ソフトウェア工学 121－7
(1998. 11. 5)

プログラム変更に対処し易いプログラム依存グラフの一変種

太田 剛，水野忠則
静岡大学 情報学部 情報科学科

〒 432-8011 静岡県浜松市城北 3-5-1
Tel: 053-478-1466, Fax: 053-478-1499
e-mail: {ohta,mizuno}@cs.inf.shizuoka.ac.jp

あらまし　プログラム依存グラフは，プログラム中の文を節点，それらの間の依存関係を辺で表現した有向グラフである．したがって，文を追加するには，その文と他の文との間の依存関係を計算しなくてはならない．これが比較的時間を要する計算であるために，プログラムの変更に対するグラフの更新作業は容易ではない．この原因は，辺を引く際に，依存関係の計算過程を全て捨て，結果だけを１本の辺の形で残すことにある．

本稿では，この計算過程をもグラフの形で残すようにした，プログラム依存グラフの一変種を示し，オリジナルのプログラム依存グラフとの得失について述べる．

キーワード　プログラム依存グラフ，更新作業，データ依存

A Variant of Program Dependence Graph Suitable for Program Modification

Tsuyoshi Ohta and Tadanori Mizuno
Department of Computer Science, Fuculty of Information,
Shizuoka University

Johoku 3-5-1, Hamamatsu 432-8011, Japan
Tel: 053-478-1466, Fax: 053-478-1499
e-mail: {ohta,mizuno}@cs.inf.shizuoka.ac.jp

Abstract　Program dependence graph (PDG) is a directed graph which represents dependencies between statements/predicates. Thus, when we introduce a new statement (i.e. vertex and edges), computing dependencies between the vertex and other vertexes is needed. Because this work is costful, updating a PDG is not easy.

This paper shows a variant of PDG which has an easier updating algorithm. We also compare it with an original PDG.

key words　Program Dependence Graph, Data Dependence

1 まえがき

プログラム依存グラフ (Program Dependence Graph; PDG) は，プログラム中の文や条件式を節点，それらの間のデータや制御による依存関係を辺で表現した有向グラフである [1]. PDG は，ソフトウェア工学の様々な分野において利用され得る有用なツールであることが示されてきた [2, 3, 4, 5].

その一方で，プログラムの変更に対する PDG の更新作業は，容易ではない．これは，PDG 上で辺を引くためには，節点間の依存関係を計算しなくてはならず，これが比較的時間を要する計算であるためである．ことに，データ依存関係の更新作業はデータフロー解析に基づくものであり，これをインクリメンタルに行なう方法 [6, 7] や，PDG 上で効率良く行なう方法に関する研究が行なわれてきた [8].

PDG の更新作業が容易ではない原因についてよく考えてみると，次の点にたどり着く．すなわち，辺を引く際に，最終的に得られた結果だけを 1 本の辺の形で残し，依存関係の計算過程を全て捨ててしまうことにある．

そこで，本稿では，この計算過程を PDG 上に残すことによって，更新作業を容易にする方式を示す．ただし，PDG の利点 — 様々な作業がグラフの探索やマッチングという単純なアルゴリズムで実現できる — を消してしまわないように，計算過程をもグラフの形で残す方法を示す．また，このプログラム依存グラフの一変種と，オリジナルのプログラム依存グラフとの得失についても述べる．

2 プログラム依存グラフの一変種

本稿で述べるプログラム依存グラフの一変種 (a variant of PDG; VPDG) は，基本的には，プログラム依存グラフ内のデータ依存辺を変数ごとにまとめて，一本のデータ依存辺を複数の制御フロー辺と節点に置き換えたものである．

2.1 単一関数の VPDG

本節で例として用いるプログラムを図 1 に示す．なお，図 1 の左端にある数値は，グラフの節点の番号を示すものであって，プログラムの一部ではない．また，このプログラムの PDG を，図 2 に示す．

