Journal of the Korean Institute
of Industrial Engineers
Vol.22, No.4, December, 1996

데이타베이스의 물리적 설계에서 분지한계법을 이용한 n-ary 수직분할문제

Branch-and-bound method for solving n-ary vertical partitioning problems in physical design of database

윤병익* · 김재련*

Byung-Ik Yoon · Jae-Yern Kim

Abstract

In relational databases the number of disk accesses depends on the amount of data transferred from disk to main memory for processing the transactions. N-ary vertical partitioning ofthe relation can often result in a decrease in the number of disk accesses, since not all attributes in a tuple are required by each transactions. In this paper, a 0-1 integer programming model for solving n-ary vertical partitioning problem minimizing the number of disk accesses is formulated and a branch-and-bound method is used to solve it. A preprocessing procedure reducing the number of variables is presented. The algorithm is illustrated with numerical examples and is shown to be computationally efficient. Numerical experiments reveal that the proposed method is more effective in reducing access costs than the existing algorithms.

1. 서 론

관계형 데이타베이스에서 트랜잭션/질의를 처리하기 위해 디스크와 주기억장치간에 이동해야 하는 자료의 양이 많으면 디스크 액세스 횟수(the number of accesses)가 증가하여 반응시간(response time)이 증가한다. 그러므로 논리적 설계(logical design) 또는 물리적 설계(physical design)를 실시할 때 반응시간을 줄이기 위하여 릴레이션(relation)을 몇 개의 단편(fragments)으로 수직분할(vertical partition), 수평분할(horizental partition) 및 혼합

* 한양대학교 산업공학과

분할(mixed partition)하여 디스크에 저장한다[6].

본 연구는 이러한 분할 방법 중에서 수직분할방법만 고려한다. 수직분할방법은 분할된 각 단편에 키 속성(primary key attribute) 또는 터플(tuple)을 구별할 수 있는 식별자((identifier)를 추가하여 저장한다. 이렇게 함으로써 키 속성에 의한 단편의 결합으로 원래의 릴레이션을 다시 만들 수 있다.

수직분할문제는 릴레이션의 속성들을 몇 개의 단편에 할당하는 조합 최적화 문제이다. m개의 속성들로 이루어진 릴레이션을 서로 다르게 수직분할하는 방법의 수는 Bell number(B(m))로 알려져 있다[12]. m이 클 경우, B(m)은 m^m과(예, $B(15) \approx 10^{15}$, $B(30) \approx 10^{23}$) 같아져, 수직분할문제는 계산량이 $O(M^m)$인 NP-Complete 문제다.

수직분할문제에 대한 기존 연구는 다음과 같다. Hoffer[7]는 각각의 부화일에 주어진 용량 제약하에서 저장, 검색, 갱신 비용을 고려하여 수직 단편을 결정하는 비선형, 0-1 정수계획모형을 개발하였다. Eisner and Severance [5]는 주기억장치(primary memory)에 저장되어 있고 모든 사용자에 의해 액세스 가능한 집합과 보조기억장치(secondary memory)에 저장되어 각 사용자에 의해 추가 비용으로 검색되는 집합들 중에서 레코드를 분할하는 수리적 기법을 제시하였다. Hoffer and Severance[8]는 속성들을 조합하여 높은 친밀도(high affinity)를 갖는 것들을 그룹핑하는 알고리즘을 제시하였다. Navathe 등[12]은 [8]의 연구를 확장하여 2단계 이진 수직분할 알고리즘을 제시하였다(1단계: 직관적 설계, 2단계: 비용을 고려한 설계). 1단계는 트랜잭션들의 속성 사용 행렬(attribute usage matrix)과 논리적 액세스 횟수(logical access frequency)를 이용하여 두 속성들 사이의 친밀도 행렬(attribute pairwise affinity matrix)을 만들어 가능한 한 block diagonal 형태로 만들기 위한 행과 열을 조합하는 클러스터링(clustering) 알고리즘을 적용한다. 위의 행렬을 두 속성 집합으로 나누어, 부적절한 속성들의 액세스를 최소화하는 분할 점(dividing point)이 결정된다. 2단계는 1단계에서 구한 단편을 비용 요인(cost factors)을 고려하여 수정한다. Navathe와 Ra[11]는 [12]의 수직분할방법의 친밀도 행렬에서 두 속성들 사이의 친밀도를 edge값으로 하는 친밀도 그래프 알고리즘(affinity graphical algorithm)을 개발하였다. Cornell and Yu[4]는 속성들을 물리적인 단편에 할당하여 디스크 액세스 횟수를 최소화하는 이진 분할 선형 정수계획법을 개발하였다. 또한 재분할에 관한 수리모형을 제시하였으나, 분할의 수가 커지면 수리적 접근 방식이 불가능하다. Chu[3]는 Cornell and Yu의 문제를 트랜잭션에 근거한 접근 방법(transaction-based approach)으로 이진 수직분할을 연구하였다. 이진 최적 수직분할은 reasonable cut 에서만 존재함을 보이고, 이진 분할에 대한 해법으로 분지한계법과 발견적 기법을 제시하였다. 알고리즘의 계산량은 $O(2^n)$으로 트랜잭션의 수 n에 영향을 받는다. 그러나 이 연구에서 제시한 분지한계법은 가지치기(fathoming)의 효과가 발생하지 않으므로 완전 열거법(exhaustive search)과 동일하다. Cheng[2]은 속성 사용 행렬의 mutually separable cluster를 찾아내는 분지한계법과 발견적 기법을 제시하였다. 분지한계법에서 CI 알고리즘[9]을 나무

