人工知能学会研究会資料SIG-IES-9901-2, pp.9-16 (1999)

いかにプログラム空欄補充問題を作るか？

柏原昭博，寺井淳裕，豊田順一

大阪大学産業科学研究所

〒567-0047　大阪府茨木市美穂ヶ丘8-1
TEL. 06-879-8426　　E-mail. kasihara@ai.sanken.osaka-u.ac.jp

プログラム空欄補充問題では,空欄を埋める過程で自ずとプログラムの処理過程を規定するアルゴリズムを考える機会を持つことができ，理解を強化することができると期待される．本論文では，こうしたプログラム空欄補充問題を作る方法について述べる．特に, PDGを用いてプログラムの処理過程の要所を特定し，それを空欄とする方法について述べる．本手法の特徴は，アルゴリズムの内容まで立ち入らずに，形式的に空欄を設定できる点にある．また，本手法の妥当性を調べた実験についても述べる．この実験では,アルゴリズムを熟知した被験者に設定してもらった空欄箇所を本手法で説明できるかどうかを確かめた．その結果，空欄設定方法の妥当性を確認することができた．

プログラム空欄補充，アルゴリズム理解，認知負荷，PDG

Making Fill-in-Blank Program Problems

Akihiro Kashihara, Atsuhiro Terai, and Jun'ichi Toyoda

I.S.I.R., Osaka University, JAPAN

8-1, Mihogaoka, Ibaraki, Osaka 567-0047
TEL. 06-879-8426　E-mail. kasihara@ai.sanken.osaka-u.ac.jp

Fill-in-blank program problem encourages learners to fill in part of the program that is blanked out and to have a deeper understanding of algorithm embedded in the program. In this paper, we discuss a method of blanking out an important point of data or control flows of the program with PDG (Program Dependence Graph) to make an instructive fill-in-blank problem. The key point of this method is to make a blank in no consideration of semantic aspects of the algorithm. This paper also describes a preliminary experiment on the blank-making method with subjects who are familiar with algorithms. In this experiment, we have ascertained that blanks made by hand follow the blank-making method.

Fill-in-Blank Program Problems, Algorithm Understanding, Cognitive Load, PDG

1. 序論

アルゴリズムの理解支援は，情報教育における重要な課題の一つであり，これまでにも流れ図（フローチャート）表現，処理の流れを視覚化するアニメーションなど様々な支援方法が提案されている[1-3]．筆者らも，アルゴリズム理解の強化を主目的として，プログラム中に空欄を設けて，それを埋めるように問う問題（空欄補充問題）を用いる試みを行っている[4,5]．

プログラム空欄補充問題では，プログラムの機能記述（プログラム仕様）とともに，一部が空欄となったプログラムが与えられる．空欄を埋めるためには，空欄の前後における処理の流れを追いながら必要な情報を得ることが求められる．こうした問題解決過程において，自ずとプログラムの処理過程（アルゴリズム）を考える機会が増え，アルゴリズム理解が強化されることが期待できる．文献[5]でもアルゴリズム理解に貢献することを実験的に確かめている．

一方，空欄の設け方によっては，理解を強化できなかったり，あるいは空欄補充が難しくなりすぎて学習者に過負荷を与えてしまうことになる．したがって，アルゴリズム理解がうまく強化・促進されるように，空欄の設定方法を工夫する必要がある．文献[4,5]では，経験的に空欄を設定した問題を用いて学習効果を調べることに主眼を置いたが，本論文では，空欄補充問題をいかにして作るかということに焦点を当てて議論したい．特に，アルゴリズムの内容に立ち入らず形式的に空欄を設定する方法を検討する．

効果的な空欄補充問題の作成に向けて重要な点は，学習者にできるだけアルゴリズム（処理過程）を考える機会を与えるように空欄を設けることである．本研究では，プログラムにおける処理の流れの要所となる部分を空欄とする方法を提案するとともに，ＰＤＧ（Program Dependence Graph）を用いてこのような要所を見つける方法について述べる．なお，ここでは，初学者が学習するような簡単なアルゴリズムを対象とする．また，C言語で記述されたプログラムを用いる．