2.1.1 VPDG の構成法

PDG の更新作業のうち最も時間を要するのは，プログラムの変更にしたがってデータ依存辺の接続換えをする作業である．例えば，文 4 と 5 の間に `plus := plus + 1;` を挿入する場合には，この文を表す新たな節点の追加のほか，文 2, 6 と自分自身からのデータ依存辺，文 6, 8 へのデータ依存辺を追加し，さらに，文 2 から 6 へのデータ依存辺を削除する必要がある．

この作業は，データフロー解析に基づいて行なわれる．データフロー解析は，変数を使用 (use) する文におけるその変数値が，どの文で定義 (define) されたかを知ることでもある．そこで，データ依存辺を変数ごとにまとめてみると見通しがよくなる．図 3 に，PDG のデータ依存辺を変数ごとにまとめたものを示す．図中，同一変数にかかわるデータ依存辺を楕円で囲んだ．

この各楕円内のデータ依存関係について，もう少し詳しく見てみる．この依存関係は，当該変数について，制御フローに基づいて計算される．これを見易く表示すると，図 4 のようになる．図中の d および u でラベル付けされた節点は，値が定義 (define) される節点，および，使用 (use) される節点であることを示す．さらに，変数 plus に関係する部分について，制御フローを省略せずに書いた図を図 5 に示す．なお，ラベルのない節点は，if 文や while 文の範囲を示すための節点である．図 5 のようにして，すべての変数について制御フローを省略せずに構成したグラフを VPDG と呼ぶ．

2.1.2 VPDG の更新

さて，この VPDG 上で，プログラムの更新作業がどのように反映されるかを見てみる．先ほど述べた例，文 4 と 5 の間に `plus := plus + 1;` を挿入する場合と，挿入後に同じ文を削除する場合を考える．

挿入する文に出現する変数は plus だけなので，図 5 の範囲だけを更新すれば良いことがわ

```
1:   dummy := 0;
2:   plus := 0;
3:   read(val);
4:   while val <> 999 do
5:      if val > 0 then
6:         plus := plus + val
       endif;
7:      read(val);
     done;
8:   write(plus);
9:   write(dummy);
```

