韓國 OA學會 論文誌
第7卷 第1號, 2002. 3.

2002-7-1-1-1

컴포넌트 검색을 위한 새로운 가중치 신경 접속 행렬

금 영 욱*

A New Weighted Synaptic Connectivity Matrix for Component Retrieval

Young-wook Keum*

요 약

최근에 컴포넌트에 기반한 소프트웨어 개발이 대학과 산업체의 중요 연구 대상이 되었다. 컴포넌트는 컴포넌트 저장소에 저장되는데 컴포넌트의 효율적인 검색은 컴포넌트에 기반한 소프트웨어 개발에 매우 중요하다. 이 논문에서 컴포넌트의 효율적인 검색을 위하여 새로운 가중치 함수를 사용한 신경 접속 행렬을 제안한다. 또한 부정 검색을 위한 신경 접속 행렬을 구하는 새로운 알고리즘을 제안하고 이를 증명한다. 마지막으로 논리 연산자를 사용한 질의에 대하여 효율적으로 행렬을 연산하는 과정을 제안한다.

Abstract

Component-based software development(CBSD) is gaining popularity. Effective search and retrieval of desired components, which are stored in a component repository, is a very important issue in CBSD. In this paper, a new weighted synaptic connectivity matrix is proposed to find more appropriate components. An algorithm is proposed for effective search with NOT operator and a proof for the algorithm is presented . A new procedure to calculate the output vector for a logically combined query is also presented.

* 성결대학교 컴퓨터학부 조교수

논문접수 : 2002. 1. 15
심사완료 : 2002. 3. 4

Ⅰ. 서론

컴포넌트를 재사용하려면 먼저 원하는 컴포넌트를 발견하여야 한다[1,2]. 따라서 컴포넌트를 저장하는 컴포넌트 저장소의 효율적인 검색은 컴포넌트 기반 기술에서 매우 중요하다. 최근에 소프트웨어 컴포넌트 저장소의 크기가 커지면서 효율적인 검색은 더욱 중요해진다.

컴포넌트를 검색하는 여러 방식들이 제안되었는데 일반적으로 인공지능 방식[3], 도서관학 방식[4], 형식 명세 방식[5], 브라우징 방식[6] 등의 4개의 종류로 분류할 수 있다. 도서관학 방식의 한 종류인 패싯 방식은 항목을 분류하는데 합성적인 방법을 사용하므로 듀이의 십진 시스템[4]과 같은 나열 방식에 비해 항목의 확장과 관리가 쉬우며 정확도와 표현력이 좋다.

본 논문에서는 패싯 방법을 사용한 신경 접속 행렬[7]에 기초하여 새로운 가중치 함수를 사용한 새로운 신경 접속 행렬을 제안한다. 논리 연산자 AND를 사용한 검색이 가능한 새로운 방법과 NOT을 사용한 검색이 가능하도록 신경 접속 행렬을 확장한다. 또한 부정 검색하는 알고리즘과 연산 과정을 단순화하여 연산의 복잡도를 향상시킨다.

본 논문의 구성은 다음과 같다. 2장에서 기존의 신경 접속 행렬을 기술하고 3장에서 새로운 가중치 함수를 사용한 검색 방법과 알고리즘을 제안한다. 4장에서 결론과 향후 연구방향을 기술한다.

Ⅱ. 신경 접속 행렬

2.1 신경 연상 기억 장치

신경 연상 기억 장치[7]는 단일층의 신경망 네트워크로 입력 패턴 X={x1, ..., xm}를 출력 패턴 Y={y1, ..., ym}로 변환하는 일을 한다. 연상 기억 장치는 변환의 집합 S={(xk, yk); k=1, ..., m}을 기억한다. 새로운 입력 x가 네트워크에 입력되면 여기에 대응하는 y가 y=xW에 의해 계산된다. 이런 W를 신경 접속 행렬이라 한다.

