Chap 9. An Intro to The Theory of NP

1. The Three General Categories of Problems
2. The Theory of NP

General Categories of Problems

- 문제의 일반적인 분류
  1. 다차시간 알고리즘을 찾은 문제
  2. 다루기 힘들다고 증명된 문제
  3. 다루기 힘들다고 증명되지 않았고,
     다차시간 알고리즘도 찾지 못한 문제

General Categories of Problems

1. 다차시간 알고리즘을 찾은 문제

- 예제
  - 정렬 문제: $\Theta(n \lg n)$
  - 정렬된 배열 검색 문제: $\Theta(\lg n)$
  - 행렬곱셈 문제: $\Theta(n^{2.38})$

- 무작정 알고리즘은 지수시간이나, 다차시간 알고리즘이 개발된 경우
  - 연쇄행렬곱셈 문제
  - 최단경로 문제
  - 최소비용신장트리(Minimum spanning tree) 문제

General Categories of Problems

2. 다루기 힘들다고 증명된 문제

- 비다항식(nonpolynomial) 크기의 결과를 요구하는 비현실적인 문제
  - 보기: 모든 해밀토니안 회로를 출력하는 문제
    - 만일 모든 정점들 간에 이음선이 있다면, (n-1)!개의 답을 얻어야 한다.
    - 이러한 문제는 하나의 해밀토니안 회로를 구하는 문제에 비해서 필요이상으로 많은 답을 요구하므로 사실상 비현실적이고, 다루기 힘든 문제로 분류된다.

- 요구가 현실적이나, 다차시간에 풀 수 없다고 **증명된** 문제
  - 놀랍게도 이런 부류에 속하는 문제는 상대적으로 별로 없다.
  - 결정불가능한 문제(Undecidable Problem)
    - 그 문제를 풀 수 있는 알고리즘이 존재할 수 없다고 증명이 된 문제
    - 예: 종료 문제(Halting Problem) 등
  - 프레스버거 산술(Presburger Arithmetic) 문제
    - Fischer와 Rabin에 의하여 증명됨(1974)

General Categories of Problems

- 종료 문제 (Halting Problem)
  - 주어진 (알고리즘 A,입력 I)쌍에 대해서 A는 입력 I에 대해서 Halt (Yes 아니면 No로 답하면서 종료되는 상태) 하게 될지를 결정하는 문제
    - 1936년 Alan Turing에 의해서 결정불가능문제임이 증명됨

  - **정리**: 종료 문제는 결정불가능하다.
  - **증명**:
    - 이 문제를 풀 수 있는 "Halt"라는 알고리즘이 존재한다고 가정하자.
    - 즉, 이 알고리즘은 (A,I)를 입력으로 받아서
      그 프로그램이 종료하면 "예"라는 답을 주고,
      종료하지 않으면 "아니오"라는 답을 줄 것이다.
    - 그러면 우리는 다음과 같은 "말도안돼" 알고리즘을 만들 수 있다.

General Categories of Problems

```
Algorithm 말도안돼
  if (Halt(말도안돼) == "예")
      while true
        do print "야호"
```

(1) 만일 "말도안돼" 알고리즘이 정상적으로 종료하는 알고리즘이라고 한다면,
Halt(말도안돼)는 "예"가 되고,
따라서 이 알고리즘은 절대로 종료하지 않는다.
이는 가정과 상반된다.

(2) 만일 "말도안돼" 알고리즘이 정상적으로 종료하지 않는 알고리즘이라면,
Halt(말도안돼)는 "아니오"가 되고,
따라서 이 알고리즘은 종료하게 된다.
이도 마찬가지로 가정과 상반된다.

- 결론적으로, "Halt"라는 알고리즘은 존재할 수 없다.

General Categories of Problems

3. 다루기 힘들다고 증명되지도 않았고,
   다차시간 알고리즘도 찾지 못한 문제

- 많은 문제들이 이 카테고리에 속한다.
  - 0-1 배낭채우기 문제
  - 외판원 문제
  - m-색칠하기 문제 ($m \geq 3$)
  - 해밀토니안 회로 문제 등

- 이러한 문제들은 NP (Nondeterministic Polynomial) 문제에 속한다.

