韓國 OA學會 論文誌
第7卷 第3號, 2002. 9.

2002-7-3-1-3

다중 패싯값과 다중 패싯을 위한 컴포넌트의 효율적인 검색 방법

금 영 욱*

An efficient Component Retrieval Scheme for multiple facet values and multiple facets

Young-wook Keum*

요 약

컴포넌트의 효율적인 검색은 컴포넌트에 기반한 소프트웨어 개발에 필수적이다. 패싯 방식은 컴포넌트 검색 방법의 하나로 많은 연구의 대상이다. 이 논문에서 여러 개의 패싯값에 대한 논리 부정 검색에 사용되는 가중치 신경 접속 행렬을 효율적으로 만드는 새로운 알고리즘을 제안한다. 이 알고리즘을 사용하여 연산에 드는 복잡도를 향상할 수 있다. 또한 여러 개의 서로 다른 패싯을 사용하는 경우 이에 대한 논리적인 검색이 가능하도록 새로운 연산 방법을 제안하다.

Abstract

Effective component retrieval is very essential for component based software development. Facet scheme is one of typical component retrieval methods and is being widely researched. In this paper, an efficient algorithm which supports a query with logical operator NOT for more than one facet values is presented. With this new algorithm the complexity to calculate a weighted synaptic connectivity matrix is enhanced. Also a new scheme is presented to support a query with logical operators for multiple facets.

* 성결대학교 컴퓨터학부 조교수

논문접수 : 2002. 6. 20
심사완료 : 2002. 9. 18

Ⅰ. 서론

컴포넌트의 효율적인 검색은 컴포넌트에 기반한 소프트웨어 개발에서 매우 중요하다. 원하는 컴포넌트를 발견하지 못하면 컴포넌트를 사용한 소프트웨어 개발이 불가능하기 때문이다[1,2].

패싯 방식[3-7]은 컴포넌트를 검색하는 방식의 한 종류인 도서관학 방식[8]의 대표적인 예이다. 패싯 방식은 합성적인 방법을 사용하여 항목을 분류하므로 듀이의 십진 시스템[8]과 같은 나열 방식에 비해 항목의 확장과 관리가 쉬우며 정확도와 표현력이 좋기 때문에 많은 연구의 대상이 되었으며 실제 제품화되어 사용된다[9].

패싯 방법을 구현하는 한 가지 방법은 신경 접속 행렬을 사용하는 것이다. 본 논문에서 기존에 제안된 가중치 신경 접속 행렬[3-5]에 기초하여 효율적인 논리 부정 검색을 가능하게 하는 알고리즘을 제안한다. 기존 논문[3-5]에서 제안한 알고리즘은 한 개의 패싯 값에 대한 논리 부정 검색을 하였을 때는 효율적인 가중치 신경 접속 행렬을 계산하지만 여러 개의 패싯 값을 사용한 부정 검색에 대하여 효율적인 연산을 제공하지 못하였다.

또한 기존 논문[3-5]에서 서로 다른 패싯 간에 논리적 관계를 고려한 검색을 지원하지 않았는데 본 논문에서 이를 지원하는 방법을 제안하다.

본 논문의 구성은 다음과 같다. 2장에서 기존의 신경 접속 행렬과 부정 검색의 문제점을 기술한다. 3장에서 논리 부정 검색을 지원하는 새로운 효율적인 알고리즘과 다른 패싯간에 논리적인 검색이 가능하게 하는 새로운 방법을 제안한다. 4장에서 결론과 향후 연구방향을 기술한다.

Ⅱ. 신경 접속 행렬

2.1 가중치 신경 접속 행렬

신경 접속 행렬의 정의와 이를 이용한 컴포넌트 검색 방법은 아래와 같다[3,4].

정의 1. 신경 접속 행렬 W의 원소

w_{ij}, *where* $1 \le i \le F$, $1 \le j \le N$

$w_{ij} = 1$ *if componenet j has facet value i*
$= 0$ *otherwise*

수식에 사용한 F 는 서로 다른 패싯 값의 개수이고 N 은 컴포넌트 저장소에 있는 컴포넌트의 개수이다.

정의 2. 사용자 질의어 벡터 Q의 원소

q_i $(1 \le i \le F)$

$q_i = 1$ *if a component with facet value i is to be retrieved*
$= 0$ *otherwise*

정의 3. 검색 출력 벡터 O

$$O = Q \cdot W$$

$Q \cdot W$ 를 계산하면 N개(컴포넌트의 개수)의 원소를 가진 출력 벡터 O가 만들어지는데 o_i 값이 클수록 컴포넌트 i는 검색을 더 만족하는 컴포넌트가 된다.