저장소에서 원하는 컴포넌트를 발견하기 위해 각 컴포넌트에 대해 컴포넌트를 표현하는 n개의 패싯이 있다고 가정한다.

패싯을 F1, F2, ... , Fn으로 표현하며 각 패싯 Fi는 유한개의 값의 집합으로

$$F_i = \{\ V_{ij};1 \le j \le N_i,\ N_i\text{는 패싯 } F_i\text{의 패싯값 개수}\}$$

으로 표현된다. 그러면 저장소의 공간 F는

$$F = F_1 \times F_2 \times \ldots \times F_n$$으로 표현된다.

2.2 신경 접속 행렬

신경 접속 행렬의 정의와 이를 이용한 컴포넌트 검색 방법은 아래와 같다.

정의 1. 신경 접속 행렬 W의 원소 w_{ij}

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

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 needed*
= 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.

이 논문에서 컴포넌트를 표현하는 한 개의 패싯 F_i

에 대해서 한 개의 신경 연상 네트워크 N_i를 할당한다. 이진 벡터가 F_i의 패싯값의 공간을 표시하는데 사용된다. 벡터의 각 비트는 패싯 값 공간의 한 개의 값을 표현한다. 예를 들어 패싯의 하나인 기능(function)에 대해 세 개의 값 remove, create, modify가 있다고 하면 이 패싯의 패싯 값 공간을 표시하는 벡터는 $[x_1, x_2, x_3]$ 이다. 벡터 [0,0,1]은 컴포넌트의 기능이 modify라는 것을 나타내며 벡터 [1,1,0]은 컴포넌트가 remove, create 기능을 다 수행함을 의미한다.

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.

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

Ⅲ. 새로운 신경 접속 행렬

3.1 가중치 신경 접속 행렬

여러 컴포넌트가 동일한 패싯 값을 가질 수 있으며 한 컴포넌트가 여러 개의 다른 패싯 값을 가질 수 있다.

예 2. 신경 접속 행렬의 문제점

질의어 [0,1,0]에 대해 $O = Q \cdot W$ =[0,1,1,1]. 패싯 값 create를 만족하는 컴포넌트가 3개가 있는데 컴포넌트 3은 create 한 개의 패싯 값만 가지고 있으므로 이 질의를 가장 만족한다. 그러나 컴포넌트 2와 4는 이 질의에서 요구하지 않은 다른 기능을 가지고 있지만 위의 출력 벡터의 결과는 1로 동일하여 정확한 만족도를 표현하지 못한다.

검색하는 패싯 값에 대한 컴포넌트의 만족도를 정확하게 계산하기 위해 이 논문에서 새로운 가중치 함수를 사용한 가중치 신경 접속 행렬을 제안한다.

정의 4. 가중치 신경 접속 행렬

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

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

수식에 사용된 N은 저장소에 있는 컴포넌트의 개수, N_z는 패싯 값 z를 가지고 있는 컴포넌트의 수, F는 서로 다른 패싯 값의 개수를 나타낸다.

예 3. 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.

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

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

질의어 [0,1,0]에 대해 $O = Q \cdot W$ = [0, 1/4, 1, 2/5]. 결과 값이 큰 순서대로 컴포넌트를 나열하면 3,4,2,1이 되어 이 결과가 패싯 값의 희소성을 제대로 표현하는 것을 알 수 있다.

3.2 논리 연산자를 이용한 검색

예 4. 논리 관계를 고려하지 않은 검색

질의어 Q= [0, 1, 1]가 들어온다고 하자. 그러면 출력 벡터 $O = Q \cdot W$ =[0, 2, 1, 2]이 되며 $O' = Q \cdot W'$ =[0, 5/8, 1, 1]이 된다. 가중치를 주었을 때의 결과를 보면 컴포넌트 2의 경우는 2개의 패싯 값을 만족하지만 패싯 값 1개를 만족한 컴포넌트 3과 4보다도 작은 값을 가지게 된다. 특히 패싯 값 2개를 동시에 만족(AND)하는 경우의 질의였다면 컴포넌트 3과 4

는 질의를 만족할 수 없다.

