고차원 데이터를 부분차원 클러스터링하는 효과적인 알고리즘

박 종 수† · 김 도 형††

요 약

고차원 데이터에서 클러스터를 찾아내는 문제는 그 중요성으로 인해 데이터 마이닝 분야에서 잘 알려져 있다. 클러스터 분석은 패턴 인식, 데이터 분석, 시장 분석 등의 여러 응용 분야에 광범위하게 사용되어지고 있다. 최근에 이 문제를 풀 수 있는 투영된 클러스터링이라는 새로운 방법론이 제기되었다. 이것은 먼저 각 후보 클러스터의 부분차원들을 선택하고 이를 근거로 한 거리 함수에 따라 가장 가까운 클러스터에 점이 배정된다. 우리는 고차원 데이터를 부분차원 클러스터링하는 새로운 알고리즘을 제안한다. 알고리즘의 주요한 세 부분은, ① 적절한 개수의 점들을 갖는 여러 개의 후보 클러스터로 입력 점들을 분할하고, ② 다음 단계에서 유용하지 않은 클러스터들을 제외하고, 그리고 ③ 선택된 클러스터들은 밀접도 함수를 사용하여 미리 정해진 개수의 클러스터들로 병합한다. 다른 클러스터링 알고리즘과 비교하여 제안된 알고리즘의 좋은 성능을 보여주기 위하여 많은 실험을 수행하였다.

An Effective Algorithm for Subdimensional Clustering of High Dimensional Data

Jong Soo Park† · Do-Hyung Kim††

ABSTRACT

The problem of finding clusters in high dimensional data is well known in the field of data mining for its importance, because cluster analysis has been widely used in numerous applications, including pattern recognition, data analysis, and market analysis. Recently, a new framework, projected clustering, to solve the problem was suggested, which first select subdimensions of each candidate cluster and then each input point is assigned to the nearest cluster according to a distance function based on the chosen subdimensions of the clusters. We propose a new algorithm for subdimensional clustering of high dimensional data, each of the three major steps of which partitions the input points into several candidate clusters with proper numbers of points, filters the clusters that can not be useful in the next steps, and then merges the remaining clusters into the predefined number of clusters using a closeness function, respectively. The result of extensive experiments shows that the proposed algorithm exhibits better performance than the other existent clustering algorithms.

키워드 : 데이터 마이닝(Data Mining), 클러스터링(Clustering), 고차원 데이터(High Dimensional Data), 알고리즘(Algorithm)

1. 서 론

클러스터링(clustering)은 데이터를 클래스나 클러스터로 그룹지우는 과정이다[7, 9]. 클러스터링 문제는 d차원을 갖는 n개의 데이터 객체(object 또는 point)들을 입력으로 하여 유사한 특성을 갖는 k개의 클러스터로 나누는 것을 말한다. 한 클러스터 내의 객체들은 다른 것들과 비교해서 높은 유사성(similarity)을 갖지만, 다른 클러스터에 속한 객체들과는 아주 다르다. 객체들 사이의 유사성의 척도는 객체를 구성하는 속성(attribute)들의 값에 기초한 거리 함수에 의해서 계산된다. 클러스터링은 여러 분야에서 사용되고 있고, 데이터 마이닝, 통계학, 생물학, 그리고 기계 학습 분야에서 많은 연구가 이루어지고 있다. 데이터 마이닝의 기능 측면에서[7], 클러스터링의 분석은 데이터의 분포에 대한 이해를 구하거나 또는 각 클러스터의 특성을 관찰하는 단독적인 도구로 사용되기도 하고, 그리고 다른 알고리즘의 전처리 단계로 사용되기도 한다. 예를 들자면 유사 탐색(similarity search), 고객 세분화(customer segmentation), 패턴 인식, 경향 분석, 분류(classification) 등의 도구로서 사용되기에 데이터베이스 분야에서 많이 연구되고 있다.

일반적으로 클러스터링 알고리즘들은 크게 두 방법으로 나누어진다[9] : 분할 방식(partitioning methods)[13]과 계층 방식(hierarchical methods)[6, 10, 15]이고, 추가적으로 밀도-기반 방식(density-based methods)[4, 5]과 격자-기반 방식(grid-based methods)[8] 등으로 나누기도 한다[7]. 대개의 클러스터링 알고리즘들은 고차원 공간을 갖는 데이터에서는 제대로 작동을 하지 못하고 있다. 데이터 자체의 고유한 희박성(sparsity)이 원인이 되어 클러스터링의 결과가 좋지

※ 본 논문은 성신여자대학교 2000년도 후기 학술연구조성비 지원에 의하여 연구되었음.
† 종신회원 : 성신여자대학교 컴퓨터정보학부 교수
†† 정 회 원 : 성신여자대학교 컴퓨터정보학부 교수
논문접수 : 2002년 11월 5일, 심사완료 : 2003년 1월 15일

않기 때문이다. 고차원 데이터를 갖는 응용 분야에서, 데이터 중에서 어느 두 객체는 몇 개의 차원에서만 값이 밀접하게 존재하고 나머지 차원에서는 아주 멀리 떨어져 있을 가능성이 많다. 그래서 어떤 클러스터링 알고리즘은 특징 추출(feature selection)을 먼저 시행하여 그 차원을 줄이는 방법을 사용하기도 한다[11, 12]. 그 과정은 데이터 상의 객체들이 관련되어 있는 특정 차원만을 선택하고, 나머지 차원들은 데이터에서 잡음으로 간주하여 삭제하는 것이다. 그러나 이런 경우, 특정 차원을 미리 추출하여 사용하게 됨으로써 정보의 손실을 가져올 수 있다. 실제로 많은 응용 예에서 어떤 데이터 집합은 한 차원 집합에 관련되고, 다른 데이터 집합에서는 다른 차원 집합에 관련이 있을 수 있기 때문에, 일률적으로 특정 차원을 삭제하는 경우에는 정보 손실로 인하여 클러스터링이 제대로 되지 못하게 된다. 고차원 데이터의 클러스터링의 다른 방안으로 부분차원(subdimension)을 고려하는 경우를 생각할 수 있다. (그림 1)은 3차원 공간에서 정의된 점들을 X-Y와 X-Z 평면에 투영한 분포를 보여주고 있다. 점들은 크게 두 그룹으로 이루어지고 있다. 왼쪽 그룹의 점들은 Y축으로 넓게 퍼져 있고, X-Z 평면에서는 서로 가까이 있음을 알 수 있다. 비슷한 관점에서, 오른쪽 그룹을 구성하는 점들은 3차원의 부분차원인 X-Y 평면에서 서로 가까이 있다. 이 그림에서 전체 차원인 3차원을 고려하여 클러스터링을 하게 되면 원하는 결과가 나오지 않게 되고, 각 클러스터에 속한 점들의 부분차원을 고려하여 클러스터링을 하게 되면 원하는 결과를 얻을 수 있다. 이러한 고차원의 데이터를 클러스터링하는 알고리즘으로는 CLIQUE[3], PROCLUS[1], ORCLUS[2], DOC[14] 등이 있다.

