Journal of the Korean Institute
of Industrial Engineers
Vol.24, N0.4, December, 1998



# 데이터베이스 설계에서 SOFM을 이용한 화일 수직분할 방법

## A Vertical File Partitioning Method Using SOFM in Database Design

신광호* · 김재련**

K. H. Shin* · J. Y. Kim**

**Abstract**

It is important to minimize the number of disk accesses which is necessary to transfer data in disk into main memory when processing transactions in physical database design. A vertical file partitioning method is used to reduce the number of disk accesses by partitioning relations vertically and accessing only necessay fragments. In this paper, SOFM(Self-Organizing Feature Maps) network is used to solve vertical partitioning problems. This paper shows that SOFM network is efficient in solving vertical partitioning problem by comparing approximate solution of SOFM network with optimal solution of N-ary branch and bound method. And this paper presents a heuristic algorithm for allocating duplicate attributes to vertically partitioned fragments. As branch and bound method requires particularly much computing time to solve large-sized problems, it is shown that SOFM network is able to overcome this limitation of branch and bound method and solve large-sized problems efficiently in a short time.

## 1. 서론

### 1.1 연구의 배경 및 목적

대규모 데이터베이스에서는 수행도가 중요한 연구의 대상이 되고 있다. 수행도란 데이터베이스에 대한 질의, 수정을 수행하는 트랜잭션(transaction)의 처리시간, 즉 응답시간(response time)을 뜻하며 이 응답시간을 줄이는 것이 물리적 데이터베이스 설계의 중요한 요소이다. 응답시간은 주기억 장치와 디스크 사이를 이동하는 데이타량에 비례한다. 본 논문에서는 트랜잭션의 처리시간을 단축하기 위하여 화일을 수직분할하여 디스크에 보관하는 방법을 연구한다.[2,3,4,6]

다수의 속성(attribute)들이 모여 이루어진 것을 데이터 화일(file)이라 한다.(또한 화일을 릴레이션(relation), 또는 테이블(table)이라고도 함. 본 논문에서는 릴레이션을 사용). 그림 1은 대학생에 대한 자료를 모은 릴레이션이며 학번, 학생이름 등은 속성을 나타내며 5 명의 학생 레코드(혹은 tuple)이 나타나 있다. 3장의 그림 3은 세 개의 속성으로 구성된 릴레이션을 네 개의 트랜잭션이 사용하는 것을 나타낸다. 이 릴레이션을 학생 릴레이션이라 할 때 a1 속성은 "학번", a2 속성은 "학생 이름"으로 볼 수 있다. 트랜잭션 t1은 "학번"과 "학생 이름"만 필요로 하는 응용 프로그램으로 생각할 수 있고 t2는 "학번"만 필요로 한다.

* SDS 근무

** 한양 대학교 공과대학 산업공학과 교수

| 학 번 | 학생이름 | 학생주소 | 단과대 |
|---|---|---|---|
| 980001 | 김 장 돌 | 서울 | 공대 |
| 980010 | 홍 길 동 | 부산 | 상대 |
| 980123 | 김 기 수 | 대전 | 공대 |
| 981234 | 이 수 연 | 광주 | 문리대 |
| 982345 | 이 영 수 | 대구 | 상대 |

그림 1. 학생 화일의 예

이와 같이 트랜잭션이 요구하는 속성을 표시하는 행렬을 속성사용행렬(attribute usage matrix)이라 한다. 그림 3에서와 같이 3개의 속성을 모아서 하나의 릴레이션으로 관리하는 경우 t2 수행시 속성 a2 및 a3도 함께 디스크에서 주기억 장치로 이동하므로 이동하는 데이타량 증가로 응답시간이 증가하게 된다.

따라서 속성사용행렬을 고려하여 속성들을 몇개의 그룹, 즉 단편(fragment)으로 나누어 디스크에 보관하여 사용하는 것이 전체 트랜잭션의 응답시간을 단축할 수 있다.(속성 a1을 하나의 단편, 속성 a2 및 a3를 다른 단편으로 분할하는 것이 하나의 예이며 이 경우 t2는 첫 번째 단편만을, t3 및 t4는 두 번째 단편만 이용하면 되므로 응답시간 단축이 가능하다. 그러나 t1은 두 단편을 모두 이동하여야 한다. 그러나 이러한 분할배치로 인하여 이동하여야 할 전체 데이터량은 감소하여 응답시간의 합계는 감소한다).

이와같이 속성별로 릴레이션을 분할하는 것을 화일수직분할(vertical partitioning)이라 하며 수직분할 방법외에 화일 수평분할(horizontal partitioning) 방법, 수직분할과 수평분할을 함께 고려하는 혼합분할방법(mixed partitioning)이 있다. 수평분할은 레코드(tuple)별로 분할하여 디스크에 보관하는 방법이다(그림 1의 학생 화일에서 단과대 별로 학생 레코드를 분할하여 보관하면 단과대 별로 처리하는 경우 데이터 이동량이 적어진다. 특히 단과대들이 지역적으로 분산되어 있는 경우 해당 학생 레코드를 해당 지역에 보관하면 유리하다).

m개의 속성으로 이루어진 릴레이션의 수직분할 방법의 수는 Bell number(B(m))로 알려져 있으며 m이 클 경우 B(m)은 $m^m$과 같아져 계산량(complexity)이 $O(m^m)$가 된다. 따라서 문제의 크기가 커지면 계산량이 지수적으로 증가하기 때문에 최적해를 구하는 데 많은 시간이 소요된다. 그러므로 이에 대한 해결 방안으로 발견적(heuristic) 가법의 개발이 필수적이라고 할 수 있다. 본 연구에서는 데이터베이스 설계에서의 수직분할 문제를 신경망의 하나인 SOFM(Self-Organizing Feature Maps, 자기조직형상지도) 네트워크의 자율적인 경쟁학습(competitive learning) 방법을 적용하여 푸는 방법을 제시한다.

