0/1 Knapsack에 대한 서브-지수 함수 알고리즘

이충세*

요 약

이 논문에서는 고정된 개수를 가진 bin들을 이용하여 실행 복잡도가 $p(n).2^{O(\sqrt{n})}$ 인 알고리즘을 제시한다, 여기서 x는 ⑤n개의 객체들에 대한 리스트의 길이에 대한 총 비트 수를 나타낸다. 이러한 방법은 수치적 크기나 비중의 합의 리스트를 이용하는 여러 가지 최적화 알고리즘이나 결정 문제등에 적용할 수 있다. 이 논문에서 제시한 알고리즘은 의사-다항식(pseudo-polynomial) 시간을 갖는 NP-Complete의 많은 문제들을 결정적인 서브-지수 시간에 해결할 수 있은 가능성을 제시한다. 여기서 제시한 알고리즘을 이용하여 생명공학의 유전자 분석에 적용하려고 한다.

Sub-Exponential Algorithm for 0/1 Knapsack

Chung Sei Rhee*

Abstract

We investigate $p(n).2^{O(\sqrt{n})}$ algorithm for 0/1 knapsack problem where x is the total bit length of a list of sizes of n objects. The algorithm is adaptable of method that achieves a similar complexity for the partition and Subset Sum problem. The method can be applied to other optimization or decision problem based on a list of numerics sizes or weights. 0/1 knapsack problem can be used to solve NP-Complete Problems with pseudo-polynomial time algorithm. We try to apply this technique to bio-informatics problem which has pseudo-polynomial time complexity.

Key words : knapsack, NP, sub-exponential, algorithm, bio-informatics, Partition, Pseudo-Polynomial

접수일(2014년 11월 28일), 수정일(1차: 2014년 12월 24일, 게재확정일(2014년 12월 31일)

* 충북대학교 소프트웨어학과

1. 서 론

전산학 분야에서 NP-Complete 문제는 실행-복잡도를 구할 경우에 실시간 처리가 어려울 경우 자주 이용되는 분야이다. NP-완비(NP-Complete)는 비-결정적(Non-determinisic) Algorithm의 약어로 어떤 문제의 해를 구할 때 결정적이 아닌 비결정적, 즉 해가 집합의 형태로 여러 개가 존재하는 것을 나타낸다. NP-완비는 어떤 문제가 NP이고 어떠한 다항 시간의 알고리즘도 발견되지 않은 문제를 나타낸다. 이러한 문제들에 대하여 $P \neq NP$ 라는 질문이 1971년에 처음 나타난 이후에 컴퓨터 분야의 깊고 복잡한 미해결 연구 주제의 하나이다. NP-Complete 속하는 문제들은 Clique, Satisfiability, Vertex Cover, 그리고 Sum of Subset등이 존재한다.[1] 요즈음, 생명공학 등의 분야에서 컴퓨터를 이용하여 유전자(Gene)를 분석하는 연구를 많이 수행하고 있다. 그러나 대부분의 경우, 유전자 분석 알고리즘은 NP-Complete에 속하는 어려운 문제들을 포함하고 있다.

1990년에 Stearns와 Hunt[2]는 파워 인덱스 i을 갖는 문제를 해결하는데 가장 빠른 알고리즘에 소요되는 과정에 걸리는 시간이 $2^{O(n)}$인 문제들을 분류하였다. Stearns 등은 Satisfiability 문제가 파워 인덱스 1을 갖는다고 가정하면, Clique와 Partition 문제는 파워 인덱스 1/2를 갖는다고 주장하였다. 이들의 분석은 시간 복잡도 $p(n).2^{O(\sqrt{n})}$ 을 갖는 두 가지 알고리즘에 기초를 두고 있는데, 여기서 x는 입력을 표현할 때에 필요한 비트의 길이를 그리고 $p(n)$은 Clique의 경우에는 그래프 상에서의 간선들의 개수를 그리고 Partition의 경우에는 입력 집합의 개수의 다항식을 나타낸다. 이러한 결과는 Clique나 Partition 문제가 Satisfiability나 다른 NP-Complete 문제보다 훨씬 쉬운 문제임을 보이고 있다.