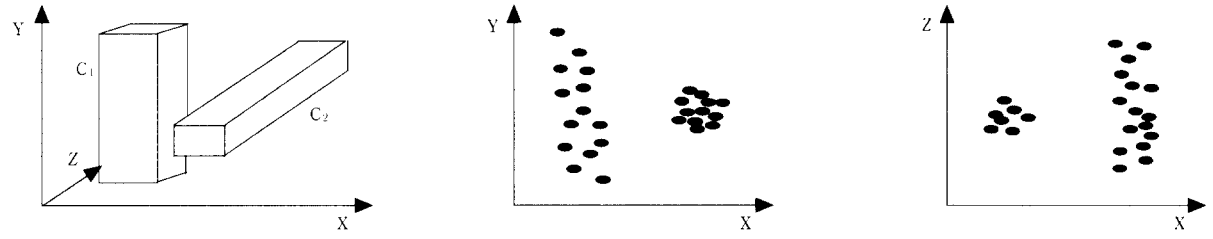
본 논문에서는 고차원 데이터를 저차원인 부분차원으로 줄여서 데이터를 클러스터링하는 방법을 연구하였다. 데이터를 각 클러스터에 배정할 때, 전체 d차원으로 거리를 계산하는 것이 아니라 각 클러스터에 따라 다르게 선택된 부분차원들만 고려된 거리를 계산하여 가까운 클러스터에 데이터를 배정하게 된다. 제안된 알고리즘 SUBCLUS(SUB dimensional CLUStering)는 분할 방식을 사용하는 알고리즘으로 비슷한 부류에 속하는 알고리즘인 PROCLUS[1]와 비교하여 특성들을 검토하였다. 본문에서는 부분차원을 찾아내어 클러스터링하는 알고리즘들의 특징과 문제점을 분석하여 효과적으로 고차원 데이터를 클러스터링하는 방안을 모색하였다. 클러스터링 알고리즘들의 결과 분석을 위해 입력 클러스터와 출력 클러스터 사이의 관계를 나타내는 혼돈 행렬(confusion matrix)을 기반으로 하는 우세 비율(dominant ratio)을 사용하여 클러스터링의 정확도를 측정하였다. 이러한 실험을 통해 제안된 알고리즘 SUBCLUS가 K-Means[7, 8], CLARANS[13], PROCLUS[1] 등의 알고리즘들에 비해 클러스터링의 결과가 우수함을 결과로 보여준다.

본 논문의 구성은 다음과 같다. 2절에서는 부분차원으로 투영하여 고차원 데이터를 클러스터링하는 알고리즘들의 주요한 특성을 살펴본다. 이를 기반으로 성능을 개선시킨 새로운 알고리즘인 SUBCLUS를 3절에서 제안하고, 이 알고리즘을 구성하는 함수들을 설명한다. 생성된 실험 데이터에 대한 성능 평가는 4절에서 보여준다. 마지막으로 5절에서는 알고리즘에 대한 결론을 맺고, 추후 연구 분야에 대해 설명한다.

2. 부분차원 선택 알고리즘의 특성

알고리즘 PROCLUS[1]는 고차원 데이터를 효과적으로 클러스터링하는 새로운 접근 방식을 제안하였다. 이전에 발표되었던 CLIQUE[3] 알고리즘에 비해서 보다 작은 개수의 클러스터들을 찾아내고 정확도 면에서 우수함을 보여주고 있다. 그러나, 알고리즘 PROCLUS도 무작위로 선택되는 초기 medoid(discrete median or representative)들의 위치에 따라서 클러스터링 결과가 많이 달라질 수 있다. 그래서 내부 반복 과정에서 클러스터링이 제대로 되지 않았다고 판단되는 소수의 점들을 포함하는 클러스터들의 대표값인 medoid를 무작위로 교체하여 앞에서 설명한 취약점을 보완하려고 하며, Greedy 함수에서도 medoid의 후보들을 가능하면 이웃하지 않은 점들로 유지하려고 하였다. 그럼에도 불구하고, 알고리즘 PROCLUS는 초기 medoid 집합에 너무 종속적으로 영향을 받는 클러스터링 결과를 만들어내고 있다.

이후 알고리즘 PROCLUS를 개선한 ORCLUS[2]와 DOC[14]가 발표되었다. ORCLUS는 클러스터들의 중심축이 고차원 데이터의 차원 축과 평행하지 않은 경우에 공분산 행렬(covariance matrix)을 이용한 고유 벡터(eigen vector)를 계산해내는 방식으로 문제를 풀어나갔다. 그리고, ORCLUS는 초기 단계에서 최종 클러스터들의 개수보다 아주 많은 개수들의 클러스터들로부터 시작하여 분할된 클러스터들의 부분차원을 찾아서 클러스터들을 합병하는 과정을 거쳐서 클러스터링을 수행한다. 선택된 부분차원들의 개수도 초기에는 전체 차원의 개수로 시작하여 점차적으로 원하는 개



(a) 3차원 공간에서 패턴 (b) X-Y 평면으로의 투영 (c) X-Z 평면으로의 투영

(그림 1) 특징 추출 방법에서의 난점

수의 부분차원으로 줄여간다. ORCLUS는 각 클러스터의 부분차원의 개수를 같은 수로 선택하기 때문에 클러스터들이 서로 다른 개수의 부분차원으로 이루어진 경우에는 처리하기가 쉽지 않은 면이 있다. DOC 알고리즘은 무작위로 medoid에 해당하는 점을 찾아서 일정한 영역안의 점들을 하나의 클러스터에 속하게 하여 클러스터들의 품질을 계산하는 공식을 적용하여 일정한 값 이상의 클러스터링이 될 때까지 계속하여 무작위로 medoid를 선택한다. Monte Carlo algorithm의 특성으로 확률적으로 좋은 결과가 나올 수 있지만, 많은 회수의 내부 반복 과정을 거쳐야 되므로 실행 시간이 길어진다는 단점이 있다.

3. 클러스터링 알고리즘 SUBCLUS

제안된 알고리즘의 이름은 SUBCLUS라고 지었고, 고차원 데이터를 부분차원(SUBdimension)으로 투영하여 클러스터링(CLUStering)하는 방법이다. 입력으로는 n개의 d차원을 가진 점(point)들이고, 사용자가 지정하는 변수는 원하는 클러스터들의 개수 k와 한 클러스터 당 평균적으로 투영된 차원들의 개수 l_{avg}이다. 이 알고리즘은 클러스터링의 방법론인 분할 방식과 계층 방식을 차례로 적용한 것으로, 먼저 점들을 여러 개의 클러스터들로 나눈 후에, 이들을 원하는 개수의 클러스터들로 병합하도록 하였다. (그림 2)에 구체적인 알고리즘이 설명되어 있다. 전체적으로는 세 단계로 구성되며, 분할 단계, 병합 단계, 그리고 정제 단계로 이루어진다. 초기 단계에서 n개의 d차원 점들을 거의 같은 크기의 소규모 클러스터들로 나눈다. 알고리즘 SUBCLUS에서는 각 클러스터에 속한 점들의 개수는 평균적으로 n/k_0개를 갖도록 한다. 여기서, 초기 분할 개수 $k_0 = A \cdot k$로 두고 A의 범위는 입력 데이터베이스 크기에 따라 결정한다. 초기 단계인 분할 단계를 거치면, 각 클러스터는 자신에게 소속된 점들과 이 점들의 중심점(centroid)을 가지게 된다.

다음 단계인 병합 단계에서는 작은 클러스터들을 결합시켜 전체적으로 클러스터의 개수가 줄어들게 된다. 사용자가 지정한 k개의 클러스터들만 남게 되면 병합 과정은 종료된다. 이 단계에서는 각 클러스터에서 점들이 서로 밀집되어 있는 투영된 부분차원들을 찾아내게 된다. 부분차원이 찾아지면 모든 점들이 각 클러스터의 중심점 사이의 거리를 부분차원을 근거로 계산하여 가장 가까운 클러스터에 배정된다. 일정 개수 이하의 점들을 갖거나 밀집도가 작은 클러스터를 제외한 후에, 병합 함수에서 클러스터들의 개수가 정해진 숫자가 될 때까지 밀접한 두 클러스터들을 한 클러스터로 계속 병합한다. 클러스터들 사이의 밀접도 계산에서는 부분차원을 고려한 중심점 사이의 거리와 공통된 부분차원의 개수에 근거한 함수를 사용한다.