図 1: 例題プログラム 1

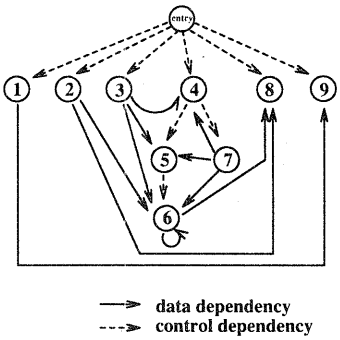

図 2: 例題プログラム 1 の PDG

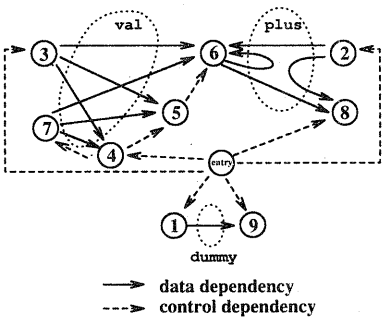

図 3: 図 2 の PDG を変形したもの 1

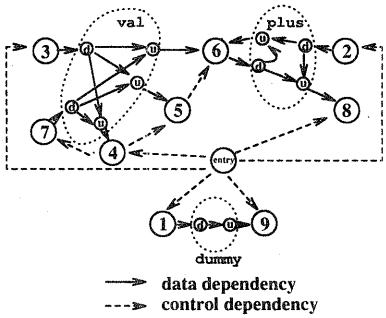

図 4: 図 2 の PDG を変形したもの 2

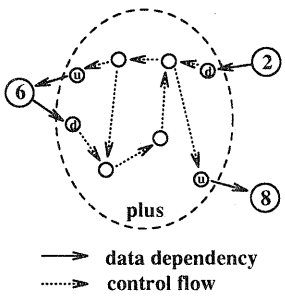

図 5: 図 4 の変数 plus 部分の詳細

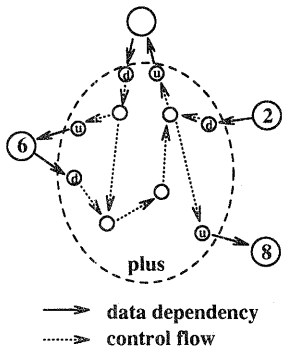

図 6: 文 plus := plus + 1; 挿入後の VPDG

かる．また，文4と5，すなわち while 文と if 文の間に挿入するので，図5上部のラベルのない節点の間を変更すれば良いこともわかる．挿入する文には，左辺と右辺にそれぞれ一度ずつ変数が現れるので，d と u のラベルを持った節点がそれぞれひとつ作られることになる．よって，更新されたグラフは，図6となる．

削除する場合には，まず，削除する文に相当する節点（この場合，図6最上部のラベルのない節点）を特定する．この節点が入力辺と出力辺を持つ場合には，当該辺と，その先の u および d のラベルが付いた節点まで削除し，その u および d 節点に接続されていた制御フロー辺を単純に継ぐことによって削除が完了する．また，入力辺または出力辺だけを持つ場合には（図6では，2や8のラベルを持つ節点が相当する），当該辺と，その先の u または d 節点および，それらに接続されている制御フロー辺を単に削除すればよい．

このように，プログラムの変更にともなう作業について，VPDG 上では，PDG のようなデータフローの再計算といった複雑な作業は必要がない．節点と辺を単純に追加したり削除したりするだけである．この作業は，追加・削除する節点と辺の数に比例する時間を要するだけであり，一般的には定数時間で済むと考えてよい．

2.1.3 PDG の再構成

与えられた VPDG から元の PDG を再構成することは容易である．すなわち，制御フロー辺だけを用いて，d のラベルの付いた節点から，d を経由せずに可達である u の節点を計算すればよい．これは単純な深さ優先探索によって可能である．

2.2 複数関数の VPDG

関数呼び出しを含むプログラムの VPDG 構成法について，図7の例題プログラムをもとにして述べる．

図7の PDG(複数関数のプログラムに対するものは，System Dependence Graph; SDG とも呼ばれる) は図8のようになる．これを 2.1 節で述べた方法によって自然に拡張したもの (図9) が，そのまま複数関数のプログラムに対する VPDG となる．

2.3 大域変数の扱い

複数関数で，かつ，大域変数を持つプログラムにおける VPDG の構成法について，図10の例題プログラムをもとにして述べる．

2.3.1 VPDG の構成法

図10の PDG を図11に，VPDG を図12に示す．PDG では，関数内部で大域変数を参照 (定義) する場合には，大域変数も当該関数に引数として与えられているかのごとく扱うために，特殊な節点 global-in (global-out) を導入する．これに対し，VPDG での大域変数の扱いは，複数関数を扱う場合と異なり，PDG の自然な拡張とはなっていない．しかし，大域変数の値の流れに関して言えば，VPDG のほうが直観的に理解し易い．そればかりか，関数内で当該大域変数が使用されたり定義されたりするかどうかのチェックをする必要もない (PDG の global-in, global-out 節点を作成するかどうか判断するためには，このチェックが必要となる)．

なお，図12中の最下部にある，c と r のラベルが付いた節点は，それぞれ関数の呼び出しと戻りを示す節点である．スライス計算等のために VPDG を探索する際には，c と r のラベルが付いた節点において，関数を呼び出した順に戻るような制御をしなくてはならない．これは，探索時に別のスタックをひとつ用意することによって実現できる．

2.3.2 VPDG の更新

PDG において，大域変数にかかわるプログラム変更が生じたときには，global-in, global-out 節点の追加，削除と，それにともなうデータ依存辺の接続換えが必要になることがある．global-in, global-out 節点の追加，削除が必要かどうかはプログラムの変更内容から容易にわかるので，データ依存辺の接続換えに要するコストが問題となる．