### 1.2 기존 연구 고찰

수직분할 방법을 사용하여 데이터베이스의 성능 향상을 위한 연구는 지금까지 활발히 연구되어 왔다.

Navathe[10]는 이 단계 수직분할 알고리즘을 제시하였다(1단계: 직관적 설계, 2단계: 비용을 고려한 설계). 1단계는 트랜잭션들의 속성사용행렬과 논리적 액세스 횟수(logical access frequency)를 이용하여 두 속성들 사이의 친밀도 행렬(attribute pairwise affinity matrix)을 만들어 가능한 한 block diagonal한 형태로 만들기 위한 행과 열을 조합하는 클러스터링 알고리듬을 적용하였다. 위의 행렬을 두 속성 집합으로 나누어, 부적절한 속성들의 액세스를 최소화하는 분할 점(dividing point)이 결정된다. 2단계는 1단계에서 구한 단편을 비용 요인(cost factor)을 고려하여 수정하였다. 또한 갱신(update) 비용은 검색(retrieval) 비용의 두 배로 가정하였다. Navathe와 Ra[11]는 수직분할 문제에 대한 친밀도 그래프 알고리듬을 개발하였다.

Cornell과 Yu[5]는 속성들을 물리적인 단편에 할당하여 디스크 액세스 횟수를 최소화하는 이진분할 선형 정수계획법을 개발하였다. 또한, 재분할에 관한 수리모형을 제시하였으나 분할의 수가 커지면 수리적 접근은 불가능하였다. Chu[4]는 트랜잭션에 근거한 접근 방법(transaction-based approach)으로 이진 분할법을 연구하였다. 여기에서 트랜잭션별로 속성들을 분석하여 화일을 분할했으며 이진 분할에 대한 해법으로 분지한계법과 발견적 기법을 제안하였다. Cheng[3]은 0-1 속성사용행렬의 mutually separable cluster를 구별해내는 분지한계법과 발견적 기법을 제시하였다. 그러나 입력 변수로 단지 0-1 속성사용행렬만을 사용하기 때문에 최적 디스크 액세스 횟수를 구한다는 보장이 없다. 윤[2]은 Chu[4]의 이진 수직분할 방

법을 개선한 물리적 디스크 액세스 횟수를 최소화시킬 수 있는 분지한계법을 이용한 N-ary 최적해법을 제시하였다.

본 논문은 다음의 5장으로 구성되어 있다. 1장에서는 연구의 배경, 기존의 연구를 설명하였고 2장에서는 SOFM 네트워크의 기본적인 개념과 학습 알고리듬에 대하여 설명한다. 3장에서는 수직분할 문제에 SOFM 네트워크를 이용한 해법을 제시하고 수치 예제를 풀어본다. 4장에서는 SOFM 네트워크를 이용하여 속성을 중복 할당하는 발견적 기법과 수치 예제를 풀어보고 5장에서는 본 연구의 결론을 제시한다.

## 2. SOFM 네트워크

### 2.1 SOFM 네트워크의 기본개념

신경망(neural networks)은 크게 지도 학습(supervised learning) 네트워크와 비지도 학습(unsupervised learning) 네트워크로 분류된다. SOFM 네트워크는 Kohonen(1984)에 의해서 제안된 것으로 뉴런(neuron)간 상호 위치는 패턴 요소들의 물리적인 관계를 반영한다는 것에 착안하였다. SOFM 네트워크는 비지도 학습과 경쟁 학습을 하는 경우로, 네트워크의 입력노드와 출력노드를 연결하는 연결 가중치($w_{ij}$)들이 주어진 학습식에 따라서 입력 패턴에 대하여 스스로 적응하여 변해가며 클러스터링, 패턴 인식(pattern recognition)과 같은 분류기(classifier)로서 많이 사용된다[7,13]. SOFM 네트워크의 구조는 그림 2와 같다.(i는 입력노드 번호, j는 출력노드 번호)

본 연구에서 수직분할 문제를 풀기 위하여 SOFM 네트워크를 사용한 이유는 다음과 같다. 첫째로, SOFM 네트워크의 주어진 문제 환경에 대한 적응성(adaptivity)을 들 수 있다. 이는 SOFM 네트워크가 지도 학습을 하는 신경망이 아니라 비지도 학습을 하는 신경망이기 때문에 특별한 목표치가 주어져 있지 않더라도 스스로 학습하여 해를 제시한다. 즉, 수직분할 문제의 해가 전혀 알려져 있지 않은 상태에서 최선의 분할 방법을 제시한다. 두 번째로, 단순하며 이해하기 쉽다는 것이다. SOFM 네트워크는 입력 벡터의 학습 과정이 1차원 혹은 2차원의 유클리디안 공간상에서 벡터 계산으로 이루어지기 때문에 학습 과정을 쉽게 이해, 검토할 수 있다. 세 번째로, 해의 일관성(consistency)을 들 수 있다. SOFM 네트워크는 적응성의 성질을 가지고 있어서 동일 조건하의 반복적인 실험에서 충분한 학습시간이 주어지면 동일한 해를 얻게된다. 네 번째로 SOFM 네트워크는 한번의 전방 패스로 학습이 이루어지기 때문에 계산 시간이 적게 소요된다.