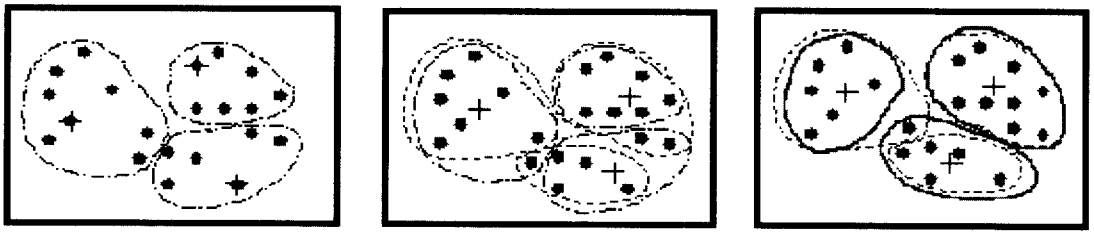
세 번째 단계에서는 k개의 클러스터들의 부분차원과 중심점을 재계산하고 모든 점들을 재배정하여 최종적으로 클러스터링의 결과가 더 좋아지도록 한다.

```
Algorithm SUBCLUS( k, l_avg )
begin
    k_0 = k_c = A · k ;                    // initial number of clusters, k_0
    (s_1, ..., s_{k_c}, C_1, ..., C_{k_c}) = Partition ( k_c ) ;
    while ( k_c > k) do begin
        (D_1, ..., D_{k_c}) = FindDimensions(k_c, l_avg, C_1, ..., C_{k_c}) ;
        (s_1, ..., s_k, C_1, ..., C_k) = AssignPoints (s_1, ..., s_k, D_1, ..., D_{k_c}) ;
        (C_1, ..., C_{k_f}) = FilterClusters (k, C_1, ..., C_{k_c}) ;
        k_n = max(k_f/2, k) ;              // new number of clusters
        (s_1, ..., s_{k_n}, C_1, ..., C_{k_n}) = Merge (C_1, ..., C_{k_f}, k_f, k_n) ;
        k_c = k_n ;
    end
    (D_1, ..., D_k) = FindDimensions (k, l_avg, C_1, ..., C_k) ;
    (s_1, ..., s_k, C_1, ..., C_k) = AssignPoints (s_1, ..., s_k, D_1, ..., D_k) ;
    return( C_1, ..., C_k ) ;
end
```

(그림 2) 알고리즘 SUBCLUS

3.1 첫 번째 단계 : 분할 단계(Partition Stage)

제 1단계인 분할 단계에서는 n개의 입력 점들을 k_0개의 클러스터들로 나누는데, 각 클러스터에 속한 점들의 개수가 가능하면 균일하도록 한다. CHAMELEON[10]에서는 그래프 이론을 적용하여 근접성에 근거하여 점들을 분할한다. 여기서는 클러스터링의 분할 방식 중의 하나인 K-Means 알고리즘을 이용하고 있다. 클러스터 C_i의 중심점(centroid)을 계산하는 방법은 $s_i = \sum_{x=1}^{t} \overline{p_x}/t$이다. 여기서 p_x는 클러스터 C_i에 속한 점이고($p_x \in C_i$), t는 C_i에 속한 점들의 개수($t = |C_i|$)이다. (그림 3)은 K-Means 알고리즘이 점들의 집합을 클러



(a) (b) (c)

(그림 3) K-Means 알고리즘에 의한 점들의 클러스터링[7]

스터링해가는 과정을 보여주고 있고[7], (그림 4)는 이에 대한 알고리즘을 기술하고 있다.

```
Partition(k)        // by K-Means algorithm[7]
begin
    Arbitrarily choose k points, s_1, ..., s_k, as the intial cluster centers ;
    local-iteration = 0 ;
    repeat
        (Re)Assign each point to the cluster to which the point is
            in the most similar, based on the mean value of the
            points in the cluster ;
        Update the cluster mean, s_i, i.e., calculate the mean
            value of the points for each cluster C_i ;
        local-iteration++ ;
    until (local-iteration >= THRESHOLD or no change) ;
    return ( s_1, ..., s_k, C_1, ..., C_k ) ;
end
```

(그림 4) 초기 분할을 수행하는 Partition 함수

K-Means 알고리즘의 시간 복잡도가 크다면 무작위로 k_0개의 점들을 추출하여 초기 단계의 클러스터의 중심점으로 사용할 수 있다[1, 2, 13]. 이 경우에는 전체 차원을 고려한 클러스터들의 분할이 이루어지도록 AssignPoints 함수를 적용해야 된다. 앞의 K-Means 방법에 비해 소규모의 클러스터들이 균일하게 퍼져있지 않을 수도 있어서 다음 단계의 클러스터링에 나쁜 영향을 미치게 된다. 이 분할 단계에서의 결과는 k_0개의 클러스터들이 균일하게 퍼져 있도록 하고, 그리고 각 클러스터에 속한 점들의 개수도 같아지도록 하는 것이 효과적이다. 분할 함수로 사용하는 K-Means 알고리즘의 시간 복잡도는 $O(mknd)$인 것은 (그림 4)에서 repeat-loop이 m번 수행하는 것을 의미하고, 이것은 점들의 재할당과 중심점을 구하는 단계를 m번 수행하는 것이다. 이 함수로 인하여 전체 클러스터링 알고리즘의 수행 시간이 너무 늦어진다면 문제가 될 수 있다. 이를 해결하는 방안으로 일정한 회수 동안만 재할당과 중심점 찾기를 수행하도록 하여 입력 점들이 분할되는 효과를 보게 하여 제안된 알고리즘의 전체 실행 시간에 미치는 영향이 적어지도록 한다. (그림 4)에서 THRESHOLD의 값을 지정하여 일정한 회수의 loop을 수행하도록 하였고, 이 값은 대략 한자리 숫자에서 정하도록 한다.

3.2 두 번째 단계 : 병합 단계(Merge Stage)

앞 단계인 분할 단계에서 입력 점들이 아주 많은 개수의 클러스터들로 분할되었다. 이 단계에서는 원하는 k개의 클러스터들이 남을 때까지 네 개의 함수를 사용하여 반복 과정을 되풀이한다. 다음은 이 단계에서 필요한 네 개의 함수들을 상세히 설명한다.

부분차원 찾기 함수 : 이 함수에서는 BIRCH[15]와 ORCLUS [2]에서 사용하는 clustering feature인, 각 클러스터에 속한 점들의 선형 합(linear sum)과 제곱 합(squared sum)인 LS와 SS를 사용한다. LS와 SS를 사용하여 각 클러스터의 점들이 각 차원에서 분포된 정도를 나타내는 표준 편차를 계산하여, 이 값을 기준으로 각 차원의 부분차원을 결정한다. (그림 5)는 이 함수를 설명하고 있다. 클러스터 C_i의 선형 합은 d개의 원소로 구성되어 있고, 각 원소는 $LS_{i,j} = \sum_{x=1}^{|C_i|} p_{x,j}$로 계산된다. 여기서도 $p_x \in C_i$이다. 그리고, 제곱 합도 점의 위치의 제곱을 계산한다는 것 이외는 비슷한 방식으로 계산된다. C_i에서 d개로 구성된 제곱 합의 j번째 원소는 $SS_{i,j} = \sum_{x=1}^{|C_i|} p_{x,j}^2$로 계산된다. 클러스터 C_i의 전체 d차원들 중에서 j번째 차원에 분포된 점들의 평균값(mean)은 $\mu_{i,j} = LS_{i,j} / |C_i|$로 계산되고, 그 표준 편차(standard deviation)는 $\sigma_{i,j} = \sqrt{(SS_{i,j} - |C_i| \cdot \mu_{i,j}^2)/(|C_i| - 1)}$로 계산된다.

```
FindDimensions (k, l, C_1, ..., C_k)
begin
    for each cluster i do begin // linear sum, squared sum,
                                   and standard deviation of C_i
        LS_{i,j} = Σ_{x=1}^{|C_i|} p_{x,j} and SS_{i,j} = Σ_{x=1}^{C_i|} p_{x,j}^2 for p_x ∈ C_i and
            1 ≤ j ≤ d ;
        μ_{i,j} = LS_{i,j}/|C_i| and σ_{i,j} = sqrt((SS_{i,j} - |C_i| · μ_{i,j}^2)/(|C_i| - 1))
            for 1 ≤ j ≤ d ;
        D_i = ∅ ;
    end
    Pick the k · l numbers with the least values of σ_{i,j} subject
        to the constraint that there are at least 2 dimensions
        for each cluster ;
    if σ_{i,j} is picked then add dimension j to D_i ;
    return ( D_1, ..., D_k ) ;
end
```

