RDF/S 및 OWL 문서에 대한 키워드 검색 알고리즘[1]

김학수, 손진현
한양대학교 컴퓨터공학과
e-mail : {hsookim,jhson}@hanyang.ac.kr

A New Keyword Search Algorithm for RDF/S and OWL Documents

Hak Soo Kim, Jin Hyun Son
Department of Computer Science and Engineering, Hanyang University in Ansan

요 약

XML 또는 RDBMS 에서의 키워드 검색은 기존의 정보 검색처럼 데이터의 구조 또는 질의 언어에 대한 사전 지식 없이 질의 처리를 수행하는 연구 분야 중의 하나이다. 오늘날 키워드 검색을 효율적으로 처리하기 위해 제안된 연구들은 그래프 기반의 질의 처리에 기반한 기법들에 초점을 두고 있다. 이러한 접근들은 XML 또는 RDBMS 안에 존재하는 데이터를 그래프 구조에 기반한 데이터로 변환한 다음에 그래프 탐색을 통해서 모든 질의 키워드를 포함하는 결과들을 찾는다. 그러나 기존의 기법들을 RDF/S 또는 OWL 문서와 같은 복잡한 그래프 구조에 적용하기에는 질의 성능 측면에서 많은 문제점을 가지고 있다. 또한, 온톨로지 언어의 의미적 단위로서의 RDF 트리플을 고려하지 않기 때문에 질의 결과에 대한 신뢰성을 보장할 수 없다. 이러한 관점에서 본 논문은 RDF/S 또는 OWL 저장소에서 효율적이고 의미적인 키워드 검색을 위한 인덱싱 기법 및 알고리즘을 설계한다.

1. 서론

일반적으로 키워드 검색은 정보 검색 분야에서 사용되는 용어로서 사용자가 키워드의 나열을 통해서 나열된 키워드를 포함하고 있는 문서를 검색하는 질의 처리 기법이다. 이와 같은 검색 기법은 사용자가 복잡한 질의 언어 또는 데이터의 구조에 대한 사전 지식을 요구하지 않는 장점을 가지고 있다. 오늘날 이러한 장점은 관계형 데이터베이스 [1,2] 또는 XML 데이터베이스[3,4,5,6]에서의 연구로 확장되고 있는 시점이다.

RDF/S 및 OWL 문서는 XML 구조를 가지고 있지만 기존의 방식을 적용할 수 없는 문제점을 가지고 있다. 기존 방식의 대부분은 XML 트리로 보고 질의 처리가 이루어지거나 그래프 구조를 가지고 있더라도 질의 처리를 위한 인덱스의 유지 관리 비용이 크다. 하지만 RDF/S 와 OWL 같은 온톨로지 문서는 복잡한 구조를 가지는 그래프이기 때문에 기존의 기법을 적용할 경우 질의 성능 및 인덱스의 유지보수 비용을 보장할 수 없는 문제점을 가지고 있다. 또한 기존 방식의 데이터는 데이터와 데이터 사이의 관계가 미약하다는 것이다. 즉, 온톨로지 문서 같은 경우에는 데이터와 데이터 사이의 관계를 기술해주기 위한 서술어(Predicate)를 기술할 수 있다. 예를 들면, "제인의 친구는 존이다"에 대해서 온톨로지 문서는 트리플(Triple)로서 기술이 되는데 주어로서 "제인", 서술어로서 "친구", 목적어로서 "존"으로 기술할 수 있다. 이와 같이 온톨로지 문서는 데이터와 데이터 사이의 관련성이 관계형 데이터베이스 또는 XML 데이터베이스에서의 데이터보다 의미적으로 밀접한 관계가 있기 때문에 키워드 검색 시 이러한 관계를 고려하도록 설계되어야만 한다.

위와 같은 문제점을 해결하기 위해서 본 논문은 RDF/S 및 OWL 에서 의미적 기술의 최소 단위인 트리플을 키워드 검색에서 고려하기 위해서 상위 트리플 인덱스(Upper Triple Index)를 구축한다. 이를 통해서 본 논문은 트리 기반의 경로 병합 알고리즘인 T-PM(Tree-based Path Merging)을 제안할 것이다.

본 논문의 구성은 다음과 같다. 2 장에서는 제안한 인덱스 구조 및 키워드 검색 알고리즘을 설명한다. 마지막으로 3 장에서는 결론 및 향후 계획으로 마무리한다.

