경로 압축을 이용한 DJ 그래프의 지연 감축 알고리즘

심 손 권[†] · 안 희 학[††]

요 약

효과적이고 정확한 데이터 흐름 문제 분석은 흐름그래프와 지배자 트리 그리고 DJ 그래프를 사용한다. 데이터 흐름 문제 해결은 흐름 그래프를 안전하게 지배자 트리로 감축하는 것이다. 흐름 그래프는 파스 트리를 대신하고, DJ 그래프는 감축 가능하거나 감축이 불가능한 흐름 그래프를 지배자 트리로 정확하게 감축하는데 이용된다. 본 연구에서는 Tarjan의 경로 압축 알고리즘을 이용하기 위하여 Top 노드 찾기 알고리즘을 제시하고 기존의 지연감축 알고리즘을 경로압축을 이용하여 개선한다. 경로압축을 이용한 지연감축 알고리즘은 DJ 그래프를 지연 감축하면서 노드를 끌어올려 지배자 트리의 경로를 압축시킨다. 실제로 제안된 알고리즘은 22% 정도 노드들을 끌어올렸고, 20% 정도 경로를 압축시켰다. 압축된 지배자 트리는 효과적인 데이터 흐름 분석을 가능하게 하고, 코드 최적화 과정의 노드 끌어올리기 효과를 가져와 코드 최적화 과정의 복잡도를 개선하는 효과를 가져온다.

Delayed Reduction Algorithms of DJ Graph using Path Compression

Son-Kweon Sim[†] · Heui-Hak Ahn[††]

ABSTRACT

The effective and accurate data flow problem analysis uses the dominator tree and DJ graphs. The data flow problem solving is to safely reduce the flow graph to the dominator tree. The flow graph replaces a parse tree and used to accurately reduce either reducible or irreducible flow graph to the dominator tree. In this paper, in order to utilize Tarjan's path compress algorithm, the Top node finding algorithm is suggested and the existing delay reduction algorithm is improved using path compression. The delayed reduction algorithm using path compression actually compresses the pathway of the dominator tree by hoisting the node while reducing to delay the DJ graph. Really, the suggested algorithm had hoisted nodes in 22% and had compressed path in 20%. The compressed dominator tree makes it possible to analyze the effective data flow analysis and brings the improved effect for the complexity of code optimization process with the node hoisting effect of code optimization process.

키워드 : 데이터 흐름 분석(data flow analysis), DJ 그래프(DJ graph), 경로 압축(path compression), 지연감축(delayed reduction)

1. 서 론

데이터 흐름 그래프는 데이터 흐름 분석을 위해 지배자 트리(dominator tree)로 감축하여 분석을 한다. 따라서 데이터 흐름 분석은 지배자 트리가 항상 정확하게 유지되어야 프로그램의 의미 변화가 일어나지 않는다. 데이터의 흐름이 분석된 지배자 트리는 컴파일 최적화 과정인 상수계산 미리하기, 공통부분식 제거, 필요없는 코드 제거, 코드 모션 등에 유용하게 이용되어 프로그램 실행시간을 단축시키고 기억 장소를 효율적으로 사용할 수 있게 한다[1-4].

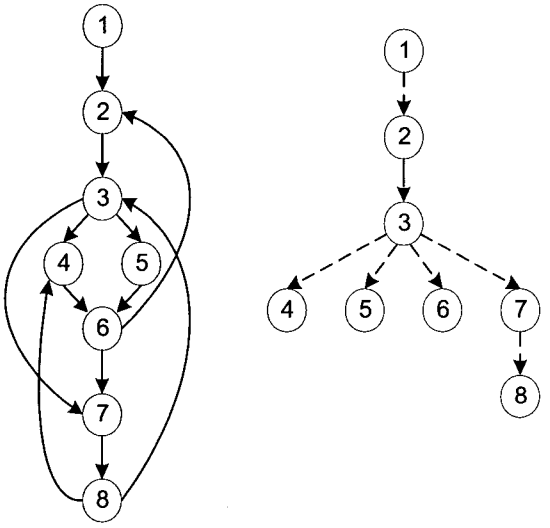
본 논문에서는 최적화 방안의 기본이 되는 데이터 흐름 분석을 효과적으로 처리하기 위하여 Sreedhar에 의해 연구된 지연제거 방법[8]에 Tarjan의 경로 압축 알고리즘[5, 10]을 합성하였다. 두 알고리즘의 합성 가능성은 지배자 트리의 특성인 직접적으로 지배되는 노드는 그 노드의 직접 지배자에만 의존한다는 것에 착안한다. 즉, 어떤 노드 y에서의 정보는 직접 지배자에 의해서만 변형될 수 있기 때문에, y 노드가 위치한 서브 트리의 Top 노드를 찾아서 Top 노드의 정보를 하향식으로 y 노드에 정보를 전달해 줌으로서 y 노드에서의 연결가지를 안전하게 제거할 수 있을 뿐만 아니라 y 노드는 Top 노드의 직접 지배자로 끌어올려 진다. 따라서 연결가지가 완전히 제거된 뒤 DJ 그래프는 전체적으로 경로가 압축되는 효과를 가져온다.

이러한 연구를 위하여 본 논문에서는 Top 노드를 찾는 알고리즘을 제안하고, Tarjan의 경로 압축 알고리즘[5, 10]을 이용하였다. 연구 결과의 타당성을 입증하기 위하여 기존 연구에서 사용하였던 흐름 그래프와 새로운 흐름 그래프의 예를 적용하였다.

† 준 회 원 : 관동대학교 대학원 컴퓨터공학과
†† 종신회원 : 관동대학교 컴퓨터공학과 교수
논문접수 : 2001년 10월 8일, 심사완료 : 2002년 3월 28일

2. DJ 그래프

흐름 그래프는 중간 코드로 표현된 프로그램에 대한 정보를 수집하는 유향 그래프로 한 노드는 기본 블록을 의미한다. 흐름 그래프 G = (N, E, START, END)로 유도된 그래프이다. 여기서, N은 노드의 집합이고, E는 흐름 그래프 가지(edge)의 집합이며, START는 N에 속한 시작 노드이며, END는 N에 속하는 끝 노드이다[3,4, 6-9]. (그림 1)은 흐름 그래프의 예를 보여준다. (그림 1)과 같이 프로그램을 흐름 그래프로 표현하는 것은 기본 블록 집합에 제어 흐름 정보를 나타낼 수 있으며, 각 변수의 사용이나 다른 요소의 흐름을 추적할 수 있기 때문이다. 그러나 흐름 그래프는 각 변수의 제어 흐름 정보를 알 수는 있으나 각 노드 사이의 지배 정보는 직접적으로 알 수가 없다. 따라서 지배정보를 알기 위하여 각 노드에서의 지배자 계산[1]을 통한 지배자 트리를 구축하여 사용한다.