Optimization Problem vs Decision Problem

- Optimization problem (최적화 문제)
  - 최적의 해를 찾는 문제
  - TSP(G): 그래프 G의 최단 일주거리는?
  - Graph-Coloring(G): 그래프 G의 꼭지점들은 최소 몇 개의 색깔로 칠할 수 있는가? (이웃한 꼭지점은 다른 색깔로 칠해져야 한다.)
  - Clique(G): 그래프 G의 최대 Clique 크기는 얼마인가?
    - Clique: 모든 꼭지점간의 연결선이 G에 있는, G의 꼭지점 부분집합
    - Clique의 크기: Clique의 원소(꼭지점) 개수
- Decision problem (결정 문제)
  - 대답이 "예" 또는 "아니오"로 이루어지는 문제
  - TSP-Decision(G,d): 그래프 G에 거리 d 이하의 최단 일주경로가 있는가?
  - Graph-Coloring-Decision(G,d): 그래프 G의 꼭지점들은 d 이하의 색깔로 칠할 수 있는가?
  - Clique(G,d): 그래프 G에는 크기 d 이상의 Clique가 존재하는가?

Optimization Problem vs Decision Problem

- 최적화 문제에 대해서 다차시간 알고리즘이 있다면, 그 문제에 대응하는 결정 문제에 대한 다차시간 알고리즘은 있다.
  - TSP(G)를 다차시간에 풀 수 있다면, TSP-Decision(G,r)에 대해서도 다차시간에 Yes/No로 답할 수 있는데, 왜냐하면,
    - 그래프 G에 대해서 TSP(G)를 풀고 최적해 d가 r보다 작거나 같으면 YES, 아니면 NO로 답하면 되기 때문이다.
  - Graph-Coloring(G) vs Graph-Coloring-Decision(G,r)
  - Clique(G) vs Clique-Decision(G,r)에 대해서도 마찬가지이다.

Optimization Problem vs Decision Problem

- 놀랍게도 많은 경우, 이 역도 성립한다. 즉, 결정문제에 대한 다차시간 알고리즘을 이용해 최적화 문제에 대한 다차시간 알고리즘도 찾아질 수 있다.
  - Parametric Searching Technique: 결정문제를 이용해 최적해의 범위내에서 Binary searching 기법을 적용시키는 방법
  - TSP(G)의 해가 [0,N]사이의 정수값임을 안다고 하면, TSP-Decision(G,r)을 이용해 Binary Search하면, log N 번 반복내에 최적해를 구할 수 있다. 즉,
  - TSP-Decision(G,N/2)를 실행한다. 결과가 Yes이면 최적해는 [0,N/2]에, No이면 [N/2,N]에 있어야 함을 알 수 있다. 새로운 범위에서 다시 TSP-Decision(G, N/4 또는 3N/4)을 실행하여 그 결과로부터 최적해의 범위를 유추할 수 있다. 최악의 경우, 이를 log N번 반복하면 최적해를 알 수 있다.
- 고로, NP와 P를 다룰 때 결정 문제만 고려해도 충분하다.

Optimization Problem vs Decision Problem

- VertexCover(G): Find a minimal size of vertex cover of G.
- DVC(G,k): Is there a vertex cover of size k?
- DVC(G,k)를 O(poly(|G|))에 해결할 수 있다면, VC(G) 도 O(poly(|G|)) 에 해결될 수 있다.
  - MinVC(G): Find the size of minimal vertex cover.

    k=0;

    while(!DVC(G,k)) k=k+1;

    return(k);
  - MinVC(G)도 O(poly(|G|) 해결가능 (DVC(G,k)를 최대 |V|번 호출)
  - VC(G, t){ // output the set of minimal vertex cover of G

    Find a vertex u of G such that

    MinVC(G\{u}) == t-1; //최악의 경우, MinVC()를 |V|번 호출

    Output "u";

    if (t>1) VC(G\{u}), t-1); }
  - $T(t)=T(t-1)+|V|$ Time(MinVC) $< T(1) + |V|^2$ Time(MinVC)

The Sets P & NP

- **P 의 정의**
  - P는 다차시간 알고리즘으로 풀 수 있는 결정 문제들의 집합이다.

  Ex) P에 속해 있는 문제들
    - 정렬문제, 검색문제, 행렬곱셈문제 등

  Ex) P에 속해 있지 않은 문제
    - Presburger Arithmetic

  - TSP-Decision(G,d)는 P에 속하는가?
    - No polynomial algorithm has been known yet.
    - So we don't know the answer yet.