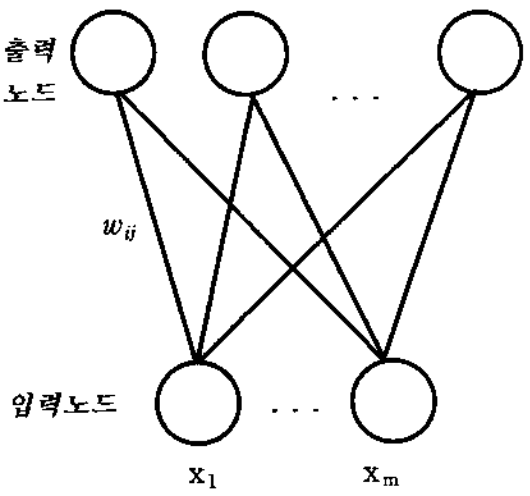


그림 2. SOFM네트워크의 구조

### 2.2 SOFM 네트워크의 알고리듬[7]

SOFM 네트워크의 학습은 다음과 같다.

step 1. 입력 노드 및 출력노드의 개수를 정한다. 연결가중치 $w_{ij}$, 학습률 모수($\alpha$ ) 및 이웃 범위값을 초기화한다.
step 2. 종결조건이 만족할 때까지 step3~8을 수행한다.
step 3. 각 입력벡타 x에 대해서 step4~6을 수행한다.
step 4. 각 출력노드 j에 대해서 D(j)를 계산한다.

$$D(j)=\sum_i (\omega_{ij}-x_i)^2 \qquad (2.1)$$

step 5. D(j)가 최소인 j(승자노드)를 찾 는다.
step 6. 승자노드로부터 이웃 범위값 안에 있는 출력노드에 대해서 연결가중치를 수정한다.

$$\omega_{ij}(new)=\omega_{ij}(old)+\alpha\ [x_i-\omega_{ij}(old)] \qquad (2.2)$$

step 7. 학습률 모수 및 이웃 범위값을 감소시킨다.
step 8. 종결조건을 확인한다.

학습률 모수 및 이웃 범위값을 감소시키는 방법에는 다

양한 함수가 사용될 수 있으며 학습률 모수에는 선형 감소 함수가 효과적인 것으로 밝혀졌다. 수직분할문제를 SOFM 네트워크로 풀기 위해 자료를 입력하는 방법 및 출력노드의 해석은 3.1절에서 설명한다.

## 3. 해법 개발

### 3.1 트랜잭션 빈도수 반영

본 연구에서의 0-1 속성사용행렬은 셀형 제조 시스템(cellular manufacturing system)환경에서의 기계-부품군(machine-parts group) 형성의 행렬과 유사하다. 그러나 0-1 속성사용행렬에서는 기계-부품군 형성의 행렬과는 달리 트랙잭션의 시간당 발생 빈도(frequency)와 속성의 길이(length)가 추가된다. 트랜잭션의 빈도수는 속성들의 그룹화, 즉 단편형성에 영향을 미친다. 즉, 빈도수가 큰 트랜잭션의 응답시간이 전체 응답시간에서 차지하는 비중이 크다. 따라서 이 트랜잭션이 사용하는 속성들을 하나의 단편으로 형성하여야 데이터 이동량이 많이 감소하기 때문에 트랜잭션의 빈도수를 함께 SOFM네트워크에 입력하는 방법을 제시한다. 제시한 해법의 타당성을 설명하기 위하여 그림 3과 같은 속성사용행렬을 사용한다.

| | | attributes | | | frequency of |
|---|---|---|---|---|---|
| | | a1 | a2 | a3 | transactions |
| trans-actions | t1 | 1 | 1 | 0 | 100 |
| | t2 | 1 | 0 | 0 | 25 |
| | t3 | 0 | 1 | 1 | 10 |
| | t4 | 0 | 1 | 1 | 5 |
| | | 5 | 2 | 10 | |

length of attributes

**그림 3. 속성사용행렬의 예**

만약 그림 3의 수치예제에서 빈도수를 고려하지 않고 0-1 액세스 패턴만으로 분할한다면 속성 a2 와 a3가 하나의 단편을 이루고 a1이 또 다른 단편이 되는 것이 유리하다. (트랜잭션 t3 와 t4가 요구하는 속성이 동일하기 때문에 SOFM은 두 속성을 하나로 묶는다). 그러나 t1의 빈도수가 다른 것에 비해 크기 때문에 t1의 비중이 크고 따라서 a1과 a2를 하나의 단편으로 하는 것이 액세스 비용을 감소시킨다. 실제로 액세스 비용을 계산하면

i) F1 = {a1, a2}, F2 = {a3}인 경우 비용은
1130 (=(5+2)*100+(5+2)*25+(5+2+10)*10+(5+2+10)*5) 이고

ii) F1={a1}, F2={a2, a3}인 경우 비용은
2005 (=(5+2+10)*100+5*25+(2+10)*10+(2+10)*5)

이다. 즉, F1 = {a1, a2}, F2 = {a3} 분할 방법이 유리하다. 그러나 속성사용행렬의 패턴만을 SOFM에 입력하면 SOFM은 0-1 패턴만 관찰하여 속성 a2와 a3를 하나의 단편으로 만드는 두번째 방법을 선택한다.

따라서 본 연구에서 제시한 트랜잭션의 빈도수를 가중치로 곱하여 SOFM 네트워크의 입력 벡터로 사용하는 경우에는 그림 3의 행렬에서 그림 4와 같은 행렬을 만들 수 있다. 이렇게 변형된 행렬을 SOFM 네트워크에 입력 벡터로 사용하면 단편의 결과는 F1 = {a1, a2}와 F2 = {a3}가 되어 트랜잭션의 빈도수를 반영할 수 있다.

| | | attributes | | | frequency of |
|---|---|---|---|---|---|
| | | a1 | a2 | a3 | transactions |
| trans-actions | t1 | 10 | 10 | 0 | 100 |
| | t2 | 2.5 | 0 | 0 | 25 |
| | t3 | 0 | 1 | 1 | 10 |
| | t4 | 0 | 0.5 | 0.5 | 5 |
| | | 5 | 2 | 10 | |

length of attributes

**그림 4. 빈도수를 고려한 속성사용행렬**

이는 트랙잭션의 빈도수를 가중치로 곱한 것을 SOFM 네트워크의 입력 벡터로 입력하면 학습 과정에서 연결가중치($w_{ij}$)가 트랜잭션 빈도수를 가중치로 곱한 것을 학습함으로써 결과적으로 a1과 a2를 하나의 단편으로 클러스터링하게 되기 때문이다. 따라서 본 논문에서 해법으로 제시한 트랜잭션의 빈도수를 가중치로 곱하여 SOFM 네트워크의 입력 벡터로 입력하는 것이 타당한 것을 알 수

있다.