一方，VPDG では，2.1.2 節で述べた通り，単純な操作だけで済み，辺の接続換えにかかわ

```
    program
1:      call mult2(P, area);
2:      call mult2(area, vol);
    end;
3:  procedure mult2(op1, result);
4:      result := op1 * 2;
    end;
```

図 7: 例題プログラム 2

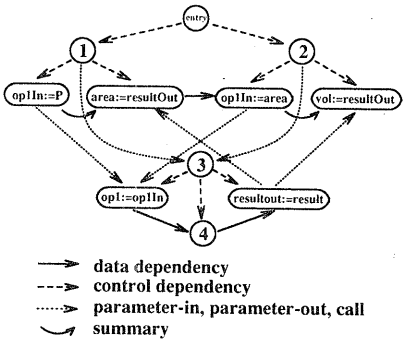

図 8: 例題プログラム 2 の PDG

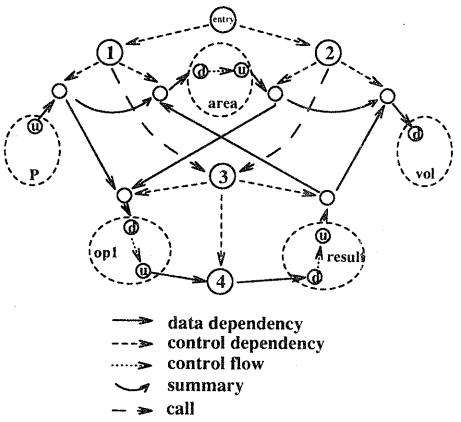

図 9: 例題プログラム 2 の VPDG

```
    program
        var gl;
5:      gl := 3;
1:      call mult2(P, area);
2:      call mult2(area, vol);
6:      writeln(gl);
    end;
3:  procedure mult2(op1, result);
4:      result := op1 * 2;
7:      gl := gl + 1;
    end;
```

図 10: 例題プログラム 3

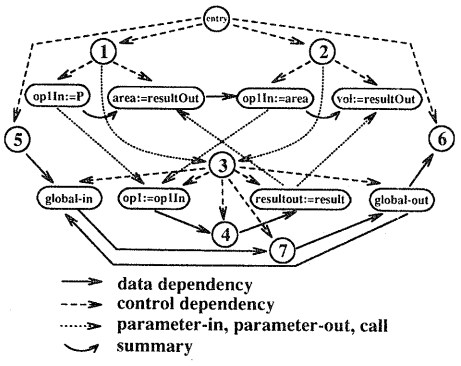

図 11: 例題プログラム 3 の PDG

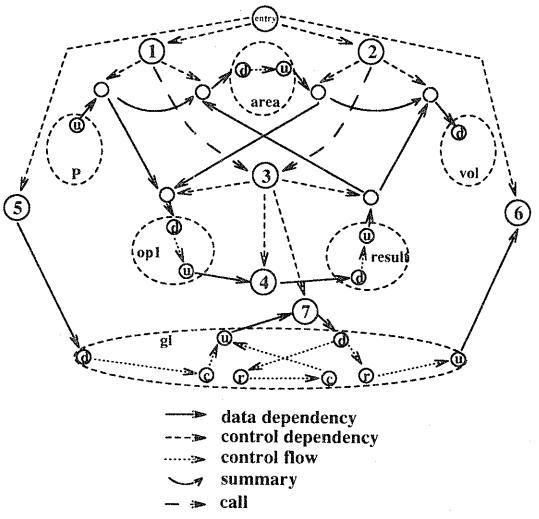

図 12: 例題プログラム 3 の VPDG

表 1: PDG と VPDG の節点と辺の数の比較

プログラム名	関数	文 (制御 , 代入)	変数	PDG 節点 (A)	PDG 辺 (B)	VPDG 節点 (C)	VPDG 辺 (D)	倍率†
rain	1	13 (2 , 8)	4	14	31	39	59	2.18
ave	1	11 (2 , 7)	8	12	33	51	77	2.84
stat	4	28 (5 , 15)	13	46	86	168	214	2.89
score	3	31 (7 , 19)	13	57	137	169	250	2.16
score2	1	28 (7 , 19)	8	29	94	128	212	2.76
gen	1	400 (67 , 333)	20	401	1478	3900	4651	4.55

† 倍率 = (C+D)/(A+B)

るコストは問題にならない.