また，本論文では，PDGに基づく空欄の設定方法の妥当性を調べるために行った実験について述べる．本実験では，プログラミング経験がありアルゴリズムを熟知した被験者に設定してもらった空欄の箇所が，提案した空欄設定方法で説明できるかどうかを確かめた．その結果，設定方法の妥当性を確認することができた．

以下，２章ではまず空欄補充問題の効果や有効性について議論する．３章では，具体例を交えて，PDGおよび空欄の設定方法について述べる．４章では，空欄設定方法の妥当性を調べた実験について述べ，５章で本論文をまとめる．

2. 空欄補充問題

空欄設定の方法について議論する前に，プログラム空欄補充問題の効果および有効性について説明しておく．

2.1 学習効果

本研究では，プログラムの機能記述と空欄つきのプログラムを与えて，空欄の補充を要求する問題を空欄補充問題としている．図１に問題例を示す．与えられたプログラムは，バブルソートのアルゴリズムを実現している．設定された空欄は，データや制御の流れを遮断する働きがある．空欄を補充することは，この遮断された部分を補完することであり，そのためには見えているプログラムコードから処理の流れに関する情報を集めることが必要となる．例えば，プログラムコードにおける変数の役割やステートメント間の因果関係を見いだしたり，個々のステートメントの役割やいくつかのステートメントを一つのブロックとみなして，その役割を考えるようなことが必要となる．

こうした空欄補充問題によって，処理過程を規定するアルゴリズムに対する理解の程度を「確認」できるだけでなく，空欄補充を通して理解を「深める」ことができる．なお，アルゴリズムを未学習あるいは理解が十分でない初学者

[問題] 次のプログラムは，入力列tbl[i] (i=1...n)を昇順に整列するプログラムである．空欄を埋めなさい．

```
#include <stdio.h>
#define MAX 5
main ()
{
  int i, j;
  int tbl[MAX], n;
  for (i=0;i<MAX; i++) {
    printf("Input number tbl[%d] = ", i);
    scanf("%d", &tbl[i]); }
  for (i=0; i<MAX; i++){
    for(j=i+1; j<MAX; j++)
      if (              ){
        n = tbl[j];
        tbl[j] = tbl[i];
        tbl[i] = n; } }
  for (i=0; i<MAX; i++)
    printf("tbl[%d] = %d\n", i, tbl[i]);
}
```

図１ プログラム空欄補充問題

で，プログラミング言語についての文法知識を有している学習者を想定している．また，理解させようとするアルゴリズムは，リスト処理のように簡単なものを想定している．

2.2 有効性

ここでは，従来のアルゴリズム理解支援方法との比較を通して，空欄補充問題の有効性を考えてみたい．流れ図やアニメーションによる理解支援では，運用の仕方にもよるが，基本的にデータや制御の流れを見せることで支援しているといえる[1,2]．一方，空欄補充問題は，プログラムコードから処理の流れを自力で見つけだすことを求める問題であり，学習者の理解に適度な負荷を与えることを意図している．そのため，理解の強化に適した問題であるといえよう．

また，流れ図の一部を空欄とした問題を考えることもできるが，流れ図の場合は処理の流れの骨格が明示されているため，比較的空欄補充は簡単となる．データや制御の流れに注意を向ける度合いを大きくして理解の強化を図るのであれば，プログラム空欄補充問題のほうが適していると考えられる．ただし，空欄の設定によっては，アルゴリズム理解を強化できないことも起こる．以下では，効果的な問題を作るための空欄設定方法について検討する．

3. 空欄設定

プログラム空欄補充問題を考える場合,「どこに」空欄を設定するかが重要となる．特に，データや制御の流れを考える機会を増やし，アルゴリズム理解が強化されるように空欄を設けることが望まれる．そのためには，処理の流れの要所となる場所を空欄にすればよいことになる．このような観点から，PDGを用いて処理の流れの要所を見つける手法について述べる．その前に，空欄設定に関わるいくつかの制約について述べておく．