(그림 5) 한 클러스터에서 부분차원을 찾는 FindDimensions 함수

모든 클러스터들의 전체 d차원에서 점의 분포를 계산한 표준 편차를 오름차순으로 정렬한 후에, 작은 값을 갖는 차원을 선택하여 해당 클러스터에 부분차원으로 지정하게 된다. 전체적으로는 $k \cdot l$개의 차원을 선택하고, 선행 조건으로 먼저 각 클러스터에서 제일 작은 표준 편차를 갖는 두 개의 차원을 선택하도록 한다. 그 후에 $k \cdot (l-2)$개의 표준 편차 값을 크기 순으로 선택하여 그것을 포함하는 클러스터의 부분차원으로 정한다. (그림 1)에서 2개의 차원이 선택된다면, C_1에 속하는 3차원 점들의 부분차원은 {X, Z}가 되고, C_2에 의해 선택되는 부분차원은 {X, Y}가 된다.

점 배정 함수 : (그림 6)의 AssignPoints 함수는 각 클러스터의 중심점(s_i)과 부분차원 D_i를 고려하여, 각 점을 가장 가까운 중심점을 갖는 클러스터로 배정하게 된다. 한 점 p_x와 한 클러스터의 중심점 사이의 거리를 계산하는 공식은 $SegPdis\ (p_x, s_i, D_i)$로 표기하고, Euclidean segmental dis-

tance인 $SegPdis(p_x, s_i, D_i) = \sqrt{\sum_{j \in D_i}(p_{x,j} - s_{i,j})^2} / |D_i|$로 계산한다. PROCLUS는 medoid를 중심으로 가장 가까운 클러스터에 점을 배정하고, SUBCLUS는 중심점(s_i)을 기준으로 가장 가까운 클러스터에 점을 배정하는 차이점이 있다. 모든 점들을 가장 가까운 클러스터에 배정한 후에, Merge 함수에서 필요한 각 클러스터의 선형 합, 제곱 합, 그리고 중심점을 다시 계산한다.

```
AssignPoints(s_1, ..., s_k, D_1, ..., D_k)
begin
    for each i ∈ {1, ..., k} do C_i = ∅ ;
    for each data point p do begin
        Determine SegPdis(p, s_i, D_i) for each i ∈ {1, ..., k} ;
        Determine the seed s_i with the least value of
          SegPdis(p, s_i, D_i) and add p to C_i ;
        end
    for each i ∈ {1, ..., k} do begin  // linear sum, square
                                          sum, and centroid
        LS_{i,j} = Σ_{x=1}^{|C_i|} p_{x,j} and SS_{i,j} = Σ_{x=1}^{|C_i|} p²_{x,j}  for p_x ∈ C_i and
                   1 ≤ j ≤ d ;
        s_{i,j} = LS_{i,j} / |C_i| for 1 ≤ j ≤ d ;
        end
    return(s_1, ..., s_k, C_1, ..., C_k) ;
end
```

(그림 6) 모든 점들을 클러스터에 배정하는 AssignPoints 함수

클러스터 여과 함수 : 점 배정 함수에서 어떤 클러스터에 배정된 점들의 개수가 너무 작거나 또는 소속된 점들이 너무 분산되어 있으면, FilterClusters 함수는 이런 클러스터를 나쁜 클러스터로 판단하여 이후의 처리에서 삭제한다. 점들의 개수와 분산 정도를 판별하는 변수는 (그림 7)에서 f_{ratio}와 σ_{ratio}이고 사용자가 그 값들을 지정한다. 삭제 부분 비율 f_{ratio}는 0과 1 사이의 값으로 클러스터가 최소로 가져야할 점들의 개수를 정하는데 사용된다. f_{ratio} = 0.1이라는 것은 어떤 클러스터에 속한 점들의 개수가 평균 개수의 10% 이하일 때는 그 클러스터를 제외한다는 것을 의미한다. σ_{ratio}는 클러스터에 소속된 점들의 밀집한 정도를 나타내기 위한 변수 값으로, 이 값보다 작으면 그 클러스터는 삭제된다. 각 클러스터에서 선택된 부분차원들의 표준 편차와 선택되지 않은 차원들의 표준 편차의 비율은 σr_i로 계산된다. 클러스터 C_i에서 부분차원으로 선택되지 않은 차원들은 표준 편차의 값에 따라 오름차순으로 정렬하여 값이 작은 $|D_i|$개의 표준 편차들이 비율 계산에 사용된다. (그림 7)에서 σr_i가 크다는 것은 선택된 부분차원들의 표준 편차가 상대적으로 아주 작아서 그 클러스터에 속한 점들이 분산되어 있지 않고 밀집되어 있다는 것을 의미한다. 사용자가 지정하는 f_{ratio}와 σ_{ratio}의 값에 대한 민감도 분석을 다음 장에서 실험을 통해 설명한다.

이 함수에서 조건에 맞는 클러스터를 선택하다 보면 최종적으로 원하는 개수(k)의 클러스터들보다 더 작은 클러스터들이 선택되는 경우가 발생한다. 이러한 경우에는 각 클러스터에 속한 점들의 개수 $|C_i|$와 점들의 밀집도에 해당하는 σr_i를 곱한 값을 기준으로 큰 값을 갖는 클러스터를 추가적으로 선택하여 원하는 개수의 클러스터들이 남도록 한다. 마지막으로, 제외된 클러스터에 속한 점들은 선택된 클러스터에 재배정된다.

```
FilterClusters (k, C_1, ..., C_{k_c})
begin
    double f_ratio, σ_ratio ;      // f_ratio는 0.01과 0.1 사이의 값,
                                      σ_ratio는 3.0과 5.0 사이의 값
    k_f = 0 ;
    small = f_ratio * n / k_c ;
    for each cluster i do begin
        σr_i = Σ_{j∈D_i}^{|D_i|} σ_{i,j} / Σ_{j∈D_i} σ_{i,j} ;  // the ratio of standard
                                                                   deviations
        if (|C_i| >= small and σr_i >= σ_ratio ) then begin
            Choose C_i ;
            k_f++ ;
            end
        end
    if (k_f < k) then
        Select the (k − k_f) largest cost clusters from the
        (k_c − k_f) remained clusters, where the cost of a
        cluster C_i is |C_i| × σr_i ;
    Reassign points of the discarded clusters into the
        chosen clusters ;
    return (C_1, ..., C_{k_f}) ;
end
```

(그림 7) 일정 개수 이하의 점들을 갖거나 밀집도가 작은 클러스터를 제외하는 FilterClusters 함수

병합 함수 : (그림 8)의 Merge 함수는 입력으로 주어진 여러 개의 클러스터들을 클러스터간의 밀접한 정도에 근거하여 일정 개수의 클러스터들이 남을 때까지 클러스터들을 병합해 간다. 두 클러스터간의 간격을 계산하여 밀접한 정도를 나타내는 $m \times m$ 행렬인 DistMatrix를 만든다. 행렬의 대각선을 중심으로 위 삼각형에 해당하는 값들만 계산된다. 나머지 값들은 대각선을 중심으로 행과 열에 대칭한 동일한 값으로 이루어지므로 계산하지 않는다. 두 클러스터 C_i와 C_j의 간격(distance)을 계산하는 함수는 Closeness이다. 이 함수에서는 먼저 두 클러스터의 부분차원에 공통된 차원이 있는지를 살핀다. 만약 공통된 차원이 없으면 두 클러스터의 중심점 사이에서 Euclidean segmental distance를 계산하고 weighting factor 값으로 전체 차원 개수인 d를 곱한다. 그렇게 되면, 두 중심점의 간격은 단순한 Euclidean distance가 된다. 공통 차원이 있는 경우에는, 두 부분차원의 합집합을 기준으로 Euclidean segmental distance를 구하고 weight-