Polynomial-time Nondeterministic Algorithm

- **TSP-Decision(G,d)를 위한 "다차시간 비결정적 알고리즘"**
- G의 꼭지점이 모두 n개라 하면,

  1. Guessing state (비결정적 추측단계)
     - n!개의 가능한 경로 중에서 하나의 경로 S를 추측한다. (말도 안 되는 추측도 상관없음)
  2. Verification state (결정적 검증단계) – 다차시간 알고리즘
     - S가 외판원경로인지와 경로길이가 d 이하인지를 검증한다.
       - 연속하는 꼭지점마다 연결선이 G에 있는지- O(|G|) 시간 내에 가능
       - 모든 꼭지점을 정확히 한번씩 다 방문했는지 – O(|G|) 시간 내에 가능
       - 경로길이가 d 이하인지 - O(|G|) 시간 내에 가능
- 다차시간 비결정적 알고리즘 vs 다차시간 (결정적) 알고리즘
- TSP-Decision(G,d)의 다차시간 (결정적) 알고리즘을 구하기 위해서는,
  - n! 개수의 추측단계를 다 거치지 않고,
  - "다차함수 개수"의 추측단계 (혹은 해라고 추측되는 경로)를 뽑는 것이 핵심
  - 이처럼, 대부분의 어려운 문제, 즉, 아직 다차시간 알고리즘을 모르는 문제는 "검증단계"가 아니라 "추측단계"가 어렵기 때문이다.

Verification

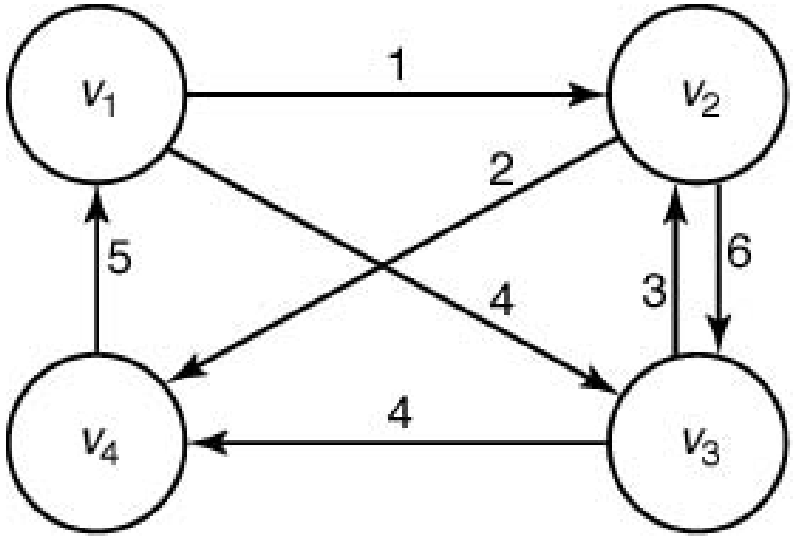
- Verification (검증)
  - 결정문제의 답이 "예"라고 할 때, 증거물이라고 주장되는 사례(Certificate)가
  - 과연 결정 문제의 답을 "예"로 만드는 증거물인지를 검증하는 절차.
  - 예를 들어, TSP-Decision(G,d)에 대해 제시된 증거물 S (Certificate; 여기서는 어떤 경로)이 외판원 결정 문제의 답을 "예"로 만드는 경로인지-즉, 일주여행경로이며 길이가 d 이하인지 검증하는 절차를 말한다.

```
bool verify(weighted_digraph G, number d, claimed_tour S) {
      if (S is a tour && the total weight of the edges in S <= d)
          return true;
      else
          return false;
 }
```

  - 이 검증 절차는 다차시간 안에 수행될 수 있다.
  - 즉, d 보다 작은 일주여행경로를 찾는 것이 아니라 (이 과정은 다차시간으로 해결하지 못할 수 도 있다), 주어진 여행경로 S(certificate)가 d보다 작게 걸리는 것인지를 알아보는(verification) 것이다.