예 4의 문제는 검색 방법이 패싯 값간에 논리 관계를 고려하지 않고 있으며 더 정확히 표현하면 논리합 관계만 고려하고 있는데 문제가 있다. 패싯으로 검색을 제안한 다른 논문들[8-10]에서도 논리 연산자를 사용한 검색을 고려하지 않고 있다. 논리 연산자를 사용한 검색은 기본적으로 필요하며 CORBA[11-15]의 트레이딩 객체 서비스[16]에서도 요구하고 있다. 이 절에서 패싯 값간의 논리 관계를 고려한 검색이 가능하도록 하였다.

(1) 논리합 검색

논리합 검색의 출력 벡터는 다음과 같이 정의한다.

정의 5. 논리합 검색의 출력 벡터 O_{OR}

논리합 검색의 만족도를 계산하는 출력 벡터는 다음과 같이 구한다.

$$O_{OR} = Q \cdot W'$$

즉 질의어 벡터와 가중치 신경 접속 행렬을 곱하여 출력 벡터를 생성한다.

예 5. 논리합 검색의 예

논리합 검색을 위한 질의어가 Q= [0, 1, 1]이라 하자. 그러면 벡터는 $O_{OR} = Q \cdot W' = [0, 5/8, 1, 1]$이 된다. 이 때 컴포넌트 2가 컴포넌트 3과 4보다 만족도가 작은데 이는 컴포넌트 2가 사용자가 원하지 않는 다른 기능을 제공하기 때문이다.

(2) 논리곱 검색

논리곱 검색을 위해 검색하는 모든 패싯 값에 대하여 컴포넌트가 가지고 있는 패싯 값의 개수를 가진 히트율을 계산한다.

정의 6. 질의어에 대한 히트율 벡터 H

어떤 질의어에 대한 히트율 벡터 H의 원소 h_j $(1 \le j \le N)$는 다음과 같이 정의한다.

$$h_j = \frac{f_j}{f}, \text{ where } f = \sum_{i=1}^{E} q_i \text{ and}$$

$$f_j = \sum_{i=1}^{E} q_i \cdot w_{ij}$$

정의 7. 논리곱 검색의 출력 벡터 O_{AND}의 i째 원소 $o_{AND\,i}$

$$o_{AND\,i} = h_i \cdot o_{OR\,i}$$

즉 O_{OR}의 i번째 원소와 히트율 벡터의 i번째 원소의 곱으로 구한다.

예 6. 논리곱 검색

질의어 Q= [0, 1, 1]이 주어지면 히트율 벡터는 H= [0, 1, 1/2, 1]이다. 최종 검색 결과는 O_{AND}= [0·0, 1·5/8, 1/2·1, 1·1]=[0, 5/8, 1/2, 1]이 된다. 결과에서 보듯이 컴포넌트 2가 2개의 패싯 값을 동시에 만족하므로 한 개의 패싯 값만 만족하는 컴포넌트 3보다 만족도가 높으며 패싯 값을 2개만 가지고 있는 컴포넌트 4보다는 만족도가 낮게 나온다.

(3) 논리부정 검색

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 8. 논리 부정 검색을 위한 신경 접속 행렬 $\overline{k}W$

W에 대해, k번째 패싯 값을 포함하지 않는 컴포넌트를 검색할 때 사용되는 신경 접속 행렬을 $\overline{k}W$로 표시하자. $\overline{k}W$의 원소인 $\overline{k}w_{ij}$는 다음과 같이 정의한다.

$$\begin{aligned} \overline{k}w_{ij} &= 1 - w_{ij}, \quad if\ i = k \\ &= w_{ij}, \quad otherwise \end{aligned}$$

정의 9. 논리부정 검색을 위한 가중치 신경 접속 행렬 $\overline{k}W'$

k번째 패싯 값을 포함하지 않는 컴포넌트를 검색할 때 사용되는 가중치 신경 접속 행렬의 원소 $\overline{k}w'_{ij}$는 다음과 같이 정의한다.