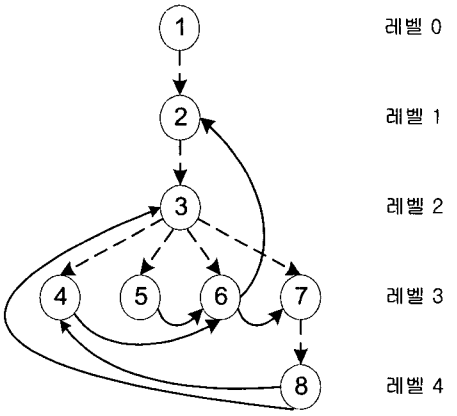


(a) 흐름그래프 (b) 지배자 트리

(그림 1) 흐름그래프와 지배자 트리

또한 지배자 트리는 지배자 정보만 나타낼 뿐 데이터의 흐름을 완벽하게 알 수 없다. 이러한 그래프의 단점을 극복하기 위하여 기존 연구에서 DJ 그래프[7-9]가 제안되었다. DJ 그래프는 기존의 감축 가능한 흐름그래프에서만 데이터 흐름 분석이 가능하였던 것을 감축이 불가능한 흐름그래프에 대해서도 데이터 흐름 분석을 가능하게 하였다. DJ 그래프는 흐름그래프의 데이터 흐름 정보와 지배자 트리에서의 각 노드간의 지배정보를 나타낼 수 있기 때문에 어떠한 흐름그래프도 데이터 흐름 분석을 가능하게 한다. (그림 1)의 흐름그래프와 지배자 트리에 대한 DJ 그래프는 (그림 2)와 같다.

DJ 그래프는 흐름그래프의 노드정보와 가지정보, 그리고



(그림 2) DJ 그래프

지배 정보를 종합하여 만들어진다. DJ 그래프를 생성하기 위하여 흐름그래프의 각 노드에서 가지 정보를 추출한 다음 지배자 계산을 하여 지배자 가지 정보를 추출한다. 흐름그래프 가지 정보에서 지배자 가지 정보를 빼면 간단하게 연결가지 정보를 얻을 수 있다. 또한 지배자 특성 중 피지배 노드는 지배노드의 레벨+1이므로 각 노드의 레벨 정보를 얻을 수 있다. 흐름그래프와 지배자 정보를 입력받아 DJ 그래프를 생성하는 알고리즘을 나타내면 (알고리즘 1)과 같다.

```
procedure make_dj(join_edge_table) ;
                    /* 흐름그래프와 지배자정보 */
begin
  while num_fgraph do          /* 흐름그래프 노드 수 */
    begin
      new(node) ;          /* DJ 그래프 노드 생성 */
      node↑.id : = num_fgraph ;     /* 노드 번호 */
      node↑.dom : = dominator_node ;  /* 지배자 노드 연결 */
      node↑.jedge : = join_edge ;        /* 연결가지 생성 */
      node↑.data : = flow_data ;  /* 각 노드의 흐름정보 저장 */
      num_fgraph : = num_fgraph - 1 ;
    end
end ;
```

(알고리즘 1) DJ 그래프 구축 알고리즘

(알고리즘 1)에서 num_fgraph는 흐름그래프 전체 노드의 갯수이고 dominator_node는 현재 노드를 지배하고 있는 노드에 대한 주소를 갖고 있다. 또한 join_edge는 현재 노드에서 지배자 가지를 제외한 나머지 연결가지에 대한 주소이다. 그리고 flow_data는 데이터 흐름 분석에 의해 분석된 정보를 현재 노드에 저장하는 과정이다.

3. 데이터 흐름 분석

데이터 흐름 분석은 프로그램에 대한 사실을 정적으로 평가하는 과정이다. 따라서 어떤 한 노드에서의 정보는 입력정보와 출력정보에 의해 나타낼 수 있다[1, 5, 8]. 즉 (그림

2)의 DJ 그래프 노드 2에서의 데이터 흐름 정보는 노드 1의 출력정보와 노드 6의 출력정보, 그리고 노드 2에서의 생성정보의 합이다. 노드에서의 데이터 흐름 정보가 계산되면 노드 6에서 노드 2로의 연결가지는 제거될 수 있다. 모든 노드에서의 데이터 흐름 정보가 계산되면 DJ 그래프는 지배자 트리로 감축된다.

3.1 데이터 흐름 정보 추출

y → x로의 연결가지가 있다고 가정하고, 각 노드에서의 입력정보와 출력정보에 대하여 식 (1)과 같은 방정식[1, 5, 8]을 세울 수 있다.

$$I_x = \bigwedge_{y \in Pred_f(x)} O_y \tag{1}$$

식 (1)은 어떤 노드 x에서의 입력정보는 노드 x의 모든 조상 노드 y의 출력정보로 표현한 것이다. I는 입력정보, O는 출력정보, ∧는 교집합 또는 합집합 연산을 의미한다. 그리고, $Pred_f(x)$는 흐름그래프에서 어떤 노드 x에 대한 모든 조상노드들 이다.

$$O_x = f_x(I_x) = P_x I_x + G_x \tag{2}$$

식 (2)는 x 노드의 출력정보이다. 여기서 f_x는 입력정보를 출력정보로 사상하는 함수이고, P_x는 노드 x에서 보존되어야 할 정보이고, G_x는 노드 x에서 새롭게 생성된 정보이다. 이 두 개의 정보는 x 노드에서 변하지 않는 상수 값을 갖는다. 따라서 어떤 한 노드에서의 출력정보는 입력된 정보와 보호되어야 할 정보를 교집합으로 연산하고 생성된 정보를 합집합 연산을 해야 한다.

$$O_x = P_x(\bigwedge_{y \in Pred_f(x)} O_y) + G_x \tag{3}$$

식 (3)은 흐름그래프에서 어떤 노드 x의 출력정보를 연산하기 위하여 식 (1)을 식 (2)에 대입한 결과이다.