3.1 空欄設定の制約

空欄補充問題では，空欄の場所以外にも，空欄のサイズおよび個数が問題解決に大きく影響する．本研究では，空欄設定にあたり，問題の難しさが適度なものとなるように空欄のサイズ・個数に制約を加えている．

3.1.1 空欄のサイズ

一つの空欄に入る内容が１変数のみであったり，１ステートメントあるいは１ブロックであるような空欄補充問題を想定することができる．こうした空欄のサイズが大きいほど，補充に必要な情報を得るのが難しくなるといえる．また，空欄内のコーディング作業が増えるため，処理過程の理解に集中するのが難しくなる．したがって，初学者を対象とする場合には，空欄のサイズを大きくすることは望ましくない．そこで，本研究では，一つの空欄に含まれる内容をせいぜい１ステートメントに限定している．

3.1.2 空欄の数

空欄のサイズが小さくても，空欄の数が多くなると，データや制御の流れが追いにくくなり，空欄補充に必要な情報を見つけるのが難しくな

```
1: x=1;          1: if (x=1)
2: x=2;          2:     y = x;
3: y = x;        3: x = x + 1;
```

(a) データ依存関係　　(b) 制御依存関係

図 2 依存関係

ることが予想される．そこで，本研究では，過度に難しくしないこと，および取り扱うプログラムのサイズが大きくないことを念頭において，空欄の数を 2 個に限定している．

文献[5]では，二つの空欄を主空欄，副空欄と呼び，重要と思われる箇所を主空欄として設定し，また主空欄の補充の難しさを変えることをねらいとして主空欄の箇所と関係（データ依存，隣接，同一ブロック内）のある箇所を副空欄として設定する方法を検討している．副空欄は主空欄が決まると形式的に決めることができるものの，主空欄についてはアルゴリズムの内容を踏まえて，処理の中でも重要と思われる箇所に設定していた．例えば，バブルソートでは，隣り合う入力列の要素の大小を逐一比較し，昇順あるいは降順となるように要素を入れ替えて整列するアルゴリズムであり，なかでも要素の大小比較する部分がこのアルゴリズムの重要なポイントであると考えられる．このような観点から，図 1 ではこの部分が空欄として設定されている．

本論文では，空欄のサイズを一文として，これまで経験的に決めていた主空欄を形式的に決める方法について議論する．

3.2 PDG（Program Dependence Graph）

ここでは，プログラム中の処理の流れの要所となる部分を見つけるために用いるPDGについて簡単に説明する．

PDG とは，プログラムの各文（ステートメント）をノード，文間の依存関係をアークとする有向グラフであり，データや制御の流れを定量化することができる表現方法である[6,7]．文間の依存関係には，(i) データ依存関係，(ii) 制御依存関係，がある．

文間（文 s1 と文 s2）のデータ依存関係とは，次のような条件が満たされる場合をいう．

- s1 において変数 v が定義されている．
- s1 から s2 へのプログラムの実行可能パス内に v を再定義している文は存在しない．
- s2 において変数 v が参照されている．

例えば，図 2 (a)に示すプログラムでは，変数 x について文2から文3にデータ依存関係が存在することになる．なお，文 1 から文 3 については，文 2 において変数 x が再定義されているため，データ依存関係は存在しない．