2. 인덱스 구조 및 키워드 검색 알고리즘

이번 장에서는 RDF/S 및 OWL 저장소에서의 키워드 검색을 위한 인덱스 구조 및 키워드 검색 알고리즘을 제안한다. 대용량의 RDF/S 및 OWL 문서를 유지관리 하기 위한 저장소에 대한 연구는 [7] 에서 분석한 시스템에서 진행되어 왔기 때문에 본 논문에서는 상세히 다루지 않는다. 다만, 이전 연구에서 핵심이 되는 RDF 트리플 (주어:subject, 서술어:predicate,

[1] 이 논문은 2007 년도 정부재원(교육인적자원부 학술연구조성사업비)으로 한국학술진흥재단의 지원을 받아 연구되었음 (KRF-2007-313-D00747).
이 논문은 2007 년도 정부(과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임 (No.R01-2007-000-20135-0).

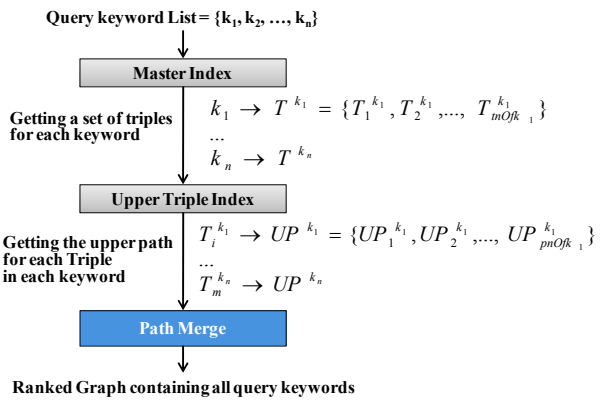
목적어:object) 구조를 관계형 데이터베이스의 테이블 형태로 유지한다는 것이기 때문에 본 논문은 키워드 검색의 대상이 되는 테이블을 RDF 트리플 테이블로 가정한다. 즉, 다시 말해서 본 논문에서 제안하는 알고리즘은 RDF 트리플 구조 기반의 온톨로지 저장소에 쉽게 적용할 수 있는 장점이 있다.

RDF/S 및 OWL 저장소에서의 키워드 검색의 결과는 모든 질의 키워드를 단 한번씩만 포함하는 최소 그래프이며, 그림 1 은 구축된 인덱스로부터 키워드 검색이 이루어지는 과정을 보여준다. 이를 통해서 실제로 질의 처리 과정을 보면서 인덱스 구축과정을 설명할 것이다.

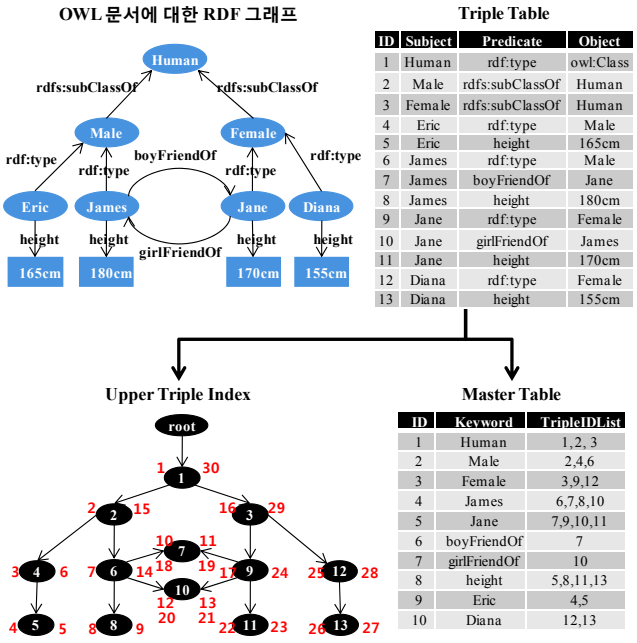
사용자가 입력하는 질의 키워드(Query Keyword)가 n 개 들어오면은 마스터 인덱스(Master Index)로부터 각 키워드를 포함하는 RDF/S 와 OWL 에 있는 트리플 아이디 리스트를 그림 1 에서 보는 것처럼 가지고 온다. 그 다음에 상위 트리플 인덱스(Upper Triple Index)로부터 각 키워드를 포함하는 트리플 아이디 리스트로부터 그래프의 루트까지의 경로인 상위 경로(Upper Path)를 검색하게 된다. 마지막으로 검색된 각 키워드에 대한 상위 경로를 이용하여 경로 병합(Path Merge)을 진행하여 최종적으로 사용자가 입력한 모든 질의 키워드를 포함하는 그래프를 순서대로 리턴 한다.