(그림 2)의 DJ 그래프에서 각 노드의 흐름 정보를 연산하면 결과는 다음 <표 1>과 같다.

〈표 1〉 (그림 2)의 각 노드에 대한 입출력 정보

x	O_x	$I_x = \bigwedge_{y \in Pred_f(x)} O_y$	$O_x = P_x I_x + G_x$
1	O_1	NULL	G_0
2	O_2	$O_1 \wedge O_6$	$P_2(O_1 \wedge O_6) + G_2$
3	O_3	$O_2 \wedge O_8$	$P_3(O_2 \wedge O_8) + G_3$
4	O_4	$O_3 \wedge O_8$	$P_4(O_3 \wedge O_8) + G_4$
5	O_5	O_3	$P_5 O_3 + G_5$
6	O_6	$O_3 \wedge O_4 \wedge O_5$	$P_6(O_3 \wedge O_4 \wedge O_5) + G_6$
7	O_7	$O_3 \wedge O_6$	$P_7(O_3 \wedge O_6) + G_7$
8	O_8	O_7	$P_8 O_7 + G_8$

3.2 출력변수 제거에 의한 연결가지 제거

연결가지 y → x가 있다고 가정하고, 노드 y의 출력정보는 위의 식 (1), 식 (2)와 식 (3)을 이용하여 식 (4)와 같은 방정식[8]을 세울 수 있다.

$$O_y = P_y(\bigwedge_{z \in Pred_f(y)} O_z) + G_y \tag{4}$$

식 (4)는 노드 z의 출력정보는 노드 y의 입력정보이므로 노드 y에서의 출력정보가 된다. 따라서 노드 x의 출력정보는 변수 O_y를 제거하여 식 (5)와 같은 방정식을 얻을 수 있다.

$$O_x = P_x(P_y(\bigwedge_{z \in Pred_f(y)} O_z) + G_y) + G_x \tag{5}$$

변수 O_y를 제거할 수 있다는 의미는 노드 y에서의 정보(보호되어야 할 값, 입력정보, 생성된 정보)를 노드 x에 대입하였으므로 프로그램의 의미 변화 없이 안전하게 y에서 x로의 연결가지를 제거할 수 있다. 여기서 연결가지 y → x의 제거는 세 가지 상황이 있다. 첫 번째, 노드 y의 레벨과 노드 x의 레벨이 같고 노드 y와 노드 x가 같지 않은 경우이다. 이러한 경우에는 노드 y와 노드 x의 직접지배자가 같기 때문에 연결가지를 완전히 제거할 수 있다. 두 번째, 노드 y의 레벨과 노드 x의 레벨이 다른 경우로 이 경우에는 두 노드의 직접 지배자가 서로 다르므로 프로그램의 의미 변형 없이 연결가지를 제거하기 위해서는 식 (5)를 이용하여 계산한 다음 노드 y의 직접 지배자와 노드 x로 연결가지를 삽입한 다음 연결가지 y → x를 제거한다. 세 번째, 위의 두 번째 경우를 이용하여 새로운 가지를 삽입하고 기존의 연결가지를 제거하다 보면 셀프루프가 발생한다. 이 경우에는 다음의 식 (6)과 같은 방정식으로 셀프루프를 제거할 수 있다.

$$O_x = P_x(O_x) + G_x = f_x(O_x) = f_x^*(O_x) \tag{6}$$

어떤 노드 x에서 셀프루프가 발생하였다고 가정하자. 위의 식 (6)을 이용하여 노드 x의 출력변수 O_x를 제거하기 위해서 x노드에서 보존되어야 할 정보 (P_x)에 x노드의 출력정보를 대입하여 교집합 연산을 하고 여기에 생성된 정보 (G_x)를 합집합 연산한다. 같은 노드에서의 반복이므로 같은 정보에 대하여 교집합 연산을 한다. 따라서 1회 이상의 셀프루프 반복은 같은 결과를 초래하고, 출력변수 O_x는 제거된다.

4. 경로압축을 이용한 지연감축 알고리즘

본 장에서는 기존의 지연감축 알고리즘[8]에 경로압축 알

고리즘을 적용하기 위하여 Top 노드 찾기 알고리즘을 제안한다. 또한 찾은 Top 노드와 지배자 특성을 이용하여 DJ 그래프를 지연 감축시키면서 경로를 압축하는 방법을 제시한다.

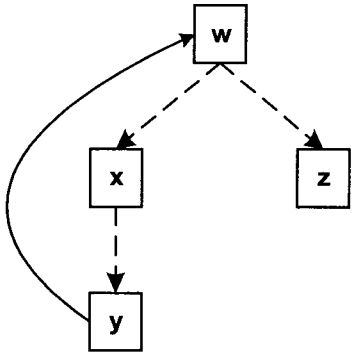
4.1 Top 노드 찾기 알고리즘

DJ 그래프에서 어떤 한 노드의 Top 노드는 그 노드에 영향을 줄 수 있는 노드이다. Top 노드는 연결가지가 존재하는 두 노드에서 도착 노드이거나 도착 노드와 같은 레벨에 있는 조상 노드이다. (그림 2)의 8→3 연결가지를 보면 시작노드 8에 직접적으로 영향을 줄 수 있는 노드는 노드 3과 7이다. 따라서 노드 7의 출력정보를 노드 8에 미리 전해주면 노드 8은 노드 7에 독립적일 수 있고, 노드 8은 노드 7과 같은 레벨로 끌어 올려질 수 있다. Top 노드를 찾는 알고리즘을 나타내면 (알고리즘 2)와 같다.

```
procedure find_topnode(from_n, to_n) ;
          /* 연결가지의 시작노드와 도착노드 */
begin
   level : = to_n↑.level ;
   top_level : = from_n↑.level ;
   repeat
        top_level : = from_n↑.dom ;
        top_node_list : = from_n↑.dom ;
                /* 찾아진 Top 노드 저장 */
        from_n : = from_n↑.dom ;
    until level = top_level
   /* 도착노드의 레벨과 같아질 때까지 */
end.
```

(알고리즘 2) Top 노드 찾기 알고리즘

(그림 3)에서 연결가지 y→w를 (알고리즘 2)에 적용하면 노드 w의 레벨인 0이 될 때까지 Top 노드를 찾게 되고, 수행과정에서 노드 y의 직접지배자인 노드 x를 찾고, 노드 x의 직접지배자인 노드 w를 찾게 된다. 따라서 Top_node_list에는 x와 w가 저장된다.



