데이터웨어하우스에서 유전자 알고리즘을 이용한 구체화된 뷰 선택 기법

이 민 수†

요 약

데이터 웨어하우스는 복잡한 질의 및 분석을 위해서 다양한 종류의 여러 정보 출처들로부터 정보를 모아서 저장한다. 일반적으로 웨어하우스에는 자주 실행되는 질의들을 미리 계산해서 구체화된 뷰의 형태로 저장한다. 웨어하우스를 설계할 때 가장 중요한 일들 중의 하나는 웨어하우스에서 유지될 구체화된 뷰의 선택이다. 이것은 뷰들의 유지를 위해 제한된 시간이 주어졌을때, 모든 질의들에 대한 총 질의 응답 시간을 최소화 하는 방법으로 일련의 뷰들을 선택하는 것이다(*유지-비용 뷰 선택 문제*). 본 논문에서는 최적에 가까운 일련의 뷰들을 계산하기 위해 유전자 알고리즘을 사용하여 유지-비용 뷰 선택 문제에 대한 효율적인 해결책을 제안한다. 특히 OR 뷰 그래프들의 관점에서의 유지-비용 뷰 선택 문제를 다룬다. 본 논문의 접근방식은 휴리스틱 방법을 사용한 기존의 탐색-기반 접근 방식들에 비해서, 시간 복잡도에서 큰 향상을 보여준다. 본 논문의 알고리즘은 최적의 질의 비용에 비해 10%이내의 추가비용만을 갖는 해결책을 제시하면서도 실행시간 측면에서는 매우 향상된 선형 증가만을 보인다. 본 논문의 알고리즘에 대한 프로토타입을 구현하였으며 이것을 사용하여 논문에서 제안하는 접근방식의 분석을 수행하였다.

A Genetic Algorithm for Materialized View Selection in Data Warehouses

Minsoo Lee†

ABSTRACT

A data warehouse stores information that is collected from multiple, heterogeneous information sources for the purpose of complex querying and analysis. Information in the warehouse is typically stored in the form of materialized views, which represent pre-computed portions of frequently asked queries. One of the most important tasks of designing a warehouse is the selection of materialized views to be maintained in the warehouse. The goal is to select a set of views so that the total query response time over all queries can be minimized while a limited amount of time for maintaining the views is given(*maintenance-cost view selection problem*). In this paper, we propose an efficient solution to the maintenance-cost view selection problem using a genetic algorithm for computing a near-optimal set of views. Specifically, we explore the maintenance-cost view selection problem in the context of OR view graphs. We show that our approach represents a dramatic improvement in terms of time complexity over existing search-based approaches that use heuristics. Our analysis shows that the algorithm consistently yields a solution that only has an additional 10% of query cost of over the optimal query cost while at the same time exhibits an impressive performance of only a linear increase in execution time. We have implemented a prototype version of our algorithm that is used to evaluate our approach.

키워드 : 데이터웨어하우스(Data Warehouse), 유전자 알고리즘(Genetic Algorithm), 뷰 유지(View Maintenance), 뷰 구체화(View Materialization), 뷰 선택(View Selection), 웨어하우스 구성(Warehouse Configuration)

1. 서 론

데이터 웨어하우스는 복잡한 질의 및 분석을 위해서 여러 가지 서로 다른 종류의 정보 근원들로부터 정보를 모아서 저장한다[1, 2]. 서로 다른 데이터 근원들로부터 가져온 관련 데이터들에 대해서 불일치와 모순점들을 검출하고 해결하는 과정을 거쳐서 통합한 후에 웨어하우스에 적재한다. 데이터 웨어하우스에 저장된 정보의 양이 매우 많은데다 질의들이 복잡해져 한번에 수많은 요약정보를 필요로 하기 때문에, 데이터 웨어하우스를 어떻게 구성하는가의 문제는 웨어하우스를 주기적으로 유지하는 것뿐만 아니라 효율적인 OLAP을 지원함에 있어 결정적인 요소가 된다. 웨어하우스의 데이터는 종종 요약 테이블들 또는 자주 실행되는 질의들을 미리 계산해 놓은 구체화된 뷰들로 구성된다[3].

※ 이 연구는 2002학년도 이화여자대학교 교내연구비 지원에 의한 연구임.
† 정 회 원 : 이화여자대학교 컴퓨터학과 교수
논문접수 : 2003년 7월 31일, 심사완료 : 2004년 3월 2일

이러한 방법으로 웨어하우스 질의처리기는 각 질의가 대량의 일련의 데이터들을 읽어들여야 하는 연산을 피하도록 하는데 특히 이러한 연산이 자주 발생하는 경우의 심한 시간 낭비를 줄일 수 있다. 그러나, 이러한 구체화된 뷰들을 사용할 경우에는 구체화된 뷰들과 근원 데이터 사이의 일관성을 유지해 주어야 한다. 이를 위하여 모든 뷰들을 주기적으로 새로 갱신하는 방법이 있지만 이 방법은 많은 시간이 소요되며 낭비가 많다. 그리하여 이보다는 근원 데이터의 변화에 영향을 받는 뷰의 일부분들만을 갱신하는 점진적 방식으로 뷰를 유지해줄 수 있다[4, 5].

구체화된 뷰는 이러한 뷰 유지 또는 갱신 비용 외에 추가적인 저장공간도 필요로 하며 이 공간은 어떤 뷰를 구체화할 것인지와 구체화할 뷰의 수를 결정할 때 고려되어야 하는 사항이다. 예를 들어 자주 실행되는 OLAP 질의의 집합이 주어졌을 때 이 모든 질의들을 구체화된 뷰로 만드는 것은 질의 응답 시간을 확실히 줄여주지만 웨어하우스에 대한 갱신 비용을 증가시키고 사용가능한 저장 용량을 초과할 수도 있다. 따라서, 질의 실행 시간을 줄이기 위해서 공간을 많이 사용하거나 반대로 사용 공간을 줄이기 위해서 질의 실행 시간을 손해보거나 함으로써 웨어하우스 관리자는 세가지 중요한 요소들인 질의 응답시간, 유지비용, 저장공간 간의 균형을 고려하여 웨어하우스의 구성을 결정하여야 한다. 이들 세가지 비용을 고려하여 웨어하우스 구성을 위한 일련의 구체화된 뷰들을 선택하는 문제를 *뷰 선택 문제* 라 한다.

본 논문에서는 유지 비용에 대한 제한된 시간이 주어졌을 때 질의 응답 시간을 최소화하려는 유지-비용 뷰 선택 문제에 대하여 새로운 알고리즘을 제안한다. 이때 저장공간은 매우 저렴하여 더 이상 결정적인 자원으로 간주되지 않는다는 가정 하에 저장공간에 대한 제한은 두지 않는다. 그리고, 각각의 뷰들이 하위 뷰들중 하나만을 가지고도 도출 가능한 OR뷰 그래프의 관점에서 유지-비용 뷰 선택 문제를 연구한다. 이 문제는 이전에도 Gupta[6], Theodoratos와 Sellis[7]에 의해서 제기되었으나, 기존 알고리즘들은 20~25개 이상의 뷰를 포함하는 웨어하우스 구성을 계산할 때 성능이 좋지 못하였다. 이러한 경우에 대해서는 탐색공간이 지나치게 커지므로 전체 탐색공간을 대상으로 하는 어떠한 탐색기법도 적용시키기 어려워진다. 가장 우수한 휴리스틱 기법들마저도 이 문제의 특별한 몇몇 경우에 적용이 가능할 뿐이다. 이에 따라 본 논문에서는 다른 복잡한 문제들에 성공적으로 적용되었던 무작위 기법[8, 9]을 사용하는 해결책을 고안하였다. 본 논문에서 제시하는 방법은 기존 방법들에 비해 결과로 얻어지는 웨어하우스 구성의 질적인 면뿐만 아니라 결과를 얻기까지의 알고리즘의 실행 시간면에서도 우수하다. 본 연구에서의 분석결과 본 논문의 유전자 알고리즘은 실행시간에서는 선형 증가만을 보이는 반면, 최적의 질의 이익의 90% 이내에 드는 결과를 찾는 것을 알 수 있었다. 특히 웨어하우스 뷰들이 지원해 주어야 할 질의들이 자주 바뀌는 경우에 본 논문의 알고리즘은 웨어하우스를 효율적이고 빠르게 재구성하는데 사용될 수 있으므로 데이터 웨어하우스 설계에 있어 매우 유용할 것으로 본다. 이와 같이 데이터 웨어하우스의 진화 또는 변경을 지원해줌으로써 데이터 웨어하우징의 유용성을 보다 더 증가시킬 수 있다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구를 다루며 3장에서는 본 논문의 주요한 기술적인 내용들을 다룬다. 특히 무작위 알고리즘 중에서 유전자 알고리즘을 설명하고 유지-비용 뷰 선택 문제에 어떻게 적용시켰는지를 소개한다. 4장에서는 프로토타입 시스템의 구현을 설명하고 5장에서는 이 프로토타입을 사용하여 실험한 결과를 보인다. 6장에서는 본 논문의 결과를 요약하고 향후 연구과제로 결론을 맺는다.

2. 관련 연구

뷰 선택에 대한 대부분의 관련 연구에서는 최적의 해를 찾기 위하여 전체 탐색공간을 찾는 것을 피하고 욕심쟁이(Greedy) 전략 또는 휴리스틱(Heuristic) 기법을 사용한다. 구체화할 추가적 구조들을 선택하는 문제는 Roussopoulos[10]에 의해 처음 연구되었는데 이때는 실제 뷰들 자체보다는 뷰 인덱스들을 구체화하는 것을 제안하였다. 뷰 인덱스는 뷰와 유사하지만 뷰에 있는 투플들을 직접 저장하는 대신, 뷰 투플을 생성하는 근원 테이블의 투플에 대한 포인터들로 구성된다. 알고리즘은 최적의 뷰 인덱스들의 집합을 찾기 위해서 A^* 알고리즘을 기반으로 하고 있으나 뷰의 갱신 비용 모델이 매우 단순해서 하위 뷰들이 이미 구체화된 경우를 고려하지 않는다. 결과적으로 선택된 뷰 집합의 유지 비용은 그다지 현실적이지 못하다.