질의 키워드가 포함될 수 있는 위치 노드는 RDF 트리플의 주어, 서술어, 목적어에 있지만 RDF/S 및 OWL 문서를 기술하는 최소 단위는 RDF 트리플이다. 다시 말해서 의미적 최소 단위가 RDF 트리플이기 때문에 질의 키워드를 포함하는 단일 노드를 RDF 트리플로 정의하는 것이 의미 있는 질의 결과를 줄 수 있다.

마스터 인덱스는 관계형 데이터베이스 기반의 키워드 검색에서 제안했던 인덱스로서 정보 검색의 역 인덱스(Inverted Index)와 같은 구조이다. 즉, 키워드를 주키(Primary Key)로 하여 키워드를 포함하는 트리플을 저장하는 구조로 되어 있다. 결과적으로, 키워드를 포함하는 트리플을 빠른 시간 안에 검색할 수 있는 장점이 있기 때문에 이러한 구조가 널리 사용되고 있다. 그림 1 의 마스터 인덱스에 의한 결과는 사용자 키워드 k_1 부터 k_n 에 대해서 T^{k_1}부터 T^{k_n}까지의 트리플 리



(그림 2) 키워드 질의 처리 과정



(그림 1) 상위 트리플 인덱스 및 마스터 인덱스를 위한 마스터 테이블

스트이다.

상위 트리플 인덱스는 마스터 인덱스로부터 검색된 트리플 리스트로부터 RDF 그래프의 최상위 루트까지의 경로를 검색하기 위해 사용된다. 예를 들면 $T_i^{k_1}$으로부터 상위 트리플 인덱스에서 범위 질의(Range Query)를 통해서 빠르게 $UP_j^{k_1}$를 검색한다. 상위 경로를 검색하는 이유는 키워드 검색의 결과가 모든 키워드를 포함하는 그래프를 찾아내는 것이기 때문에 결과 그래프를 생성할 때 필요하다.

그림 2 는 OWL 문서에 대한 RDF 그래프로부터 추출된 RDF 트리플을 저장하는 트리플 테이블(Triple Table)로부터 상위 트리플 인덱스(Upper Triple Index)와 마스터 테이블(Master Table)을 생성하는 예를 나타낸다. 트리플 테이블은 RDF/S 와 OWL 저장소로서 가장 기본이 되는 테이블 구조이다. RDF/S 와 OWL 문서는 RDF 트리플의 집합으로 표현할 수 있기 때문에 이러한 구조는 기존의 온톨로지 저장소에서 일반적으로 많이 사용하는 구조이다.

마스트 테이블은 마스터 인덱스를 나타내는 것으로서 정보 검색 분야에서 다루어지는 역 인덱스와 같은 구조이다. 테이블 스키마는 (ID, Keyword, TripleIDList)로 되어 있다. 이러한 구조는 입력으로 들어오는 키워드로부터 키워드를 포함하는 트리플을 빠르게 검색하기 위한 것이다. 예를 들면 Human 을 포함하는 트리플은 "1,2,3" 으로 인덱스 검색을 통해서 빠르게 검색할 수 있다. 만약에 마스터 인덱스가 없다면 모든 RDF 트리플을 검사해야만 한다.

마스터 인덱스가 사용자 질의 키워드를 포함하는 트리플 아이디를 빠르게 검색하는 것이라면 상위 트리플 인덱스는 검색된 트리플 아이디로부터 상위 경로를 빠르게 검색하기 위한 인덱스이다. 마스터 인덱스와 마찬가지로 상위 트리플 인덱스 또한 트리플 테