(그림 3) DJ 그래프에서 Top 노드 찾기

4.2 경로 압축 알고리즘

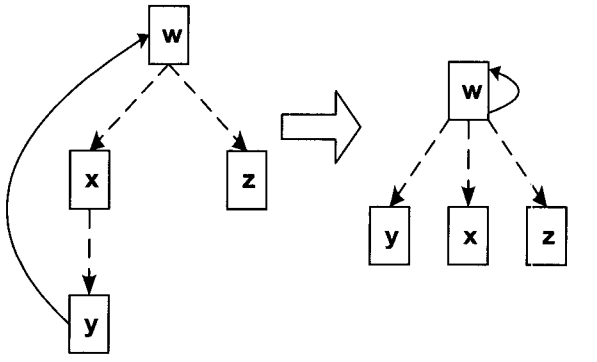
경로 압축 알고리즘은 Tarjan의 감축가능한 흐름그래프에서 데이터 흐름 분석을 효과적으로 하기 위하여 제안된 알고리즘이다. 본 연구에서는 이 알고리즘을 이용하여 기존의 지연감축 알고리즘에 적용하여 데이터 흐름 분석에 효과적인 압축된 지배자 트리로 변형한다.

경로 압축 알고리즘은 어떤 한 노드가 직접 지배자 노드에만 의존한다는 지배자 특성에 근거한다. 직접 지배자의 출력정보를 연결가지가 있는 시작노드에 Top-down 방식으로 전달해 줌으로서 시작노드는 직접 지배자에 독립할 수 있고 한 수준 위로 경로를 압축할 수 있게 된다.

(그림 4)의 (a) DJ 그래프에서 연결가지 y→w를 경로압축 알고리즘에 적용하면 노드 y는 알고리즘의 from_n이되고 노드 w는 to_n이된다. y→w의 Top 노드는 (알고리즘 2)에 의해 노드 x와 노드 w로 찾아진다. 연결가지는 지연감축 알고리즘에 의해 w→w로 되고, 노드 x에서의 출력정보를 노드 y에 주면 노드 y는 직접지배자인 노드 x에 의존하지 않게 되고 노드 w의 자식노드로 변형될 수 있다. 따라서 DJ 그래프는 (그림 4)의 (b)와 같이 압축되게 된다. 경로 압축 알고리즘을 나타내면 (알고리즘 3)과 같다.

```
procedure djpath_compress(from_n, to_n, top_node_list)
begin
  while Top_node_list
      from_n↑.data : = from_n↑.data + Top_node↑.data ;
      /* 하향식으로 지배자 노드의 정보를 전달 */
  from_n↑.dom : = top_node↑.child ;
end.
```

(알고리즘 3) 경로압축 알고리즘



(a) (b)

(그림 4) DJ 그래프의 경로 압축

4.3 경로압축을 이용한 지연 감축 알고리즘

지연감축 알고리즘[8]은 어떤 노드에 연결가지가 존재할 때 그 연결가지의 레벨 정보를 이용하여 두 레벨 이상의

차이가 존재하면 연결가지에 있는 시작 노드의 지배자를 차례로 찾아서 도착노드와 같은 레벨에 위치하는 지배 노드와 도착 노드로 직접 연결하는 알고리즘이다.

본 연구에서는 지배자 트리의 특성인 직접적으로 지배되는 노드는 그 노드의 직접 지배자에만 의존한다는 것에 착안하여 Tarjan의 경로 압축 알고리즘[5, 10]을 지연감축 알고리즘에 적용하였다. 경로 압축 알고리즘을 적용하기 위하여 연결가지의 목표노드와 같은 레벨에 있는 직접지배자 노드를 찾기 위하여 Top 노드 찾기 알고리즘인 (알고리즘 2)를 제안하였다. 제안된 (알고리즘 2)와 경로 압축 알고리즘을 기존의 지연감축 알고리즘에 추가 적용하여 기존 알고리즘[8]을 (알고리즘 4)로 개선하였다. (알고리즘 4)의 CUT 4는 시작 노드와 도착 노드의 레벨이 서로 다른 경우이므로 (알고리즘 2)인 find_topnode 프로시저를 호출하여 시작 노드의 Top 노드를 찾아서 Top 노드와 지배자 노드들을 top_node_list에 반환하고 연결가지를 제거한다. 그리고 나머지 CUT1, CUT2, CUT3 알고리즘에서 top_node_list가 null이 아닌 경우 경로 압축 프로시저인 (알고리즘 3)을 호출하여 변수제거를 한 후 해당 노드의 경로 압축을 실행한다. CUT1, CUT2, CUT3, CUT4 제거 규칙은 다음과 같다.

(1) CUT1 규칙

연결가지 y → y로의 셀프루프를 제거하는 규칙이다.

- 변수 제거 : $compute\_cut1(\langle O_y = f_y(O_y)\rangle)$

$$\Rightarrow \langle O_y = f_y^*(O_y)\rangle$$

- DJ 그래프 감축 : $cut1(\langle g^i,\ N,\ E,\ y \xrightarrow{J} y\rangle)$

$$\Rightarrow \langle g^{i+1},\ N,\ E-\{y \xrightarrow{J} y\}\rangle$$

(2) CUT2, CUT3 제거 규칙

y.level = z.level이고 y → z인 연결가지를 제거하는 규칙이다.