Verification의 예들

Ex)



S	Output	Reason
$[v_1, v_2, v_3, v_4, v_1]$	False	Total weight is greater than 15
$[v_1, v_4, v_2, v_3, v_1]$	False	S is not a tour
#@12*&%a_1\	False	S is not a tour
$[v_1, v_3, v_2, v_4, v_1]$	True	S is a tour with total weight no greater than 15

NP의 정의

- NP (Nondeterministic Polynomial)의 정의
  - 다차시간 비결정적 알고리즘에 의해서 풀 수 있는 모든 결정 문제의 집합
  - 답이 "예"라고 가정할 때, 증거물(Certificate)이 존재하고,
    이 증거물이 "예"라는 답을 주는 사례임을 검증(Verification)하는
    다차시간 알고리즘이 있는 모든 결정 문제들의 집합

- NP에 속하는 문제들
  - TSP-Decision(G,d): TSP-Decision(G,d)의 답이 "예"라고 가정할 때, 제시된 증거물에 대해서 다차시간 내에 검증할 수 있다.
    - Certificate S = 길이가 d 이하인 G안의 일주여행경로
    - Verification: S가 일주여행경로이며, 경로길이가 d 이하인지 확인
      - S에서 연속하는 꼭지점마다 연결선이 G에 있는지- O(|G|)
      - 모든 꼭지점을 정확히 한번씩 다 방문했는지 – O(|G|)
      - 경로길이가 d 이하인지 - O(|G|)

NP의 정의

- Graph-Coloring(G,d): 그래프 G가 d가지 이하의 색으로 coloring될 수 있을 때, 제시된 증거물에 대해서 다차시간 내에 검증할 수 있다.
  - Certificate S={(v1, color 1), (v2, color 2)...}; 사용된 color d개 이하
  - Verification: S의 coloring이 합당한 것인가, 즉
    - G의 모든 꼭지점이 칠해졌는지 - O(|G|)
    - 모든 연결선에 대해 양 끝점이 같은 색으로 칠해진 것은 없는지 – O(|G|)

- Clique(G,d): 그래프 G가 크기 d 이상의 Clique를 가지고 있다고 가정할 때, 제시된 증거물에 대해서 다차시간 내에 검증할 수 있다.
  - Certificate S = G의 꼭지점 d개 이상을 포함한 어떤 complete graph
  - Verification: S가 정말 G의 크기 d 이상인 Clique인지 확인, 즉
    - S의 꼭지점 개수가 d 이상이며 모두 G의 꼭지점인지 - O(|G|)
    - S의 모든 연결선에 대해 그것이 G안에 포함된 연결선인지 체크 - O(|G|)

NP의 특징

- NP에 들어가지 않는 혹은 들어가는지 여부가 불확실한 문제가 있기나 할까? 즉, 증거물을 다차시간에 검증하는 것은 당연한 게 아닐까? 당연하지 않다.
  - Not-TSP-Decision(G,d): 그래프 G에 거리 d 이하의 최단 일주경로가 없는가? (Example 9.15)
    - Certificate S = { G안의 모든 일주여행 경로 }
    - Verification: 검증할 증거물의 크기가 n!이므로 입력크기(|G|)에 대해서 다차시간에 검증할 수 없다.
  - Not-Clique(G,d) – 그래프 G에 크기 d 이상의 Clique이 없는가?
    - Certificate S = { 크기 d 이상의 모든 꼭지점 부분집합 }
    - Verification: 검증할 증거물의 크기가 $2^{|V|}$이므로 입력크기(|G|)에 대해서 다차시간에 검증할 수 없다.
  - 다루기 힘들다(intractable)고 증명이 된 문제,
    즉 Halting 문제, Presburger Arithmetic 문제 등도 있다.

P와 NP의 특징

- P에 포함되는 모든 문제는 당연히 NP에도 포함된다.
- P에 포함되는 문제의 complementary 문제 또한 P에 포함된다.
- NP에 속하는 문제의 complementary 문제가 NP에 속하라는 보장은 없다.
  - TSP-Decision(G,d)는 NP, Not-TSP-Decision(G,d)는 NP 포함여부 아직 모름
  - Clique(G,d)는 NP, Not-Clique(G,d)는 NP 포함여부 아직 모름
- NP에 속한 문제 중에서 P에 속하지 않는다고 증명이 된 문제는 하나도 없다. 따라서 아마도 NP - P 는 공집합일지도 모른다.
- 그럼 P = NP ?
- 사실 많은 사람들이 $P \neq NP$일 것 이라고 생각하고 있기는 하지만 아무도 이것을 증명하지 못하고 있는 것이다

Polynomial-time Reducible

- Transformation (변환) 알고리즘
  - 결정문제 A에 대한 입력 x를 결정문제 B에 대한 입력 y로 변환하되
  - 결정문제 A에 대해 입력 x의 답이 "예" 이면
    결정문제 B에 대한 입력 y의 답이 "예",
    결정문제 A에 대해 입력 x의 답이 "아니오" 이면
    결정문제 B에 대한 입력 y의 답이 "아니오"인 성질을 만족해야 한다.
  - 즉, 결정문제 A에 대한 입력 x의 결과와 결정문제 B에 대한 입력 y의 결과가 같은 성질을 만족한다.