Ross[11]는 전체 탐색(Exhaustive search)에 기반한 탐색 알고리즘을 사용하였으며 탐색공간의 불필요한 부분을 없애는 휴리스틱 기법과 몇몇 최적화 기법들을 제안하였다. 그는 하위 뷰들에 대한 국부적 선택을 통한 최적화로는 이 문제를 해결할 수 없고, 대신 전체적인 최적화를 고려하는 방법을 이용하여야 함을 보였다. Labio[12]의 연구 역시 A^*를 기반으로 하고 있으며, 인덱스를 고려하고 알고리즘의 최적성을 향상시킴으로써 이전 연구를 더욱 확장하였다. Labio[12]는 제안한 알고리즘이 해결책의 복잡성으로 인해 적절한 시간 내에 최적의 웨어하우스 구성을 계산할 수 없는 경우에 이용할 수 있도록 일련의 뷰들과 인덱스들을 선택하는 지침과 유익한 일련의 규칙들을 처음으로 제공하였다.

또한 실험을 통해 뷰 대신에 인덱스를 저장하므로써 제한된 공간을 보다 효율적으로 사용하는 방법을 보였다. 연구 결과 주요 뷰의 키 애트리뷰트에 대한 인덱스를 만드는 방법이 유지 비용을 줄이면서 아주 적은 저장공간만을 차지함을 보였다.

Gupta[6]는 선택된 뷰들을 구체화하는데 필요한 총 공간을 제약함으로써 뷰 선택 문제를 논의하였다. AND 뷰 그래프, OR 뷰 그래프, AND-OR 뷰 그래프, 이 세가지 구성 형태를 구별하고, 서로 다른 종류의 욕심쟁이 알고리즘들을 적용하기 위한 기반을 마련하였다. 각 뷰 구성에 대해 먼저 갱신을 전혀 고려하지 않는 경우, 갱신을 포함하는 경우, 마지막으로 갱신과 인덱스 모두를 고려하는 경우의 순서로 복잡성을 증가시키면서 이들 각각의 경우들에 적용할 수 있는 알고리즘들이 고안되었다. 제안된 욕심쟁이 알고리즘들이 최적의 해법의 63% 이내의 성능을 보장해 준다는 것이 분석적으로 증명되었다. 알고리즘들은 시간 복잡도면에서 뷰들의 수에 대하여 다항적인 증가를 보인다. 욕심쟁이 알고리즘은 비교적 좋은 결과를 보장해주는 것이 증명되었지만, 알고리즘 상에서 초기의 선택이 최종적인 결과에 크나큰 영향을 주기 때문에 단지 국부적 최적해 만을 찾아내는 잠재적 문제가 있다. 또한, 욕심쟁이 알고리즘은 한 번에 한 개의 뷰만을 고려한다. 그러므로, 두 개의 뷰들이 서로 돕는 효과가 있는 경우에 한 번에 한 개의 뷰만을 고려하는 것은 최적의 해로부터 동떨어진 결과를 찾을 가능성이 높다.

본 연구는 유지-비용 뷰 선택 문제를 해결하기 위한 Gupta[13]의 연구와 가장 밀접하게 연관되어 있는데, 이들의 연구에서는 OR나 AND 뷰 그래프 및 일반적인 경우의 AND-OR 뷰 그래프의 관점에서 A^* 알고리즘뿐만 아니라, 욕심쟁이 알고리즘 두 가지 모두를 이용했다. 이들의 접근 방식은 제한된 저장 공간에서 질의 응답 시간과 뷰 유지 비용 사이의 균형을 맞추려고 한다. 유지-비용 뷰 선택 문제는 갱신이 없이 공간 제약만을 고려한 뷰 선택 문제보다 복잡도가 훨씬 크므로 근사 알고리즘들이 고안되었다. OR 뷰 그래프의 경우, 반전-트리(Inverted Tree) 욕심쟁이 알고리즘이 제안되었다. AND-OR 뷰 그래프와 같이 보다 복잡한 문제에 대해서는 A^* 휴리스틱을 채택하고 반전 트리 욕심쟁이 알고리즘은 더 이상 적용하지 않는다. 휴리스틱 기법이 최적의 해결책을 찾아내는데 반해, 욕심쟁이 알고리즘은 최적의 해결책의 63% 범위의 해결책만을 제공한다. 하지만, 해결책을 찾아내는 시간의 관점에서 보면, 반전-트리 욕심쟁이 알고리즘이 A^* 휴리스틱 기법에 비해 일반적으로 빠르다는 것을 OR 뷰 그래프들에 대한 실험 결과를 통해 알 수 있다.

데이터베이스 분야에서 무작위 알고리즘들은 이제껏 주로 질의 최적화의 관점에서만 연구되어져 왔다. 구체적인 예로는 Swami[14]의 연구에서처럼 다중-조인 최적화 문제와 같은 매우 복잡한 문제들에 가장 활발히 응용되었다. 데이터 마이닝 분야에서는 데이터 집합들 내에서 일련의 초기 규칙들을 밝히기 위해서 유전자 알고리즘을 이용하기도 한다[15, 16]. 대규모 또는 심지어 무제한의 탐색 공간을 다루는 복잡한 문제의 출현과 함께 이들 알고리즘들은 보다 광범위하게 사용될 것으로 예상된다.

3. 유전자 알고리즘을 이용한 접근 방법

뷰 선택 문제는 NP-hard[17, 18]이다. 이러한 부류의 문제들은 문제의 범위가 커지면 해결책의 탐색 공간도 기하급수적으로 증가하기 때문에 최적의 해결책을 찾는 것은 어렵다. 일반적인 NP-hard 문제와 특정 경우의 뷰 선택 문제에 대한 몇 가지 가능한 해결책들이 존재하기는 하지만, 이들은 문제의 크기가 특정 한계를 넘어서게 되면 성능과 관련해서 심각한 문제가 생긴다. 그리하여 최근의 접근 방식들은 무작위 알고리즘을 사용하여 NP-hard 문제들을 해결하고자 한다.

무작위 알고리즘은 통계학적 개념들을 기반으로 하며 큰 탐색공간에서 무작위로 탐색하면서도 평가함수라는 것을 이용해서 원하는 목표에 보다 가까이 접근하고자 한다. 무작위 알고리즘은 해결책을 찾기 위한 실행시간과 결과로 나온 해결책의 질을 맞바꿈으로써 상대적으로 짧은 시간 안에 비교적 적절한 해결책을 찾을 수 있다. 결과로 얻은 해결책은 최적에 매우 가까운 해결책으로서 약간의 질적 저하는 있지만 실행 시간의 단축에 비교하면 매우 미미한 차이다. 일반적으로 이렇게 해서 얻어진 해결책은 최적의 몇 퍼센트 이내 일 뿐이어서 무작위 알고리즘들은 2장에서 소개한 이전의 접근 방식들에 비하면 상당히 매력적인 대안이 된다.

무작위 알고리즘들 중에서, 가장 잘 알려진 알고리즘들은 경사 오르기 기법(*hill-climbing methods*)[19], 단조 모방 기법(*simulated annealing*)[20], 및 유전자 알고리즘(*genetic algorithms*)[8]이다. 경사 오르기 기법은 탐색 공간의 단일 지점에서 시작하여 보다 좋은 지점을 찾기 위해 지속적으로 그들의 주위를 탐색하는 반복적 향상 기법(iterative improvement)에 기반하고 있다. 주위에 보다 좋은 지점이 없으면 탐색이 종료된다. 이 접근방식은 시작 지점에 의존적이어서 국부적 최적해 만을 제공하는 단점을 가지고 있다.

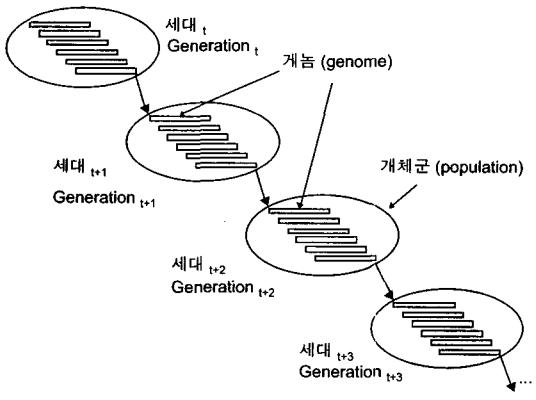
유전자 알고리즘은 탐색 공간의 후보 지점들을 여러 개 유지하는 방법으로 다중 방향성 탐색을 사용한다. 후보 지점들 사이의 정보 교환을 통해서 좋지 못한 후보들은 없어지고 좋은 후보들이 남는 방향으로 탐색이 수행된다. 이러

한 다중 방향성의 진화적인 접근방식은 유전자 알고리즘이 효율적으로 공간을 탐색하고 전역적 최적해에 가까운 지점을 찾도록 해준다.

하지만, 유전자 알고리즘은 자동적으로 좋은 해결책을 제시하지는 못하며, 그 성패는 종종 문제를 정확하게 표현하는 것에 달려있다. 이에는 알고리즘의 초기화 및 많은 테스트 수행 동안 이루어져야 하는 알고리즘의 세심한 조정이 해당된다. 다음 절에서는 유전자 알고리즘에 대하여 간략하게 소개한 후에 유지-비용 뷰 선택 문제에 어떻게 유전자 알고리즘을 적용시켰는지에 대해 자세히 설명한다. 특히, 해결책을 적절하게 표현하는 방법과 유전자 알고리즘의 탐색 방향을 인도하는 평가 함수에 대하여 설명한다.

3.1 유전자 알고리즘

유전자 알고리즘의 기본 아이디어는 살아있는 유기체가 한 세대에서 다음 세대로 넘어가면서 더 뛰어난 개체군으로 진화하는 방식을 모방하는 것으로부터 시작되었다. 유전자 알고리즘은 다음과 같이 진행된다. 먼저 게놈(genome)들의 집단이 생성된다. 각각의 게놈은 문제에 대한 하나의 가능한 해결책을 나타낸다. 이 게놈들의 집단을 개체군(population)이라고 부른다. 개체군은 변화를 겪으면서 새로운 개체군을 형성한다. 이러한 변화의 과정에서 각 단계의 개체군을 하나의 세대(generation)라고 부른다. 각 세대를 generation_t, generation_{t+1}, generation_{t+2}, … 등과 같이 이름을 붙여 지칭할 수 있다. 여러 세대가 지난 generation_{t+k}에 해당하는 개체군은 generation_t에 해당하는 개체군보다 더 뛰어난 게놈들로 구성되어 있을 것이다. (그림 1)은 세대별로 개체군이 나열된 모습을 보여준다.