Impagliazzo 등은[3] NP-완비 문제들의 비교하기 위한 다른 방안을 제시하였다. Stearns 등의 파워 인덱스를 채택하는 대신에 이들은 약한 지수 함수나 $(2^{n^{\Omega(1)}})$ 또는 강한 지수 함수$(2^{\Omega(n)})$을 하한으로 하는 방안에 기초하여 문제들을 분류하였다. Stearn 등은 Satisfiability 문제는 언제가 강한 지수를 갖는 문제로 증명될 것이라고 가정하였다. 만약 각 상수 $\epsilon > 0$ 에 대하여 $M(x)$가 어떤 다항식 함수 p에 대하여 기껏해야 $p(n).2^{O(\sqrt{n})}$ 의 실행시간을 갖는다고 가정하면 알고리즘 $M(x)$는 부분-지수(sub-exponential) 시간 복잡도를 갖는다고 생각할 수 있다.

최근의 보다 복잡도 연구에서는 Subset Sum에 속하는 문제들을 전형적인 입력의 길이를 인수로 사용하지 않는 연구가 수행되었다. Subset Sum이나 0/1 Knapsack의 상한은 리스트에 있는 객체의 수를 복잡도로 사용할 경우에 $2^{O(n/2)}$가 된다.[8] 배낭-등(Knapsack)에 대한 하한 $\Omega(2^{n/2}/\sqrt{n})$이 [2]에서 상세히 설명되어 있다.

2. 분할과 부분 집합의 합

분할(Partition)과 부분집합의 합(Subset Sum)을 이용하여 유전자 분석에 적용할 수 있다. 먼저, 분할 문제는 다음과 같이 정의할 수 있다. 주어진 양의 정수들의 집합 S에 대하여 두 개의 부분집합이 같은 합을 갖는지를 결정한다. 부분집합의 합은 비슷하지만 보다 일반적인 특징을 갖는다. 주어진 양의 정수들의 집합 S와 목표가 되는 정수 t에 대하여 S의 어떤 부분집합이 합 t을 갖는지를 결정한다. 분할을 부분집합의 합으로 변형할 수 있다.[1]

부분집합의 합에 대한 알고리즘을 구현하려면 연속적으로 모든 부분집합을 생성하고 이들의 합을 계산하는 지수-시간의 많은 검색을 수행해야 한다. 집합 S가 n개의 원소들을 갖고 있으면 2^n개의 부분집합이 존재하고, 검색 알고리즘에 걸리는 시간은 어떤 다항식 $p(n)$에 대하여 $O(p(n).2^n)$이 소요된다. 다른 많은 NP-Complete 문제와 마찬가지로, 검색은 검색 크기의 공간을 줄이기 위하여 여러 가지 경계조건을 가지고 깊이-우선-검색을(Depth-First-Search) 사용하여 실행할 수 있다.

부분집합의 합을 구하는 문제는 동적 프로그램을 이용하여 지수 시간이 아닌 실시간에 처리할 수가 있다. 이러한 문제는 목표가 되는 합 t보다 작거나 같

은 양의 합 k에 대하여 S가 합 k을 갖는 부분집합을 결정하는 문제가 된다. 이러한 문제는 보통 불리언 배열을 이용하여 동적프로그램으로 해결할 수 있다. 또한 이러한 알고리즘에 필요한 과정의 수는 $n.t$가 된다. 간단한 경우가 아닌 값 t는 S에 속하는 원소들의 합보다 작아야 되고 따라서 배열의 길이는 S에 속하는 최대값 m에 대하여 $n.m$ 초과할 필요가 없다. 만일 m이 $O(n)$이면 알고리즘에 걸리는 시간은 $O(n^3)$가 되고 이러한 문제의 복잡한 경우에는 $m = O(2^n)$이 된다.[6, 7] 이 경우의 실행 시간은 $O(n.(2^n)^2)$이 되는데, 많은 NP-Complete 문제들은 이와 비슷한 분석 결과를 갖게 된다.

3. 일반화된 동적 할당에 의한 동적 프로그래밍

분할에 대한 Stearns와 Hunt의 알고리즘은 동적프로그램과 backtracking을 결합하여 구현하였다. 이러한 복합적인 접근 방법은 종종 operation research에서 사용되기도 한다. 입력 집합은 순서화하고 밀집한(denser) 집합과 희소(sparse) 집합으로 분할한다. 희소 부분집합에는 backtracking을 적용하고 반면에 밀집 부분집합에는 동적프로그램을 적용한다. 이러한 결과들을 결합하여 시간 복잡도 $2^{O(\sqrt{n})}$의 해를 얻을 수 있는데, 여기서 x는 입력에 있는 비트들의 총 길이를 나타낸다.