- 변수 제거 : $compute\_cut2$ or $cut3(\langle O_z = P_zO_y + G_z\rangle)$

$$\Rightarrow \langle O_z = P_z(P_yO_x + G_y) + G_z\rangle$$

- DJ 그래프 감축 : $cut2$ or $cut3(\langle g^i, N, E, y \xrightarrow{J} z\rangle)$

$$\Rightarrow \langle g^{i+1}, N, E-\{y \xrightarrow{J} z\}\rangle$$

(3) CUT4 제거 규칙

y.level ≠ z.level이고 y → z인 연결가지를 제거하는 규칙이다.

- 변수 제거 : $compute\_cut4(\langle O_z = P_zO_y + G_z\rangle) \Rightarrow$ 지연
- DJ 그래프 감축 : $cut4(\langle g^i, N, E, y \xrightarrow{J} z\rangle)$

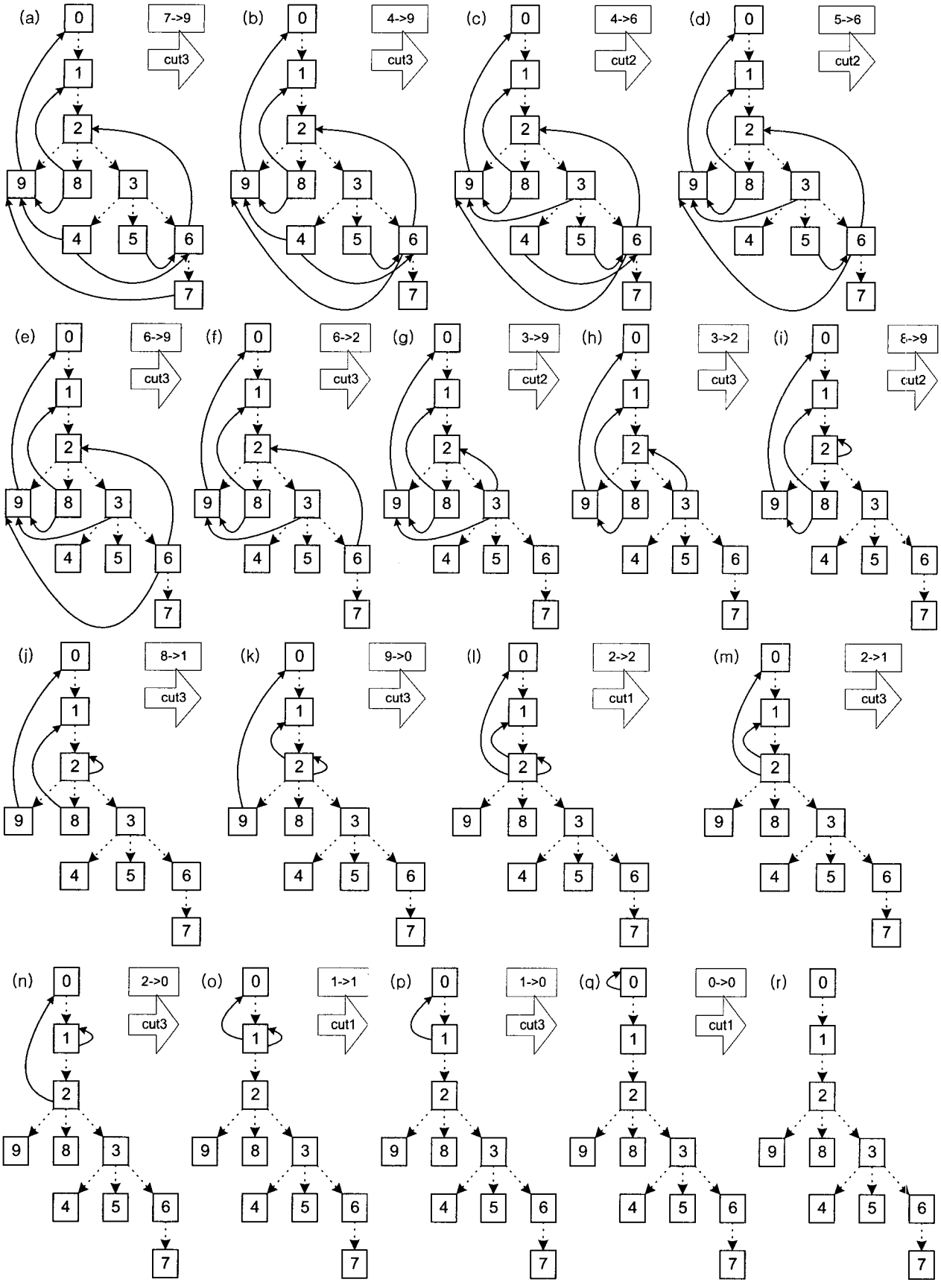
$$\Rightarrow \langle g^{i+1}, N, (E-\{y \xrightarrow{J} z\}) \cup \{Top_{\ y\ J\to z} \xrightarrow{J} z\}\rangle$$

제거 규칙에 기반한 알고리즘은 (알고리즘 4)와 같다. (알고리즘 4)는 만들어진 DJ 그래프에서 데이터 흐름 분석을 하여 지배자 트리로 변환하는 알고리즘이다. 여기서 j_sequency()는 DJ 그래프에서 연결가지가 있는 노드에 도달하였을 때 연결가지를 제거하는 프로시져이고, djpath_compress()는 경로압축 알고리즘을 호출하는 함수이다. 또한 deljedge()는 연결가지를 제거하는 알고리즘을 호출하는 함수이며, addjedge()는 연결가지를 삽입하는 알고리즘을 호출하는 함수이다. 그리고 find_topnode()는 연결가지가 있는 노드의 최상위 직접지배자를 찾는 알고리즘을 호출하는 함수이다.