그림 4에서 SOFM 네트워크에서의 입력벡터 x는 속성 사용행렬의 컬럼(column) 벡터이다. 예를 들면, 속성 a1에 대해서는 (10, 2.5, 0, 0)$^T$가 된다. 속성 a2 및 a3에 대해서도 마찬가지로 해당 컬럼벡타가 입력된다. 입력벡터는 가중치 벡터와 2차원 유클리디안 공간상에서의 거리 D(j)를 계산한다(식(2.1)참조). 3개의 입력 벡터 중에서 가장 거리가 작은 입력 벡터를 SOFM 네트워크의 학습식(식(2.2)참조)에 의거해서 가중치 벡터를 수정하게 된다. 이와 같은 과정을 여러번 반복하면 2차원의 유클리디안 공간상에서 속성 a1과 a2는 속성 a3에 비해서 상대적으로 가까운 거리에 있게 되어 하나의 클러스터, 즉 단편을 형성하게 된다.

그림 3의 0-1 액세스 패턴만 가지고 SOFM 네트워크를 이용하여 단편을 구하면 속성 a2와 a3가 상대적으로 a1보다 거리가 가깝기 때문에 하나의 단편이 된다. 이것은 0-1 액세스 패턴만을 고려한다면 속성 a2와 a3가 유사하기 때문에 당연한 결과라고 할 수 있다. 그러나 그림 4의 속성사용행렬을 SOFM 네트워크에 입력하면 t1의 액세스 빈도수 100이 가중치가 되어 최종 가중치 벡터가 바뀌게 되어 속성 a1과 a2가 하나의 단편이 되며 이러한 결과는 t1의 빈도수가 높기 때문에 합리적인 결과라고 할 수 있다. 즉, 빈도수를 가중치로 곱하여 입력하면 SOFM 네트워크가 학습 과정에서 이를 학습 결과인 클러스터, 즉 생성된 단편에 반영한다는 것을 알 수 있다.

다음은 임의의 문제에 대해서 트랜잭션의 빈도수를 곱한 행렬과 곱하지 않은 행렬에 대해서 각각 SOFM 네트워크로 해를 구해 보았다. 표 1에서 알 수 있듯이 트랜잭션의 빈도수를 가중치로 곱한 것의 평균 초과율은 5.8%이고 0-1 액세스 패턴만 입력한 것은 11.5%로 나와 트랜잭션의 빈도수를 가중치로 곱하여 나온 결과가 좋게 나왔다. 따라서 트랜잭션의 빈도수를 가중치로 곱하여 학습하는 것이 액세스 패턴만을 학습하는 것보다 더 분지한계법의 최적해에 근사하다는 것을 알 수 있다. 여기서 '초과율'이란 SOFM 네트워크의 해가 분지한계법으로 구한 최적해에 대해서 얼마나 초과했는지를 나타낸다.

또한, SOFM 네트워크의 초기 학습률값은 0.9와 0.5로 하고 실험을 하였고 학습률 함수의 형태는 지수형과 선형 함수로 실험을 수행했는데 최종적으로 생성된 단편의 클러스터링 결과에 영향을 미치지 않았다.

**표 1. 트랜잭션의 빈도수 실험 결과**

| 크 기 | 초 과 율 (%) | |
|---|---|---|
| | 0-1 패턴만 입력 | 트랜잭션 빈도수를 가중치로 곱하여 입력 |
| 7×7 | 9.6 | 5.5 |
| 7×7 | 13.3 | 5.4 |
| 7×7 | 5.2 | 3.9 |
| 7×7 | 5.4 | 0.0 |
| 7×7 | 0.0 | 0.0 |
| 12×12 | 12.6 | 7.1 |
| 12×12 | 16.0 | 9.8 |
| 12×12 | 27.1 | 8.9 |
| 12×12 | 13.7 | 10.3 |
| 12×12 | 11.4 | 6.2 |
| 평 균 | 11.5 % | 5.8 % |

### 3.2 동일 그룹 속성의 결합

본 논문에서의 화일 수직분할 문제는 속성의 수가 크면 검토하여야 할 가지 수가 증가하므로 속성의 수를 줄이면 계산량이 줄어든다. 즉, 액세스 패턴이 같은 동일 그룹 속성은 항상 같은 단편에 할당함으로써 액세스 비용을 감소시킬 수 있다.

먼저 그림 5의 (a)와 같은 속성 사용 행렬을 CI(Cluster Identification) 알고리듬을 적용하여 액세스 패턴이 유사한 속성들끼리 모아 (b)와 같은 형태의 행렬을 만든다 [3,5,9]. (b)에서 a1 및 a3는 모든 트랜잭션에 대해 사용되는 액세스 패턴이 동일한 것을 알 수 있다. 이러한 속성을 동일 그룹 속성(identical group attributes)이라 하고 같은 단편에 할당시켜 액세스 비용을 감소시킬 수 있다. 즉 동일 그룹 속성을 묶어 하나의 속성으로 처리할 수 있으며 속성의 길이는 동일 그룹 속성의 길이의 합으로 증가한다. 그림 5의 (c)는 동일 그룹 속성인 a1과 a3를 묶어 하나의 속성으로 만든 결과이다. 속성의 수는 5개에서 4개로 줄었고 a1'속성의 길이는 두 속성의 길이의 합인