(tree)의 각 노드(node)마다 적용하여 inter-submatrix attributes을 찾는다. 이 연구는 입력변수로 단지 속성 사용 행렬만을 사용하므로 최적 디스크 액세스 횟수를 구한다는 보장이 없다.

이와 같이 수직분할문제에 대한 기존 연구는 논리적 설계단계와 물리적 설계단계에서의 분할로 나눌 수 있다. 전자의 경우 친밀도를 최대화하는 연구가 있고[2, 8, 11, 12], 후자는 디스크 액세스 횟수를 최소화하는 연구가 있다[3, 4]. 두 부류의 연구는 속성들을 두개의 단편으로만 분할하는 이진 분할에 관한 알고리즘을 개발하여, 두 개 이상의 단편으로 분할할 경우, 알고리즘을 반복 적용하여 해를 구하므로 최적 단편이 여러 개인 문제에서는 최적해를 구한다는 보장이 없다. 또한 저장, 검색 등의 비용을 고려하여 수직분할하는 연구도 수행되었다[5, 7].

본 연구는 수직분할문제의 최적해를 찾기 위한 n-ary 분할 방법의 수리모형을 제시하고, 이의 해법으로 분지한계법을 사용한다. 또한 계산량을 줄이기 위한 선행처리과정을 제시한다.

다음절에서는 수직분할문제에 대한 개념을 소개한다. 3.1에서는 본 연구에서 개발한 수직분할에 대한 수리모형을 제시하고, 3.2에서는 이 모형에 대한 해법으로 분지한계법을 제시하며, 3.3에서 알고리즘의 계산량을 줄이는 선행처리방법을 제시한다. 4절에서 수치예제를 제시하고, 분지한계 알고리즘을 평가하며, 5절에서 결론을 제시한다.

2. 수직분할문제

트랜잭션은 대상 릴레이션의 모든 속성을 필요로 하지 않는 경우가 대부분이다. 이러한 경우에 모든 속성들을 참조한다면 필요하지 않은 속성의 자료들이 디스크와 주기억장치 사이를 이동하여야 하므로 디스크 액세스 횟수는 증가한다. 그러므로 릴레이션을 몇 개의 단편으로 수직분할하여 불필요한 자료의 이동 양을 줄이면 디스크 액세스 횟수를 줄일 수 있다.

디스크 액세스 횟수는 트랜잭션의 액세스 패턴(access pattern), 액세스 빈도수(access frequency), 액세스 방법(access method)에 따라 달라진다[3]. 그러므로 액세스 횟수에 영향을 주는 요인 분석이 필요하다. 본 연구에서는 액세스 횟수에 영향을 주는 요인을 입력변수와 액세스 방법으로 나누고, 입력변수는 릴레이션에 관련된 변수와 트랜잭션에 관련된 변수로 나누어 분석한다.

입력변수 중 릴레이션에 관련된 변수는 릴레이션을 구성하는 속성의 수와 각 속성의 길이(byte) 및 터플의 수가 있다. 트랜잭션에 관련된 변수로는 트랜잭션 수행에 필요한 속성을 나타내는 액세스 패턴(access pattern)과 단위 시간에 발생하는 상대적 발생 횟수(relative arrival frequency), 명제의 선택 조건(selectivity of predicate) 등이 있다. 이러한 입력변수값은 시스템 분석 및 설계단계에서 얻어진다. 특히, 트랜잭션에 관련된 변수는 일반적으로 "80/20" 규칙을 따른다[1]. 즉, 중요한 20%의 트랜잭션을 효과적으로 처리하면 전체 업무 중 80%를 효과적으로 처리하는 결과가 된다.

디스크 액세스 횟수는 액세스 방법에 따라 다르기 때문에 액세스 방법을 고려하여야 한다[10]. 데이타베이스 시스템에서 트랜잭션을 처리할 때, 시스템의 질의 최적기(query optimizer)가 디스크 액세스 횟수를 최소화하는 스캔방법을 선택한다. 시스템마다 스캔방법은 다를 수 있으므로 관계형 데이타베이스에서 흔히 사용하는 세그먼트 스캔방법(segment scan method)과 인덱스 스캔방법(index scan method)을 설명한다[4].