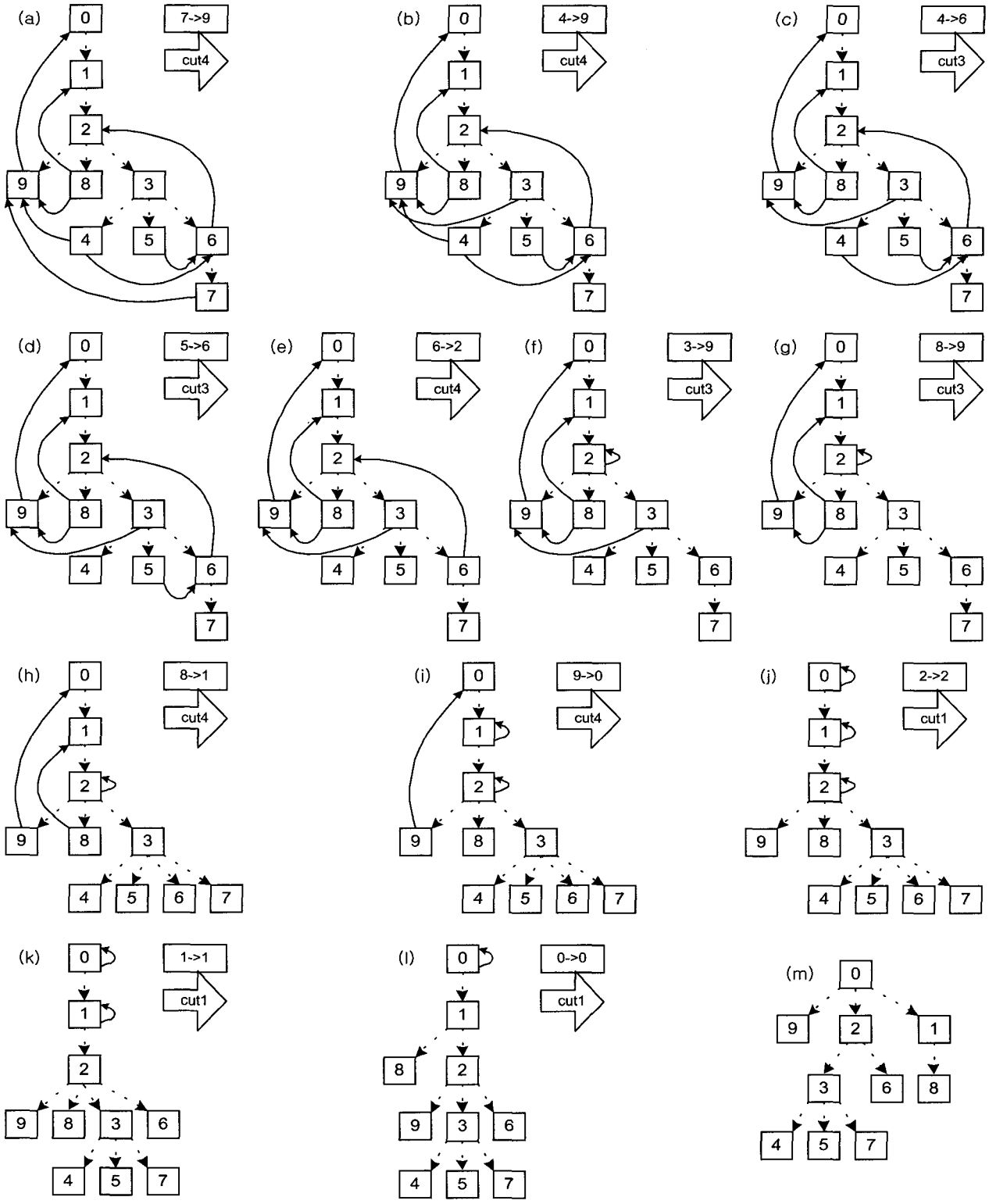
```
procedure del_jedges_sequency()
begin
  for i := 0 to DJ_tree[i] do
  begin
    if DJ_tree[i]↑.jedges then
      for j = 0 to tmp[j] do
      begin
        j_sequency(DJ_tree[i], tmp[j]) ;
      end ;
    end ;
end ;
procedure j_sequency(nodeptr from_n, nodeptr to_n)
begin
  if from_n = to_n then                    /* CUT1 */
    while top_node_list
      djpath_compress(from_n, to_n, top_node_list) ;
    deljedge(from_n, to_n) ;
    else if from_n_parent = to_n then
    begin                                  /* CUT2 */
      while top_node_list
        djpath_compress(from_n, to_n, top_node_list) ;
      to_n↑.data = to_n↑.data + from_n↑.data ;
      deljedge(from_n, to_n) ;
      addjedge(to_n, to_n) ;
    end ;
    else if from_n_level = to_n_level then
    begin                                  /* CUT3 */
      while top_node_list
        djpath_compress(from_n, to_n, top_node_list) ;
      to_n↑.data = to_n↑.data + from_n↑.data ;
      deljedege(from_n, to_n) ;
    end ;
      else
        begin                              /* CUT4 */
          find_topnode(from_n, to_n) ;  /* Top 노드 찾기 */
          from_n↑.jedge := null ;
          j_delay(from_n_ancestor, to_n) ;
        end
end ;
```

(알고리즘 4) 경로압축을 이용한 지연 감축 알고리즘

본 연구에서 제시한 알고리즘의 성능을 측정하기 위해 기존 연구에 적용된 그래프[8]를 이용하여 각각의 알고리즘을 적용하여 지배자 트리로 변환하였다.



(그림 5) 기존 알고리즘에 의한 지배자 트리로의 변환



(그림 6) 경로압축을 이용한 지연감축 알고리즘에 의한 지배자 트리로의 변환

(그림 6)의 (a)에서 노드 7에서 노드9로의 연결가지를 예를 들면, 시작노드 7과 도착노드 9는 레벨이 서로 다르므로 (알고리즘 4)에 의해 CUT4를 수행한다. 알고리즘에 의해 연결가지 7→9의 Top 노드를 찾는 프로시저를 호출하여 (알고리즘 2)에 의해 top_node_list에 노드3과 6이 저장된다. 또한 연결가지 7→9는 제거되고 새로운 연결가지 3→9가 생성된다. 차례로 연결가지들이 제거되고 난 후, 연결가지 3→9를 만나면 CUT2를 수행하게 되고 djpath_compress()

를 호출한다. 여기서 Top 노드 3에 저장된 노드 3의 출력 정보를 하향식으로 노드 7에 전달함으로 인해서 노드 7은 부모노드인 6번 노드에 독립되게 된다. 따라서 노드 7이 갖고 있는 지배자 노드의 포인터에 노드 3의 자식 포인터를 대입함으로서 노드 7은 노드 3의 자식으로 끌어 올려지고 경로는 압축되게 된다.