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

예 7. 논리 부정 검색

질의어 Q= {0, $\bar{1}$, 0}가 있다. $\bar{1}$은 논리 부정 검색을 의미한다. k가 2이므로 W에서 $\bar{2}W$와 $\bar{2}W'$를 다음과 같이 얻는다.

$$W = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}, \quad \bar{2}W = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

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

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

O_{NOT}=Q · $\bar{2}$W'={2/3, 0, 0, 0}의 결과를 얻는다.

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

정의 10. 가중치 합 벡터 S

S의 원소 s_j는 다음과 같이 정의한다.

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

이와 같이 정의된 S를 미리 계산하여 저장하고 있으면 패싯 값 k를 부정한 신경 접속 행렬의 i 번째 행은 아래에 있는 알고리즘으로 구할 수 있으며 이 알고리즘에 대한 증명을 한다.

$\bar{k}w'_{ij}$ $(1 \le j \le N)$을 구하는 복잡도가 아래의 알고리즘을 사용하면 $O(N \cdot F)$에서 $O(N)$으로 향상된다.

신경 접속 행렬의 i번째 행

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

```
for (j=1; j≤N; j++)
 { if i=k
     if w'kj=0
        k̄w'ij = N/(N−nk) / (sj + N/(N−nk))
     else
        k̄w'ij = 0
   else
     if w'kj=0
        k̄w'ij = w'ij · sj / (sj + N/(N−nk))
     else
        k̄w'ij = w'ij · sj / (sj − N/nk)
 }
```

알고리즘의 증명

w'_{ij}정의와 s_j 정의에 의해 w'_{ij}는 다음과 같이 표현된다.

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

1) $i=k$라 하자. 그러면 $w_{ij}=1-w_{ij}$. 따라서 k행에 0아닌 개수는 원래 n_k이나 부정 검색을 가정할 경우 k행에 0아닌 값의 개수는 $N-n_k$이다.

따라서 $w'_{kj}=0$이면

$$\bar{k}w'_{kj} = \frac{N/(N-n_k)}{s_j + (N/(N-n_k))}$$

만약 $w'_{kj} \neq 0$이면 부정 검색이므로 $\bar{k}w'_{ij}=0$.

2) $i \neq k$라 하자.

$w'_{kj}=0$이면 $\bar{k}w'_{ij} = \dfrac{w_{ij} \cdot (N/n_i)}{S_j + (N/(N-n_k))}$

그런데 $w_{ij} \cdot (N/n_i) = w'_{ij} \cdot S_j$이므로

$$\bar{k}w'_{ij} = \frac{w'_{ij} \cdot s_j}{s_j + (N/(N-n_k))}.$$

w'_{kj}=1이면 $\overline{k}w'_{ij}=\frac{w_{ij}\cdot(N/n_i)}{s_j-N/n_k}$.

그런데 $w_{ij}\cdot(N/n_i)=w'_{ij}\cdot s_j$이므로

$\overline{k}w'_{ij}=\frac{w'_{ij}\cdot s_j}{s_j-N/n_k}$.

3.3 행렬의 연산 과정과 복잡도

출력 벡터를 구하는 간단한 과정은 다음과 같다.

(1) 질의어를 입력받는다.

(2) 질의어에 1 또는 $\overline{1}$로 표시된 인덱스의 집합 A를 구한다.

(3) 질의어에 $\overline{1}$로 표시된 부정 검색이 있으면 신경 접속 행렬의 i 번째 행을 구하는 알고리즘을 사용하여 집합 A에 있는 모든 원소에 해당하는 신경 접속 행렬의 행을 계산한다.

(4) 위에서 계산된 신경 접속 행렬의 행을 추출하여 새로운 임시 행렬 TW'을 구한다.