ing factor로 합집합과 교집합의 비율을 곱하게 된다. 만약, 두 클러스터의 부분차원이 같으면 이 비율이 1이 되어 간격은 Euclidean segmental distance가 되지만, 합집합과 교집합의 비율이 1보다 크면 간격이 앞의 경우보다 더 커지게 된다. 다시 설명하면, 두 클러스터의 부분차원들 중에서 공통된 차원이 많으면 많을수록 간격(distance)은 작아지고, 그렇지 않으면 간격은 커지게 된다. 두 클러스터간의 간격이 작을수록 더 밀접하다는 것을 의미한다.

```
Merge( C_1, ..., C_{k_c}, k_c, k_n )
begin
    m = k_c ;
    Make an m × m matrix, DistMatrix[ i, j ]
          = Closeness( s_i, s_j, D_i, D_j ) for i < j ;
    while ( k_c > k_n ) do begin
       Find indices (i, j) such that DistMatrix[ i, j ] is
         minimum from the matrix ;
       C_i = C_i ∪ C_j ;        // Merge the corresponding
            clusters C_i and C_j into C_i
       LS_i = LS_i + LS_j and then, s_{i,x} = LS_{i,x} / | C_i |
             for 1 ≤ x ≤ d ;
       SS_i = SS_i + SS_j and then,
         σ_{i,x} = sqrt( (SS_{i,x} - | C_i | · s^2_{i,x}) / (| C_i | - 1) ) for 1 ≤ x ≤ d ;
       Pick max(2, (| D_i | + | D_j |)/2) numbers with the least
         values of σ_{i,x} for 1 ≤ x ≤ d ;
       D_i = ∅ ; if σ_{i,x} is picked then add dimension x to D_i ;
       Discard C_j and set the elements of DistMatrix[ ]
            with index j to infinite, ∞ ;
       Update the elements of DistMatrix[ ] with index i
          using the function Closeness( ) ;
       k_c = k_c - 1 ;
       end
    return( s_1, ..., s_{k_new}, C_1, ..., C_{k_new} ) ;
end

Closeness ( s_i, s_j, D_i, D_j )        // distance between C_i and C_j
begin
    if | D_i ∩ D_j | = ∅ then do distance = SegPdist( s_i, s_j, D ) × d ;
                                  //  d = | D |
    else do distance = SegPdist( s_i, s_j, D_i ∪ D_j )
                       × ( | D_i ∪ D_j | / | D_i ∩ D_j | ) ;
    return(distance) ;
end
```

(그림 8) 클러스터를 병합하는 Merge 함수와 두 클러스터간의 밀접도를 계산하는 함수

다음으로 클러스터의 개수가 일정 개수가 될 때까지 클러스터들을 병합한다. 두 클러스터 사이가 가장 밀접한 것은 두 클러스터 사이의 간격이 제일 작은 것이므로, 행렬 DistMatrix의 원소 중에서 제일 작은 값을 가지는 원소의 인덱스 i와 j를 구한다. 그러면, 두 클러스터 C_i와 C_j는 병합되어 보다 큰 새로운 클러스터 C_i가 된다. 병합된 클러스터 C_i에 속하는 값들인 선형 합, 제곱 합, 중심점, 그리고 부분차원을 계산한다. 부분차원을 계산하려면 먼저 각 차원에 분포된 점들의 값의 표준 편차를 계산한 후에, 이 표준 편차들 중에서 작은 순서로 $\max(2, (|D_i|+|D_j|)/2)$개의 차원을 선택하여 새로운 클러스터의 C_i 부분차원으로 한다. 여기서, 값 2는 클러스터의 부분차원들의 개수가 최소한 2가 되도록 한 FindDimensions 함수의 내용을 반영한 것이다. 두 클러스터가 병합되면, 행렬 DistMatrix에서 i와 j에 관련된 원소의 값들은 변경되어야 한다. 클러스터 C_j는 병합되어 없어졌으므로 인덱스 j에 연관된 행렬의 원소 값은 무한대로 두어서 다음 병합에서 제외되도록 한다. 새로운 클러스터인 C_i의 인덱스 i와 연관된 클러스터 중에서 현재 클러스터로 남아있는 것에 해당되는 행렬의 원소 값은 Closeness 함수를 이용하여 다시 계산된다.

3.3 세 번째 단계 : 정제 단계(Refine Stage)

이 단계는 두 개의 함수 호출로 이루어진다. 먼저 두 번째 단계에서 k개의 클러스터들을 병합한 결과를 넘겨받으면, 부분차원 찾기 함수에서 각 클러스터에 속한 점들의 분포를 나타내는 각 차원에서의 표준 편차를 계산하여 각 클러스터의 부분차원을 구한다. 그리고 두 번째 함수에서는 각 클러스터의 중심점과 부분차원에 따라 점들을 가까운 클러스터에 배정하게 된다. 두 번째 함수의 호출에 의한 결과는 n개의 점들을 우리가 원하는 k개의 클러스터로 분할한 것이다. 결과적으로, 한 점은 한 클러스터에 속하게 되고, 한 클러스터 내의 점들 사이의 거리는 다른 클러스터에 있는 점과의 거리보다 근접해 있게 된다.

3.4 알고리즘의 시간 복잡도

알고리즘 SUBCLUS는 5개의 주요 함수로 구성되어 있다. 이 함수 각각이 전체 알고리즘에서 수행하는 시간 복잡도를 먼저 계산하고, 그 이후에 전체 알고리즘의 시간 복잡도를 분석한다. 함수 Partition의 시간 복잡도는 (그림 4)에서 설명한 것과 같이 $O(mk_0nd)$로 나타낼 수 있고, 여기서 n은 입력 점들의 개수이고 k_0는 초기 단계에서 분할하고자 하는 전체 클러스터의 개수이다. 그리고, d는 한 점의 전체 차원의 수를 나타내고, m은 분할 함수가 실행하는 반복 횟수를 나타낸다. m은 보통 한자리 숫자로 주어지므로, 분할 함수의 시간 복잡도는 $O(k_0nd)$로 표현할 수 있다.

부분차원 찾기 함수는 각 클러스터에 소속된 점들로 선형 합과 곱셉 합을 계산하여 표준 편차를 구해서 일정한 개수의 차원을 선택한다. 이 함수에서 다루어지는 것은 모든 입력 점들의 각 차원에 해당되는 값이므로, 함수 자체에 대한 시간 복잡도는 $O(nd + k_c d\log(k_c d))$가 된다. 두 번째 항목은 표준 편차를 구하기 위하여 $k_c \cdot d$개의 표준 편차를 정렬하는 비용으로 첫 항목에 비해서 너무 작으므로 $O(nd)$로 둔다. 이 함수가 수행되는 전체 회수를 고려한 시간 복잡도는 $O\left(ndk_0 \sum_{i=0}^{\lceil \log_2 A \rceil} 2^{-i}\right)$가 되고, 이를 다시 정리하면 $O(k_0nd)$가

된다. 점 배정 함수 자체의 시간 복잡도는 $O(k_c n l_{avg} + nd)$로 나타낼 수 있다. 첫 항목은 각 점에 대해서 모든 클러스터의 부분차원으로 거리를 구하는 것이고, 두 번째 항목은 할당된 모든 점에 대해서 선형 합과 곱셈 합을 구하는 것에 해당된다. 앞의 분석과 같은 방법을 적용하면, 이 함수가 적용되는 전체 회수에 대한 복잡도는 $O(k_0 nd)$가 된다. 작은 클러스터를 제거하는 함수의 시간 복잡도는 $O(k_0^2)$로 나타낼 수 있고, 이는 다른 함수에 비해서 복잡도가 작으므로 전체 시간 복잡도에는 별 영향을 미치지 않는다. 그리고, 병합 함수에서의 전체 시간 복잡도는 $O(k_0 l_{avg}^2 + k_0^3 + k_0 d)$로 나타낸다. $k_0 l_{avg}^2$은 거리 행렬식을 만드는 비용이고, k_0^3은 거리 행렬에서 최소값을 찾는 것이고, $k_0 d$는 표준 편차를 구하고 거리 행렬을 수정하는 비용이다. 이 식에서 차수가 높은 것으로만 다시 정리하면 $O(k_0 l_{avg}^2 + k_0^3)$으로 나타낼 수 있다.