세그먼트 스캔은 한 릴레이션의 모든 레코드를 검색한다. 스캔은 순차적이기 때문에 대상 릴레이션의 터플들을 차례로 모두 주메모리(main memory)로 가져와야 한다. 인덱스 스캔방법에서는 색인되어 있는 속성의 조건을 만족하는 터플의 수에 따라 트랜잭션에 의해 검색되는 페이지 수가 달라진다. 만약, 후보키(candidate key)가 아닌 속성으로 색인되어 있고, 이 속성에 대하여 클러스터링(clustering)되어 있다면, 이를 clustered index scan이라 한다. 후보 키 또는 주 키(primary key)에 대하여 색인되어 있는 경우를 unclustered index scan이라 하며, 이 방법은조건을 만족하는 레코드를 검색하기 위하여 하나의 블럭을 이동하여야 하는 것이 일반적이다[6].

이와 같은 관계형 데이타베이스에서 사용하는 스캔방법들의 액세스 횟수는 다음과 같이 추정할 수 있다[3].

· Segment scan method:

$$\text{Number of accesses} = \frac{(number\ of\ tuple)(length\ of\ tuple)}{(page\ size)(prefetch\ blocking\ factor)} \quad (2.1)$$

· Clustered index scan method:

$$\text{Number of accesses} = \frac{(number\ of\ tuple)(selectivity)(length\ of\ tuple)}{(page\ size)} \quad (2.2)$$

· Unclustered index scan method:

$$\text{Number of accesses} = (\text{number of tuple})\,(\text{selectivity}) \quad (2.3)$$

위의 식에서, 액세스 횟수는 segment scan 방법과 clustered index scan 방법에서 터플의 길이에 비례하므로, 터플의 길이를 사용하여 액세스 횟수를 구한다. 다음 절에서 트랜잭션 수행에 필요한 터플의 길이를 고려하여 액세스 횟수를 최소화하는 수리모형을 제시한다.

3. 수직분할 모형

본 연구에서 개발한 수직분할문제에 대한 수리모형, 알고리즘 및 선행 처리를 설명한다.

3.1 사용기호 및 모형

본 연구에서 사용하는 기호는 다음과 같다.

－사용기호

m: 릴레이션의 속성 개수

n : 릴레이션을 액세스하는 트랜잭션 수

p : 단편 개수

i : 속성 번호 (i=1, …, m)

j : 트랜잭션 번호 (j=1, …, n)

k : 단편 번호 (k=1, …, p)

－입력변수

ID : 키 속성의 길이(byte)
a_i : 속성 i의 길이(byte)
F_k : 단편 k의 터플 길이
C_R : 릴레이션의 터플의 수

$$b_{ij} = \begin{cases} 1 & \text{속성 } i\text{를 트랜잭션 } j\text{가 사용할 경우} \\ 0 & \text{사용하지 않을 경우} \end{cases}$$

f_j : 트랜잭션 j의 단위 시간당 상대적 발생 횟수
S_j : 트랜잭션 j의 명제만족률
A_j : 트랜잭션 j의 액세스 비용

$$T_{jk} = \begin{cases} 1 & \text{트랜잭션 } j\text{가 단편 } k\text{를 필요로 하는 경우} \\ 0 & \text{필요로 하지 않는 경우} \end{cases}$$

－결정변수

$$X_{ik} = \begin{cases} 1 & \text{속성 } i\text{가 단편 } k\text{에 할당될 경우} \\ 0 & \text{그렇지 않을 경우} \end{cases}$$

수직분할문제에 대한 본 연구의 수행 척도는 트랜잭션에 필요한 디스크 액세스 비용(본 연구에서 액세스 횟수 대신 액세스 비용을 수행척도로 사용한다. 만약, 각 단편에 대해 액세스 횟수를 구체적으로 계산하려면 시스템에서 정의하는 page size와 blocking factor를 모형에 포함시켜서 본 알고리즘을 사용할 수 있다(식 (2.1), (2.2) 참조))이다. m개의 속성으로 구성된 릴레이션을 액세스하는 트랜잭션이 n개가 있다고 하자. 이 릴레이션이 p개의 단편으로 분할되었다고 가정할 때, 트랜잭션 j가 필요한 단편들을 접근하는 데 필요한 액세스 비용은 분할된 단편에 영향을 받으므로 $A_j = \Sigma_{k=1}^{p} l(C_R \cdot S_j \cdot (F_k + ID) \cdot T_{jk})$로 나타낼 수 있다. 이 식에서 l은 트랜잭션 j가 단편 k를 액세스할 때 필요한 I/O횟수를 계산하는 비용 함수이며, 이 값은 액세스 방법에 따라 달라진다. 예를 들어, segment scan 방법을 사용할 경우, 트랜잭션 j에 단편 k만 필요하면, 단편을 차례로 주 메모리로 이동시켜야 하므로, 이에 필요한 액세스 비용은 $A_j = C_R \cdot (F_k + ID)$가 된다. 또한, 단위 시간에 발생하는 트랜잭션의 상대적인 발생 횟수가 다를 수 있으므로, 트랜잭션 j의 단위 시간에 발생하는 액세스 비용은 $f_j \cdot A_j$가 된다. 그러므로 단위 시간에 발생하는 모든 트랜잭션의 총 액세스 비용은 $\Sigma_{j=1}^{n} f_j \cdot A_j$가 된다. clustered index scan방법을 사용할 경우는 S_j가 고려되어야 한다. 액세스 방법을 고려한 트랜잭션의 총 액세스 비용(Z)을 최소화 하는 0-1 정수계획모형은 다음과 같다.