次に，文間（文 s1 と文 s2）の制御依存関係とは，次の条件が満たされる場合をいう．

- s1 は条件文かループ文である．
- s2 が実行されるかどうかは s1 の文が実行されるかどうかに直接依存する．

図 2 (b)に示すプログラムでは，文 1 の実行結果が文 2 を実行するかどうかを左右することになるため，文1から文2に制御依存関係が存在することになる．

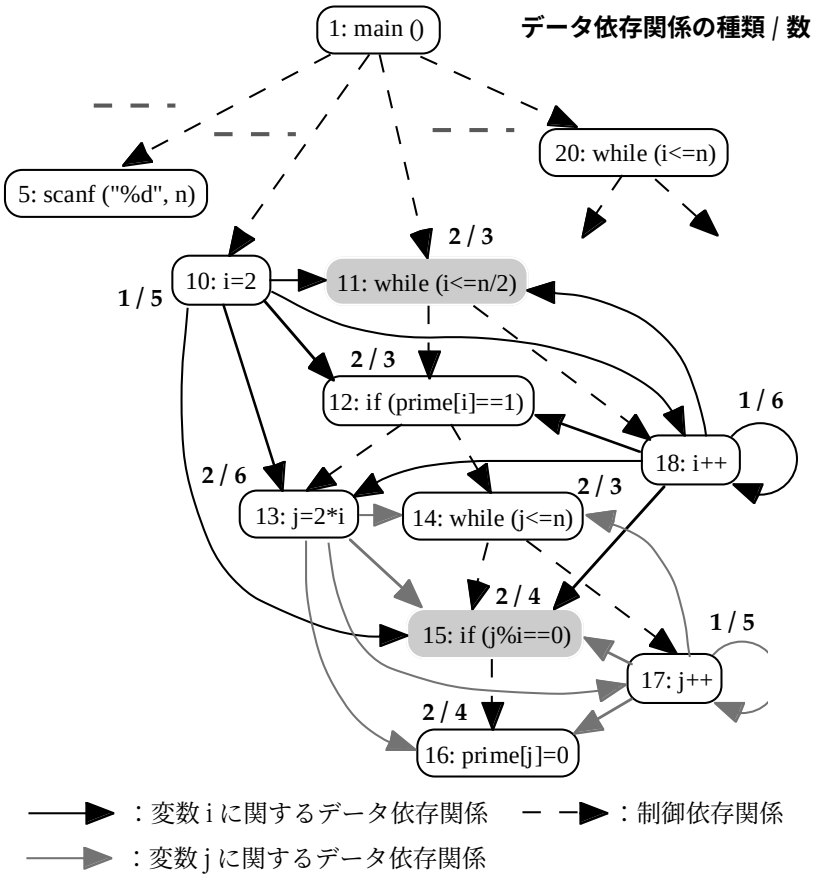
図 3 に素数を求めるプログラムとそのPDG表現を示す．ここでは，プログラムの 10 行目から 18 行目までの部分の PDG を示している．この図が示すように，制御依存関係はしばしば被制御文が制御文となるような入れ子構造を形成する．この構造をプログラムの制御構造と呼ぶ．また，データ依存関係は，文中の変数ごとにアークが付けられる．図 3 (b)では，変数 x と変数 y に限定してデータ依存関係を表しており，他の変数に関する依存関係については省略している．

3.3 設定方針

PDG では，ノードに出入りするアークの種類・本数によって，処理過程おける各文の重要度を定量的に判断することができる．つまり，アークの種類・数が多いノードほど，処理の流れの中心となる文であると考えることができる．このようなノードを空欄とすることで，そのノードと依存関係にある文まで参照するように

```
     #include <stdio.h>
     #define NUM 1000
1:   main ()
2:   {
3:     int n, i, j, prime [NUM+1];
4:     printf ("Input natural number: ");
5:     scanf ("%d", &n);
6:     i=2;
7:     while (i<=n) {
8:       prime[i] =1;
9:       i++;}
10:    i=2;
11:    while (i<=n/2) {
12:      if (prime[i]==1) {
13:        j=2*i;
14:        while(j<=n){
15:          if (j%i==0)
16:            prime[j]=0;
17:          j++;}}
18:      i++;}
19:    i=2;
20:    while(i<=n) {
21:      if (prime[i]==1)
22:        printf(""%5d", i)
23:      i++;} }
```

(a)素数を求めるプログラム



(b)PDG表現

図3 PDG の例（1）

仕向けることができ，アルゴリズムを考えさせる機会を増やすことができる．特に，制御文が空欄である場合は，処理の流れ方についてより注意深く考える必要が出てくるため，制御文以外の文を空欄とするよりも，アルゴリズム理解の強化にとってより効果があると考えることができる．このような観点から，本研究では次に示すように，制御文（if 文，while 文）と制御文以外に分け，それぞれ論理積で表される条件を数多く満たすノードを，プログラムにおける処理過程の要所となる部分とみなしている．