- Polynomial-time Reducible
  - A에서 B로 다차시간에 변환하는 변환 알고리즘이 있다면,
  - A는 B로 "polynomial-time reducible"하다고 하고,
  - $A \propto B$로 쓴다.

Polynomial-time Reducible

- Transformation (변환) 의 예(ex.9.8)
  - 결정문제 A: Given logical variables $x_1, x_2, .., x_n$, does anyone have the value "True"?
  - 결정문제 B: Given integers $k_1, k_2, .., k_n$, is the largest of them positive?
  - 변환 알고리즘 및 A∝B 의 증명:
    - 변환 알고리즘:
      - for i=0…n, $k_i = 1$ if $x_i =$ "True" , 0 otherwise.
    - 위 알고리즘에 의해 결정문제 A의 임의의 input $I_A = \{x_1, x_2, .., x_n\}$ 은 결정문제 B의 어떤 input $I_B = \{k_1, k_2, .., k_n\}$ 로 O(n) 시간에 변환
    - Input I_A 에 대한 결정문제 A의 답이 "Yes"

    => 어떤 i에 대하여, $x_i =$ "True" => $k_i = 1$ => largest is positive

    => Input I_B 에 대한 결정문제 B의 답이 "Yes"
    - Input I_A 에 대한 결정문제 A의 답이 "No"

    => 모든 i에 대하여, $x_i =$ "False" => $k_i = 0$ => largest is zero.

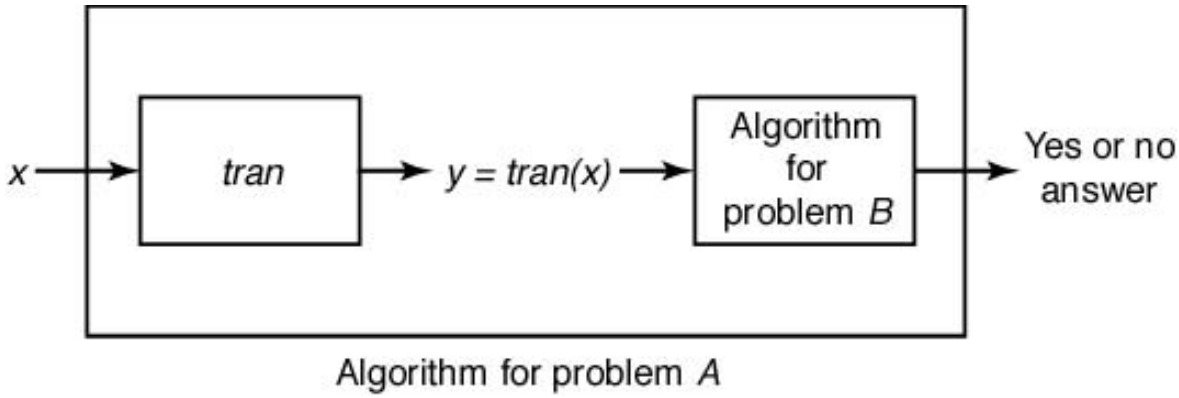
    => Input I_B 에 대한 결정문제 B의 답이 "No"

정리 1

결정 문제 B가 P에 속하고 A∝B이면 결정 문제 A는 P에 속한다.