$$Minimize\ Z = \sum_{j=1}^{n} f_j \cdot A_j \tag{3.1}$$

subject to

$$A_j = \sum_{k=1}^{p} l(C_R \cdot S_j \cdot (F_k + ID) \cdot T_{jk}),\ \ ^{\forall}J \tag{3.2}$$

$$F_k = \sum_{i=1}^{m} a_i X_{ik},\ \ ^{\forall}k \tag{3.3}$$

$$\sum_{k=1}^{p} X_{ik} = 1,\ \ ^{\forall}i \tag{3.4}$$

$$(\sum_{i=1}^{m} b_{ij} \cdot X_{ik})/m \le T_{jk} \le \sum_{i=1}^{m} b_{ij} \cdot X_{ik},\ \ ^{\forall}j,k \tag{3.5}$$

$$X_{ik} = \{0,1\},\ T_{jk} = \{0,1\},\ ^{\forall}i,j,k \tag{3.6}$$

목적식 (3.1)은 모든 트랜잭션의 액세스 비용의 합으로, 각 트랜잭션이 접근해야 하는 데이타의 양이 감소하면 총 액세스 비용은 감소한다. 제약식 (3.2)는 트랜잭션 j의 액세

스 비용을 나타낸다. 제약식 (3.3)은 단편 k에 할당된 속성들의 속성 길이의 합으로, 단편 k의 터플 길이를 나타낸다. 식 (3.4)는 모든 속성은 하나의 단편에만 할당되어야 하는 제약이다. 본 연구에서는 중복 할당을 고려하지 않는다. 식 (3.5)는 트랜잭션 j가 단편 k에 할당된 속성들 중에서 적어도 하나의 속성을 필요로 하면, 단편 k를 액세스 해야하는 조건이다. 식 (3.6)은 변수가 0 또는 1의 값을 갖는 제약이다.

액세스 횟수를 수행 척도로 하는 수직분할 문제에 대한 기존의 해법으로는 트랜잭션에 기초한 최적 이진 분할 방법[3]과 이진 분할 선형 정수 계획법[4] 등이 있다. 이러한 기존 해법은 이진 분할에 대한 해법만을 다루어 셋 이상의 단편에 최적해가 존재할 때, 최적해를 구하기 어려운 단점이 있다. 다음에 모형에 대한 최적 n-ary 수직분할 알고리즘을 제시한다.

3.2 분지한계법

본 연구에서 제시하는 분지한계법은 다음 두 가지 가정을 이용한다.

1) 액세스 비용은 터플의 길이에 대하여 비감소함수(non-decreasing function)이다.

2) 단편에 속성을 추가하면, 추가되기 전보다 액세스 비용은 증가한다.

이와 같은 가정하에 속성을 하나씩 단편에 추가해서 모든 속성이 할당된 하나의 가능해(feasible solution)를 구한 후 다른 조합으로 생성되는 열등해를 제외시킬 수 있다. 이렇게 할당 순서를 변경시켜 가면서 최적해를 찾는 분지한계법은 다음과 같다.

단계 0. [초기화]

- b_{ij}, a_i, f_j, S_j, C_R, ID, P
- P에 따른 모든 가능해를 표현하는 나무(tree)를 구성한다.
- node_cost = 0
- $Z_u = \infty$

단계 1. [분기] 깊이 우선순위로 가지치기 하지 않은 새 노드를 선택한다.

- 선택된 새 노드의 node_cost를 구한다.

단계 2. [가지치기]

- 만약 node_cost가 Z_u보다 크면, 그 노드의 자식 노드를 가지친다.

단계 3. [한계]

- 만약 노드가 잎새(leaf)노드이고 node_cost가 Z_u보다 작으면 Z_u를 node_cost로 바꾼다.

단계 4. [종료 조건]

- 나무에서 가지치기 안된 남아 있는 노드가 없으면 종료하고, 현재 Z_u를 최적해로 한다. 그렇지 않으면 단계 1로 간다.