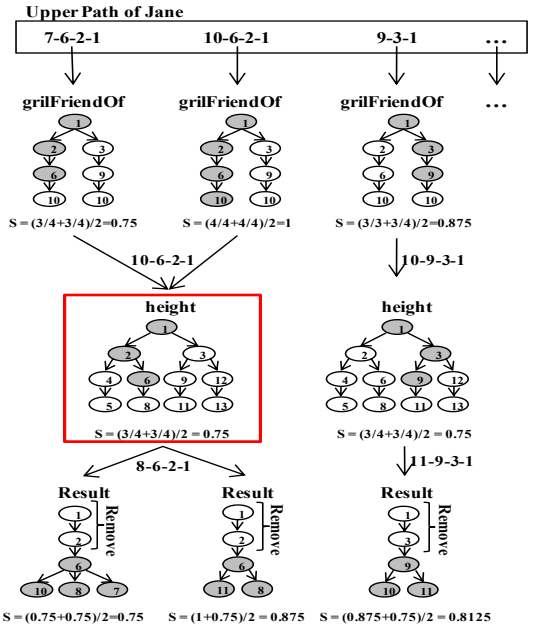
이블로부터 생성된다. 각각의 트리플은 트리플 테이블에서 유일한 아이디를 가지고 있으며 상위 트리플 인덱스는 트리플 아이디를 노드로 가지면서 루프가 없는 그래프 구조로 되어 있다. 트리플들 사이의 연관성은 주어와 서술어만을 고려하며 주어와 서술어에 기술된 클래스 또는 인스턴스를 정의한 트리플만을 고려한다. 다시 말해서, 속성(Predicate)가 "rdf:type"인 것을 참조 대상으로 한다. 예를 들면, 트리플 아이디 (7, James, boyFriendOf, Jane)의 경우에 주어와 목적어인 James, Jane 을 정의한 트리플 6, 9 와 연관이 있게 된다. 따라서 트리플 아이디 6, 9 번으로부터 노드 7 번이 구성되었기 때문에 그림 2 의 상위 트리플 인덱스에서 보는 것처럼 화살표 방향이 7 번으로 들어오는 형태로 된 것이다.

상위 트리플 인덱스로부터 빠르게 상위 경로를 검색하기 위해서 XML 문서에서 XPath 질의 처리를 위한 인덱스 기법[8]을 확장한 Multi-Numbering Scheme 을 통하여 각 노드에 숫자를 부여한다.

기존의 Numbering Scheme 는 XML 노드의 부모/자식, 조상/자손, 형제 관계를 빠르게 질의하기 위한 인덱스로서 각 XML 노드는 (시작 번호:start, 종료 번호:end)로 구성되어져 있다. 하지만 Numbering Scheme 을 적용할 수 있는 XML 문서는 트리 형태일 경우에 가능하다. 이러한 제한 때문에 본 논문에서는 각 노드가 다수의 Numbering Scheme 를 가질 수 있도록 확장하였다. 상위 트리플 인덱스는 루프가 없는 그래프이기 때문에 노드가 다수의 Numbering Scheme 을 가질 수 있게 변형이 가능하다. 예를 들면, 그림 2 의 상위 트리플 인덱스에서 보는 것처럼 노드 7 번 같은 경우에 Numbering Scheme 이 (10,11), (18,19) 두 개를 가지고 있다. 따라서, 7 번 노드의 상위 경로는 "1-2-6-7"과 "1-3-9-7" 두 개가 나올 수 있다. 이 두 경로를 찾아내기 위해서 7 번 노드의 Numbering Scheme (10,11)에 의해서 start 가 10 보다 작고 end 가 11 보다 큰 노드를 찾으면은 6(7,14), 2(2,15), 1(1,30)을 범위 질의에 의해서 빠르게 찾을 수 있다. 또한 (18,19)에 대해서도 마찬가지 방법으로 범위 질의를 하면 9(17,24), 3(16,29), 1(1,30)을 찾을 수 있다.

3. T-PM 경로 병합 알고리즘

2 장에서 본 것처럼 마스터 인덱스와 상위 트리플 인덱스가 구축된 다음에 그림 1 의 키워드 검색 절차에 따라서 최종 질의 결과를 가지고 온다. 예를 들면 질의 키워드가 "Jane, girlFriendOf, height"라고 했을 때 마스터 인덱스로부터 MI(Jane) = {7,9,10,11}, MI(girlFriendOf) = {10}, MI(height) = {5,8,11,13}을 검색한다. 그런 다음에 각 키워드로부터 검색된 트리플 아이디로부터 상위 경로를 상위 트리플 인덱스로부터 검색한다. 즉, UP^{Jane} = {7-6-2-1, 9-3-1, 10-6-2-1, 7-9-3-1(X), 10-9-3-1(X), 11-9-3-1(X)}, $UP^{girlFriendOf}$ = {10-9-3-1, 10-6-2-1}, UP^{height} = {5-4-2-1, 8-6-2-1, 11-9-3-1, 13-12-3-1} 와 같이 각 키워드에 대한 상위 경로를 검색한다. 이때 X 표시는 상위 경로에 포함되지 않음을 나타낸다. 왜냐하면 X 가 표시된 경로는 "Jane" 키워드를 포함하는 노드가 2 개 이상 포함되기 때문이다.