- 制御文で要所となるノード

制御構造上より深い場所にあるノード　&
データ依存関係アークの種類が多いノード　&
データ依存関係アークの数が多いノード

- 制御文以外で要所となるノード

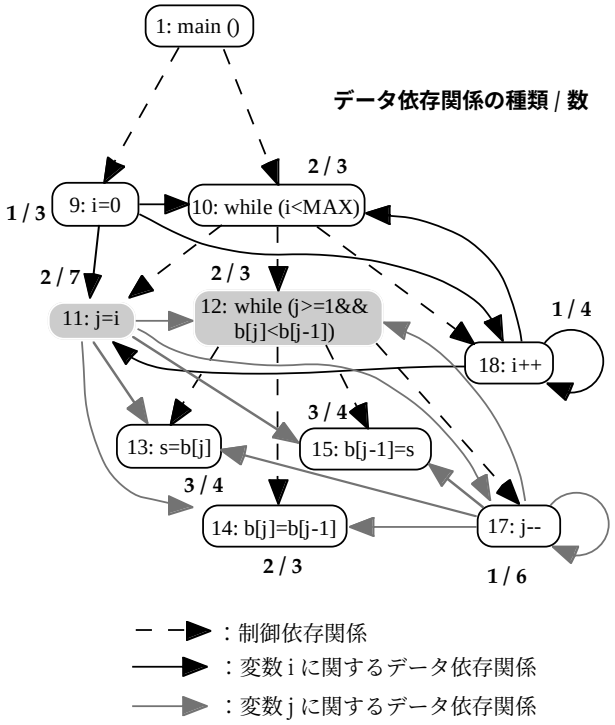
データ依存関係アークの種類が多いノード　&
データ依存関係アークの数が多いノード

なお，現時点では，どの条件を優先して空欄を設定するかについては考慮していない．

図３を例にして要所となる部分を考えてみよう．図３の各ノードの右肩あるいは左肩にノードに出入りするデータ依存関係の種類と数を記述している．制御文で要所となる候補としては，制御構造上最も深い位置にあり，データ依存関係アークの種類が2で，数が4の15行目の文（if (j%i == 0)）が挙げられる．制御文以外では，データ依存関係の種類・数が最も多い（種類が2，数が6）13行目の文（j=2*i）が候補として挙げられ，また，データ依存関係の種類が同じで数が4である16行目の文（prime[j]=0）も次の

```
     #include <stdio.h>
     #define MAX 5
 1:  main ()
 2:  {
 3:    int i, j, s, b [MAX];
 4:    i=0;
 5:    while (i<MAX) {
 6:      printf("Input number b[%d]", i);
 7:      scanf ("%d", &b[i]);
 8:      i++;}
 9:    i=0;
10:   while (i<MAX) {
11:     j=i;
12:     while (j>=1&&b[j]<b[j-1]){
13:       s=b[j];
14:       b[j]=b[j-1];
15:       b[j-1]=s;
16:       j--;}
17:     i++;}
18:   i=0;
19:   while(i<MAX) {
20:     printf("%5d", i)
21:     i++;} }
```