위의 알고리즘에서 모든 가능해를 표현하는 나무는 다음과 같다(Figure 1 참조). 뿌리 노드는 단편 F_1에 속성 1을 할당 시켰을 때의 액세스 비용(node_cost)이다. 자식 노드의 개수는 단편의 수 p와같고, 깊이 우선순위의 첫번째 자식 노드는 단편 F_1에 속성 1과 2를 할당시켰을 때의 액세스 비용을 나타낸다. Figure 1에서 Level 2의 왼쪽 첫번째 노드는 단편 F_1에 속성 1, 2 및 3을 할당시켰을 때의 액세스 비용을 나타낸다. 이러한 순서로 할당하여 첫번째 잎새 노드는 모든 속성을 단편 F_1에 할당시켰을 때의 액세스 비용이며,

하나의 가능해(feasible solution)를 나타낸다. 두번째 잎새 노드는 마지막 속성만 단편 F_2에 할당시키고, 나머지 속성은 모두 단편 F_1에 할당했을 경우의 액세스 비용을 나타낸다. 모든 가능해는 잎새 노드로 표현되며, 자식 노드의 값은 부모 노드의 값보다 크거나 같다. 알고리즘은 최소화 문제에 대한 것이므로, Zu는 액세스 비용의 상한(upper bound)을 나타낸다. 뿌리 노드에서 잎새 노드로 탐색할 때 노드 값은 증가하므로 임의의 노드 값이 Zu보다 크면 그 노드의 자식 노드 값도 Zu보다 크게 되어 자식 노드를 가지칠(fathom) 수 있다. 이와 같은 방법으로 가지치기 되지 않은 나무의 모든 노드를 탐색했을 때, 현재의 Zu가 최적해가 된다.

p=m로 할 경우, 자식 노드의 수는 m개가 되고, 가능해의 수는, m이 클 경우, m^m과 같아지므로 나무를 구성하는 노드 수와 계산량은 m이 커짐에 따라 지수적으로 증가한다. 그러나 최적 단편 수 p*는 일반적으로 m보다 작은 값이 되므로, 본 알고리즘에서는 단편의 수 p값을 작은 값에서 증가시켜 가면서 p*를 구한다. 또한 단편의 수 p를 큰 값으로 하여 풀어도 p*는 동일한 값을 얻는다. p값을 p*보다 큰 값으로 하여 알고리즘을 적용하면 최적해가 p*이므로, p가 $p^* < p \le m$의 단편에는 속성을 할당하지 않는다. 이러한 이유는 단편을 많이 생성시키면 생성된 단편에 ID를 추가하여야 하고, 추가된 ID는 총 액세스 비용의 증가를 초래하기 때문이다.

3.3 선행처리

본 연구에서 제시한 모형의 해법은 속성의 수 m에 영향을 받는다. 그러므로 해법에서 고려해야 할 속성의 수를 줄이면 변수가 줄어들어 계산량을 줄일 수 있다.

Table 1(a)는 속성 사용 행렬(attribute usage matrix)과 속성 i의 길이(byte)를 나타낸다. Table 1(a)에서 속성 1과 2가 각 트랜잭션에 사용되는 액세스 패턴이 동일한 것을 알 수 있다.

정의〉 속성 i와 i'(i≠i')를 사용하는 트랜잭션의 액세스 패턴이 동일하면, i와 i'를 동일 그룹 속성(identical group attributes)이라 한다.

정리〉 동일 그룹 속성을 모두 하나의 단편에 할당시키는 경우의 액세스 비용은 다른 단편에 분산시켜 할당 시킬 경우의 액세스 비용보다 작거나 같다.

Table 1. Parameters related in attributes and transactions

		Attributes				
		1	2	3	4	5
Transactions	t1	1	1	1	0	0
	t2	0	0	1	1	1
	t3	1	1	0	0	0
	t4	0	0	1	1	0
	ai	10	8	6	5	4

(a)

		Attributes			
		1'	3	4	5
Transactions	t1	1	1	0	0
	t2	0	1	1	1
	t3	1	0	0	0
	t4	0	1	1	0
	ai	18	6	5	4

(b)

증명〉 수직분할 알고리즘에 의해 얻어진 최적 분할이 F1, F2, …, Fp라 하자. 또한 릴레이션에 관련된 모든 트랜잭션 집합을 T라 하고, 동일 그룹 속성을 액세스하는 트랜잭션 집합을 T_i라 할 때, 정의로부터 $T = T_I + T_I^c$의 관계식이 성립한다. 3절의 식 (3.2)로부터, T_i의 한 트랜잭션을 t_i라 할 때, t_i의 액세스 비용은 단편의 터플 길이에 비례하므로 다음과 같이 표현할 수 있다.

$$A_{t_i} = k(C_R \cdot (F_1 + ID) \cdot T_{1k}) + \cdots + k(C_R \cdot (F_k + ID) \cdot T_{1k}) + \cdots + k(C_R \cdot (F_{k'} + ID) \cdot T_{1k}) + \cdots + k(C_R \cdot (F_p + ID) \cdot T_{1k})$$

이 식에서, 속성 i가 단편 F_k에 속하고, i'가 $F_{k'}$ $(k \neq k')$에 속해 있다고 하자. 이 때 다음의 두 경우가 존재한다.