(그림 1) 개체군의 진화

$t=0$으로 초기화하고, 최초의 개체군인 P(0)는 무작위로 게놈의 집단을 생성하거나 특정한 게놈 하나를 복제하여 집단을 만든다. 이 최초의 개체군은 generation_0이라고도 한다. 그런 다음, 유전자 알고리즘은 다음 4단계를 반복적으로 수행한다.

① $t = t + 1$

② P(t)를 P(t-1)으로부터 선택한다(Selection).

③ P(t)를 재구성한다(Recombination).

④ P(t)를 평가한다(Evaluation).

단계 ①에서, t값을 하나 증가시킴으로써 변수 t에 의해 표시되는 새로운 개체군이 생성된다. 단계 ②에서는 이전의 개체군인 P(t-1) 중에서 가장 뛰어난 게놈들이 선택되고 새로운 개체군인 P(t)를 구성하기 위해서 사용된다. 가장 뛰어난 게놈을 선택하는데에는 룰렛 바퀴(roulette wheel)[9] 기법과 같은 통계적인 방법을 사용한다. 단계 ③에서는 개체군에 새로운 게놈들을 만들기 위해 기존의 게놈들을 가지고 쌍을 이루거나 또는 각각에 대해 개별적으로 여러 연산을 수행해서 개체군을 재구성한다. 이러한 연산을 각각 교차(crossover) 연산 및 돌연변이(mutation) 연산이라고 부른다. 교차 연산은 한 쌍의 게놈들이 각각의 게놈의 뛰어난 형질들이 통합되기를 기대하면서 그들 자신 사이에 정보를 교환하는 것을 말한다. 돌연변이 연산은 하나의 게놈에 무작위로 변화를 가하여 게놈에 새로운 뛰어난 형질이 출현하기를 기대하는 연산이다. 단계 ④는 만들어진 개체군을 평가한다. 이 과정에서 각각의 게놈의 우월성을 평가하는 적합성 함수(fitness function)가 사용된다. 각각의 게놈의 적합성을 합산하면 새로운 세대가 얼마나 개선되었나를 평가하기 위한 기준으로 사용될 수 있다. 이 적합성 값은 또한 다음 반복과정의 선택단계(단계 ②)에서 다음 개체군을 형성할 우수한 게놈들을 선택하는 데에도 사용된다. 매번 평가시에 그때까지 가장 좋은 적합성을 가진 게놈을 저장해둔다. 이제 이 알고리즘이 어떻게 유지 비용 뷰 선택 문제를 해결하는데 적용될 수 있는지를 설명한다.

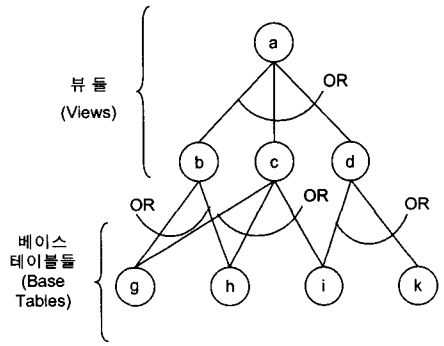
3.2 유지 비용 뷰 선택 문제에의 적용

유지 비용 뷰 선택 문제에 유전자 알고리즘(GA)을 적용하기 위해서는 다음의 세 가지 세부문제를 해결해야 한다. ① 후보 해결책을 게놈으로 표현할 수 있는 방법을 찾아야만 한다. 간단한 문자열을 사용해서 표현될 수 있는 것이면 더 좋다. ② 개체군을 초기화하는 방법을 결정하고, 교차와 돌연변이 연산을 설계하고, 유전자 알고리즘의 종료조건을 결정해야 한다. ③ 마지막으로 가장 중요한 것으로 앞에서 설명한 적합성 함수를 정의해야 한다.

3.2.1 유지 비용 뷰 선택 문제의 정의

Gupta와 Mumick[13]의 정의에 따라 문제를 요약하여 기술하면 다음과 같다. OR 뷰 그래프 G 및 전체 유지 비용

시간에 대한 제한값이 주어졌을때 전체 질의 응답시간을 최소화하면서 전체 유지 비용 시간 제한을 초과하지 않도록 구체화된 뷰의 집합을 선택하여야 하는 것이다. OR 뷰 그래프는 데이터 웨어하우스에서 수행될 질의들을 기반으로 만들어진다. 각각의 질의는 한 개의 뷰로서 그래프에서 한 개의 노드를 형성한다. 데이터 웨어하우스에서 수행될 질의들에 해당하는 뷰를 생성하는데에 도움이 되는 이외의 뷰들도 추가적으로 노드로서 포함될 수 있다. 뷰들간의 관계는 노드사이에 간선으로 표현한다. OR 뷰 그래프에서 각각의 뷰는 간선으로 연결된 다른 근원 뷰들로부터 유도될 수 있으며 연결된 근원 뷰들중에서 한 개의 근원 뷰만을 사용하여 유도가 가능하다. 다시 말하자면, 특정 뷰를 만들어내는데에는 근원 뷰들 중에서 단지 하나의 뷰만이 필요하다. OR 뷰 그래프의 예는 데이터 큐브이다. 여기에서 각각의 뷰는 다양한 방법으로 생성될 수 있으나 각각의 방법은 단지 하나의 뷰만을 사용한다.



(그림 2) 네 개의 뷰를 포함하는 예제 OR 뷰 그래프

(그림 2)에 OR 뷰 그래프 예제가 있다. 이 그림에서 예를 들자면 뷰 a는 b, c, d 중의 어느 뷰로부터도 계산될 수 있다. 뷰 b는 다시 g나 h 중의 어느 뷰로부터도 계산될 수 있다. 뷰 c, d를 자식 뷰로부터 계산하는 것도 마찬가지이다. 모든 OR들을 AND로 바꾸면 이 그래프는 AND 뷰 그래프가 되며 이것은 각각의 부모 뷰를 계산하는 데에 모든 자식 뷰들이 필요하다는 것을 의미한다. 그래프 안에 OR와 AND가 혼재하면 AND-OR 뷰 그래프라고 부른다. 본 연구에서는 OR 뷰 그래프 자체가 주요 연구 대상은 아니지만 알고리즘에서 필요로 하는 비용모델을 구성하기 위해서 OR 뷰 그래프에 관련된 변수들을 사용한다. 유지 비용 뷰 문제의 복잡도는 곧 탐색 공간의 크기로써 알 수 있다. 그래프 내에 n개의 뷰가 존재하면 뷰의 부분집합의 총 갯수는 2^n이므로 문제는 지수적인 복잡도를 갖는다.

서론에서 언급했던 것처럼 OR 뷰 그래프만을 고려하는 이유는 비용 모델이 다른 두 개의 뷰 그래프 모델(i.e., AND와 AND-OR 뷰 그래프)보다 덜 복잡하므로 광범위한 변수들을 고려해야 함으로써 생기는 부정확성을 줄일 수 있다는 장점이 있기 때문이다. 간결한 비용 모델을 채택함으로써 복잡한 비용 모델과 관련된 논쟁과 분리해서 본 실험의 결과를 분석할 수 있다. 실제로 OR 뷰 그래프는 더 간단한 비용 모델을 가짐에도 불구하고 매우 유용하며, 앞에 언급한 것처럼 데이터 큐브의 형태로 데이터 웨어하우스에서 자주 사용된다.

다음과 같이 유지 비용 뷰 선택 문제를 정의할 수 있다.

[정의] OR 뷰 그래프를 위한 유지 비용 뷰 선택 문제
OR 뷰 그래프 G, 그리고 총 유지 비용 제한 S가 주어졌을 때, 다음 두 가지 조건을 만족하는 구체화된 뷰의 집합 M을 선택한다. ① OR 뷰 그래프 G에 대한 질의 비용의 총합 τ를 최소화한다. 즉, 구체화된 뷰들을 생성함으로써 얻어지는 질의 비용에 대한 이익인 B를 극대화한다. ② 구체화된 뷰들의 유지 비용의 총합 U가 S보다 적다.

위의 공식적인 정의에 의하면 유지 비용 뷰 선택 문제에서유지 비용 제한 S는 엄격하게 지켜져야 할 것 같지만 실제적인 데이터웨어하우스 설계 상황에서는 어느 정도의 융통성을 발휘할 수 있는 부분이기도 하다. 즉 데이터웨어하우스 설계시에 정확한 제한값을 100% 만족시킬 수도 있고 약간의 유지 비용이 더 들더라도 성능이 훨씬 좋아질 수 있다면 허용되는 경우도 있다.

3.2.2 유전자 알고리즘의 설계

(1) 1단계 : 해결책의 표현 방법

하나의 게놈은 해결하고자 하는 문제에 대한 하나의 후보 해결책을 표현한다. 문제의 해결책은 스트링 형태로 표현되어야 한다. 스트링은 0과 1로 구성된 이진 숫자 스트링이거나 영문자숫자의 스트링일 수 있다. 스트링은 다양하게 구성할 수 있으나 탐색 공간 내의 모든 가능한 해결책들을 표현할 수 있도록 설계되어야 한다.