유전자를 분석하기 위하여 앞에서 소개한 알고리즘을 이용한다. 이러한 접근 알고리즘은 Subset Sum과 Partition에서 처음으로 도입되었다. 일반적인 동적 프로그램과 마찬가지로 부분적 해들은 너비 우선(breadth-first)으로 나열하여 표현한다. 프로그램은 객체들의 리스트로 나타내는데, 각 객체는 크기를 가지고 있다. 객체들의 리스트는 크기에 의해 순서화되어 있고, 가장 큰 객체를 먼저 처리한다. 일반적인 동적 프로그램과는 다르게 해들의 모임은 동적으로 할당된다. 해들의 모임은 처음에는 커지다가 더 많은 객체들을 처리함에 따라 줄어든다. 각 객체에 대하여 해들의 전체 모임을 조사하고, 해들의 남은 용량으로부터 가능한 현재 객체의 크기를 제거함으로써 각각의 해를 갱신한다. 각각의 해는 처리해야할 객체들의 크기의 합에 따라 상대적으로 계산한다. 각각의 해는 또한 처리해야 할 객체들의 크기의 합과 연관하여 계산한다. 남아 있는 크기들의 합을 이용하여 문제의 의미에 종속하는 해들의 풀(pool)을 제거할 수 있다.

4. 배낭-등(Knapsack) 문제

0/1 배낭-등 문제는 다음과 같이 정의된다. 주어진 n개의 객체들에 대하여 크기가 $s[1..n]$인 S와 값 $v[1..n]$에 대하여 크기가 C보다 작거나 같은 경우에 객체들의 부분 집합들을 구하여 최적이 되는 해를 구하는 문제가 된다. 이러한 문제는 결정 문제로도 표현할 수 있는데, 목표가 되는 V보다 작거나 같은 부분집합이 구하여 최적이 되는 지를 결정하는데, 이러한 문제는 경영학이나 최적화 알고리즘에서 어떤 문제의 최적의 해를 구하는 문제와 관련되어 있다. 배낭-등 문제를 다음과 같은 알고리즘으로 구현한다. 다음 절에서는 이러한 알고리즘에 걸리는 시간을 보인다. 다음 프로시저를 이용하여 NP-완비 시간이 걸리는 문제를 서브-지수 알고리즘, 즉 의사 다항식 알고리즘에 해결할 수 있음을 보인다. 이것은 실시간에 처리할 수 없는 문제를 의사 실시간에 처리하는 방법을 찾는 것이 된다.

```
procedure Knapsack( )
{
    best_value := 0;
    size_of_rest :=0;
      for i:=0 to n do
        size_of_rest += s[i];
      value_of_rest :=0;
      for i := 0 to n do
        value_of_rest += v[i];
    Pool := {(C, 0)};
    for i:=1 to n do
    {
        size := s[n-i+1];
```

```
    value := v[n-i+1];
    new_list := { };
      for each sack in POOL do
      {
        if (sack. capacity < size) then
          continue
          else if
            (sack.capacity > size_of_rest)
            bestval :=
              max(bestvalue.sack.value +
              value_of_rest)
              remove sack from POOL;
        else
        bestval := max(bestvalue, sack.value
                + value);
        NewList.append((sack.capacity - size,
                sack.value+value));
      end for
    size_of_rest := size_of_rest - size;
    value_of_rest := value_of_rest - value;
    Pol := merge(Pool, Newlist);
  end for
  return bestval;
}
```

배낭-등 문제의 분석

정리 1 : 앞에서 보인 배낭-등 문제는 어떤 다항식 $p(n)$에 대하여 $O(p(n)2^{\sqrt{b}})$의 시간이 걸린다.