1) 트랜잭션 t_I가 F_k의 속성 중에서 i만을 사용할 경우
2) 트랜잭션 t_I가 F_k의 속성 중에서 i 및 다른 속성을 사용할 경우

만약, 속성 i를 $F_{k'}$로 이동시키면, 1)의 경우에서 t_I가 F_k를 액세스 하지 않으므로 액세스 비용이 감소하고, 2)의 경우는 액세스 비용에 변화가 없다. 이것은 T_I의 모든 트랜잭션에 대해서도 같으며, 이러한 이동 결과는 T_I^c의 액세스 비용에 영향을 주지 않는다. 그러므로 동일 그룹 속성을 항상 같은 단편에 할당시켜 최적해를 구할 수 있다.

이와 같이 동일 그룹 속성을 같은 단편에 할당시켜 최적해를 구할 수 있으므로, 동일 그룹 속성들을 묶어 하나의 속성으로 처리할 수 있다. Table 1(b)는 선행처리 후의 결과로 새로운 속성 1'는 동일 그룹 속성 1과 2를 합쳐 놓은 것이며, 속성 길이는 동일 그룹 속성 길이의 합으로 증가한다.

이러한 동일 그룹 속성들을 합치는 선행처리 과정을 거치면 속성 개수는 m에서 m'(m ≥ m')로 줄어들어 본 연구에서 제시한 분지한계법의 계산량 $O(m^m)$이 $O(m'^{m'})$으로 감소한다. 선행 처리 과정은 nm의 속성 사용 행렬에서 같은 속성을 찾기 위해 행을 쌍으로 비교하여야 하므로 계산량은 $O(nm^2)$으로, 분지한계법의 계산량 $O(m^m)$에 비하여 작으므로 무시할 수 있다.

4. 수치예제 및 알고리즘 평가

예제를 통하여 알고리즘을 설명하고, 임의로 만든 문제와 기존 연구의 문제로 알고리즘을 평가한다.

4.1 수치예제

속성과 트랜잭션이 각각 4개인 문제를 풀어 본다. 입력변수의 값이 Table 2에 주어졌으며, ID=2(byte), S_j=1로 사용하였다. C_R값은 모든 단편에서 동일하므로 최적해를 구하는 과정에 영향을 미치지 않으므로 C_R=1로 사용한다.

Figure 1에는 단편의 수를 p=3개로 하였을 때의 해법 과정을 나타낸다. 속성을 단편에 할당할 때, 임의의 속성 하나를 하나의 단편에 고정하여 할당할 수 있다. 따라서 속성1을 항상 단편 F_1에 할당시킬 수 있다. 뿌리 노드가 이러한 경우를 나타내며, 이때의 node_cost는 트랜잭션 1과 3이 속성 1을 사용하므로 (8 + 2) * 30 + (8 + 2)25 = 550이 된

Table 2. Input parameters

Attribute usage matrix(b_{ij})						Arrival frequencies of transactions per unit time(f_i)
		Attributes				
		1	2	3	4	
Trnsactions	1	1	1	0	0	30
	2	0	0	1	1	5
	3	1	1	0	0	25
	4	0	0	1	0	10
Length of attributes	a_i	8	6	4	2	

{1, 2}, F_2={3}, F_3={4}가 된다. 이와 같이 분할할 경우 트랜잭션 1의 액세스 비용은, 단편 F_1만 액세스하면 되므로, A_1=(8+6+2)*30=480이 된다. 나머지 트랜잭션의 액세스 비용도 같은 방법으로 구하고, 이것들을 합하면 총 액세스 비용(990)이 된다. 잎새 노드 중에서 최적해(990)와 같은 값을 갖는 경우는 분할 방법이 같다. 즉, 속성 1과 2를 하나의 단편에 할당하고, 속성 3과 4는 각각 다른 두 개의 단편에 할당하는 경우이다. 이 문제에서 최적 단편 수는 p*=3이므로 p=4로 하여