(a)整列プログラム

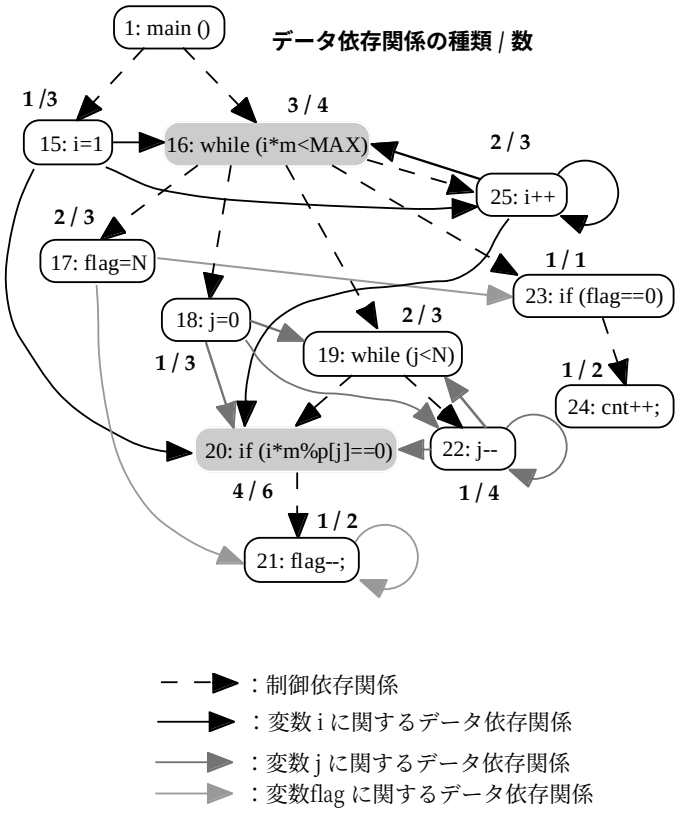


(b)PDG表現

図4 PDG の例（2）

```
     #include <stdio.h>
     #define N 1000
     #define MAX 1000
 1:  main ()
 2:  {
 3:    int i, j, m, flag, p [N], cnt=0;
 4:    i=0;
 5:   while (i<N)
 6:      printf("Input natural number: ");
 7:      scanf ("%d", &p[i]);
 8:      i++;}
 9:    m=p[0];
10:   i=1;
11:   while (i<N)
12:     if (p[i]>m)
13:       m=p[i];
14:     i++;
15:   i=0;
16:   while (i*m<MAX){
17:     flag=N;
18:     j=0;
19:     while (j<N)
20:       if (i*m%p[j]==0)
21:         flag--;
22:       j++;
23:     if (flag==0)
24:       cnt++;
25:     i++;
26:   printf("Number of common multiple is %d", cnt)}
```

(a)公倍数を求めるプログラム



(b)PDG表現

図5 PDG の例（3）

表１ 実験結果

アルゴリズム	選択された文	選択回数	制御文	制御構造上の位置	データ依存関係 種類・関係
素　数	15: if (j%i==0)	7	○	最も深い	最多・最多
	11: while (i<=n/2)	4	○	―――	最多・２番目
整　列	12: while(j>=1&&b[j]<b[j-1])	9	○	最も深い	最多・最多
	11: j=i	3	×		２番目・最多
公倍数	16: while(i*m<MAX)	6	○	―――	２番目・２番目
	20: if(i*m%p[j]==0)	5	○	最も深い	最多・最多

候補として挙げられる．

以上のように候補にあがった文をアルゴリズムの内容から考えてみよう．図３の「ｎまでの自然数の中から素数を求める」アルゴリズムでは，２からｎ/２まで順に，その倍数がｎまでの範囲にあれば消去し，最終的に残った数を素数としている．このアルゴリズムの中で重要と思われる点は，倍数であるかどうかを判断する部分（15行目）と，倍数を見つける範囲を決める部分（13行目），どの範囲の倍数を求めるのかを決める部分（11行目）であろう．また，素数かどうかのフラグのセット，リセットも（特にリセット）大切な部分と考えられる（16行目）．これらの部分は，11行目を除き，PDGで形式的に重要と考えた部分に相当しており，PDGに基づく空欄設定方針がほぼ妥当であるということを示唆するものとなっている．

4. 妥当性検証

4.1 実験

実験の目的は，PDGをもとに立てた空欄設定方針の妥当性を調べることにある．本実験では，プログラミング経験がありアルゴリズムを熟知した理工系大学生，大学院生および大学教員10名を被験者として，プログラム中の一文に空欄を設定してもらい，設定箇所が，提案した空欄設定方針に沿っているかどうかを確かめることで妥当性を評価する．