본 논문에서는 이진 스트링 표현을 사용한다. OR 뷰 그래프에서 뷰의 전체 갯수를 m이라고 할 때 각 뷰를 v_1, v_2, …, v_m과 같이 표시할 수 있다. 이러한 뷰들에 대한 선택을 m 비트의 이진 스트링으로써 표현할 수 있다. 위치 i에서 비트가 1이라면 뷰 v_i가 선택된 것이다. 만약 그렇지 않으면 뷰 v_i가 선택되지 않은 것이다. 예를 들면, 비트 스트링 001101001 은 전체 9개의 뷰 집합으로부터 단지 v_3, v_4, v_6, v_9만 선택된 것을 표시한다. 이것을 일반화하면 게놈을 다음과 같이 표시할 수 있다.

$$\text{게놈(genome)} = (b_1, b_2, b_3, \cdots, b_m),$$

단, $b_i = 1$: 뷰 v_i가 구체화를 위해 선택된 경우
$b_i = 0$: 뷰 v_i가 구체화를 위해 선택되지 않은 경우

(2) 2단계 : 개체군의 초기화

초기의 개체군은 무작위로 생성된 길이 m인 비트 스트링의 집합이다. 추후에는 무작위의 방법보다는 문제에 대한 지식을 반영하는 좀더 적절한 초기 개체군을 생성하는 것으로 연구의 범위를 확장하고자 한다. 한 가지 방법은 OR 뷰 그래프의 구성 정보를 이용하여 초기의 게놈들을 생성할 수 있다. 즉 높은 질의 빈도를 가진 뷰는 구체화를 위해서 대부분 선택되므로 이들을 초기 게놈에 반영하는 것이다. 이러한 방법이 알고리즘의 실행뿐만 아니라 해결책의 질에도 어떤 영향을 미치는지를 알아보는 것도 매우 흥미로울 것이다. 본 논문의 실험에서는 개체군의 크기를 30으로 하였는데 이는 [8]에 기술되어 있는 De Jong의 실험에서 일반적으로 사용되는 수치이다.

(3) 3단계 : 선택, 교차, 돌연변이, 종료

선택 과정은 유명한 룰렛 바퀴(roulette wheel) 방법을 사용한다[9]. 교차(crossover)와 돌연변이(mutation) 연산에는 각각 확률 p_c, p_m이 지정된다. 본 실험에서 이들 확률에 사용한 수치는 각각 0.9와 0.001이다. 이 값을 택한 이유는 관련된 문제를 풀 때 다른 연구자들이 같은 값을 사용하여 성공했기 때문이다[9]. 1975년 De Jong에 의한 유전자 알고리즘의 매개변수에 대한 연구[8]에서는 높은 교차 확률, 낮은 돌연변이 확률 그리고 적당한 개체군 크기를 권장한다. 무작위 알고리즘에 관한 여러 연구 결과들을 참조하여 실험을 해본 결과 본 논문에서 사용된 매개변수 값들이 유지 비용 뷰 문제를 해결하는 데에 가장 적절한 결과를 내는 것을 발견하였다.

룰렛 바퀴 방법은 각각의 게놈의 적합성 값을 사용한다. 적합성 값은 게놈이 문제에 대한 해결책으로써 얼마나 좋은지를 나타낸다. 적합성 값에 대한 자세한 계산 방법은 뒤에 설명한다. 룰렛 바퀴 방법은 다음과 같다.

```
// 각 게놈이 1과 pop_size 사이의 숫자를 지정받았다고 하자.
// 각 게놈의 적합성 값은 f_i라고 하자.
개체군의 총 적합성 값 F를 각 게놈들의 적합성 값을 합산함으로써 계산할 수 있다.

    F = Σ_{i=1, pop_size} f_i

각 게놈의 선택 확률 p_i는 각 게놈의 적합성 값을 F로 나눔으로써 계산할 수 있다.

    p_i = f_i / F

각 게놈의 누적 확률 c_i를 게놈 1에서 i까지의 p_i 값들을 합산함으로써 계산할 수 있다.

    c_i = Σ_{j=1,i} p_j

다음의 과정들을 pop_size 회 만큼 반복하되 이것은 룰렛 바퀴를 pop_size 회 돌리는 것과 같은 효과를 갖는다 :
    무작위의 숫자 Rn이 0과 1사이의 값을 갖도록 한다.

        Rn = random (0, 1)

    만약 Rn < c_1이면 첫 번째 게놈 1을 선택한다.
    그렇지 않으면 c_{i-1} < Rn ≤ c_i를 만족하는 게놈 i를 선택한다.
```

교차 연산은 두 게놈에 적용되며 이들 게놈 사이에 정보를 교환하여 두 개의 새로운 게놈을 만든다. 교차 연산은 다음과 같이 이루어진다.

```
각 게놈은 p_c의 확률로 선택될 수 있다.
선택된 게놈들의 쌍을 지어준다.
각 쌍에 대해서 다음과 같은 연산을 수행한다 :
    // 두 개의 게놈들 g1과 g2를 가정하자.
    //     g1 = (b_1, b_2, ···, b_pos   b_{pos+1}, ···, b_m)
    //     g2 = (c_1, c_2, ···, c_pos   c_{pos+1}, ···, c_m)
    무작위로 교차점 pos를 결정한다.

    게놈들간에 정보를 교환하여 아래와 같이 g1, g2를 g1', g2'
    으로 대체한다.
    // (예) g1' = (b_1, b_2, ···, b_pos   c_{pos+1}, ···, c_m)
    //      g2' = (c_1, c_2, ···, c_pos   c_{pos+1}, ···, b_m)
```

돌연변이 연산자는 한 개의 게놈에 변화를 주며 다음과 같이 작동한다.

```
모든 게놈들에 대하여,
    게놈의 모든 비트에 대하여,
        비트를 p_m의 확률로 반전시킨다.
```

선택, 교차, 돌연변이, 평가(아래의 4단계에서 설명됨) 과정은 종료조건이 만족될 때까지 반복된다. 본 연구에서 종료조건은 400세대이다. 종료 조건은 이렇게 설계된 유전자 알고리즘을 이용한 여러 번의 실험을 통해서 결정할 수 있는 조정 가능한 반복 횟수 값이다. 비록 Michalewicz[9]의 연구에서는 대략 500세대 후에 0/1 배낭 문제에 대해서 유전자 알고리즘이 더 이상 향상되지 않는 것이 관찰되었는데 본 연구에서는 설계된 알고리즘이 더 빠르게 수렴하기 때문에 이 값을 400으로 줄일 수 있었다.

(4) 4단계 : 평가 과정

적합성 함수는 다음과 같이 적합성 값을 제공함으로써 해결책(게놈)이 얼마나 좋은 것인지를 평가하도록 해준다. 만약 적합성이 높으면 해결책은 목표를 만족하는 것이며 적합성이 낮으면 게놈은 해결책으로서 적절하지 못한 것이다.

높은 적합성 값을 가진 해결책이 원하는 대상이며 다음 세대를 위해서 보통 선택되지만, 낮은 적합성 값을 가진 해결책도 다음 세대에 낮은 확률로 포함되게 된다. 이것은 유전자 알고리즘이 진화를 위해서 다양한 가능성을 갖도록 해준다. 각 게놈에 대한 평가는 개체군의 총체적인 적합성을 평가하는 기초를 형성한다. 그러므로 적합성 함수를 정확하게 정의하는 것은 유전자 알고리즘의 성공을 좌우하는 중요한 요소이다.

본 문제에서 적합성 함수는 질의 이익(즉 뷰의 형태로 질의 결과를 구체화시킴으로 해서 얻어지는 질의 비용의 감소량)과 유지 비용 제약을 고려하여 게놈을 평가한다. 이

것은 0/1 배낭(knapsack) 문제에서 배낭의 용량 제한을 만족하는 동시에 배낭에 든 물건들로 생기는 이익을 최대화하려는 것과 비슷하다. 다른 점이라면 유지 비용 뷰 선택 문제에서는 어떤 뷰가 선택될 때의 이익 계산은 해당 뷰 자신 뿐만 아니라 이미 선택된 다른 뷰에도 영향을 준다는 것이다.

이러한 복잡한 문제를 모델하기 위한 좋은 방법으로는 적합성 함수의 일부로 벌칙(penalty) 개념을 첨가한다. 이 벌칙 함수는 유지 비용 제약이 만족되지 않으면 적합성을 감소시킨다. 유지 비용 제약이 만족되면 벌칙 함수는 영향력이 없고, 질의 이익만 평가된다. 본 논문에서는 적합성을 계산할 때 세 가지 방법으로 벌칙 값을 적용시킨다. 이들은 뺄셈 모드(S : Subtract), 나눗셈 모드(D : Divide), 뺄셈&나눗셈 모드(SD : Subtract&Divide)이다. 뺄셈 모드는 질의 이익에서 벌칙 값을 빼는 방법으로 적합성을 계산한다. 적합성 값은 음수를 가질 수 없기 때문에, 계산 결과가 음수가 되면(즉, 벌칙 값이 질의 이익을 초과한다면) 적합성은 0이 된다. 나눗셈 모드는 질의 이익을 감소시키기 위한 방법으로 질의 이익을 벌칙 값으로 나눈다. 벌칙 값이 1보다 작으면, 적합성이 증가하는 것을 방지하기 위해서 나눗셈을 하지 않는다. 뺄셈 & 나눗셈 모드는 위에서의 두 방법을 조합하였다. 질의 이익이 벌칙 값보다 크면 뺄셈 모드가 사용된다. 벌칙 값이 질의 이익보다 크면, 나눗셈 모드가 사용된다. 벌칙 값은 뒤에서 설명할 벌칙 함수를 사용하여 계산한다.

B는 질의 이익 함수, Pen은 벌칙 함수, x는 유전자, G는 OR 뷰 그래프, M은 x로부터 얻어진 선택된 뷰의 집합이라고 하자. 그리고 만일 x안의 비트 i가 1로 설정되어 있다면 이 비트에 의해 표현되는 뷰 v_i는 M에 포함되며 그렇지 않으면 이 뷰는 M에 포함되지 않는다. 이때 Eval이라고 하는 적합성 함수를 다음과 같이 정의할 수 있다.