각각의 알고리즘을 적용하여 지배자 트리로 변환해 본 결과 기존 알고리즘은 지배자 트리로 감축하는데 18단계가 필요하였고, 경로압축 알고리즘은 13단계가 필요하였다.

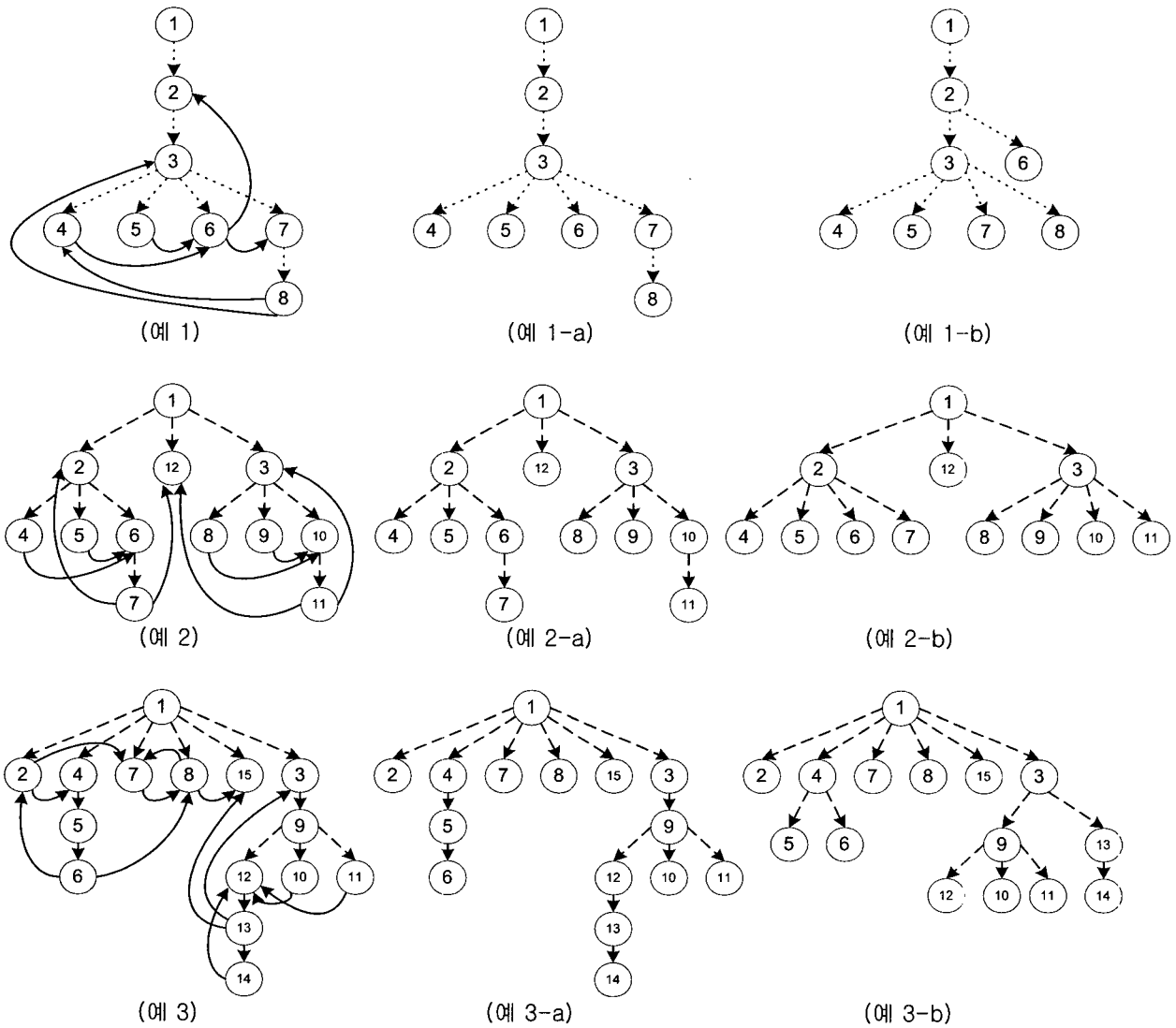
5. 성능 평가

본 장에서는 기존 알고리즘과 경로압축을 이용한 지연감축 알고리즘을 DJ 그래프에 적용하여 노드단위로 압축된 성과와 그래프 전체 단위로 압축된 성과를 비교 분석한다.

(그림 6)은 몇 가지 DJ 그래프의 예에 대하여 지연감축 알고리즘과 경로압축을 이용한 지연감축 알고리즘을 적용한 사례이다. 그림에서 (예 1-a), (예 2-a), (예 3-a)는 기존의 지연감축 알고리즘을 적용한 결과이고, (예 1-b), (예 2-b), (예 3-b)는 경로압축을 이용한 지연감축 알고리즘을 적용한 결과이다.

(그림 7)에서와 같이 기존 알고리즘은 단순히 연결가지만 제거할 수 있었다. 그러나 경로압축을 이용한 지연감축 알고리즘은 지배자 특성에 의하여 연결가지를 제거하면서 노드를 끌어올려 경로를 압축시킬 수 있다.

제안한 알고리즘을 이용하여 끌어올려진 노드의 개수와 그래프 압축 정도를 도표로 나타내면 <표 2>와 같다. 예4는 (그림 5)와 (그림 6)을 예로 든 것이다.