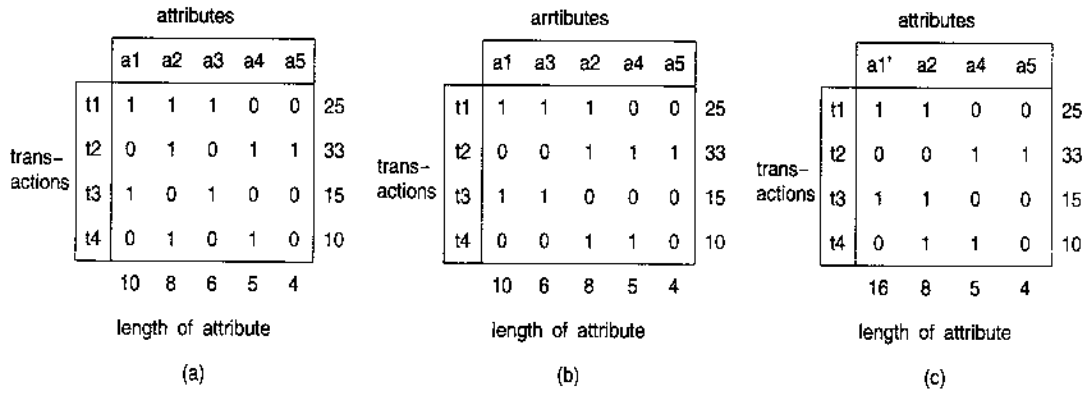


그림 5. 동일 속성 그룹의 예

16이 된다.

### 3.3 실험 및 결과 분석

Navathe[10]에 있는 속성이 10개, 트랜잭션이 8개인 문제의 해와 SOFM 네트워크의 해와 비교하였다. 표 2에서와 같이 SOFM 네트워크의 출력노드의 개수가 4일 때 분지한계법으로 구한 최적해와 동일하고 출력노드의 개수가 3일 때 Navathe의 발견적 기법의 해와 동일한 것을 알 수 있다.

3가지 크기(속성이 7개, 트랜잭션이 7개인 5문제, 속성이 8개, 트랜잭션이 10개인 10문제 및 속성, 트랜잭션이 모두 12개인 문제가 25개)의 40문제를 임의로 만들어 실험하였다. 모든 실험에 대해 주키의 길이는 2로 하였다. 실험 결과는 표 3과 같다. 트랜잭션이 두 단편에 있는 속성들을 요구할 때는 두 단편을 결합하여야 한다. 결합시 분할전의 레코드를 만들기 위하여 주키 역할을 하는 별도의 속성을 모든 단편에 첨가하여야 한다. 즉, 주키에는 일련번호를 보관하여 두 단편 결합시 동일 주키값의 레코드를 결합하면 분할전의 레코드 속성값을 구할 수 있다.

실험 결과로 알 수 있듯이 SOFM 네트워크의 근사해는 분지한계법의 최적해에 비해 평균 7.3%~11.0% 정도의 액세스 비용이 초과되었다. 또한, 크기가 큰 문제에 대해서도 비슷한 초과율을 나타내어 큰 문제를 SOFM 네트워크로 효과적으로 풀 수 있다는 것을 알 수 있다.

보다 현실적인 문제를 풀기 위하여 크기가 30×30인

표 2. Navathe[10] 문제에 대한 실험결과

| 사용방법 | 출력 노드 수 | 단 편 | 액세스 비용 |
|---|---|---|---|
| SOFM 방법 | 3 | F1 = {1, 5, 7}<br>F2 = {2, 3, 8, 9}<br>F3 = {4, 6, 10} | 8860 |
| | 4 | F1 = {1, 5}<br>F2 = {2, 7, 8}<br>F3 = {3, 9}<br>F4 = {4, 6, 10} | 7540 |
| 분지 한계법 | | F1 = {1, 5}<br>F2 = {2, 7, 8}<br>F3 = {3, 9}<br>F4 = {4, 6, 10} | 7540 |
| Navathe의 방법 | | F1 = {1, 5, 7}<br>F2 = {2, 3, 8, 9}<br>F4 = {4, 6, 10} | 8860 |

문제를 임의로 만들어 풀어 보았다. 문제의 크기가 커서 분지한계법으로 최적해를 구하는 것이 현실적으로 불가능하기 때문에 해를 상호 비교할 수는 없었다. SOFM 네트워크의 계산 시간은 출력노드의 개수에 따라서 차이가 있었지만 대략 160~180초 정도의 짧은 시간 안에 해를 얻을 수 있었다.

SOFM에서 출력노드의 수는 입력자료이다(2.2절의 step 1. 참조). 즉 표 2의 결과는 출력노드(단편)수를 3이나 4

표 3. 실험의 결과

| 크기 | 분지한계법 | SOFM | 초과율(%) | 평균 | 크기 | 분지한계법 | SOFM | 초과율(%) | 평균 |
|---|---|---|---|---|---|---|---|---|---|
| 7 × 7 | 4398 | 5404 | 22.8 | | 12 × 12 | 11856 | 11856 | 0.0 | |
| 7 × 7 | 7929 | 9070 | 14.4 | | 12 × 12 | 9220 | 10600 | 14.9 | |
| 7 × 7 | 3493 | 3493 | 0.0 | 10.5% | 12 × 12 | 4961 | 5403 | 8.9 | |
| 7 × 7 | 3665 | 4230 | 15.4 | | 12 × 12 | 10193 | 11013 | 8.0 | |
| 7 × 7 | 2217 | 2217 | 0.0 | | 12 × 12 | 6082 | 6138 | 0.9 | |
| 8 × 10 | 10215 | 10215 | 0.0 | | 12 × 12 | 5824 | 6011 | 3.2 | |
| 8 × 10 | 4302 | 6668 | 54.0 | | 12 × 12 | 14841 | 14993 | 1.0 | |
| 8 × 10 | 3267 | 3537 | 8.2 | | 12 × 12 | 10354 | 10554 | 1.9 | |
| 8 × 10 | 3569 | 3604 | 1.0 | | 12 × 12 | 10856 | 11253 | 3.6 | |
| 8 × 10 | 4847 | 5361 | 10.6 | | 12 × 12 | 6586 | 7193 | 9.2 | |
| 8 × 10 | 5007 | 5496 | 9.8 | 11.0% | 12 × 12 | 15442 | 15830 | 2.5 | 7.3% |
| 8 × 10 | 7375 | 8171 | 10.8 | | 12 × 12 | 21211 | 23194 | 9.3 | |
| 8 × 10 | 3879 | 3879 | 0.0 | | 12 × 12 | 16993 | 17651 | 3.9 | |
| 8 × 10 | 14427 | 15363 | 6.5 | | 12 × 12 | 4241 | 4599 | 8.4 | |
| 8 × 10 | 10896 | 11893 | 9.1 | | 12 × 12 | 10220 | 13315 | 30.2 | |
| 12 × 12 | 10082 | 11558 | 14.6 | | 12 × 12 | 10509 | 10509 | 0.0 | |
| 12 × 12 | 8926 | 9322 | 4.4 | | 12 × 12 | 23055 | 23175 | 0.5 | |
| 12 × 12 | 6783 | 8861 | 30.6 | | 12 × 12 | 13508 | 15432 | 14.2 | |
| 12 × 12 | 10243 | 11055 | 8.0 | | 12 × 12 | 8977 | 9356 | 4.2 | |
| 12 × 12 | 7203 | 7316 | 1.6 | | 12 × 12 | 12113 | 12342 | 1.9 | |