예 1. 신경 접속 행렬

패싯 F={remove, create, modify}. C1=(remove), C2=(remove, create, modify), C3=(create), C4=(create, modify)라 하자. 이를 표시하는 신경 접속 행렬 W는 아래와 같이 정의된다.

$$W = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

정의 4. 가중치 신경 접속 행렬

가중치 신경 접속 행렬 W'의 원소 w'_{ij} 다음과 같이 정의한다

$$w'_{ij} = w_{ij} \cdot \frac{N/n_i}{\sum_{z=1}^{E} w_{zj}(N/n_z)}, \quad where \ \sum_{i=i}^{E} w'_{ij} = 1$$

수식에 사용된 N_z는 패싯 값 z를 가지고 있는 컴포넌트의 수를 나타낸다.

예 2. 가중치 신경 접속 행렬

예 1의 행렬 W에 가중치 함수를 적용하여 가중치 신경 접속 행렬 W'를 구한다. 먼저 N/N_i를 표시하는 벡터 IW를 계산하고 이를 이용하여 W'을 아래와 같이 구한다.

$$IW = \begin{bmatrix} 4/2 & 4/2 & 0 & 0 \\ 0 & 4/3 & 4/3 & 4/3 \\ 0 & 4/2 & 0 & 4/2 \end{bmatrix},$$

$$W' = \begin{bmatrix} 1 & 3/8 & 0 & 0 \\ 0 & 1/4 & 1 & 2/5 \\ 0 & 3/8 & 0 & 3/5 \end{bmatrix}$$

2.2 기존의 논리 부정 검색

부정 검색을 하는 아래에 표시된 알고리즘이 논문[3,4]에서 제안되었다. 이 알고리즘은 한 개의 패싯 값에 대한 부정 검색에는 효율적으로 사용할 수 있으나 여러 개의 패싯 값에 대한 부정 검색에는 효율적이지 못하다.

신경 접속 행렬의 i번째 행

$\overline{k}w'_{ij}\ (1 \le j \le N)$을 구하는 알고리즘

```
for (j=1; j≤N; j++)
 { if i=k
     if w'_kj=0
        w̄k'_ij = N/(N−n_k) / (s_j + N/(N−n_k))
     else
        w̄k'_ij = 0
   else
     if w'_kj=0
        w̄k'_ij = w'_ij · s_j / (s_j + N/(N−n_k))
     else
        w̄k'_ij = w'_ij · s_j / (s_j − N/n_k)
 }
```

예 3. 두 개의 패싯 값을 부정하는 검색

질의어 Q= [$\overline{1}$, $\overline{1}$, 0]가 주어질 때 논리 부정 검색을 위한 신경 접속 행렬의 1행을 계산하는 과정은 다음과 같다.

(1) k=1일 때 위의 알고리즘을 이용하여 1행의 원소 값을 구한다.

(2) 패싯 값 1의 부정을 고려한 s_j(정의 8 참조)를 계산한다.

(3) k=2 일 때 위의 알고리즘을 이용하여 1행의 원소 값을 다시 계산한다.

이런 반복적인 연산은 모든 행에 적용되며 연산의 회수는 부정 검색하는 패싯 값에 정비례하게 된다.

이렇게 연산의 회수가 증가할 뿐 아니라 위의 알고리즘에 s_j값을 재계산하는 방법이 제시되어 있지 않다. 이런 문제점을 해결하기 위해 다음 절에서 다중 논리 부정 검색을 위한 효율적인 알고리즘을 제안한다.

Ⅲ. 효율적인 검색을 위한 방법

3.1 논리 부정 검색

정의 5. 부정 검색을 하는 패싯 값의 인덱스의 집합을 F_{NOT}으로 정의한다.

예 4. F_{NOT}의 예

질의어 Q= [$\overline{1}$, $\overline{1}$, 0]가 주어지면 F_{NOT} = {1,2}.

신경 접속 행렬에서 k행에 표시된 패싯 값을 포함하지 않는 컴포넌트를 검색하는 것은 기본 신경 접속 행렬에서 $w_{kj}=0$ 이면 컴포넌트 j가 선택되고 $w_{kj}=1$이면 컴포넌트 j가 제외되는 것을 의미한다. 이 논리부정 검색을 일반 검색과 동일하게 적용하려면 새로운 행렬을 다음과 같이 구하면 된다.