Subtract mode (S) :

$$Eval(x) = B(G,M) - Pen(x) \quad (if\ B(G,M) - Pen(x) \geq 0) \qquad (3.1)$$
$$= 0 \quad (if\ B(G,M) - Pen(x) < 0)$$
$$for\ all\ x[i]=1,\ v_i \in M$$
$$for\ all\ x[i]=0,\ v_i \notin M$$

Divide (D) :

$$Eval(x) = B(G,M) / Pen(x) \quad (if\ Pen(x) > 1) \qquad (3.2)$$
$$= B(G,M) \quad (if Pen(x) \leq 1)$$

Subtract&Divide (SD) :

$$Eval(x) = B(G,M) - Pen(x) \quad (if\ B(G,M) > Pen(x)) \qquad (3.3)$$
$$= B(G,M) / Pen(x) \quad (if\ Pen(x) \geq B(G,M)\ and\ Pen(x) > 1)$$
$$= B(G,M) \quad (if\ Pen(x) \geq B(G,M)\ and\ Pen(x) \leq 1)$$

벌칙 함수 자체도 다양한 형태를 가질 수 있다. 예를 들면, 수식 (3.4), (3.5), (3.6)에 보인 것과 같이 본 연구에서는 로그(logarithmic) 벌칙, 선형(linear) 벌칙, 지수(exponential) 벌칙 함수를 가지고 실험하였다. 이들 함수들은 나열된 순서로 벌칙 값이 증가한다. 로그 벌칙 함수는 가장 적은 벌칙 값을 적용하고, 지수 벌칙 함수는 가장 큰 벌칙 값을 적용한다. 수식에 사용된 함수 U는 구체화된 뷰의 집합 M의 총 유지 비용을 계산한다. 상수 ρ는 질의 이익 함수와 총 유지 비용 함수로부터 계산되는 값이다. S는 총 유지 비용 제약을 나타내는 값이다.

Logarithmic penalty(LG) :

$$Pen(x) = log_2 (1 + \rho (U(M) - S)) \qquad (3.4)$$

Linear penalty(LN) : $Pen(x) = (1 + \rho (U(M) - S))$ (3.5)

Exponential penalty(EX) : $Pen(x) = (1 + \rho (U(M) - S))^2$ (3.6)

유지 비용 뷰 선택 문제를 해결하기 위한 가장 최선의 전략을 평가하고 결정하기 위하여 세 개의 벌칙 모드(즉 S, D, SD)와 세 개의 벌칙 함수(LG, LN, EX)를 조합하였다. 이들 9개의 조합(LG-S, LG-D, LG-SD, LN-S, LN-D, LN-SD, EX-S, EX-D, EX-SD)을 평가해본 결과 몇 가지 가능성 있는 전략들을 확인하였다.

3.2.3 비용 모델과 공식

질의 이익 함수 B(G, M)와 총 유지 비용 U(M) 및 ρ의 공식을 다음에 설명하고자 한다. 공식을 설명하기 전에, 먼저 OR-뷰 그래프에 할당된 비용들을 설명한다. <표 1>은 비용과 관련된 변수들을 보여준다.

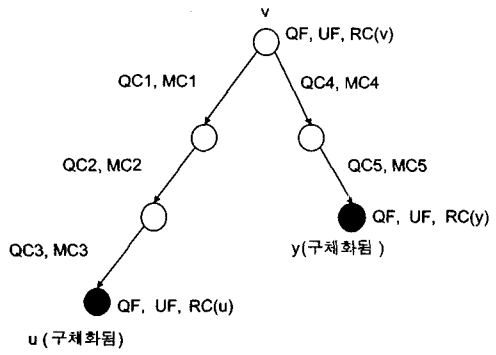
〈표 1〉 OR 뷰 그래프의 비용 변수들

	변수	설 명
노드(뷰)	RC	뷰의 읽기 비용. 뷰의 크기를 나타내기도 함.
	QF	질의 빈도. 주어진 시간 간격 동안에 뷰에 던져지는 질의의 갯수
	UF	갱신 빈도. 주어진 시간 간격 동안에 뷰에 가해지는 갱신 갯수
간선	QC	질의 비용. 주어진 뷰를 소스 뷰 로부터 계산하는 비용
	MC	유지비용으로서 소스 뷰를 사용하여 뷰를 갱신하는 비용

(그림 3)은 OR-뷰 그래프의 예를 보여준다. 그래프에서 각각의 노드는 뷰를 나태내며, 읽기 비용(RC), 질의 빈도(QF), 갱신 빈도(UF)가 지정된다. 또한 그래프의 간선은 뷰들 간의 관계를 표시하며 질의 비용(QC)과 유지비용(MC)이 지정된다.

그래프의 각각의 간선은 질의 비용(QC1 에서 QC5)과 유지비용(MC1에서 MC5)을 갖는다. 그래프의 흰색 노드는 비구체화 뷰(즉 구체화를 위해 선택되지 않은 뷰)를 나타내며, 흑색 노드는 구체화를 위해 선택된 뷰를 나타낸다. 비

구체화 뷰와 구체화 뷰 모두 읽기 비용을 갖는다. 그러나 다음 예에서 설명을 단순화 시키기 위해서 본문에서 사용된 읽기 비용만을 표시하였다. 그래프에서 보는 바와 같이 구체화된 노드 u와 y, 그리고, 구체화되지 않은 노드 v는 각각 읽기 비용인 R(u), R(y), R(v)가 지정된다. 질의 빈도(QF)와 갱신 빈도(UF)도 각각의 노드에 모두 할당되지만, 그림을 간단히 하기 위해서 u, v, y노드에 대한 것만 보여주고 있다.



(그림 3) 비용 변수들을 표시한 OR 뷰 그래프

질의 이익 함수 B(G, M)을 정의하기 위해서, 우리는 먼저 Q(v, M)와 τ(G, M) 두 개의 함수를 정의해야 한다. 다음의 정의에서 구체화된 뷰의 집합 M은 유전자 알고리즘에서 하나의 계놈과 같다.

[정의 1] 구체화된 뷰의 집합을 M이라고 할 때 Q(v, M)은 v(뷰이거나 베이스 테이블)를 이용하여 **질의에 응답하는 비용**이다. G에서 베이스 테이블의 집합을 L이라고 했을 때 이 함수는 v에서 어떤 $u \in$(M U L)로 가는 경로에 대한 질의 길이의 최소값을 계산한다. 질의 길이는 다음과 같이 계산한다.

v에서 u로 가는 경로에 대한 질의 길이(*Query-length*)
= $RC(u)$ + (v에서 u로의 경로의 간선에 관련된 모든 질의 비용의 합 즉 *sum(query-costs)*)

베이스 테이블이 언제나 접근 가능하다고 가정하면, Q(v, ϕ)는 베이스 테이블을 이용해서 직접 질의를 응답하는 비용이다. (그림 3)의 예에서 RC(u) + QC3 + QC2 + QC1 < RC(y) + QC5 + QC4이라고 하자. 그러면 구체화된 뷰의 집합이 M일때 v에 대한 *질의에 응답하는 비용*은 Q(v, M) = RC(u) + QC3 + QC2 + QC1으로 계산된다.

[정의 2] 총 질의 비용인 τ(G, M)은 구체화된 뷰의 집합 M이 있을 때 OR 뷰 그래프 G에 대해서 정의된다. 이것이 바로 유지 비용 뷰 선택 문제에서 최소화되어야 하는 값이다.

$$\tau(G, M) = \Sigma QF_v \times Q(v, M), \text{ 단 } v \in V(G)$$

이 공식에서 QFv는 뷰 v의 질의 빈도이며, V(G)는 OR 뷰 그래프 G의 *모든(구체화되었거나 구체화되지 않은) 뷰*들을 가리키며, Q(v, M)은 위에서 정의된 것처럼 구체화된 뷰의 집합 M이 있을 때 v에 대한 *질의 비용*이다.

[정의 3] [정의 1]과 [정의 2]를 사용하여 **질의 이익 함수** B(G, M) 를 정의하면 아래와 같으며 이때 주어진 OR 뷰 그래프 G에서 선택된 구체화된 뷰의 집합이 M이다.

$$B(G, M) = \tau(G, \phi) - \tau(G, M)$$

Gupta와 Mumick[13]은 B를 M의 *절대 이익*이라고 한다. 본 논문에서의 B(G, M)의 의미는 저자들이 제안했던 B(M, ϕ)와 같다.

벌칙 함수는 **총 유지 비용 함수** U(M)을 사용한다. U(M)을 정의하기 위해서, UC(v, M) 함수를 먼저 정의해야 한다.

[정의 4] 구체화된 뷰의 집합을 M이라고 할 때 UC(v, M)은 **뷰 v를 유지하는 데에 드는 비용**이다. 이것은 G에서의 베이스 테이블의 집합을 L이라고 할 때 v에서 어떤 y $\in$ (M U L)-{v}로 가는 경로에 대한 *최소 유지 비용 길이*(간선과 관련된 유지비용들의 합)로서 계산된다.

(그림 3)의 예에서 MC4 + MC5 < MC1 + MC2 + MC3라고 가정한다면 UC(v, M) = MC4 + MC5로 계산된다.

[정의 5] M이 구체화를 위해 선택된 뷰의 집합이며 UFv는 뷰 v의 갱신 주기라고 가정하자. 그러면 **총 유지 비용 함수** U(M)은 다음과 같이 정의된다.

$$U(M) = \Sigma UF_v \times UC(v, M), \text{ 단 } v \in M$$

이 비용은 모든 뷰가 아닌 오직 *구체화된 뷰*만을 대상으로 계산된다는 점에 주의해야 한다.

[정의 6] 벌칙함수에서 사용되는 ρ를 계산하는 공식은 다음과 같다.

$$\rho = Max(\frac{B(G, \{v_i\})}{U(\{v_i\})}), \text{ 단 } v_i \in V(G)$$

본 논문에서의 뷰 그래프의 대상을 AND-OR 뷰 그래프로 확장하려면 Q(v, M)과 UC(v, M)을 변경함으로써 가능하다. 즉 Q(v, M)에서 v가 OR 노드이면 원래대로 계산을 하고, 만약 v가 AND 노드이면 v 노드의 모든 자식 노드들을 u1, u2, …, uk라고 할때 모든 Q(ui, M)의 합을 구하는 것으로 계산한다. 단, 이때 $1 \le i \le k$이다. 또한 UC(v, M)에서 v가 OR 노드이면 원래대로 계산을 하고, 만약 v가 AND 노

드이면 v 노드의 모든 자식 노드들을 u1, u2, …, uk라고 할때 모든 UC(ui, M)의 합을 구하는 것으로 계산한다. 단, $1 \le i \le k$이다.

실제로 점진적 관리(Incremental Maintenance)가 가능한 데이터웨어하우스의 설계를 지원하기 위해서는 본 연구에서 제시한 비용 평가 모델을 수정하면 된다. 현재 각 간선마다 유지비용과 관련된 변수인 MC 값들을 할당하고 있고 각 노드에 대한 유지비용을 계산할 때에는 단순히 자식 노드들 중에서 최소의 유지비용으로 갱신이 가능한 값을 구하고 있다. 이를 점진적 관리가 가능하도록 변경한다면 실제 점진적 관리를 위해 드는 비용을 정확하게 계산하는 MC 값 함수들을 새로 정의하여 간선에 지정함으로써 구현할 수 있다.