(그림 3) 경로 병합의 예

위와 같이 상위 경로가 계산되면 그 경로 들에 대한 경로 병합이 이루어져야 최종 질의 결과를 구할 수 있다. 그림 3 은 "Jane" 키워드의 상위 경로를 첫 번째 경로 병합 대상으로 선정하였을 경우이다. "Jane" 키워드의 상위 경로와 병합 대상이 되는 그 다음 키워드의 상위 경로는 키워드 질의 입력 순서대로 진행된다. 왜냐하면 앞 키워드 일수록 중요도가 높기 때문이며 경로 병합이 상위 경로 사이의 유사도를 계산하여 상위 경로를 선택할 경우에 키워드 우선 순위와 가장 유사한 그래프를 질의하기 때문이다. 우선, "Jane" 키워드의 첫 번째 상위 경로인 "7-6-2-1"을 선택하고 그 다음에 "girlFriendOf"의 상위 경로들과 경로 병합을 진행한다. 이때 경로 병합을 효율적으로 하기 위해서 상위 경로들에 대한 트리를 구성하게 된다. 그림 3 과 같이 트리를 구성함으로써 빠르게 유사도가 가장 높은 상위 경로를 선택할 수 있다. 본 논문에서 제안한 유사도 함수는 아래와 같다.

$$Similarity\ S = \frac{\left(\frac{the\ number\ of\ the\ same\ node}{Path_i\ length} + \frac{the\ number\ of\ the\ same\ node}{Path_k\ length}\right)}{2}$$

유사도 함수는 선택된 상위 경로와 그 다음 키워드의 상위 경로중에서 가장 유사한 상위 경로를 선택하도록 하는 함수이다. 그림 3 에서 보는 것처럼 "7-6-2-1"과 가장 유사한 상위 경로는 "girlFriendOf" 키워드의 상위 경로인 "10-6-2-1"을 선택할 수 있다. 실제로 트리 검색을 통해서 자동으로 유사도가 가장 높은 상위 경로를 선택하게 된다. 또한 유사도는 최종 그래프에서 그래프들 사이의 랭킹을 계산시에 사용된다. 이렇게 그 다음 상위 경로가 선택되면 "10-6-2-1"과 "height" 키워드의 상위 경로와 경로 병합을 한다. 마찬가지 방법으로 "height" 키워드의 상위 경로 트리로부터 "8-6-2-1"이 선택된다. 마지막으로 지금까지 선택된 상위 경로들을 병합하게 되면 그림 3 과 같이 질

의 결과 그래프가 얻어진다. 이때 병합된 상위 경로 중에서 중복되는 노드는 제거한다. 질의 결과 그래프는 지금까지 경로 병합을 하면서 계산된 유사도 합의 평균을 랭킹 점수로 계산한다. 그리고 다시 “Jane”의 상위 경로 중에 그 다음 상위 경로를 선택하여 위에서 설명한 과정을 반복하게 된다. 이때 트리 검색중에 중복이 되는 검색이 있을 수 있기 때문에 그림 3 에서 보는 것처럼 캐쉬 기법을 통해서 계산 과정을 공유함으로써 효율적인 경로 병합을 할 수 있다.

마지막으로 모든 상위 경로에 대한 경로 병합이 완료되면 랭킹 점수를 가지는 그래프들이 계산되고 랭킹 점수가 가장 높은 그래프 순으로 정렬한다. 그런 다음에 노드 아이디는 트리플 아이디를 나타내기 때문에 그 노드 부분을 트리플로 변환함으로써 질의 키워드를 모두 포함하는 최종 그래프를 얻을 수 있다.