2.3.3 PDG の再構成

大域変数を持つプログラムの VPDG から PDG を再構成するには, 大域変数にかかわる部分 (図 12 では gl で示した楕円の部分) について, 次の操作を行なえばよい.

まず, c のラベルが振られた節点の全てについて, それに対応する r のラベルが振られた節点に向かってデータ依存辺と制御フロー辺をたどる. その際, d のラベルが振られた節点については global-out 節点に, u のラベルが振られた辺については global-in 節点に変更する (図 12 では, 節点 7 に直接接続している u と d の節点がこれに相当する). このとき, これらの節点に対して制御依存辺を接続する必要があるが, これは, 当該 u と d の節点に相当する文が属する関数に相当する節点 (図 12 では 3 の節点) から接続することになる. 次に, 制御フロー辺だけを用いて, d および global-out のラベルの付いた節点から, d および global-out 節点を経由せずに可達である u および global-in の節点を求める. 2.1.3 節で述べたように, これは, 単純な深さ優先探索によって可能である.

3 探索効率

プログラムスライス [9] を求めたり, プログラム変更にともなう影響を調査したりするために PDG を利用する場合には, その探索が行なわれる. PDG のかわりに VPDG を利用する場合には, その探索の効率が問題となる.

PDG や VPDG のような疎グラフの探索は, 節点と辺の合計数に比例する時間が必要であることが知られている [10]. そこで, 探索にかかる時間を見積るために, 市販されている書籍の中から選んだいくつかの例題プログラムについて, PDG と VPDG を構成して節点と辺の数を数えた. この結果を表 1 に示す. なお, 表中のプログラム score2 は, score の関数呼び出しを展開して単一関数にした, 同一内容のプログラムである.

探索効率を評価するために, VPDG の節点と辺の合計数を PDG の節点と辺の合計数で割ったものを用いる. 以下, これを「倍率」と表す. 倍率が N であれば, VPDG の探索には PDG の探索の約 N 倍の時間を要すると見積ることができる.

この表から, 次のことがわかる.

- 相当大きな関数 (gen) でも, 倍率は数倍程度でおさまる.
- 文の数が 30 程度までの小さな関数を組み合わせて作ったプログラムの場合, 倍率は 2 から 3 の間となる.

一般に, プログラムは, 機能を明確に定めた, あまり大きくない関数の集まりとして作成することが多い. したがって, 数倍のメモリ使用量と数倍の探索時間が許容できる環境を仮定した場合, プログラムの変更が頻繁に生じる状況では, プログラム変更をグラフに反映させるコストがより低い VPDG を使うことによって, システムの応答時間を改善することができる. このことは, VPDG をデバッグ環境において利

```
1:  v := 0;
2:  while v < 999 do
3:     v := v + 1;
4:     if x > 100 then
5:        x := 100;
       endif;
    done;
```