4. 프로토타입의 구현

제안한 유전자 알고리즘은 윈도우 2000 운영체제를 갖춘 펜티엄 4에 1GHz 컴퓨터를 사용하여 개발되고 실험되었다. Galib[21]이라 불리는 MIT의 유전자 알고리즘 툴킷 버전 2.4.3을 사용하여 알고리즘을 구현하였다. 이 툴킷은 다양한 유전자 알고리즘들과 돌연변이 연산, 교차연산, 1차원 또는 2차원 스트링들과 같은 게놈 라이브러리, 그리고 유전자 알고리즘이 수행될 때의 각 세대에 대한 요약된 통계 정보를 제공하는 도구들을 가지고 있다. 프로토타입은 마이크로소프트 비쥬얼 C++로 작성되었다.

Galib 툴킷은 본 연구에서 필요로 하는 적합성 함수를 제공하지 않으므로 직접 작성을 하였다. 구현된 적합성 함수는 각 벌칙 모드와 각 벌칙 함수의 종류가 쌍을 이루는 9가지의 다른 방법으로 적합성을 계산할 수 있다. 특정 변수를 이용해서 원하는 전략을 설정하도록 함으로써 벌칙 값이 계산되는 방식과 적합성을 적용하는 방법을 제어할 수 있다. 이러한 방법으로 실험을 할 때에 서로 다른 벌칙 모드들을 손쉽게 실험할 수 있었다. 적합성 함수를 구현할 때 주어진 게놈의 적합성을 계산하기 위해서는 선택된 구체화된 뷰들의 총 유지 비용과 총 질의 비용을 계산하는 공식을 구현해야 한다. 이들 비용을 계산하려면 많은 시간이 걸리므로 효율적인 방법이 필요하다. 그러므로 본 프로토타입에서는 OR 뷰 그래프에 대해서 깊이 우선 탐색을 하는 방법을 이용하였고 깊이 우선 탐색 동안 임시적인 결과들을 OR 뷰 그래프의 노드들에 저장함으로써 중복된 계산들을 제거할 수 있었다. 비용 계산을 위한 자세한 공식과 예들은 이미 3.2.3절에서 설명하였다.

본 연구에서 제안한 새로운 교차연산도 제공되지 않아서 직접 작성하였고 돌연변이 연산은 이미 제공되는 것을 사용하였다.

또한 실험을 위해서 그래프의 변수 값들과 노드 밀도가 주어지면 이에 따라서 OR 뷰 그래프를 무작위로 생성할 수 있는 OR 뷰 그래프 생성기를 작성하였다. OR 뷰 그래프를 결정하는 변수들은 베이스 테이블의 갯수, 뷰의 갯수, 밀도, 질의 비용 범위, 유지 비용 범위, 베이스 테이블의 읽기 비용 범위, 질의 빈도 범위, 갱신 빈도 범위이다. 이와 함께 최적의 해결책을 찾을 수 있는 전체(exhaustive) 탐색 알고리즘을 구현하여 본 논문에서 제안한 유전자 알고리즘(GA)에서 나온 해결책의 질과 최적의 해결책을 비교할 수 있도록 하였다.

5. 알고리즘의 성능 평가

실험을 통해 2가지를 평가하였다. 첫째는 적합성 함수의 9가지 전략들을 최적의 해결책에 대해서 질적인 면에서 비교하였다. 둘째는 유전자 알고리즘의 실행시간을 전체 탐색 알고리즘의 것과 비교하였다.

본 실험에 사용된 데이터의 특성을 설명하기 위하여 생성된 OR 뷰 그래프에 대한 사항들을 <표 2>와 <표 3>에 정리하였다. 베이스 테이블의 개수는 10개로 고정되었고 뷰의 개수는 5개에서 20개까지 16가지로 변하였다. 본 논문에서 제안한 유전자 알고리즘을 실제로는 20개 이상의 뷰를 가지고도 실험을 하였으나 다른 연구들에서는 이러한 경우는 제외하거나 불가능하여서 비교할 수 있는 대상이 없어 분석 대상에서 제외하였다. 또한 25개 이상의 뷰로 실험을 하였을 때에는 전체(exhaustive) 탐색 알고리즘의 실행시간이 지나치게 길어져서 비교 자체가 불가능해졌다. 예를 들면 펜티엄 4 컴퓨터로 256메가의 램을 갖춘 컴퓨터를 사용하였을 경우, 30개의 뷰를 갖춘 웨어하우스에 대한 전체(exhaustive) 탐색 알고리즘은 12시간 이상의 실행시간이 걸렸다. 실험에서 그래프의 간선 밀도는 15%에서 30%, 50%, 75%까지 변화하였다. 4가지의 간선 밀도에 대해서 각각 16가지의 뷰의 개수에 대한 경우를 조합한 것은 총 64(=16×4)가지의 상황을 만들며 각각의 상황에 대해서 그래프는 10개씩을 생성하여 평균을 계산하는 방식으로 실험하였다. 생성된 OR뷰 그래프의 노드 및 간선에는 3.2.3절에서 설명된 모든 중요한 비용모델 변수들의 범위가 <표 3>과 같이 할당되었다. 문제에 대한 유지비용 제한은 50, 100, 300, 500으로 각각 설정되었다. 이러한 유지비용 제한이라는 수치를 몇 가지로 해석할 수 있는데 유지를 위해서 얼마나 오래 동안 웨어하우스가 다운되어 있을 수 있는지에 대한 제한 시간으로 볼 수도 있고 읽혀져야 하는 데이타의 양 등으로 다르게 해석할 수도 있다. 64가지의 그래프 형태와 4가지

유지비용 제한을 조합하여 총 256가지의 경우에 대한 실험을 하였다.

〈표 2〉 실험에 사용된 OR 뷰 그래프의 특성 및 실험 환경 변수의 범위

	실험 환경 변수	값
그래프의 형태 (64가지)	베이스 테이블의 개수	10개로 고정
	간선의 밀도(4가지)	15%, 30%, 50%, 75%
	뷰의 개수(16가지)	5에서 20개
그래프의 개수	각 그래프 형태별로 생성한 개수	10개씩
문제 정의 변수 (4가지)	유지비용 제한	50, 100, 300, 500

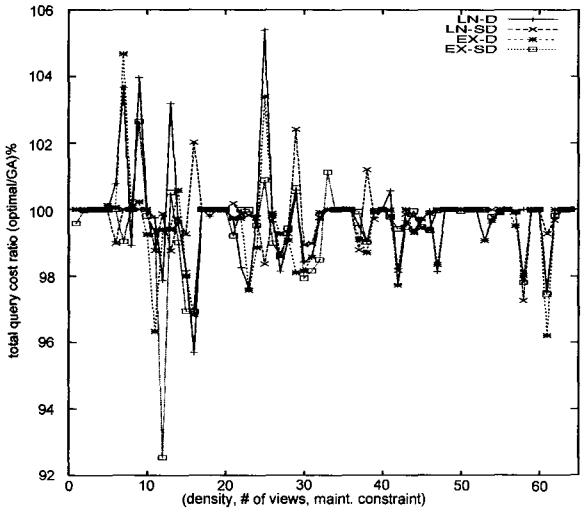
〈표 3〉 각 OR 뷰 그래프에 할당된 비용모델 변수값의 범위

노드 (뷰)			간선	
RC	*QF*	*UF*	*QC*	*MC*
베이스 테이블은100~10,000, 뷰들의 RC 값은 근원뷰들로부터 계산됨.	0.1~0.9	0.1~0.9	근원 뷰의 RC 값의 10 ~80 %	QC의 10~150%

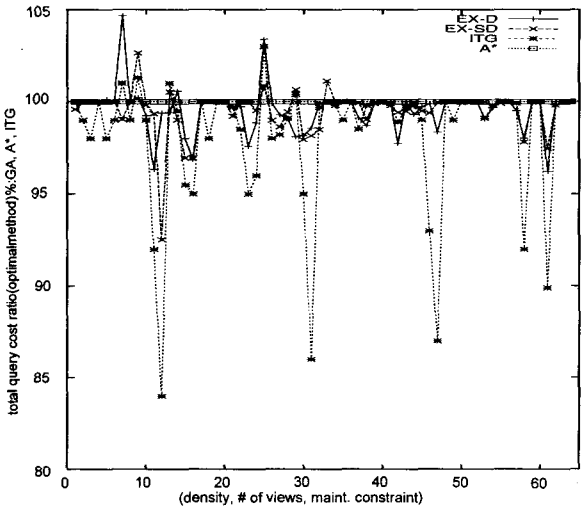
5.1 해결책의 성능

먼저 9가지의 조합에 따른 적합성 함수들을 사용하여 실험을 하였다. 해결책의 질에 대한 평가는 유전자 알고리즘을 사용해서 얻은 해결책의 질의 비용에 대한 최적 질의 비용의 비율을 계산하여 평가하였다. 이때 최적 질의 비용은 전체 탐색 알고리즘을 사용해서 얻은 해결책의 질의 비용이다. 이 비율을 여러 번의 실험을 통해 평균 값을 구하였다. 실험 전에는 이 비율은 항상 100% 이하일 것이라고 예상하였다. 하지만 유전자 알고리즘은 유지 비용 제한을 약간만 초과함으로써 더 적은 질의 비용을 얻을 수 있는 경우도 발견하여 때때로 유지 비용 제한을 약화시키기도 하였다는 것을 알았다. 이런 경우, 얻어진 질의 비용은 실제로 엄격한 유지비용 제한 속에서 계산된 최적 질의 비용보다 낮았다. 즉 실제로 비율이 100%를 초과하였다. 이런 현상은 유지 비용 제한을 둔다고 하여도 이것은 단지 참고가 되는 값일 뿐이지 엄격히 지켜지는 값이 아니라는 점에서 매우 흥미롭다고 볼 수 있다. 실제로 Gupta와 Mumick [13]의 반전-트리 욕심쟁이 휴리스틱(inverted-tree greedy heuristic)도 엄격한 유지 비용 제한을 보장하지 못하고, 제한 값의 2배수 범위 내의 제한 값을 만족한다. 9개의 전략은 다음과 같이 표기되었다 : LG-S, LG-D, LG-SD, LN-S, LN-D, LN-SD, EX-S, EX-D, EX-SD. 여기에서 LG, LN, EX, S, D, SD는 3.2.2절에서 설명된 서로 다른 벌칙들과 함수들을 나타낸다.