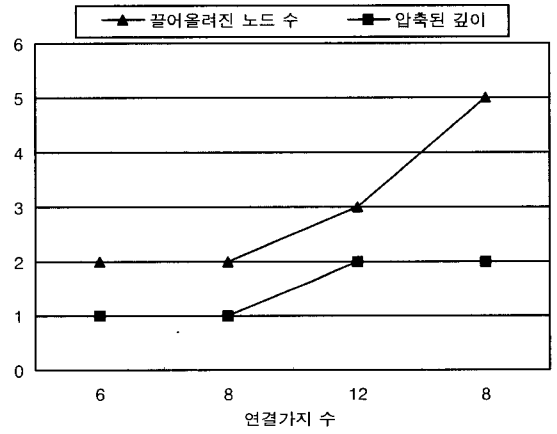


(그림 7) 경로압축을 이용한 지연감축 알고리즘에 적용한 DJ 그래프의 예

〈표 2〉 경로압축을 이용한 알고리즘 적용 후 그래프 변화에 대한 정보

구분 / 지배자 연결그래프	알고리즘 적용 전 연결지배자 그래프			경로압축을 이용한 지연감축 알고리즘 적용 후	
	연결 가지 수	노드 수	깊 이	끌어올려진 노드 수	압축된 깊이
예 1	6	8	5 레벨	2	1 레벨
예 2	8	11	4 레벨	2	1 레벨
예 3	12	15	6 레벨	3	2 레벨
예 4	8	10	6 레벨	5	2 레벨

<표 2>에서 예3의 연결지배자 그래프는 전체 노드 15개 중 3개의 노드가 끌어 올려지고, 그래프 깊이는 6 레벨에서 4 레벨로 줄어들었다. 전체적으로 연결지배자 그래프에서 끌어올려지는 노드는 22%정도이고, 그래프의 깊이는 평균 1.5 레벨 정도가 줄어들었다. 위의 <표 2>에서 연결가지 개수에 대한 그래프 변화를 차트로 나타내면 (그림 8)과 같다.



(그림 10) 연결가지에 대한 그래프 변화

실험에서 알고리즘 적용 후 그래프의 변화는 노드 수 또는 그래프의 깊이 보다는 연결가지 개수에 더 많이 영향을 받고 연결가지를 의미하는 제어문이 많으면 많을수록 알고리즘은 더욱 효과적이다.

6. 결 론

앞선 연구에서 제안한 DJ 그래프는 지배자 트리에 연결가지를 합성한 그래프이다. 따라서 DJ 그래프는 지배자 트리의 특성을 포함하고 있다. 이 점에 착안하여 본 연구에서는 지배자 트리의 특성 중 피지배 노드는 지배 노드에만 의존한다는 특성을 DJ 그래프에 적용하여 감축이 완료된 지배자 트리는 원래의 그래프보다 경로가 20% 정도 압축되는 성과를 보였다. 또한 DJ 그래프의 감축과정에서 연결가지가 있는 노드 중 22% 정도의 끌어 올려지는 노드가 발생한다. 이것은 코드 최적화 과정인 노드 끌어올리기 효과에 해당하므로 코드 최적화 과정의 복잡도를 줄일 수 있는 효과가 있다.

현재 프로그램 개발시 소스코드의 수정, 추가, 삭제시 발생되는 갱신 문제에 대하여 효과적으로 데이터 흐름을 분석하기 위해 점진적 알고리즘에 필요한 어떤 한 노드의 지배영역과 반복적 지배영역에 관한 연구가 진행중이다. 또한 본 연구의 결과를 토대로 코드 최적화 과정인 노드 끌어올리기 효과를 데이터 흐름 분석에 포함할 수 있는지의 가능성에 대한 연구가 필요하다.

참 고 문 헌

[1] Aho, A. V., Sethi, R., and Ullman, J. D. "*Compilers Principles, Techniques, and Tools*," Addison-wesley Publishing Co., 1986.

[2] Carroll, M. and Ryder, B. G. "Incremental data flow update via attribute and dominator updates," *In ACM SIGPLAN-SIGACT Symposium on the Principles of Programming Languages*, pp.274-284, Jan., 1988.

[3] Ramalingam, G. and Reps, T., "An incremental algorithm for maintaining the dominator tree of a reducible flowgraph," *In ACM SIGPLAN-SIGACT Symposium on the Principles of programming Languages*, pp.314-325, Jan., 1994.

[4] Ramalingam, G. "Identifying loops in almost linear time," *ACM Transactions on Programming Languages and Systems*, Vol.21, No.2, pp.175-188, 1999.

[5] Ryder, B. G. and Paull, M. C. "Elimination algorithms for data flow analysis," *ACM Comput. Surv.* Vol.18, No.3, pp. 277-316. Sep., 1986.

[6] Sreedhar, V. C., and Gao, G. R., "A linear time algorithm for placing ϕ-nodes," *In ACM SIGPLAN-SIGACT Symposium on the Principles of programming Languages*, pp. 62-73, 1995.

[7] Sreedhar, V. C., and Gao, G. R., and Lee, Y. F. "Identifying Loops Using DJ Graphs," *ACM Transactions on Programming Languages and Systems*, Vol.18, No.6, pp.649-658, 1996.

[8] Sreedhar, V. C., and Gao, G. R., and Lee, Y. F. "A New Framework for Exhaustive and Incremental Data Flow Analysis Using DJ Graphs," *ACM In Proceedings of the SIGPLAN'96 Conference on Programming Language Design and Implementation*, pp.278-290, 1996.

[9] Sreedhar, V. C., Gao, G. R., and Lee, Y. F. "Incremental Computation of Dominator Trees," *ACM Transactions on Programming Languages and Systems*, Vol.19, No.2, pp. 239-252, 1997.

[10] Tarjan, R. E. "Fast algorithms for solving path problems," *Journal of ACM*, Vol.28, No.3, pp.594-614. 1981.

심 손 권

e-mail : sksim@kwandong.ac.kr

1996년 관동대학교 전자계산공학과 졸업 (공학사)

1998년 관동대학교 대학원 전자계산공학과 졸업(공학석사)

2001년 관동대학교 대학원 전자계산공학과 박사수료

1998년~1999년 관동대학교 산업기술연구소 책임연구원

1998년~현재 관동대학교 컴퓨터공학과 강사

2000년~현재 동우대학 컴퓨터정보과 강사

관심분야 : 컴파일러, 병렬컴파일러, 프로그래밍 언어, 웹 프로그래밍 언어 등

안 희 학

e-mail : hhahn@kwandong.ac.kr

1981년 숭실대학교 전자계산학과 졸업 (공학사)

1983년 숭실대학교 대학원 전자계산학과 졸업(공학석사)

1994년 숭실대학교 대학원 전자계산학과 졸업(공학박사)

1994년~1996년 관동대학교 전자계산소 소장

2001년~현재 관동대학교 전자계산소 소장

1984년~현재 관동대학교 컴퓨터공학과 교수

관심분야 : 컴파일러, 병렬컴파일러, 프로그래밍 언어, 함수 언어, 오토마타 등