정의 6. 논리 부정 검색을 위한 F_{NOT}에 대한 신경 접속 행렬 $\overline{\overline{W}}$

W에 대해, F_{NOT}에 속한 패싯 값을 포함하지 않는 컴포넌트를 검색할 때 사용되는 신경 접속 행렬을 $\overline{W}$로 표시하자. $\overline{W}$의 원소인 $\overline{w_{ij}}$는 다음과 같이 정의한다.

$$\begin{aligned}\overline{w_{ij}} &= 1-w_{ij}, \quad if\ i \in F_{NOT}\\ &= w_{ij}, \quad otherwise\end{aligned}$$

정의 7. 논리부정 검색을 위한 가중치 신경 접속 행렬 $\overline{W'}$

F_{NOT}에 속한 패싯 값을 포함하지 않는 컴포넌트를 검색할 때 사용되는 가중치 신경 접속 행렬의 원소 $\overline{w'_{ij}}$는 다음과 같이 정의한다.

$$\overline{w'_{ij}} = \overline{w_{ij}} \cdot \frac{N/n_i}{\sum_{z=1}^{F} \overline{w_{zj}} \cdot (N/n_z)}$$

예 5. 논리 부정 검색

질의어 Q= {0, $\overline{1}$, 0}가 있다. $\overline{1}$은 논리 부정 검색을 의미한다. F_{NOT} = {2} 이므로 W에서 $\overline{W}$와 $\overline{W'}$를 다음과 같이 얻는다.

$$W=\begin{bmatrix}1&1&0&0\\0&1&1&1\\0&1&0&1\end{bmatrix},\ \overline{W}=\begin{bmatrix}1&1&0&0\\1&0&0&0\\0&1&0&1\end{bmatrix},$$

$$\overline{W}=\begin{bmatrix}4/2&4/2&0&0\\4/1&0&0&0\\0&4/2&0&4/2\end{bmatrix}$$

$$\overline{W'}=\begin{bmatrix}1/3&1/2&0&0\\2/3&0&0&0\\0&1/2&0&1\end{bmatrix}$$

부정 검색 결과는 다음과 같다.

$O_{NOT} = Q \cdot \overline{W'}$ = {2/3, 0, 0, 0}

논리 부정 검색을 하기 위해 가중치 신경 접속 행렬 $\overline{W'}$ 전체를 새로 만드는 것은 복잡도가 $O(N \cdot F)$이다. 또한 한 개의 원소인 $\overline{w'_{ij}}$를 구하는 복잡도는 $O(F)$이다. 복잡도를 줄이기 위해 다음과 같은 벡터 S를 정의한다.

정의 8. 가중치 합 벡터 S

S의 원소 s_j는 다음과 같이 정의한다.

$$s_j = \sum_{i=1}^{F} w_{ij} \cdot (N/n_i)\ (1 \le j \le N)$$

이와 같이 정의된 S를 미리 계산하여 저장하고 있으면 패싯 값들을 부정한 신경 접속 행렬의 i 번째 행을 효율적으로 계산할 수 있다.

부정 검색에 대한 가중치 합 벡터 S는 원래 가중치 합 벡터와 값이 다르게 된다. 정의 8을 사용하여 S를 재계산하면 복잡도는 $O(N \cdot F)$이다. 아래에 있는 알고리즘 1은 복잡도를 향상한다.

알고리즘 1. 부정검색을 위한 가중치 합 벡터 S_{NOT}

S_{NOT}의 원소 $s_{NOT\,j}\ (1 \le j \le N)$는 다음과 같이 구한다.

$s_{NOT\,j} = s_j$
for *every* $k \in F_{NOT}$
{ *if* $w_{kj}=0$ $s_{NOT_j} = s_{NOT_j} + N/(N-N_k)$
else $s_{NOT_j} = s_{NOT_j} - N/N_k$
}

위의 알고리즘의 복잡도는 F_{NOT} 에 들어 있는 원소의 개수가 일반적으로 적으므로 $O(N)$이다. 정의 8의 방법으로 $s_{NOT\,j}$를 다시 구하면 $O(N \cdot F)$이므로 알고리즘 1은 복잡도를 향상시킨다.

알고리즘 1에 근거하여 이번에는 부정 검색을 위한 신경 접속 행렬의 i번째 행을 구하는 알고리즘을 제시한다.

알고리즘 2. 신경 접속 행렬의 i번째 행

$\overline{w'_{ij}}\ (1 \le j \le N)$을 계산
for $(j=1; j \le N; j++)$
{ if $i \in F_{NOT}$
if $w'_{ij}=0$
$\overline{w'_{ij}} = \dfrac{N/(N-n_i)}{s_{NOT\,j}}$
else
$\overline{w'_{ij}} = 0$
else
$\overline{w'_{ij}} = \dfrac{w'_{ij} \cdot s_j}{s_{NOT\,j}}$
}

$\overline{kw'_{ij}}\ (1 \le j \le N)$을 구하는 복잡도가 위의 알고리즘을 사용하면 $O(N \cdot F)$에서 $O(N)$으로 향상된다.