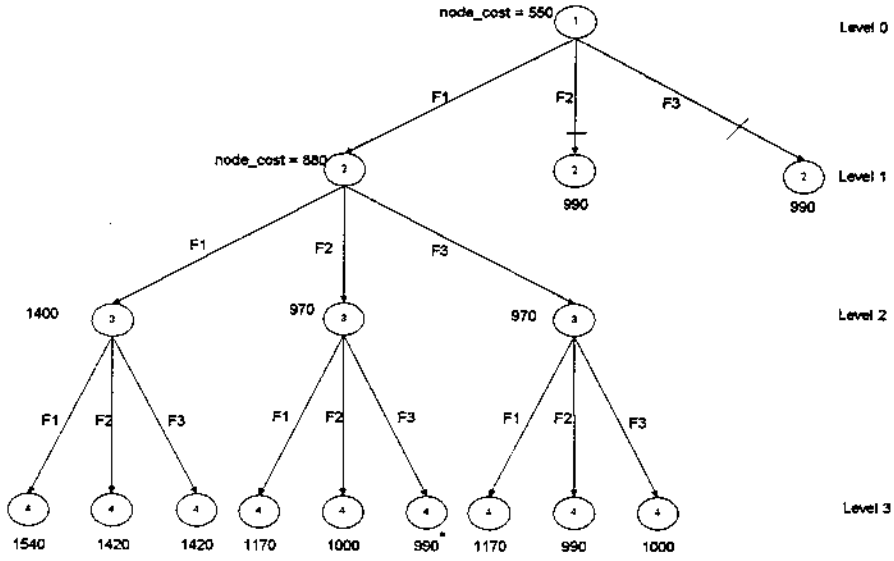

Figure 1. Example search tree used by branch-and-bound algorithm

다. Level 1의 첫번째 노드의 node_cost는 속성 1과 2가 단편 F_1에 할당되었을 때의 액세스 비용을 나타낸다. 최적해는 깊이 우선순위를 적용하여 잎새 노드인 가능해 중에서 선택되므로, 여섯번째 잎새 노드 값(Z*=node_cost=990)이다. 그때의 최적 분할 단편은 F_1=

풀었을 때도 p*=3을 얻는다. 최적해는 액세스 비용이 터플의 길이에 비례하는 특성을 이용하였으므로, 이 방법으로 분할하면 segment scan과 clustered index scan을 사용할 때 액세스 횟수를 줄일 수 있다.

4.2 알고리즘 평가

알고리즘을 평가하기 위하여 기존 연구의 문제와 임의로 만든 문제를 풀어 본다. 첫째, 속성이 10개, 트랜잭션이 8개인 기존의 Navathe[12] 문제를 본 연구와비교한다. 문제의 변수를 ID=10, C_R=5,000 및 S_j=1로 가정하여, 본 연구에서 제시한 방법으로 푼 결과가 Table 3에 나타나 있다. Table 3은 릴레이션을 분할하지 않고 사용하는 경우와 분할하여 사용하는 경우의 비용과 분할 단편을 비교하는 것이다. 또한 Chu[3]의 결과는 트랜잭션에 기반한 이진 분할해를 나타낸다. 이 결과에서, 분지한계법을 사용하여 4개의 단편으로 분할한 최적해는 분할하지 않을 때의 액세스 비용($1.08*10^8$)의 52%를 절감하였고, 이 값은 Chu의 이진 분할해보다 19%가 더 절감된 것을 알 수 있다. 본 알고리즘으로 최적해를 얻는 데 걸린 수행 시간은 Chu의 이진해를 구하는 수행시간과 차이가 없었다.

Table 3. Experimental results on the problem in [12]

	Access cost	Cost Reduction (%)	Fragment
B & B Method	$5.23*10^7$	52	F1={1,5} F2={2,7,8} F3={3,9} F4={4,6,10}
Chu's Binary Method	$7.24*10^7$	33	F1={1,2,3,5,7,8,9} F2={4,6,10}

둘째로, 20개 문제를 Table 4에 따라 임의로 만들어 풀어본다. 각 문제의 속성과 트랜잭션의 수는 12개씩이다. 이 문제들 중에서 10문제는 트랜잭션이 속성을 사용할 확률(Pr)을 0.25로 하였고, 나머지 10문제는 0.35로 하여, 두 경우로 나누어 만들었다. IBM 호환기종 펜티엄급, CPU clock 90Mhz 컴퓨터를 사용하고, C/C++언어로 구현하여 푼 결과가 Table 5에 나타나 있다. 모든 문제에 대하여 단편의 수(p)를 증가시켜 가면서 풀었고, 8보다 작은 단편 수에서 최적해를 구했기 때문에 p=8까지만 나타냈다. (2)열은 p값에서 알고리즘의 나무를 구성하는 잎새 노드 수이고, (3)열은 알고리즘에 의해서 탐색된 평균 잎새 노드 수며, (4)열은 평균 실행 시간을 나타낸다. 본 알고리즘의 계산량은 나무의 잎새 노드 수에 크게 영향을 받는다. 알고리즘이 탐색한 평균 잎새 노드 수가 나무의 모든 잎새 노드 수보다 상당히 작으므로 평균 실행 시간이 짧고, 알고리즘의 효율이 좋은 것을 알 수 있다. 만약, 동일 그룹 속성이 존재하여 선행처리를 실시할 경우, 더 빠른 시간에 최적해를 구할 수 있다. (5)열은 분할하지 않았을 때에 대하여 p개의 단편으로 분할했을 때의 평균 액세스 비용절감을 Pr값으로 구별하였다. Pr=0.25일 때가 Pr=0.35보다 평균 액세스 비용 절감이 2배 정도 크게 나타났다. 또한, Pr=0.25인 10문제는 최적 단편수 p*=6,

Table 4. Parameter value used in the algorithm

Parameters	Values
ID	12
Length of attribute a_i	$1a_i \le 15$
Selectivity of transaction j	$S_j=1$
Frequency of transaction f_i	$5 \le f_i \le 50$
A probability of an attribute accessed by a transaction(Pr)	(0.25) or (0.35)
Cardinality	5,000