증명 : $S=\{y_1,y_2,...,y_n\}$에 대하여 크기는 비-감소 형태로 저장되어 있다고 가정하자.($s[i] \leqq s[i+1]$) 총 과정의 수는 Pool의 크기에 의해 결정된다. 외부 for 루프의 각각의 반복문에 대하여, Pool을 조사하고 가능하면 확장하는데, 이러한 과정에 두 개의 패스가 필요한데 하나는 for 루프에 대한 내부 반복문이고 다른 하나는 순차적인 병합 과정이다. 총 작업의 양은 $\sum_{i=1}^{n}|Pool(i)|$을 이용하여 측정할 수 있는데, 여기서 $|Pool(i)|$는 바깥 반복문에서 i가 시작할 때의 $Pool$의 길이를 나타낸다. merge 연산이 용량들의 중복을 제거하기 때문에 $Pool(i)$의 길이를 최대 $MaxCapacity(i)$로 나타낼 수 있는데, 이것은 리스트에서 반복문 i가 시작할 때의 임의의 부대(sack)의 최대 용량을 나타낸다. 0과 최대값 사이에 있는 모든 용량이 존재하지 않기 때문에 리스트는 실제에 있어 이러한 값보다 작다. 또한 각 루프를 반복하는 동안 리스트의 길이는 많아야 2배가 되기 때문에 리스트에 있는 최대값에 관계없이 리스트의 길이는 2^i을 초과할 수 없다. 따라서 다음의 결과를 얻는다.

$$|Pool(i)| \leqq \min(2^i, MaxCapacity(i))$$

$Pool$의 길이는 급격히 커지지만 뒤에 i가 n에 접근하면서 줄어든다. $MaxCapacity(i)$에 대한 상한을 다음과 같이 구한다. 초기에 $MaxCapacity(1)=C$가 되는데, 이것은 초기 sack의 용량이 된다. 작용량의 sack 만이 리스트에 첨부되고 최종적으로 보다 큰 용량의 sack 들은 제거되기 때문에 다음의 수식을 얻는다.

$$MaxCapacity(i) \leqq \sum_{i=1}^{n}\min(2^i, MaxCapacity(i))$$
$$\leqq n.2^n \leqq n.2^{\sqrt{b}}$$

$MaxCapacity(i)$의 경계를 구하는 문제는 $s[n-i+1]$의 상한을 구하는 문제로 제한된다. 이러한 분석을 수행하기 위하여 과정의 수를 배열 s의 총 비트 수를 나타내는 b의 함수로 표현한다. 두 가지를 경우를 고려하자.

경우 1. $n \leqq \sqrt{b}$. 이 경우에 분석은 다음과 같다.

$$\sum_{i=1}^{n}|Pool(i)| \leqq \sum_{i=1}^{n}\min(2^i, MaxCapacity(i))$$
$$\leqq n.2^n \leqq n.2^{\sqrt{b}}$$

경우 2. $n > \sqrt{b}$. 이 경우에는 $i=\sqrt{b}$일 때 합을

분할한다.

$$\sum_{i=1}^{n}|Pool(i)| \leqq \sum_{i=1}^{n}\min(2^i, Max\,Capacity(i))$$
$$\leqq \sum_{i=1}^{\sqrt{b}-1}\min(2^i, Max\,Capacity(i)) + \sum_{i=\sqrt{b}}^{n}\min(2^i, Max\,Capacity(i))$$

$$\leqq (\sqrt{b}-1).2^{\sqrt{n}-1} + \sum_{i=\sqrt{b}}^{n}\min((2^i, Max\,Capacity(i))$$

$$\leqq (\sqrt{b}-1)(2^{\sqrt{b}-1}+(n-\sqrt{b}+1).Max\,Capacity(\sqrt{b})$$
$$\leqq (\sqrt{b}-1).2^{\sqrt{b}-1} + (n-\sqrt{b}+1)\sum_{i=1}^{n-\sqrt{b}+1} s[j]$$
$$\leqq (\sqrt{b}-1).2^{\sqrt{b}-1} + (n-\sqrt{b}+1)s[n\sqrt{b}+1]$$
$$\leqq (\sqrt{b}-1).2^{\sqrt{b}-1} + (n-\sqrt{b}+1)\,s[n\sqrt{b}+1]$$

위의 식에서 $k=n-\sqrt{b}+1$인 $s[k]$의 경계값을 구하기 위하여 $s[n-\sqrt{b}+1]$을 $2^{\sqrt{b}+1}$로 대입하면 수식은

$$< (\sqrt{b}-1).2^{\sqrt{b}-1}+(n-\sqrt{b}+1)^2.2^{\sqrt{b}+1}$$
$$< (\sqrt{b}+(n+1)^2).2^{\sqrt{b}+1}$$
$$< (2n^2+6n+2).2^{\sqrt{b}}.$$

이러한 결과는 배낭-등에 대한 시간 복잡도가 어떤 다항식 $p(n)$에 대하여 $O(p(n)2^{\sqrt{b}})$가 되는 것을 보여준다.