로 고정한 후 얻은 값이다. 다수의 문제를 실험한 결과 단편의 수에 대해서 엑세스 비용은 볼록 함수(convex function)의 형태를 나타내었다. 따라서 단편의 수를 증가시키면서 실험을 수행하면 최소비용의 단편수를 구할 수 있다. 표 3에 있는 SOFM 네트워크의 실험결과 값은 최소 비용을 나타낸다.

### 3.4 '1' 밀도에 대한 분석

크기가 10×10인 5문제의 0-1 속성사용행렬에서 '1'의 밀도를 0.2와 0.5의 두 가지 경우에 대하여 실험을 수행해 보았다. 여기서 '1'의 밀도란 0-1속성사용행렬에서 1이 차지하는 비율을 말한다. 이에 대한 결과는 표 4와 같다. 실험결과에서 SOFM 네트워크의 해는 '1'의 밀도가 낮으면 단편의 수가 많아지고 높으면 단편의 수가 적어지는 경향이 있어 분지한계법과 동일한 경향이 있음을 확인할 수 있었다. 또한, SOFM네트워크의 해는 '1'의 밀도 값이 낮을 때 분지한계법의 최적해에 더욱 가까운 해를 얻을 수 있었다.

SOFM 네트워크로 해를 구하는 실험은 IBM PC 호환 기종 메모리 32Mb인 펜티엄 150 MHz CPU 기종을 사용했고 C++언어로 구현하였다.

표 4. '1'의 밀도에 대한 해의 변화

| 예제 | 밀도 = 0.2 | | 밀도 = 0.5 | |
|---|---|---|---|---|
| | 단편 수 | 초과율(%) | 단편 수 | 초과율(%) |
| 1 | 7 | 3.4 | 6 | 6.2 |
| 2 | 8 | 1.0 | 3 | 14.4 |
| 3 | 8 | 5.0 | 5 | 5.3 |
| 4 | 8 | 6.8 | 5 | 8.2 |
| 5 | 8 | 0.5 | 5 | 1.2 |
| 평균 | | 3.4 % | | 7.1 % |

## 4. 속성의 중복 할당

앞 장에서는 화일수직 분할시 속성을 중복할당하지 않는 경우만을 고려하였으나 본 장에서는 SOFM 네트워크를 이용하여 속성을 중복 할당하는 발견적 기법을 제시한다. 속성의 중복 할당 방법을 그림 5 (c)를 이용하여 설명한다. F1 = {a1', a2}, F2 = {a4, a5}로 분할하는 방법은 비중복 분할 방법이며 F1 = {a1', a2, a4}, F2 = {a4, a5}는 속성을 중복 할당하는 방법이다. 즉 a4 속성을 두 단편에 모두 배치함으로써 t4 수행시 F2 단편을 이동할 필요가 없다.

이와 같이 단편에 속성을 중복 할당함으로써 트랜잭션이 액세스하는 단편의 수를 줄여 중복을 허용하지 않는 방법보다 디스크 액세스 비용을 줄일 수 있다.

### 4.1 속성의 중복 할당을 위한 발견적 알고리듬

SOFM 네트워크를 이용하여 속성을 중복 할당하는 발견적 기법은 다음과 같다.

step 1. 비중복 할당 최선해의 발견
- SOFM네트워크를 적용하여 액세스 비용을 최소로 하는 비중복 할당 단편을 결정한다.

step 2. 노드 탐색
- 각 속성에 대하여 출력층에서 승자노드 다음으로 거리가 가까운 노드를 탐색한다.

step 3. 비용절감 효과가 있는 속성 발견
- 각 속성에 대해서 step 2.에서 선택한 노드에 중복 할당을 하고 비용을 계산 하여 비중복 할당시의 액세스 비용보다 비용절감 효과가 있는지 검사한다.

step 4. 속성을 단편에 중복 할당
- 비용절감 효과가 있는 속성들을 선정하고 비용이 최소화가 되는 속성의 조합을 찾아 속성을 중복 할당한다.

