, and G. Tortora, "Understanding Cloned Patterns in Web Applications", *Proc. of 13th Int'l Workshop on Program Comprehension*, (2005), pp.333-336.

[18] De Lucia, A., G. Scanniello, and G. Tortora, "Identifying Cloned Navigational Patterns in Web Applications", *Int'l Journal of Web Eng.*, Vol.5, No.2(2006), pp.150-174.

[19] Mediawiki, http://www.mediawiki.org/.

[20] Ricca, F. and P. Tonella, "Using Clustering to Support the Migration from Static to Dynamic Web Pages", *Proc. of Int'l Workshop on Program Comprehension*, (2003), pp.207-216.

[21] Synytekyy, N., J. R. Cordy, and T. Dean, "Resolution of Static Clones in Dynamic Web Pages", *Proc. of Int'l Workshop on Web Site Evolution*, (2003), pp.49-56.

◈ 저 자 소 개 ◈

이 은 주 (ejlee@knu.ac.kr)

서울대학교 전기컴퓨터공학부에서 박사학위를 취득했으며, 현재 경북대학교 IT대학 컴퓨터학부 부교수로 재직 중이다. 주요 관심분야는 웹 공학, 소프트웨어 리파지토리 마이닝, 소프트웨어 유지보수 등이다.

정 우 성 (wsjung@cbnu.ac.kr)

서울대학교 전기컴퓨터공학부에서 박사학위를 취득했으며, LG전자 선임연구원 등을 지내고 현재 충북대학교 전자정보대학 컴퓨터공학과 조교수로 재직 중이다. 주요 관심분야는 소프트웨어 마이닝, 웹공학, 소프트웨어 진화, 소프트웨어 아키텍처 등이다.