기본적인 실험 후에, 로그 벌칙 함수(LG-S, LG-D, LG-SD)는 성능이 좋지 않음을 발견하였으며 그 중에서도 LG-S와 LG-SD가 성능이 가장 좋지 못했다. 그 이유는 로그 벌칙은 벌칙 값을 너무 작게 만들어서 적합성 값에 영향력을 충분히 행사하지 못했기 때문이다. 그래서, LG-S와 LG-SD는 오히려 유지 비용의 제약을 무시하고 질의 이익을 극대화하려는 경향을 보여 결국 모든 뷰를 구체화하는 해결책을 항상 제시하였다. LG-D와 LN-S, EX-S 들은 위와 같은 극단적인 결과는 나타내지 않았지만, 유지 비용 제한 값에서 크게 벗어나거나 때로는 10000%까지 초과하는 경향을 보였다. 그리하여 본 접근 방법에서는 이런 결과들을 제외한 LN-D, LN-SD, EX-D, EX-SD 의 결과들만을 (그림 4)에 보인다.



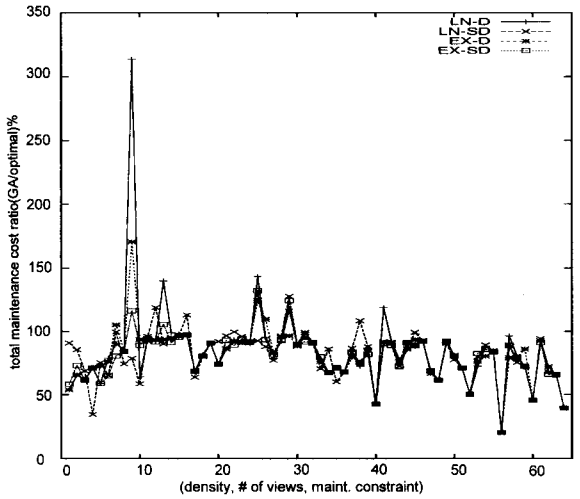
(그림 4) GA 질의 비용에 대한 최적 질의 비용의 비율의 평균



(그림 5) A*, 반전-트리 욕심쟁이(ITG) 알고리즘의 질의 비용에 대한 최적 질의 비용의 비율의 평균

〈표 4〉 실험 상황에 사용된 OR 뷰 그래프의 특성과 유지비용 제한값의 범위

실험상황	OR 뷰 그래프의 특성		유지비용 제한값 (4가지)	총 실험상황의 개수
	간선 밀도	뷰 개수(4가지)		
1~16	15%	5, 10, 15, 20	50, 100, 300, 500	16가지
17~32	30%	5, 10, 15, 20	50, 100, 300, 500	16가지
33~48	50%	5, 10, 15, 20	50, 100, 300, 500	16가지
49~64	75%	5, 10, 15, 20	50, 100, 300, 500	16가지



(그림 6) 유지 비용 제한에 대한 GA 총 유지 비용의 비율의 평균

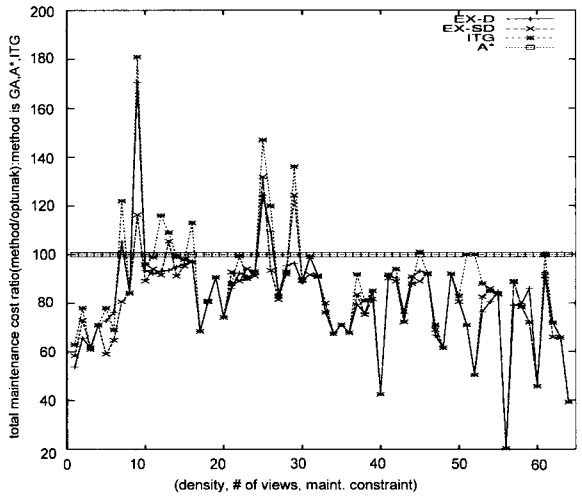
(그림 4)는 유전자 알고리즘(GA)의 총 질의 비용에 대한 최적의 총 질의 비용의 비율을 보여주고 있다. (그림 5)는 유전자 알고리즘들 중에서 EX-D와 EX-SD를 기존의 기법들인 A* 알고리즘과 반전-트리 욕심쟁이 알고리즘(ITG)과 비교하여 이들 총 질의 비용에 대한 최적의 총 질의 비용의 비율을 보여주고 있다. 이들 결과값들은 x축에 대하여 다음과 같은 좌우선 정렬의 우선순위를 갖는 순서로 정렬되었다.

(*간선 밀도, 뷰의 개수, 유지 비용 제한*) = (*density, number of views, maintenance constraint*)

예를 들어, 간선밀도가 10 또는 20이고 뷰의 개수가 4 또는 5이며 유지비용 제한이 800이나 900이라고 하면 다음과 같이 나열되고 순서대로 1에서 8번으로 번호가 붙여진다.

(10, 4, 800), (10, 4, 900), (10, 5, 800), (10, 5, 900), (20, 4, 800), (20, 4, 900), (20, 5, 800), (20, 5, 900)

밀도(density)의 변화는 x축의 1, 17, 33, 49에서 발생했고, 각각의 지점에서 밀도가 15%, 30%, 50%, 75%로 증가했다. 각 밀도 내에서 뷰의 개수는 5, 10, 15, 20개로 증가하는 4가지 경우로 나뉜다. 또한 각 뷰의 개수 내에서 유지 비용은 50, 100, 300, 500으로 증가하는 4가지의 경우가 존재한다. 이들 총 64가지의 상황을 <표 4>에 정리하였다. 한가지 주목할 점은 원래 실험에서는 뷰의 개수를 5개부터 20개까지의 16경우 모두에 대한 결과를 얻었지만 (그림 4)와 (그림 5)의 그래프에서는 결과를 단순화시켜 보기 쉽도록 하기 위해서 4가지의 중요한 경우에 대한 것만 고려하여 총 64(=4×4×4)가지의 상황에 대한 결과를 보여준다.



(그림 7) 유지 비용 제한에 대한 ITG, A*의 총 유지 비용의 비율의 평균

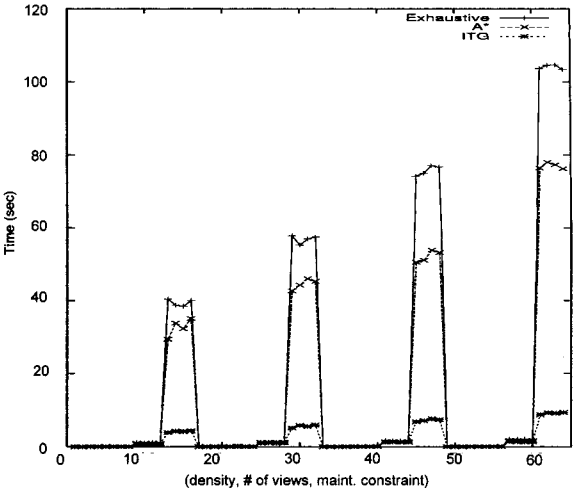
(그림 6)은 유지비용 제한에 대한 GA의 총 유지 비용의 비율의 평균을 보여주고 있다. 그림에서 LN-D와 LN-SD는 여전히 유지 비용의 기준에 대해 약 380%의 변동폭을 가지고 있음을 알 수 있다. 이러한 경향은 특히 저밀도 OR 뷰 그래프들에서 두드러지는데 이것은 벌칙 값이 작아서 유지 비용 제한을 지키기에 불충분하기 때문인 것으로 보인다. 이 두 방법을 고려의 대상에서 제외하고 나면, (그림 4)에서 남은 EX-D와 EX-SD 방법이 총 질의 비용의 비율이 항상 90% 이상이며 110% 이하를 유지하여 최적 해결책에 매우 가까운 결과를 보인다는 것을 알 수 있다. 그리고, 유지 비용은 항상 유지 비용 제한 값의 2배수 안의 범위에 있다. 그러므로 EX-D와 EX-SD는 유지 비용 뷰 선택을 위한 유전자 알고리즘에 알맞는 적합성 함수임을 알 수 있다. 이러한 결과는 Gupta와 Mumick[13]의 반전-트리 욕심쟁이 휴리스틱에서 얻은 결과와 비교해볼 때 해결책의 질적인 면에서 매우 유사하다. 이들의 휴리스틱에서 얻은 해결책의 유지 비용은 제한 값의 2배 이내임을 이론적으로 증명하였으며 최적 해결책의 질의 이익의 63% 이내의 해

결책을 제공하였다. (그림 7)에서 반전-트리 욕심쟁이 알고리즘(ITG)의 유지 비용이 제한 값의 200% 이내임을 보이고 있고 A*는 항상 유지비용 제한 값을 만족하는 최적의 해를 찾으므로 언제나 100%라는 결과를 얻는다. 또한 (그림 5)에서 반전-트리 욕심쟁이 알고리즘은 총 질의 비용에 대한 최적의 질의 비용의 비율이 항상 80% 이상임을 보여 유전자 알고리즘의 90% 이상과 비교해서 우수하지 못하며 A* 알고리즘은 항상 최적의 해를 찾아서 100%의 비율을 보임을 알 수 있다.