결과적으로 알고리즘 SUBCLUS의 시간 복잡도는 $O(k_0 nd + k_0 l_{avg}^2 + k_0^3)$이 된다. 이 식에서 시간 복잡도는 처음에 입력을 분할하는 클러스터 개수 k_0에 가장 큰 영향을 받는 것으로 나타났다. k_0가 크면 많은 실행시간이 걸리고 더 좋은 결과가 나올 수 있다. 3절에서 언급했듯이, $k_0 = A \cdot k$로 계산되는데, A의 값에 대한 상세한 분석은 다음 절에서 이루어진다.

4. 실험 결과 및 분석

실험 데이터를 생성하여 제안된 알고리즘의 성능을 측정하였다. 실험은 1.8GHz Pentium4 CPU, 1GB 주메모리, 그리고 32GB SCSI Hard Disk를 가진 PC에서 이루어졌다. 운영체제는 Windows XP Professional이고, 사용된 언어는 MS Visual C++이다. 실험은 제안된 알고리즘인 SUBCLUS의 타당성과 자체 성능 평가에 주안점을 두었다. 다음과 같은 정확도와 성능을 측정하고자 하였다. ① 입력 클러스터와 출력 클러스터에서 제대로 짝을 맞추는 정도의 정확도를 측정하였다. ② 클러스터 여과 함수에서 사용자가 지정하는 f_{ratio}와 σ_{ratio}의 값에 따라 클러스터링의 결과가 변하는 것을 분석하였다. ③ 초기 분할 단계에서 클러스터의 개수에 따른 수행 성능을 분석하였다. ④ 앞의 분석을 기초로 하여 사용자가 정하는 변수 값을 결정한 후에, 알려진 다른 클러스터링 알고리즘들과 비교하였다. 그리고, 입력 데이터베이스 크기에 따른 실행 시간을 측정하여 알고리즘 SUBCLUS의 실행 시간이 입력 데이터 개수에 선형적으로 비례한다는 것을 보여주었다.

4.1 실험 데이터 생성 및 혼돈 행렬

실험 데이터를 생성하기 위해서 [1]에서 제시된 방법으로 데이터를 만들었다. <표 1>은 입력 데이터의 하나의 예로 입력 클러스터와 각 클러스터의 부분차원과 소속된 점들의 개수를 보여주고 있다. 이 데이터에서 클러스터들의 평균적인 부분차원들의 개수 l_{avg}는 4이다. 입력되는 점들의 개수는 10,000개(n = 10,000)이고 각 점은 25차원(d = 25)으로 이루어지고 있다. 각 클러스터는 정해진 중심점에서 부분차원들의 표준 편차에 따라 소속된 점들이 분포되어 있다. d차원 중에서 선택되지 않은 부분차원들에 해당되는 점의 값은 균일 분포(uniform distribution)로 퍼져있다. 두 클러스터 A와 B는 같은 부분차원을 공유하지만, 클러스터의 중심점인 anchor point는 완전히 다르고 상당히 떨어져 있다. 10,000개의 점들 중에서 5%인 500개는 어느 특정 클러스터에 소속시키지 않고 모든 영역에 골고루 분포시켜 놓아 outlier 또는 잡음의 특성을 가지도록 하였다.

〈표 1〉 입력 데이터의 클러스터와 부분차원

Input	Dimensions	Points
A	3 14 17	1867
B	3 4 6 9 14 16 17 21	2032
C	5 17	1592
D	16 17 24	1373
E	7 14 17 18 24	1428
F	16 17 20	1208
Outlier	-	500

클러스터링의 정확도를 평가하기 위하여 혼돈 행렬(confusion matrix)을 사용하였다[1, 2]. <표 1>의 입력 데이터에 대한 PROCLUS[1] 알고리즘의 결과는 <표 2>의 혼돈 행렬로 표시하였다. 혼돈 행렬은 출력 클러스터가 어떻게 입력 점들과 짝이 맞추어지는가를 보여주고 있다. 혼돈 행렬의 (i, j)항목은 입력 클러스터 j에 속한 점들 중에서 출력 클러스터 i에 속한 점들의 개수를 표시한다. <표 2>의 (4, D) 항목은 입력 클러스터 D에 속한 점 1,373개 중에서 1,298개가 출력 클러스터 4로 짝이 맞추어진 것을 보여주고 있다. 만약 클러스터링의 알고리즘이 좋은 성능을 가진다면, 각 행과 열은 다른 것에 비해서 월등히 높은 숫자를 가진 한 항목을 가질 가능성이 높아진다[2]. <표 2>에서 입력 클러스터 A, B, E열은 특정 출력 클러스터에 점들이 집중되어 있어 클러스터링이 잘 되었다는 것을 보여준다. 반면에 입력 클러스터 F에 속하는 열의 숫자들은 여러 행으로 분산되어 있다. 이것은 입력 클러스터 F에 속한 점들이 클러스터링이 잘 되지 못하여 여러 출력 클러스터들에 분산되어 있음을 보여준다. 예상할 수 있듯이, outlier에 속한 점들도 역시 여러 출력 클러스터들에 걸쳐 분산되어 있다.

혼돈 행렬의 값에서 클러스터링 알고리즘의 실제 우수성은 우세 비율(dominant ratio)이라는 측정치로 평가한다[2]. 이것은 가장 우세한 입력 클러스터에 의해 차지하게 된 각 클러스터의 평균 부분 값이다. 이것을 구하는 과정은 다음과 같다. 각 출력 클러스터에서 가장 높은 숫자를 갖는 입력 열의 숫자를 계산한다. 각 행에서 가장 높은 값을 갖는 열을 정해서 행과 열의 짝을 맞춘다. 그리고 짝을 이루는

값을 더해서 입력 클러스터에 속한 전체 점의 개수로 나누면 우세 비율을 얻을 수 있다. <표 2>에서 짝을 이루는 (출력 클러스터 행, 입력 클러스터 열)은 (5, B), (3, A), (2, C), (6, E), (4, D), (1, F)가 되고, 각 항의 값을 더하면 8,619가 된다. Outlier를 제외한 입력 클러스터에 속한 전체 점의 개수는 9,500이므로, <표 2>의 혼돈 행렬에 대한 우세 비율은 0.91이 된다. 우세 비율이 1에 가까울수록 클러스터링은 깨끗하게 잘 된 것이다.

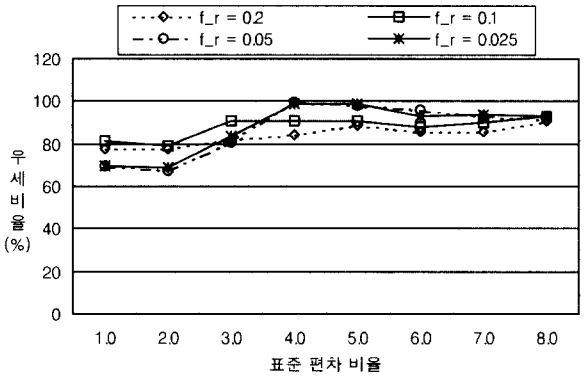
<표 2> <표 1>의 입력에 대한 클러스터링 결과의 혼돈 행렬(Confusion Matrix)

입력 클러스터 / 출력 클러스터	A	B	C	D	E	F	Outlier
1	0	1	9	28	0	412	100
2	0	0	1583	0	0	0	135
3	1867	0	0	40	0	432	106
4	0	0	0	1298	0	225	87
5	0	2031	0	2	0	12	32
6	0	0	0	5	1428	127	40

4.2 클러스터 여과 함수에서 변수의 민감도 분석

제안된 알고리즘인 SUBCLUS에서는 세 개의 변수의 값을 설정해야 하는 문제가 있다. 이 절과 다음 절에서 이러한 변수의 값을 설정하는 실험 결과에 대해 설명한다. 먼저 클러스터 여과 함수 FilterClusters의 지역 변수인 삭제 부분 비율 f_{ratio}와 표준 편차 비율 σ_{ratio}를 조정하였을 때 클러스터링의 결과에 어떠한 영향을 미치는지를 조사하여 값을 정하도록 하고, 다음 절에서 초기 분할하는 과정에서의 클러스터들의 개수인 k_0를 결정하는 변수 A에 대해서 실험을 수행한다. 이 절에서는 SUBCLUS의 변수를 $k=6$, $l_{avg}=4$, $A=16$으로 두고, 초기 단계인 함수 Partition에서 THRESHOLD의 값은 3으로 두어 실험 결과를 얻었다.