Table 5. Experimental results of 12*12 problems

(1) P	(2) No. of leaf node	(3) Avg. no. of searched leaf node	(4) Avg. processing time(sec)	(5) Avg. cost reduction(%) Pr=0.25	Pr=0.35
2	2,048	34	0.055	25	12
3	177,147	450	0.81	36	18
4	4,194,304	4,042	7.77	41	21
5	4,8,828,125	28,259	53.4	47	23
6	362,797,056	189,399	315.6	52	24
7	1,977,326,743	240,076	1,461	54	25
8	8,589,934,592	1,274,605	11,843	55	-

p*=7 및 p*=8이 각각 1, 5 및 4문제였고, Pr=0.35인 경우는 p*=5, p*=6 및 p*=7이 각각 3, 6 및 1문제가 나왔다. 이 결과는 Pr=0.35로 만든 문제는 Pr=0.25로 만든 문제보다 적은 단편의 수에서 최적해가 존재함을 알 수 있다. 위의 결과로부터 다음과 같은 수직분할 문제의 특성을 알 수 있다. 속성 사용 행렬에 "1"의 밀도가 높으면(Pr 값이 1.0에 가까우면), 트랜잭션들이 릴레이션의 대부분의 속성을 사용하는 것이므로, 최적 단편의 수가 적어지는 경향이 있으며, 분할하여 얻을 수 있는 액세스 비용 절감의 효과도 적어진다. 반대로, "1"의 밀도가 낮으면(Pr 값이 0.0에 가까우면), 많은 수의 단편으로 분할될 가능성이 크고, 액세스 비용도 많이 감소한다.

위 문제에 대한 액세스 비용 감소 효과를 본 알고리즘과 Chu의 방법을 비교하여 Table 6에 나타냈다. 본 알고리즘의 최적해는 분할하지 않았을 때보다 최대 59%에서 최소 18%의 액세스 비용을 절감할 수 있었고, Chu의 이진 분할방법은 최대 34%에서 최소 9%를 절감하였다. 또한 본 알고리즘은 Chu의 해보다 평균적으로 20%를 더 절감할 수 있었다.

Table 6. Comparison of two algorithms

Algorithms	Avg. Cost Reduction (%)	Max. Cost Reduction (%)	Min. Cost Reduction (%)
B & B Method	39	59	18
Chu's Binary Method	19	34	9

5. 결 론

본 연구는 데이타베이스에서 디스크 액세스 비용을 최소화하는 0-1 정수 계획 모형을 개발하고, 이 모형의 해법으로 분지한계법을 제시하였다. 관계형 데이타베이스에서 segment scan과 clustered index scan방법을 사용할 경우, 본 연구에서 제시한 n개의 수직분할 최적해를 사용하면 기존의 분할 방법을 사용할 때보다 디스크 액세스 횟수를 줄이는데 더 효과적임을 보여주었다. 또한 동일 그룹 속성이 존재할 경우 선행처리를 실시하면 최적해를 효율적으로 구할 수 있음을 제시하

였다. 실험 결과에서도, 속성 사용 행렬의 "1"의 밀도가 낮으면, 여러 개의 단편으로 분할하여 많은 액세스 비용을 감소시킬 수 있음을 알 수 있었다.

참 고 문 헌

[1] Ceri, S., S. Navathe and G. Wiederhold, "Distribution design of logical database schemas," IEEE Trans. Software Eng., vol. SE-9, no.4, 1983.

[2] Cheng Ch, "Algorithms for vertical partitioning in database physical design," Omega, Int. J. Mgnt. Sci., vol.22, no.3, pp. 291-303, 1994.

[3] 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.

[4] Cornell D.W. and P.S. Yu, "An Effective approach to Vertical Partitioning for Physical Design of Relational Database," IEEE Trans. Software Eng., vol. 16, no. 2, pp. 248-258, 1990.

[5] Eisner M. and D.G. Severance, "Mathematical techniques for efficient record segmentation in large shared database," J. ACM, vol.23, no.4, pp.619-635, Oct. 1976.

[6] Elmasri R. and S. Navathe, Fundamentals of database systems. Benjamin/Cummings, second edition, 1994.

[7] Hoffer J. A., "An integer programming formulation of computer database design problems," Inform. Sci. vol. 11, pp.29-48, 1976.

[8] Hoffer J. A. and D. G. Severance, "The use of cluster analysis in physical database design," in Proc. First VLDB, 1975.

[9] Kusiak A. and W. S. Chow, "An efficient cluster identification algorithm," IEEE Trans. on Syst., Man and Cybern., vol. SMC-17, no.4, pp.696-699, 1987.

[10] Livadas P. E., File Structures: Theory and Practice. Englewood Cliffs, NJ: Prentice-Hall, 1990.

[11] Navathe S. and M. Ra, "Vertical Partitioning for database design: A graphical algorithm," in Proc. ACM SIGMOD Int. Conf. Management Data, 1989.

[12] Navathe S., S. Ceri, G. Wiederhold, and J. Dou, "Vertical partitioning algorithms for database design," ACM Trans. Database Syst., vol.9, no.4, pp.680-710, 1984.

96년 1월 최초 접수, 96년 8월 최종 수정