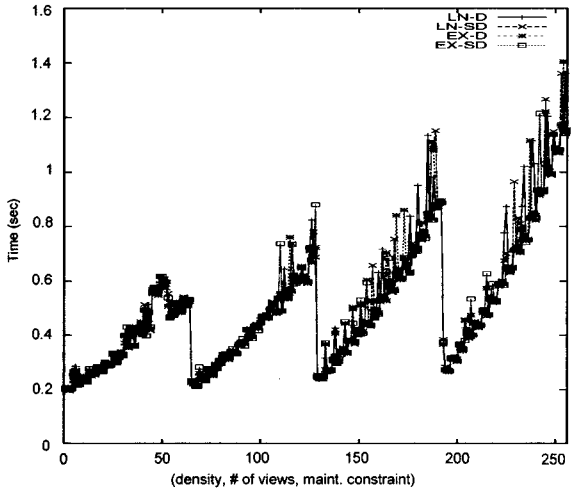
5.2 실행 시간

(그림 8)과 (그림 9)는 전체(exhaustive) 탐색 알고리즘, A*, 반전-트리 욕심쟁이 알고리즘과 유전자 알고리즘의 실행시간의 평균을 보여준다. (그림 8)에 보인 전체 탐색 알고리즘은 원래 유전자 알고리즘이 제공하는 해결책의 질을 측정하는 벤치마크로서 사용하기 위해 개발되었다. 이 알고리즘을 사용함으로써 20개 이상의 뷰가 존재할 경우, 최적의 해결책을 찾는다는 것은 극단적으로 시간 낭비임이 증명되었다. 따라서 본 연구에서는 20개의 뷰로 실험을 한정했다. 비록 더 낮은 휴리스틱 방법들이 존재하기는 하지만 (그리고 이들 역시 다항적인 시간 복잡도를 갖기는 하지만), 이 특정 실험은 성능에 대한 기본적인 이해를 하기 위한 것이다.

그림을 보면 각각의 밀도 내에서 뷰의 갯수가 증가함에 따라 전체 탐색 알고리즘의 실행 시간이 지수적으로 증가함을 알 수 있다. A*도 지수적인 증가와 비슷한 형태를 보이며 반전-트리 욕심쟁이 알고리즘은 다항적인 시간 복잡도를 갖는다[12, 13]. 본 논문의 제안한 유전자 알고리즘의 실행 시간은 이에 반해서 (그림 9)에 보인 것과 같이 선형적인 증가를 보인다. (그림 9)에서는 모든 256가지의 상황에 대한 데이터를 포함한다. 뷰의 갯수가 작은 경우(5이하), 유전자 알고리즘이 필요로 하는 추가적인 작업들은 전체 탐색을 사용하는 것보다 느려지도록 한다. 하지만 유전자 알고리즘의 강점은 웨어하우스 내의 뷰의 수가 클 경우 확연해진다. 간선 밀도가 증가함에 따라, 유전자 알고리즘의 실행 시간을 나타내는 선형 그래프의 기울기는 단지 조금밖에 증가하지 않는다. 75%의 간선 밀도와 20개의 뷰를 가진 OR뷰 그래프에서 구체화된 뷰를 선택하기 위해서 유전자 알고리즘이 사용하는 시간은 전체 탐색 기법에 의해 사용된 시간의 80분의 1정도이다. 뷰의 수가 30이나 그 이상으로 감에 따라, 이보다 더 큰 차이가 예상된다. 그럼에도 불구하고 여전히 유전자 알고리즘에 의해 생성되는 해결책은 최적에 매우 가깝다.



(그림 8) 전체(Exhaustive) 탐색, A*, 반전트리 욕심쟁이(ITG) 알고리즘의 실행시간



(그림 9) 유전자 알고리즘의 실행시간

〈표 5〉 유전자 알고리즘의 9가지 전략에 대한 비교.

전 략	벌칙함수	벌칙모드	최적에 대한 질의비용	유지비용 제한	실행시간
LG-S	로 그	델 셈	최상(모든 뷰를 구체화)	제한을 무시	선형증가
LG-D	로 그	나눗셈	60~110%	약 10000%	선형증가
LG-SD	로 그	델셈&나눗셈	최상(모든 뷰를 구체화)	제한을 무시	선형증가
LN-S	선 형	델 셈	60~110%	약 10000%	선형증가
LN-D	선 형	나눗셈	90~110%	약 380%	선형증가
LN-SD	선 형	델셈&나눗셈	90~110%	약 380%	선형증가
EX-S	지 수	델 셈	아주 좋음	10000%	선형증가
EX-D	지 수	나눗셈	90~110%	200% 이내	선형증가
EX-SD	지 수	델셈&나눗셈	90~110%	200% 이내	선형증가

6. 결 론

본 연구에서는 유전자 알고리즘이 OR 뷰 그래프를 이용한 유지 비용 뷰 선택의 문제에 대해 기존의 방법들보다 매우 우수한 성능을 가짐을 보였다. 특별히 본 연구의 유전자 알고리즘은 선형 증가하는 실행 시간을 보이면서도 최적의 해결책의 질의 비용의 90에서 110% 이내의 해결책을 항상 제공한다. 실험에서 20개까지의 뷰들에 대해서 전체 탐색 알고리즘을 수행하고 비교함으로써 이를 입증하였다. 적합성 함수에는 벌칙 함수를 포함시켰으며 9가지의 조합으로 구성된 적합성 함수들을 가지고 실험한 결과, EX-D와 EX-SD 방법들이 유지비용 뷰 선택 문제에 대해서 최상의 결과를 제공함을 알 수 있었다. <표 5>에 이들 9가지의 전략을 비교하여 정리하였다. 본 알고리즘은 웨어하우스의 진화, 특히 많은 뷰들을 포함하며 웨어하우스의 질의들이 자주 변하는 경우에 매우 중요한 도구가 되리라고 본다.

추후로 연구할 사항들은 다음과 같다. 첫째, 최초 개체군을 생성할 때 무작위로 하기보다는 문제에 대한 지식을 기반으로 최초 개체군을 생성한다. 이것은 알고리즘이 더욱 빠르게 원하는 해결책을 찾도록 해줄 것이다. 둘째, 400번의 세대를 모두 계산하기 보다는 해결책이 특정 범위 안에 들 경우에 알고리즘을 중단시키는 좀더 융통성 있는 종료 조건을 개발한다. 셋째, 범위를 넓혀서 AND-OR 뷰 그래프와 인덱스도 고려하도록 한다. 이것은 새로운 비용 모델이 필요할 뿐 아니라 해결책을 표현하는 방법도 변경해야 한다. 넷째, 유전자 알고리즘은 병렬화를 하기에 매우 좋다. 본 알고리즘의 병렬 버전을 개발하여 성능을 향상시키는 것도 가능하다.

참 고 문 헌

[1] W. H. Inmon and C. Kelley, *Rdb/VMS : Developing the Data Warehouse*, QED Publishing Group, Boston, London, Toronto, 1993.

[2] J. Widom, Research Problems in Data Warehousing, in *Proceedings of the Fourth International Conference on Information and Knowledge Management*, Baltimore, Maryland, pp.25-30, 1995.

[3] N. Roussopoulos, Materialized Views and Data Warehouses, in *Proceedings of the Workshop on Knowledge Representation meets Databases(KRDB)*, 12.1-12.6, Athens, Greece, 1997.

[4] A. Gupta and I. S. Mumick, Maintenance of Materialized Views : Problems, Techniques, and Applications, *Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing*, Vol.18, No.2, pp.3-18, 1995.

[5] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom, View Maintenance in a Warehousing Environment, *SIGMOD Record(ACM Special Interest Group on Management of Data)* Vol.24, No.2, 316-27, 1995.

[6] H. Gupta, Selection of Views to Materialize in a Data Warehouse, in *Proceedings of the International Conference on Database Theory*, Delphi, Greece, pp. 98-112, 1997.

[7] D. Theodoratos and T. K. Sellis, Data Warehouse Configuration, in *Proceedings of the Twenty-third International Conference on Very Large Databases*, Athens, Greece, pp.126-135, 1997.

[8] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, Mass, 1989.

[9] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Sringer-Verlag, NewYork, NY., 1994.

[10] N. Roussopoulos, View Indexing in relational Databases, *ACM Transactions on Database Systems* Vol.7, No.2, pp.258-290, 1982.

[11] K. A. Ross, D. Srivastava, and S. Sudarshan, Materialized view maintenance and integrity constraint checking : Tradingspace for time, *SIGMOD Record (ACM Special Interest Group on Management of Data)*, Vol.25, No.2, pp.447-458, 1996.

[12] W. Labio, D. Quass and B. Adelberg, Physical Database Design for Data Warehouses, in *Proceedings of the International Conference on Database Engineering*, Birmingham, England, pp.277-288, 1997.

[13] H. Gupta and I. Mumick, Selection of Views to Materialize Under a Maintenance Cost Constraint, in *Proceedings of the International Conference on Management of Data*, Jerusalem, Israel, pp.453-470, 1999.

[14] A. Swami, Optimization of large join queries : combining heuristics and combinational techniques, *SIGMOD Record*, Vol.18, No.2, pp.367-76, 1989.

[15] S. Augier, G. Venturini and Y. Kodratoff, Learning First Order Logic Rules with a Genetic Algorithm, in *Proceedings of the First International Conference on Knowledge Discovery and Data Mining(KDD-95)*, Montreal, Canada, pp.21-26, 1995.

[16] I. W. Flockhart and N. J. Radcliffe, A Genetic Alg-

orithm-based Approach to Data Mining, in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining(KDD-96)*, Portland, Oregon, pp.299-302, 1997.

[17] S. A. Cook, The Complexity of Theorem Proving Procedure, *Annual ACM SIGACT Symposium on Theory of Computing*, Vol.3, pp.151-158, 1971.

[18] M. R. Garey and D. S. Johnson, *Computers and Intractability-A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[19] E. H. L. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines*, John Wiley, Chichester, UK, 1989.

[20] P. J. M. v. Laarhoven and E. H. L. Aarts, *Simulated Annealing : Theory and Applications*, Kluwer, Dordrecht, Holland, 1987.

[21] MIT Technology Lab, GAlib : A C++ Library of Genetic Algorithm Components, URL, http://lancet.mit.edu/ga/.

이 민 수

e-mail : mlee@ewha.ac.kr

1992년 서울대학교 컴퓨터공학과 학사

1995년 서울대학교 대학원 컴퓨터공학과 공학석사

1995년~1996년 LG전자 미디어통신연구소 연구원

2000년 University of Florida 컴퓨터공학과 공학박사

2000년~2002년 미국 Oracle Corporation, Senior Member of Technical Staff

2002년~현재 이화여자대학교 컴퓨터학과 조교수

관심분야 : 데이타웨어하우스, 지식기반 시스템, 웹 데이타베이스