예 6. 논리 부정 검색

질의어 Q= {$\overline{1}$, 0, $\overline{1}$}가 있다. 따라서 F_{NOT} = {1,3}. 행렬의 정의를 사용하여 W에서 $\overline{W}$, W', $\overline{W'}$를 다음과 같이 계산한다.

$$W=\begin{bmatrix}1&1&0&0\\0&1&1&1\\0&1&0&1\end{bmatrix},\ \overline{W}=\begin{bmatrix}0&0&1&1\\0&1&1&1\\1&0&1&0\end{bmatrix}$$

$$IW=\begin{bmatrix}4/2&4/2&0&0\\0&4/3&4/3&4/3\\0&4/2&0&4/2\end{bmatrix}$$

$$W'=\begin{bmatrix}1&3/8&0&0\\0&1/4&1&2/5\\0&3/8&0&3/5\end{bmatrix}$$

$$I\overline{W}=\begin{bmatrix}0&0&4/2&4/2\\0&4/3&4/3&4/3\\4/2&0&4/2&0\end{bmatrix}$$

$$\overline{W'}=\begin{bmatrix}0&0&3/8&3/5\\0&1&1/4&2/5\\1&0&3/8&0\end{bmatrix}$$

이제 알고리즘 1과 2를 적용하여 $\overline{W'}$을 구한다. 먼저 S를 정의를 사용하여 구하면 다음과 같다.

$S=[2, 16/3, 4/3, 10/3]$

알고리즘 1에 의해 S_{NOT}의 원소를 다음과 같이 구한다.

$s_{NOT1}=s_1-4/2+4/2=2$

$s_{NOT2}=s_2-4/2-4/2=4/3$

$s_{NOT3}=s_3+4/2+4/2=16/3$

$s_{NOT4}=s_4+4/2-4/2=10/3$

알고리즘 2에 의해 $\overline{W}$의 1행을 구하면 다음과 같다.

$\overline{w'_{11}}=0$, $\overline{w'_{12}}=0$,

$$\overline{w'_{13}}=\frac{4/2}{s_{NOT3}}=\frac{2}{16/3}=3/8$$

$$\overline{w'_{14}}=\frac{4/2}{s_{NOT4}}=\frac{2}{10/3}=3/5$$

알고리즘 2에 의해 $\overline{W}$의 2행을 구하면 다음과 같다.

$\overline{w'_{21}}=0$

$$\overline{w'_{22}}=\frac{w'_{22}\cdot s_2}{s_{NOT2}}=\frac{1/4\cdot 16/3}{4/3}=1$$

$$\overline{w'_{23}}=\frac{w'_{23}\cdot s_3}{s_{NOT3}}=\frac{1\cdot 4/3}{16/3}=1/4$$

$$\overline{w'_{24}}=\frac{w'_{24}\cdot s_4}{s_{NOT4}}=\frac{2/5\cdot 10/3}{10/3}=2/5$$

$\overline{W}$의 3행은 1행과 동일한 방법으로 구하면 다음과 같다.

$\overline{w'_{31}}=1$, $\overline{w'_{32}}=0$, $\overline{w'_{33}}=3/8$, $\overline{w'_{34}}=0$

알고리즘 1과 2를 적용한 결과가 위에서 정의를 사용하여 구한 결과와 동일함을 알 수 있다.

3.2 패싯간의 논리적 검색

저장소에서 컴포넌트가 M개의 패싯으로 표현된다고 가정한다. 이 때 패싯을 F1, F2, ... , FM으로 표현하며 각 패싯 Fi는 유한개의 값의 집합으로 다음과 같이 표현할 수 있다.

$F_i=\{\ V_{ij};1\le j\le M_i,$
M_i는 패싯 F_i의 패싯값 개수$\}$

그러면 저장소의 공간 F는 $F=F_1\times F_2\times\ldots\times F_n$으로 표현된다.

(1) 논리합 검색

논리합 검색에 사용된 서로 다른 패싯의 개수가 N_{OR} 이라 하자. 각각의 패싯에 속한 패싯 값에 대해 검색한 각각의 출력 벡터를 O_i $(1\le i\le N_{OR})$ 로 정의한다.