### 4.2 수치 예제

그림 6과 같이 12개의 속성과 12개의 트랜잭션이 있는 수치 예제에 대하여 본 연구에서 제안한 발견적 기법을 적용하여 다음과 같이 해를 구해 보았다.

step 1. 비중복 할당 최선해의 발견

SOFM 네트워크를 적용하여 액세스 비용을 최소로 하는 비중복 할당 단편을 구한다. 그림 6의 수치 예제를 SOFM 네트워크로 적용하여 액세스 비용이 최소인 해를 구하면 F0 = {a9}, F1 = {a8}, F2 = {a10, a11}, F3 = {a7}, F4 = {a1, a4, a5}, F5 = {a2}, F6 = {a3, a6}, F7 = {a12}의 8개의 단편이 되고 이 때의 액세스 비용은 '11725'이다. 주커의 길이는 2로 하였다.

step 2. 노드 탐색

모든 속성에 대하여 출력층에서 승자노드 다음으로 거리가 가까운 노드를 찾는다. 주어진 수치 예제에 대해 이를 계산하면 표 5와 같다. 표 5의 승자노드가 단편이 되며 step1.에서 사용한 단편번호는 승자노드 번호와 동일한 값이다.

step 3. 비용절감 효과가 있는 속성 발견

step2.에서 발견한 승자노드 다음으로 가까운 노드에 속성을 중복 할당하고 비용을 계산하여 비용절감 효과가 있는 속성을 찾는다. 표 5에서 속성 a1, a4, a5는 승자노드 번호 4에 모두 속하므로 하나의 단편을 이룬다. 속성 a4 및 a5는 또한 노드번호 5에도 인접한 위치에 있으므로 속성 a2가 있는 F5단편에 속성 a4및 a5를 중복 할당하면 비용 절감이 발생할 것을 예상할 수 있다. 같은 방법으로 각각의 속성에 대하여 거리가 두번째로 가까운 노드에 속성을 중복시킬 때의 비용계산 결과는 표 5와 같다. 표 5의 마지막 열에 '*'로 표시되어 있는 것은 그 속성을 중복 할당할 경우 비용이 감소한 것을 나타낸다. 따라서 a2, a4, a5 및 a11 속성을 단편에 중복 할당할 경우 비용 절

| trans-actions \ attributes | a1 | a2 | a3 | a4 | a5 | a6 | a7 | a8 | a9 | a10 | a11 | a12 | frequency of transactions |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| t1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 32 |
| t2 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 30 |
| t3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 36 |
| t4 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 14 |
| t5 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 5 |
| t6 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 15 |
| t7 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 47 |
| t8 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 28 |
| t9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 38 |
| t10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 32 |
| t11 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 18 |
| t12 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 22 |
| | 11 | 2 | 11 | 5 | 3 | 3 | 14 | 9 | 12 | 10 | 1 | 7 | |

length of attributes

그림 6. 속성의 중복 할당의 수치 예제

감 효과가 있는 것을 알 수 있다.

표 5. 수치 예제에 대한 결과

| 속성 | 승자노드 | 2 번째 가까운 노드 | 중복할당시 액세스 비용 | |
|---|---|---|---|---|
| a1 | 4 | 3 | 12242 | |
| a2 | 5 | 4 | 11673 | * |
| a3 | 6 | 4 | 13735 | |
| a4 | 4 | 5 | 11532 | * |
| a5 | 4 | 5 | 11500 | * |
| a6 | 6 | 0 | 12175 | |
| a7 | 3 | 7 | 12467 | |
| a8 | 1 | 6 | 12876 | |
| a9 | 0 | 6 | 13529 | |
| a10 | 2 | 3 | 12195 | |
| a11 | 2 | 7 | 11173 | * |
| a12 | 7 | 3 | 12451 | |

step 4. 속성을 단편에 중복 할당

비용 절감 효과가 있는 속성들을 단편에 중복 할당한다. 표 5에서 a2, a4, a5 및 a11 속성을 중복 할당할 경우 비용 절감의 효과가 있으므로 이 4개의 속성을 선택적으로 중복 할당하여 비용 절감 효과가 큰 조합을 찾는다. 선택 가능한 조합에 대하여 각각 액세스 비용 계산을 하면 표 6과 같다.

표 6. 중복 할당에 대한 비용 계산

| 중복 할당한 속성의 조합 | 액세스 비용 |
|---|---|
| 2 | 11673 |
| 4 | 11532 |
| 5 | 11500 |
| 11 | 11173 |
| 2, 4 | 11673 |
| 2, 5 | 11673 |
| 2, 11 | 11232 |
| 4, 5 | 11307 |
| 4, 11 | 11091 |
| 5, 11 | 11059 |
| 2, 4, 5 | 11673 |
| 2, 4, 11 | 11232 |
| 2, 5, 11 | 11232 |
| 4, 5, 11 | 10866 |
| 2, 4, 5, 11 | 11232 |

속성 중복시 중복 조합중 최선해을 찾는 방법은 발견

적 기법을 사용한다. 표 6의 예제는 4개의 속성이 중복에 사용되므로 모든 조합의 경우를 고려하였으나 중복배치할 속성의 수가 큰 경우에는 불가능하다. 이러한 경우에는 비용 절감 효과가 큰 순서대로 일부의 속성만을 선택하여 이들의 조합만 고려하는 등의 방법을 사용할 수 있고 따라서 구하는 해는 최적해가 아닌 최선해가 된다.

표 6에서 속성 a4, a5, a11 속성을 중복 할당할 때 가장 비용이 낮게 나왔으므로 a4, a5, a11 속성을 중복 할당한다. 중복 할당하면 단편은 F0 = {a9}, F1 = {a8}, F2 = {a10, a11}, F3 = {a7}, F4 = {a1, a4, a5}, F5 = {a2, (a4), (a5)}, F6 = {a3, a6}, F7 = {a12, (a11)}이 되며 액세스 비용은 '10866'이다. 따라서 a4, a5, a11 속성을 중복 할당을 할 경우 중복 할당하지 않은 경우보다 7.3% 비용이 감소하였다. 수치예제의 결과를 종합하면 표 7과 같다. 원으로 표시된 숫자는 단편에 중복 할당된 속성을 나타낸다.