5.. Bio-Brick과 Minimum Genome의 설계

생물정보학의 발전과 소프트웨어 기술의 발전을 활용하는 합성생물학 분야의 연구가 최근 많이 진행되고 있다. 합성생물학은 컴퓨터 분야의 발전에 비교할 때, 아직도 초기 단계의 상황에 처해 있지만 앞으로 의료 분야에 신약개발 분야 등에 지대한 영향을 미칠 것으로 기대되고 있다.

필수 유전자는 생명체가 살아가는데 필요한 유전자를 나타내고 모든 생명체는 모두 필수 유전자를 가지고 있다. 최소유전체는 필수 유전자로 구성된 유전체이고 이를 설계하는데 오류를 최소화하고 또한 효율을 최대화해야 한다. 합성생물학은 범위가 방대하고 다양한 응용분야를 가지고 있다.

현재 초기 단계의 연구가 이루어지고 있는 합성생물학 분야에서 필수유전자들을 이용하여 최소유전체를 설계하기 위해 필요한 유전체의 특성에 대한 연구가 필요하다. 특히 합성생물학을 이용하여 연쇄상구균에서 필수 유전자와 특이 유전자를 유추하기 위하여 데이터베이스를 구축하고 또한 유전체들의 특성을 분석하여 타당성을 입증하기 위하여 통계적 신뢰도 검사를 수행하는 것이 필요하다. 현재 시행하고 있는 연구에서는 필수 유전자 478개로 이러한 유전자에서 필수유전자를 유추하여 유전체 서열을 밝혀, 밝혀진 모든 박테리아의 필수 유전자를 생명정보학적으로 분석하는데 활용하려고 한다.

현재, 연구가 수작업으로 이루어지고 있기 때문에 동적 프로그래밍 기법을 활용하면 시간이 많이 단축될 것으로 기대하는데, 이 논문에서 다룬 내용을 변형하면 실시간 처리가 어려운 유전자 분석에 분할이나 배낭-등 방법을 적용하여 유전자를 효율적으로 분할하여 검색 시간을 주일 수 있을 것으로 기대한다. 이러한 생명 공학에 적용하려는 기초 연구를 수행하고 있다.

6. 결 론

이 논문에서 제시한 0/1 knapsack은 동족인 분할을 이용한 동적 알고리즘을 이용하여 구현하였다. 이러한 알고리즘에 걸리는 실행 시간 복잡도는 $p(n).2^{O(\sqrt{n})}$ 이 된다. 여기서 x는 n개의 객체들에 대한 리스트의 길이에 대한 총 비트 수를 나타낸다. 이러한 방법은 수치적 크기나 비중의 합의 리스트를

이용하는 여러 가지 최적화 알고리즘이나 결정문제등에 적용할 수 있다. 앞으로, 여기서 제시한 알고리즘을 생명공학의 유전자 분석에 적용하려고 한다.

참고문헌

[1] M. Alehnovich, A. Borodin, J. Buresh-Oppenheim, "Toward a Model for Backtracking and Dynamic Programming,", Proc of the 20th Annual IEEE Conference n Computational Complexity, pp. 308-322, 2005.[2] M Garey and D. Johnson, Computers and

Intractability : A Guide to the Theory of NP Completenes, Freeman Press, San Fransisco, CA,

1979.

[3] R. Impagliazzo, R. Paturi and F. Zane, " Which Problems have Strongly Exponential Complexity?.", Journal of Computer and System Sciences 63, pp. 512-530, Elsevier Science 2001.

[4] S. Martello and P. Toth, "A mixture of Dynamic programming and branch-and-bound for subset sum problem,", Management Science 30(6), pp. 765-771, 1984.

[5] R. Stearns and H. Hunt, "Power indices and Easier Hard Problems", Mathematical System Thoery 23, pp. 209-225, 1990.

[6] G. J. Woeginger, "Exact Algorithms for NP-Hard Problems: A Survey,", Lecture Notes in Computer Science 2570, pp. 185-207, Spring-Verlaugh, Berlin, 2003.

[저 자 소 개]

이 충 세 (Chung Sei Rhee)

1973년 수학과 학사
1979년 8월 University of South Carolina 석사
1989년 8월 University of South Carolina 박사

email : csrhee@cbnu.ac.kr