(증명)

- A∝B이므로, A의 입력 x를 B에 대한 입력 y로 다차시간 f(|x|)에 바꿀 수 있으며, 이때 |y|=g(|x|)이다.
- B는 P에 속하므로, 입력 y에 대해 "예""아니오"로 답할 수 있는 다차시간(h(|y|)시간)-알고리즘이 있다.
- 변환 알고리즘의 정의에 의해, 입력 y가 문제 B에서 갖는 "예""아니오" 답은 입력 x가 문제 A에서 갖는 "예""아니오" 답과 같다.
- 고로, 입력 x에 대해 문제 A에 대한 "예""아니오" 답을 f(|x|)+h(g(|x|)) 시간내에 찾았다



NP-Complete 정의 및 성질

- 정의: 만일 문제 B가 (1) NP에 속하고, (2) NP에 속해 있는 모든 다른 문제 A에 대해서 A∝B이면, B는 NP-Complete (완전)이라고 한다.
- NP-Complete의 정의에 의하여,
  - NP의 부분집합인 NP-Complete에 속하는 문제 중에서
  - 어느 하나라도 P에 속한다는 것이 밝혀지면,
  - NP의 모든 문제가 P에 속하게 된다. 즉, NP가 P에 속하고, P=NP를 의미한다.
- 어떤 문제가 NP-Complete 인지를 위의 정의에 근거해서 증명하는 일은 매우 어렵다. 왜냐하면 NP에 속한 모든 문제가 그 문제로 축소가능 (reducible)하다는 것을 보여야 하기 때문이다. 그러나 다행스럽게도, 1971년 Cook이 다음 2 개의 정리를 증명했다.
- 정리 2 (Cook's Theorem): CNF-SAT 문제는 NP-complete이다.
- 정리 3: 만일 문제 C가 (1) NP에 속하고 (2) 어떤 NP-Complete 문제 B에 대해서 B∝C이면(NP-hardness), C는 NP-Complete 이다.

NP-Complete

- NP의 부분집합으로서,
  - NP-Complete에 속하는 문제 중에서
  - 어느 하나라도 P에 속한다는 것이 밝혀지면,
  - NP의 모든 문제가 P에 속하게 된다. 즉, NP가 P에 속하고, P=NP를 의미한다.
- CNF (Conjunctive Normal Form): x를 논리변수(logical variable)라고 하면, x가 참이라는 말은 $\bar{x}$ 는 거짓이라는 말과 동일하다.
  x나 $\bar{x}$ 는 리터럴(literal)이라 하고, ∨연산자로 리터럴을 결합하면 절(clause)이라고 한다. ∧로 절을 연결하면 CNF가 된다.
  예를 들면, $(\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_4}) \wedge (\overline{x_2} \vee x_3 \vee x_4)$는 CNF이다.
- CNF-SAT(P): 주어진 CNF P가 참이 될 수 있도록 논리값 (참 또는 거짓)을 지정할 수 있는 지의 여부를 결정하는 문제.

  - $\Phi = (x_1 \vee x_2) \wedge (x_2 \vee \overline{x_3}) \wedge \overline{x_2}$ 이면, 답은 "yes"

  - $\Phi = (x_1 \vee x_2) \wedge \overline{x_1} \wedge \overline{x_2}$ 이면, 답은 "no"
- 이 문제는 NP에 속한다.

정리3을 이용한 NP-Complete 증명 예

- Clique-Decision(G,d)는 NP-Complete이다.

(증명) Clique-Decision(G,d)가 NP임은 이미 보였다. 아래 보조정리1과 정리2, 정리3에 의하여, Clique-Decision(G,d)는 NP-Complete이다.

- 보조정리1: CNF-SAT(P) $\propto$ Clique-Decision(G)

(증명) 입력 P는 n개의 논리변수 x1, .., xn과 k개의 clause C1, .., Ck로 만들어진 CNF 라 하자.

- P로부터 어떤 그래프 G를 만들어서, Property(P,G)를 만족함을 보이면 된다.
  - Property (P,G): CNF-SAT(P)의 답이 "Yes"임과 Clique-Decision(G,k)의 답이 "Yes"임이 동치임 (즉, P가 참이 되도록 x1~xn에 참/거짓을 대입할 수 있는 방법이 있으면 그래프 G에 크기 k 이상의 Clique이 존재하고, 대입할 수 있는 방법이 없으면 그래프 G에 크기 k 이상의 Clique이 존재할 수 없음)
- 그래프 G의 Construction: 다음 그래프 G=(V,E)를 P의 크기에 대한 다차시간 내에 건설
  - V = {(y,i) : y is a literal in clause Ci}
  - E = {((y,i),(z,j)) : i와 j는 같지 않고, not(z)가 y가 같지 않다.}
- 논리식 P와 그래프 G는 Property(P,G)를 만족한다. (reducible.pdf 참고)