(그림 9)는 클러스터 여과 함수에서 삭제 부분 비율 f_{ratio}와 표준 편차 비율 σ_{ratio}의 값을 변화하였을 때 클러스터링 결과의 우세 비율의 변화를 보여주고 있다. 사용된 데이터베이스는 <표 1>과는 다른 10,000개의 점으로 이루어진 데이터베이스로서 참고문헌[1]의 방법에 따라 만든 것이다. 이 데이터베이스는 6개의 클러스터를 포함하고 부분차원들의 전체 개수는 25이다. 이 그림은 클러스터 여과 함수의 두 지역 변수의 값을 미리 결정하기 위한 실험 결과이다. 표준 편차 비율은 1.0에서 8.0까지 변화시키고 삭제 부분 비율은 0.2, 0.1, 0.05, 0.025로 변화시켰을 때 알고리즘 SUBCLUS의 클러스터링 결과에 대한 우세 비율을 얻었다. 이 그림에서 알 수 있는 것은 SUBCLUS의 클러스터링 결과가 두 변수에 상당히 민감한 것을 보여주고 있다. 여러 실험 결과에 따르면, 좋은 클러스터링 결과를 얻기 위해서는 부분 삭제 비율을 0.025정도나 그 이하로 두고 표준 편차 비율을 4.0이나 5.0 정도로 설정하면 우세 비율이 100%에 가까워진다.



(그림 9) 삭제 부분 비율 f_{ratio}에서 표준 편차 비율 σ_{ratio}의 여러 값에 따른 우세 비율

4.3 초기 분할 단계에서 A의 변화에 따른 클러스터링 성능 분석

알고리즘 SUBCLUS의 초기 단계에서 클러스터의 개수는 $k_0=A\cdot k$에 의해서 결정되어 전체 알고리즘의 성능에 영향을 미치므로 A의 크기에 따른 클러스터링의 결과 분석이 필요하다. <표 3>에서는 100,000개의 점으로 이루어진 데이터베이스에 대한 클러스터링의 결과를 요약하고 있다. 앞 절에서 사용한 데이터베이스를 사용하였으며, 각 클러스터의 평균점이나 각 차원에서의 표준 편차는 같고 생성된 데이터의 개수만 다르다. 실험 결과는 이런 데이터베이스를 입력으로 하여 SUBCLUS를 20번 실행시킨 결과의 평균값이고, 사용자가 지정하는 변수의 값은 $f_{ratio}=0.025$와 $\sigma_{ratio}=4.0$으로 두었다. 이 표에서는 A가 커짐에 따라 일정하게 우세 비율이 높아지다가, 너무 많은 클러스터로 분할되면 클러스터링의 결과가 나빠짐을 알 수 있다. 실행 시간은 A가 커짐에 따라 점차적으로 증가하는 것을 보여주고 있다. 데이터베이스를 분할하였을 때 각 클러스터에 속할 수 있는 평균적인 점의 개수가 너무 작게 되면 다음 단계인 병합 단계에서 좋은 결과를 얻지 못함을 알 수 있다. 실험 결과에 의하면 데이터베이스의 점의 수가 10,000개에서 100,000개 정도이면 A를 8이나 16으로 두고, 그 이상의 데이터 크기에서는 A를 증가시켜서 좋은 결과를 얻도록 한다.

<표 3> $k_0=A\cdot k$에서 A의 변화에 따른 클러스터링 실행 시간과 우세 비율

	A	2	4	8	16	32	64
|DB| = 100K	우세 비율(%)	94.17	95.11	98.30	99.31	92.39	68.06
	실행 시간(sec)	5.42	8.83	16.29	29.41	55.87	108.84

4.4 클러스터링 알고리즘들의 성능 비교

고차원 데이터를 클러스터링하는 알고리즘으로 알려진 것은 참고문헌에 소개된 정도이다. 제안된 알고리즘과 성능을 비교하기 위하여 기존의 알고리즘을 모두 비교하기란 쉽지 않다. 본 논문에서도 제안된 알고리즘의 성능을 평가하기 위해서 몇 개의 제한된 알고리즘을 실제 구현하여 그 성능을

측정하였다. PROCLUS[1] 알고리즘을 구현하였고, 고유 벡터(eigen vector)를 이용한 알고리즘인 ORCLUS[2]는 한 클러스터에서 일정한 개수의 부분차원들을 요구하고 있기에 비교 평가를 하지 않았다. 그리고, 대용량의 데이터를 클러스터링 하는 알고리즘인 CLARANS[13]와 널리 알려진 K-Means 알고리즘[7]을 구현하여 SUBCLUS와 결과를 비교하였다.

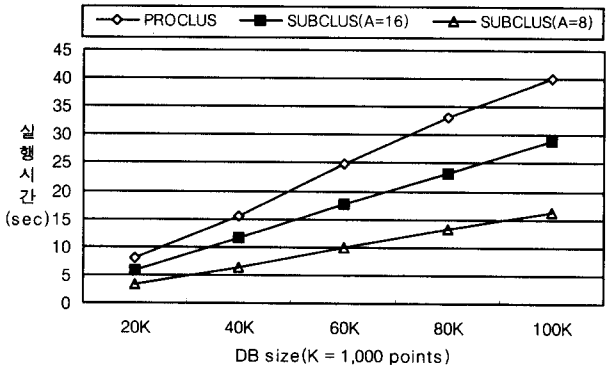
<표 4>는 <표 1>의 데이터를 입력으로 하여 20번을 실행하여 그 평균 값을 낸 실험 결과를 보여주고 있다. SUBCLUS에서의 변수는 앞의 절에서 언급한 것과 같이 $A = 16$, $f_{ratio} = 0.025$, $\sigma_{ratio} = 4.0$으로둔 실험 결과이다. CLARANS는 그 알고리즘의 특징이 저차원인 2차원이나 3차원에 적합할 것으로 판단되고, 고차원 데이터의 입력에서는 제대로 클러스터링을 수행하지 못함을 보여주고 있다. K-Means 알고리즘은 그 단순성에 비해 좋은 결과를 나타내고 실행시간도 짧음을 알 수 있다. CLARANS와 K-Means 알고리즘은 전체 차원을 대상으로 하여 클러스터링을 수행한다. PROCLUS는 고차원 데이터를 클러스터링하는 과정에서 각 클러스터에 관련된 부분차원을 선택하여 클러스터링하는 알고리즘으로 CLARANS와 K-Means에 비해 좋은 클러스터링 결과를 얻고 있다. 제안된 SUBCLUS는 PROCLUS에 비해서 우세 비율이 높고 실행 시간이 더 작음을 이 표에서 알 수 있다. 두 알고리즘의 실험 결과 비교는 다음 절에서 상세히 설명한다.