各被験者には，(i) 素数を求める，(ii)整列する，(iii)公倍数を求める，という３つのアルゴリズムをそれぞれフローチャートで見せて確認をしてもらった後，初学者の理解を強化するためにはプログラム中のどの文に空欄を設ければよいかという質問を与え，空欄を選択してもらった．この際，PDGは見せなかった．空欄の設定は，１つのプログラムに対して２回試行してもらった．（１カ所空欄にしてもらった後，再び空欄のないプログラムから別の箇所を１カ所選んでもらった．）

4.2 結果と考察

表１に実験結果を示す．表では，空欄として選択された回数が多かった文を上位２つ示している．また，選択された文が制御文かどうか，制御構造上の位置およびデータ依存関係の種類・数の多さを記述している．図３，図４，図５に実験で用いたプログラムとそのPDGを示しているが，PDG中の網掛けされたノードが実験で選択回数の多かったノードを表している．

実験結果で明らかなように，制御文が空欄として選択される場合が多いことがわかる．これは当初予想していた通りで，処理の流れの仕方に注意を向けさせることができるためと考えられる．また，PDGにおいてデータ依存関係の種類・数ともに最多あるいは２番目に多い文が選択されており，予想どおり処理の流れの要所がアルゴリズム理解を強化する上で空欄とすべき

箇所と見なされている．制御構造上の位置については，各アルゴリズムにおいて，一番深い制御文が選ばれているものの，位置に関係のないところも選択されている．制御構造上の位置が空欄設定条件になりうるかどうかについては，さらに実験を重ねる必要がある．

5. 結論

本論文では，アルゴリズムの理解の確認および強化を主目的としたプログラム空欄補充問題を作成する方法について述べた．特に，PDG を用いてプログラムの処理過程の要所を特定し，それを空欄として設定する方針を立てた．本手法の特徴は，アルゴリズムの意味的な内容までに立ち入らずに，空欄を設定できる点にある．また，本手法の妥当性を調べた実験についても述べた．実験の結果，空欄設定方針が妥当であるという見通しを得ることができた．今後，実験を重ねることで空欄設定方針の妥当性を確認するとともに，さらに洗練したい．

謝辞

本研究の一部は，（財）カシオ科学振興財団の援助による．また，本研究の実験にご協力をいただいた，知的教育システムに関わる若手研究者のメーリングリスト参加者に感謝いたします．

参考文献

[1] Merrill,D.C., Reiser,B.J., Beekelaar,R., & Hamid.A.: Making Processing Visible: Scaffolding Learning with Reasoning-congruent Representations, Proc. of 2nd International Conference on ITS (ITS ユ92), pp. 103-110 (1992).

[2]松田憲幸, 柏原昭博, 平嶋宗, 豊田順一: プログラムの振舞いに基づく再帰プログラミングの教育支援, 電子情報通信学会論文誌 Vol. J80-D-II, No.1, pp.326-335 (1997).

[3]井上勝行，上和田徹，魚井宏高，首藤勝：可視化技術を援用したアルゴリズム自習支援環境の構築, 教育システム情報学会誌, Vol. 15, No.2, pp.65-74 (1998).

[4]M.Soga, A.Kashihara, and J.Toyoda: Adaptive Fill-in-blank Program Problems from The View of Cognitive Load and Application Systems on WWW, Proc. of ICCE'95, pp.575-582 (1995).

[5]柏原昭博, 曽我真人, 豊田順一: 空欄補充問題を用いたプログラム理解支援, 教育システム情報学会誌, Vol. 15, No.3, pp.129-138 (1998).

[6] 下村隆夫: "Program Slicing 技術とテスト，デバッグ，保守への応用 ", 情報処理学会誌, Vol. 33, No.9, pp.1078-1086 (1992).

[7] Horwitz,S., Reps, T., and Binkley, D.: Interprocedural Slicing Using Dependence Graphs, ACM Transactions on Programming Languages and Systems, Vol.12, No.1, pp.26-60 (1990).