정의 9. 패싯간의 논리합 검색 출력 벡터 O_{FOR}

$O_{FOR}=\max\{O_i\}$, $where\ 1\le i\le N_{OR}$

(2) 논리곱 검색

논리곱 검색에 사용된 서로 다른 패싯의 개수가 N_{AND} 라 하자. 각각의 패싯에 속한 패싯 값에 대해 검색한 각각의 출력 벡터를 O_i $(1\le i\le N_{AND})$ 로 정의한다.

정의 10. 패싯간의 논리곱 검색 출력 벡터 O_{FAND}

$$O_{FAND}=\sum_{i=1}^{N_{FAND}}O_i\ /N_{FAND}$$

예 7. 패싯 간의 논리합과 논리곱 검색

3개의 패싯에 대한 출력 벡터가 각각 $O_1=0.4$, $O_2=0.8$, $O_3=0$이라 하자.

그러면 논리합과 논리곱에 대한 결과는 아래와 같이 계산된다.

$$O_{FOR}=\max\{0.4, 0.8, 0\}=0.8$$

$$O_{FAND}=(0.4+0.8+0)/3=0.4$$

(3) 논리부정 검색

어떤 패싯에 대한 논리 부정 검색은 패싯 값에 대한 논리 부정 검색으로 변환할 수 있다. 패싯 값에 대한 논리 부정 검색은 3.1절에 제시한 방법으로 계산할 수 있으므로 별도로 다룰 필요가 없다.

Ⅳ. 결론

본 논문에서 가중치 신경 접속 행렬에 기초한 새로운 부정 검색 알고리즘을 제안하였다. 이 알고리즘을 사용하면 다중의 패싯 값에 대한 논리 부정 검색을 효율적으로 할 수 있다.

또한 이 논문에서 서로 다른 패싯 들간에 논리합과 논리곱 검색을 가능하게 하는 새로운 방법을 제안하였다.

향후 연구 과제로 논리적 검색 외에 CORBA 트레이딩 객체 서비스[10-12]에서 제공하는 다양한 검색이 가능하도록 신경 접속 행렬을 확장하는 것이 필요하다.

참고문헌

[1] R. Mili, A. Mili and R. Mittermeir, "Storing and Retrieving Software Components: A Refinement Based System," IEEE Transactions on Software Engineering, Vol. 23, No. 7, pp. 445-460, 1997.

[2] W.P. Frakes and T.P. Pole, "An Empirical Study of Representation Methods for Reusable Software Components," IEEE Transaction on Software Engineering Methodology, vol. 20, no. 8, pp. 617-630, Aug. 1994.

[3] 금영욱 "컴포넌트 검색을 위한 새로운 가중치 신경 접속 행렬", 한국OA학회, 제 7권 제 1호, p.1-7, 2002년 3월

[4] 금영욱 "확장된 소프트웨어 컴포넌트 서술자에 기초한 컴포넌트 저장소의 검색", 한국정보처리학회, 제 9-D권 제 3호, p.417-426, 2002년 6월

[5] Zhiyuan Wang, "Component-Based Software Engineering," Doctorial Dissertation, Department of Computer and Information Science, New Jersey Institute of Technology, Newark, New Jersey, 2000

[6] R. Priesto-Diaz, "Implementation Faceted Classification for Software Reuse," CACM, Vol. 34, No. 5, pp. 89-97, May 1991.

[7] Hsian-Chou Liao et al, "Using a Hierarchical Thesaurus for Classifying and Searching Software Libraries," Proceedings of the COMPSAC '97, pp. 210-216, 1997

[8] M. Dewey, "Decimal Classification and Relative Index," 19th ed., Forest Press Inc., Albany, New York, 1979.

[9] J. Guo and Luqi, "A survey of Software Reuse Repositories," Proceedings of the 7th IEEE Int. Conf. & W/S on the Engineering of Computer Based System, pp.88-96, 2000

[10] 금영욱, 장연세, "CORBA 3 Programming Bible" 도서출판 그린, 2000년 7월

[11] 금영욱, "CORBA 컴포넌트의 구현", 한국정보처리학회지, 제 7권 제 4호, 2000년 7월, p.60-69.

[12] CORBA Trading Object Service, ftp://ftp.omg.org/pub/docs/formal/00-06-27.pdf

저자소개

금 영 욱

1978년 : 서울대학교 수학과 (학사)

1992년 : 서강대학교 정보처리학과(석사)

1997년 : 서강대학교 전자계산학과(공학박사)

1982년 - 1992년 : IBM advisory systems engineer

1993년 - 1997년 : 수원과학대학 사무자동화과 조교수

1997년 - 현재 : 성결대학교 컴퓨터 학부 조교수

관심 분야 : 분산·병렬처리, 분산객체기술, 컴포넌트 기술