〈표 4〉 클러스터링 알고리즘들의 실험 결과

알고리즘	우세 비율(%)	실행 시간
CLARANS	45.81	281.75 sec
K-Means	79.05	1.39 sec
PROCLUS	86.47	4.05 sec
SUBCLUS	97.53	3.18 sec

4.5 입력 데이터베이스 크기에 따른 실행 시간

(그림 10)은 입력 데이터베이스의 크기를 20,000개(20K)에서 100,000개(100K) 까지 변화시키면서 두 클러스터링 알고리즘인 PROCLUS와 SUBCLUS의 실행 시간을 측정한 결과이다. 사용되어진 데이터베이스는 4.2절에서 사용되어진 데이터베이스로 각 클러스터의 평균점이나 각 차원에서의 표준 편차는 같고 생성된 데이터의 개수만 다르다. 알고리즘 SUBCLUS의 초기 분할되는 클러스터의 개수에 영향을 주는 A의 값을 16과 8로 두었을 때 성능 평가의 결과를 표시하고 있다. (그림 10)은 두 알고리즘의 시간 복잡도는 모두 입력 데이터베이스 크기에 선형적으로 비례한다는 것을 보여주고 있다. <표 5>는 (그림 10)의 실행 시간을 측정할 때 클러스터링 결과의 우세 비율을 보여주고 있다. <표 3>에서의 결과에서 예측할 수 있듯이, A를 8로 두면 실행 시간은 거의 절반으로 줄어드는 대신 우세 비율이 조금 떨어지지만 PROCLUS의 클러스터링 결과보다는 우수하다는 것을 보여주고 있다.



(그림 10) 데이터베이스 크기에 따른 두 클러스터링 알고리즘의 실행 시간 비교

〈표 5〉 데이터베이스 크기에 따른 클러스터링 결과의 우세 비율(%)

|DB|	20K	40K	60K	80K	100K
PROCLUS	86.01	87.89	88.95	85.06	87.31
SUBCLUS (A = 16)	98.81	99.99	99.31	98.34	99.37
SUBCLUS (A = 8)	94.82	97.10	96.21	98.71	98.30

5. 결 론

이 논문에서는 고차원 데이터를 클러스터링하는 알고리즘을 제안하였다. 제안된 알고리즘 SUBCLUS는 먼저 사용자가 원하는 개수의 클러스터들보다 몇 배 많은 클러스터들로 분할하고, 분할된 클러스터들의 점의 분포에 따라 표준 편차를 구하여 각 클러스터의 점들을 밀집하게 분포시키는 부분차원들을 선택하였다. 각 클러스터의 부분차원에 근거한 거리 함수를 계산하여 입력 점을 가장 가까운 클러스터에 배정하고, 그런 후에 작은 개수의 점들이 배정되었거나 선택된 부분차원들의 표준 편차가 상대적으로 큰 클러스터는 제외하였다. 마지막 단계에서 클러스터들 사이의 밀접도(closeness)를 계산하여 병합 과정을 거친다. 이런 일련의 과정이 원하는 개수의 클러스터들이 남을 때까지 반복된다. 클러스터링 알고리즘의 성능은 입력 클러스터와 출력 클러스터의 사이의 점의 분포를 표시하는 혼돈 행렬(confusion matrix)을 사용하여 우세 비율(dominant ratio)로 계산하였다. 실험 데이터에서는 SUBCLUS가 기존의 알고리즘인 K-Means, CLARANS, 그리고 PROCLUS에 비해 실행 시간이나 우세 비율에서 성능이 좋음을 보여주었다.

다음과 같은 사항이 이루어지면 고차원 데이터를 효과적으로 클러스터링하는 보다 좋은 알고리즘이 되리라 여겨진다. 알고리즘 SUBCLUS의 초기 단계인 분할 과정에서 각 클러스터들이 일정한 개수의 점들을 가지게 하는 분할 함수, 각 클러스터에서 데이터의 분포가 밀집된 부분차원을 선택하는 과정, 그리고, 두 클러스터 사이의 밀접한 정도를

계산하는 함수를 개선하는 연구가 필요하다. 또한 알고리즘들의 성능 평가에서 실제 데이터를 기반으로 한 실험 결과가 요구된다.

참 고 문 헌

[1] C. C. Aggarwal, C. Procopiuc, J. L. Wolf, P. S. Yu and J. S. Park, "Fast Algorithms for Projected Clustering," In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp.61-72, 1999.

[2] C. C. Aggarwal and P. S. Yu, "Finding generalized projected clusters in high dimensional spaces," In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp.70-81, 2000(also, IEEE TKDE, Vol.14, No.2, pp. 210-225, 2002).

[3] R. Agrawal, J. Gehrke, D. Gunopulos, P. Raghavan, "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications," In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp.94-105, 1998.

[4] M. Ankerst, M. M. Breunig, H.-P. Kriegel and J. Sander, "OPTICS : Ordering Points to Identify the Clustering Structure," In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp.49-60, 1999.

[5] M. Ester, H.-P. Kriegel, J. Sander and X. Xu, "A density-based algorithm for discovering clusters in large databases," In *Proceedings of 1966 International Conference on Knowledge Discovery and Data Mining (KDD '96)*, pp. 226-231, 1996.

[6] S. Guha, R. Rastogi and K. Shim, "CURE : An Efficient Clustering Algorithm for Large Databases," In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp.73-84, 1998.

[7] J. Han and M. Kamber, *Data Mining : Concepts and Techniques, Morgan Kaufmann Publishers,* San Francisco, CA, 2001.

[8] A. Hinneburg and D. Keim, "Optimal Grid-Clustering : Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering," In *Proceedings of the 25th VLDB Conference*, pp.506-517, 1999.

[9] A. K. Jain, M. N. Murty and P. J. Flynn, "Data Clustering : A Review," *ACM Computing Surveys*, Vol.31, No.3, pp. 264-323, 1999.

[10] G. Karypis, E.-H. Han and V. Kumar, "CHAMELEON : A Hierarchical Clustering Algorithm Using Dynamic Modeling," *COMPUTER*, 32, pp.68-75, 1999.

[11] R. Kohavi and D. Sommerfield, "Feature Subset Selection Using the Wrapper Method : Overfitting and Dynamic Search Space Topology," In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, 1995.

[12] H. Liu and H. Motoda, *Feature Extraction, Construction and Selection : A Data Mining Perspective*, Kluwer Academic Publishers, Boston, 1998.

[13] R. Ng and J. Han, "Efficient and Effective Clustering Methods for Spatial Data Mining," In *Proceedings of the 20th VLDB Conference*, pp.144-155, 1994(also, IEEE TKDE Vol. 14, No.5, pp.1003-1016, 2002).

[14] C. M. Procopiuc, M. Jones, P. K. Agarwal and T. M. Murali, "A Monte Carlo Algorithm for Fast Projective Clustering," *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp.418-427, 2002.

[15] T. Zhang, R. Ramakrishnan and M. Linvy, "BIRCH : An Efficient Data Clustering Method for Large Databases," *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp.103-114, 1996.

박 종 수

e-mail : jpark@sungshin.ac.kr

1981년 부산대학교 전기기계공학과(공학사)

1983년 한국과학기술원 전기및전자공학과 (공학석사)

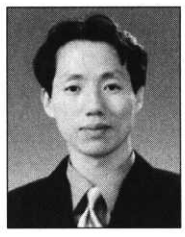
1990년 한국과학기술원 전기및전자공학과 (공학박사)

1983년~1986년 국방부 군무설계기좌

1990년~현재 성신여자대학교 컴퓨터정보학부 교수

1994년~1995년 미국 IBM T. J. Watson Research Center 객원연구원

관심분야 : 데이터베이스, 데이터 마이닝, 알고리즘



김 도 형

e-mail : dkim@sungshin.ac.kr

1985년 서울대학교 컴퓨터공학과(공학사)

1987년 한국과학기술원 전산학과(공학석사)

1992년 한국과학기술원 전산학과(공학박사)

1992년 한국과학기술원 정보전자연구소 연수연구원

1992년~현재 성신여자대학교 컴퓨터정보 학부 부교수

1997년~1998년 미국 State University of New York at Stony Brook 컴퓨터과학과 객원교수

관심분야 : 프로그래밍언어, 컴파일러, 알고리즘