(5) 이 행렬의 열을 더하여 출력 벡터 O를 구한다.

(6) O의 원소를 정렬하여 큰 순서로 첨자를 출력한다.

위의 과정을 이용하여 연산하는 예를 아래에 보인다.

예 8. 연산 과정

(1) 질의어 Q= {0, $\overline{1}$, 1}를 입력받는다.

(2) A = {2,3}

(3)-(4)

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

(5) 출력 벡터 O = {2/3 1/2 0 1}

(6) 정렬된 첨자 = {4,1,2,3}

위의 연산 과정에 있는 것처럼 전체 행렬과 질의어 벡터를 곱하지 않고 필요한 행렬만 선택하여 이들의 열을 더하기 때문에 연산 복잡도는 $O(N\cdot F)$[7]에서 $O(N)$으로 줄일 수 있다.

Ⅳ. 결론

현재까지 알려진 소프트웨어 재사용을 위한 최선의 방법은 컴포넌트에 기반한 소프트웨어를 만드는 것이다. 컴포넌트에 기반한 소프트웨어 제작을 위해서 먼저 해결해야 하는 중요한 문제는 소프트웨어 저장소에서 원하는 소프트웨어 컴포넌트를 효율적으로 검색하는 일이다.

본 논문에서 신경 접속 행렬에 기초한 새로운 검색 방법을 제안하였다. 새로운 가중치 함수를 사용한 새로운 신경 접속 행렬을 제안하였고 이에 기초하여 논리적 검색을 가능하게 하는 행렬의 연산 과정을 개발하였다. 또한 부정 검색의 복잡도를 줄이는 간단한 알고리즘을 개발하고 이를 증명하였다.

향후 연구 과제로 컴포넌트 저장소의 검색과 이와 유사한 속성을 갖는 CORBA 트레이딩 객체 서비스의 검색을 통합하는 것이 필요하다.

참고 문헌

[1] 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.

[2] Christopher G. Drummond, Dan Ionescu, and Robert C. Holte, "A Learning Agent that Assists the Browsing of Software Libraries," IEEE Transactions on Software Engineering, Vol. 26, No. 12, pp. 1179-1196, 2000.

[3] P. Devanbu, R. Brachman, P. Selfridge,

and B. Ballard, "LaSSIE: A Knowledge-Based Software Information System," Comm. ACM, vol. 34, no. 5 pp. 34-49, May 1991.

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

[5] 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.

[6] F.R. Campagnoni and K. Ehrlich, "Information Retrieval Using a Hypertext-Based Help System," Proc. 12th Int'l Conf. Research and Development in Information Retrieval, pp. 212-220, 1989.

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

[8] R. Prieto-Diaz, "A Software Classification Scheme," Doctorial Dissertation, Department of Computer Science, University of California, Irvine, California, 1985

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

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

[11] 금영욱, 장연세 "CORBA와 DCOM의 통합", 한국정보과학회지, 제 17권 제 7호, p.55-64, 1999년 7월

[12] 금영욱, "CORBA - 분산 객체 통합 기술", 한국정보과학회 소프트웨어공학회지,제 12권 제 2호, p.5-14, 1999년 6월

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

[14] 금영욱, "CORBA 3 - 코바 하부 구조의 완성", 소프트웨어공학연구회지, 제 3권 제 2호, p.22-34, 2000년 6월.

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

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

저 자 소 개

금 영 욱

1978년 : 서울대학교 수학과 (학사)

1992년 : 서강대학교 정보처리학과(석사)

1997년 : 서강대학교 전자계산학과(공학박사)

1982년 - 1992년 : IBM advisory systems engineer

1993년 - 1996년 : 수원과학대학 사무자동화과 교수

1997년 - 현재 : 성결대학교 컴퓨터 학부 조교수

관심 분야 : 분산·병렬처리, 분산객체기술, 컴포넌트 기술