図 13: 例題プログラム 4

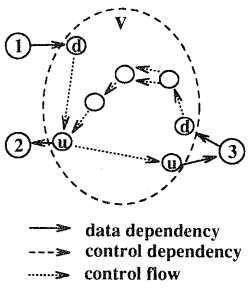

図 14: 例題プログラム 4 の VPDG

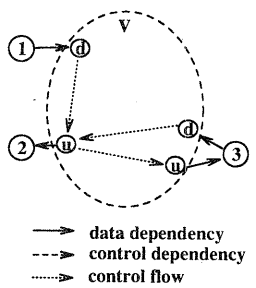

図 15: 図 14 の縮約グラフ

用する場合には，特に威力を発揮することを意味する．

4 VPDG の縮約

数倍程度の倍率が許容できない場合，グラフ変更の効率を多少儀牲にして，VPDG の節点と辺を削除することもできる．例えば，図 13 のプログラムを例にとると，VPDG の中で，変数 v にかかわる部分は図 14 のようになるが，これを図 15 のように，変数 v に影響を与える制御文にかかわる節点だけを残して縮約しても，データ依存関係の計算に支障はない．ただし，図 13 の 5 行めの直後に `v := 3;` を挿入しようとした場合に，内側の `if` 文に相当する節点も作成しなくてはならず，その分だけコストがかかることになる．

表 1 と同じプログラムに対して，このような縮約をした場合の結果を表 2 に示す．プログラム ave のように，プログラムによっては全く縮約できない場合もあるが，大きな関数ほど縮約の効果があると言える．この理由は，一般に，制御文の入れ子の深い部分で定義・参照される変数の数は少ないためである．

この表から，縮約 VPDG は，オリジナルの PDG に比べて，記憶空間の容量と探索に要する時間をおよそ 3 倍以内に抑えることができると言える．

表 2: 縮約 VPDG の効果

名前	VPDG			縮約 VPDG		
	節点	辺	倍率	節点	辺	倍率
rain	39	59	2.18	37	57	2.09
ave	51	77	2.84	51	77	2.84
stat	168	214	2.89	152	198	2.65
score	169	250	2.16	157	238	2.04
score2	128	212	2.76	92	176	2.18
gen	3900	4651	4.55	2480	3231	3.04

5 むすび

本稿では，プログラム依存グラフの一変種を示した．プログラム依存グラフがデータ依存辺の計算過程を全て捨て，最終結果だけを残してあるのに対して，この変種は，データ依存辺を引く計算過程をグラフの形で残したものである．その結果として，プログラム変更にともなうグラフ更新作業が容易になる．一方で，グラフの節点と辺の数が増加するため，グラフを保持するための記憶容量と探索にかかる時間は数倍かかる．ただし，グラフを縮約することによって，3 倍程度にまで抑えることができる．また，この変種からプログラム依存グラフを再構成することは容易である．

この変種は，データ依存関係の計算を，本当に必要になるまで — グラフを探索するときまで — 遅らせるようにしたものであると見ることができる．文献 [8] の方法との違いは，この点にある．

参考文献

[1] K. Ottenstein and L. Ottenstein, "The program dependence graph in a software development environment," ACM SIGPLAN Notices, vol. 19, no. 5, pp. 177–184, 1984.

[2] A. Aho, R. Sethi, and J. Ullman, "Compilers: Principles, Techniques, and Tools," Addison

Wesley, 1986.

[3] S. Horwitz and T. Reps, "The use of program dependence graphs in software engineering," Proc. of the 14th International Conference on Software Enginerring, pp. 392–411, 1992.

[4] 下村隆夫, "プログラムスライシング技術と応用, " 共立出版, 1995.

[5] B. Korel, "The program dependence graph in static program testing," Information Processing Letters, vol. 24, pp. 103–108, 1987.

[6] L. Pollock and L. Soffa, "An incremental version of iterative data flow analysis," IEEE Trans. on Softw. Eng., vol. 15, no. 12, pp. 1537–1549, 1989.

[7] G. Ryder and C. Pall, "Incremental data-flow analysis algorithm," ACM Trans. on Programming Languages and Systems, vol. 10, no. 1, pp. 1–50, 1988.

[8] 高田智規, 佐藤慎一, 井上克郎, "プログラム依存グラフの効率的な更新手法, " 信学論 (D–I), vol. J81–D–I, no. 3, pp. 253–260, 1998.

[9] M. Weiser, "Program slicing," IEEE Trans. on Softw. Eng., vol. SE–10, no. 4, pp. 352–357, 1984.

[10] 例えば R. Sedgewick, "Algorithms, 2nd ed.," Addison-Wesley (1988).