제안된 키워드 검색은 모든 질의 결과 그래프를 계산함을 보장해야만 한다. 그림 3 의 예제에서는 실제로 모든 질의 결과 그래프를 찾아내지 못한다. 왜냐하면 “Jane” 키워드의 상위 경로부터 경로 병합이 진행되면 그래프 {10-9-3,13-12-3}, {5-4-2,10-6-2}을 찾아내지 못한다. 이러한 문제를 해결하기 위해서 경로 병합의 과정은 키워드에 대한 상위 경로가 UP^{k1} 부터 UP^{kn} 까지 있을 때 가장 많은 상위 경로가 있는 키워드의 상위 경로부터 경로 병합을 진행하게 된다. 첫 번째 경로 병합의 대상을 이렇게 선택하는 이유는 가능한 모든 질의 결과를 계산하기 위해서이다. 질의 결과는 모든 키워드가 단 한번씩만 포함하는 그래프이기 때문에 최대 그래프 개수는 키워드에 대한 상위 경로의 개수를 초과하지 않기 때문이다. 지금까지 설명한 경로 병합에 대한 알고리즘은 있는 **Function pathMerge** 에서 볼 수 있다.

4. 결론 및 향후 연구 계획

지금까지 RDF/S 및 OWL 와 같은 온톨로지 저장소에서의 키워드 검색을 지원하기 위한 인덱스 구조 및 키워드 검색 기법에 대해서 알아보았다. 또한 트리 기반의 경로 병합 알고리즘인 T-PM 을 제안함으로써 효율성을 향상 시켰다.

향후 연구 계획으로는 온톨로지가 가지고 있는 시맨틱적 요소인 owl:inverseOf, owl:sameAs, owl:TransitiveProperty 등을 고려한 키워드 검색 기법으로 확장할 것이다.

참고문헌

[1] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou, "DISCOVER: Keyword Search in Relational Databases", *Proceedings of the 29th VLDB Conference*, pp. 670-681, 2003.

[2] Agrawal Sanjay, Chaudhuri Surajit, and Das Gautam, "DBXplorer: A System for Keyword-based Search over Relational Databases", *Proceedings of IEEE 18th International Conference on Data Engineering*, pp. 5-16, 2002.

[3] Hao He, Haixun Wang, Jun Yang, and Philip S. Yu, "BLINKS: Ranked Keyword Searched on Graphs", *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pp. 305-316, 2007.

[4] Yunyao Li, Cong Yu, and H. V. Jagadish, "Schema-Free XQuery", *Proceedings of the 30th VLDB Conference*, pp. 72-83, 2004.

[5] Yu Xu and Yannis Papakonstantinou, "Efficient Keyword Search for Smalledst LCAS in XML Databases", *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pp. 527-538, 2005.

[6] Vagelis Hristidis and Yannis Papakonstantinou, "Keyword Proximity Search in XML Trees", *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 4, pp. 525-539, 2006.

[7] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin, "LUBM: A Benchmark for OWL Knowledge Base Systems", *Journal of Web Semantics*, vol. 3, no. 2, pp. 158-182, 2005.

[8] Shurug Al-Khalifa et al., "Structural Joins: A Primitive for Efficient XML Query Pattern Matching", *Proceedings of the 18th International Conference on Data Engineering*, pp. 141-152, 2002.

부록

Function pathMerge(TargetUP, UP(k_i) … UP(k_n), direction)
Input : the selected upper path,
the upper paths of comparing keywords
Output : the merged graph for the merged paths

```
01  var midSim = 0.0
02  var mergedGraph

03  tmpUP
      ← getUPWithMaxSimInT-PM (targetUP, UP(ki), midSim)

04  if tmpUP == null
05    return null
06  end if

07  if i == n
08    insert tmpUP into mergedGraph
09  else if
10    insert tmpUP into mergedGraph
11    if direction == 0
12      insert pathMerge(tmpUP, UP(ki+1) … UP(kn))
              into mergedGraph
13    else if
14      insert pathMerge(tmpUP, UP(ki-1) … UP(k1))
              into mergedGraph
15    end if
16  end if
17  return mergedGraph
```

Function getUPWithMaxSimInT-PM(targetUP, UP(k_i))
Input : the selected upper path,
the upper paths for the keyword k_i
Output : the upper path with maximum similarity

```
01  var sim
03  var tree
04  var currentNode
05  var index
06  var maxSimUP

07  sim ← 0.0
08  index ← the number of nodes in targetUP
09  tree ← construct(UP(ki))
10  currentNode ← tree.root

11  while nextNode != null or index < 1
12    if( nextNode == targetUP.getNode(index) )
13      maxSimup ← maxSimup + currentNode
14      currentNode ← currentNode.nextNode
15      sim++
16    end if
17    index--
18  end while
19  return maxSimUP
```