**표 7. 수치 예제의 중복 할당 결과**

| 속성을 비중복 할당할 경우 | 속성을 중복 할당할 경우 |
|---|---|
| F0 = {a9} | F0 = {a9} |
| F1 = {a8} | F1 = {a8} |
| F2 = {a10, a11} | F2 = {a10, a11} |
| F3 = {a7} | F3 = {a7} |
| F4 = {a1, a4, a5} | F4 = {a1, (a4), (a5)} |
| F5 = {a2} | F5 = {a2, a4, a5} |
| F6 = {a3, a6} | F6 = {a3, a6} |
| F7 = {a12} | F7 = {a12, (a11)} |
| 액세스 비용 = 11725 | 액세스 비용 = 10866 |

### 4.3 실험 및 결과 분석

크기가 5×5인 문제 5개와 12×12인 문제 5개를 속성을 중복한 경우와 중복하지 않은 경우에 대하여 실험을 하였다. 실험결과는 표 8과 같다. 실험 결과에서 속성을 중복할 경우 액세스 비용은 속성을 중복하지 않은 경우의 액세스 비용보다 평균 6.8 % 절감되는 것을 알 수 있다. 표 8의 예제에서는 비용감소 효과가 있는 속성의 모든 조합을 고려한 후 가장 좋은 중복 조합을 선택한 값이다.

본 연구에서 속성을 중복 할당할 때 조회 트랜잭션만을 대상으로 하였다. 왜냐하면 조회 트랜잭션일 때는 속성을 중복 할당하여 액세스하는 단편의 수를 줄임으로써 액세스 횟수를 줄일 수 있지만 갱신 트랜잭션의 경우 데이터의 일관성을 유지하기 위해서 중복된 속성을 모두 갱신해야만 하는 추가적인 작업이 필요하기 때문에 속성을 중복 할당할 경우 디스크 액세스 비용이 증가하게 된다.

**표 8. 실험 결과**

| 크 기 | 액세스 비용 | | 비용절감(%) |
|---|---|---|---|
| | 속성 비중복 | 속성 중복 | |
| 5×5 | 1710 | 1594 | 6.8 |
| 5×5 | 7696 | 7121 | 7.5 |
| 5×5 | 1182 | 1106 | 6.5 |
| 5×5 | 4165 | 3949 | 5.2 |
| 5×5 | 8198 | 7334 | 10.6 |
| 12×12 | 10272 | 9479 | 7.8 |
| 12×12 | 9195 | 8643 | 6.0 |
| 12×12 | 7193 | 6883 | 4.3 |
| 12×12 | 11725 | 10866 | 7.3 |
| 12×12 | 9934 | 9350 | 5.9 |
| 평균 | | | 6.8 % |

## 5. 결론

본 연구에서는 데이터베이스 설계에서 수직분할 문제를 해결하기 위하여 신경망의 하나인 SOFM 네트워크를 적용하는 방법에 대하여 연구하였다. SOFM은 비지도 학습 방식이므로 속성사용행렬의 패턴에 의하여 유사한 속성들을 결합하여 단편을 형성한다. 액세스 비용을 감소시키기 위하여 속성사용행렬에 트랜잭션의 빈도수를 가중치로 곱하는 방법을 제시하였다. 계산시간을 단축하기 위하여 동일 그룹 속성을 결합하였다. 실험을 통하여 SOFM 네트워크의 해가 n-ary 분지한계법을 이용한 최적해에 근사한 것을 발견하였다. 특히 큰 문제의 경우 분지한계법 사용에 시간적 제약이 있으므로 본 기법의 사용이 효과적이다. 또한 속성을 단편에 중복 할당하여 비 중복할당

보다 비용을 감소시키는 발견적 기법을 제시하였다.

## 참고문헌


[1] 김대수, 신경망 이론과 응용(I), 하이테크 정보, 1993.

[2] 윤병익, 김재련, "데이터베이스의 물리적 설계에서 분지한계법을 이용한 N-ary 수직분할문제", 대한산업공학회, 제 22권, 4호, pp 567-578, 1996.

[3] Cheng, C., "Algorithms for Vertical Partitioning in Database Physical Design", *Omega, Int. J. Mgnt. Sci.*, Vol 22, No 3, pp 291-303, 1994.

[4] Chu, W. W. and I. T. Ieong, "A Transaction-Based Approach to Vertical Partitioning for Relational Database Systems", *IEEE Trans. on Software Eng.*, Vol 19, No 8, pp 804-812, 1993.

[5] Cornell, D. C. and Yu, P. S., "An Effective Approach to Vertical Partitioning for Physical Design of Relational Databases", *IEEE Trans on Software Eng.*, Vol. 16, No. 2, pp. 248-258, 1990.

[6] Elmasri, R. and S. Navathe, *Fundamentals of Database Systems*, Benjaming/Cummings, second edition, 1994.

[7] Fausett, L., *Fundamentals of Neural Networks*, Prentice-Hall ,Inc , Englewood Cliff, New Jersey, 1994.

[8] Kulkarni, Y. R. and M. Y. Kiang, "Dynamic Grouping of Parts in Flexible Manufacturing Systems: A Self-Organizing Neural Networks Approach", *European Journal of Operation Research*, Vol. 84, pp. 192-212, 1995.

[9] Kusiak, A., *Intelligent Manufacturing System*, Prentice-Hall, Inc., Englewood Cliff, New Jersey, 1990.

[10] Navathe, S., S. Ceri, G. Wiederhold, and J. Dou, "Vertical Partitioning Algorithms for Database Design", *ACM Trans on Database Syst.*, Vol 9, No 4, pp 680-710, 1984

[11] Navathe, S. and M. Ra, "Vertical Partitioning for Database Design A Graphical Algorithm", *Proc. ACM SIGMOD Int Conf Management Data*, 1989

[12] Pandya, A. S and Macy, R. B., *Pattern Recognition with Neural Networks in C++*, CRC Press, Inc., Boca Raton, Florida, 1996.

[13] Simon Haykin, *Neural Networks*, Prentice Hall, Inc., Upper Saddle River, New Jersey, 1994.


98년 1월 최초 접수, 98년 9월 최종 수정