論　文

グラフ処理プログラム —GRAMP*—

滝内政昭** 高見沢一彦***
西関隆夫*** 斎藤伸自***

Abstract

We have developed a program package GRAMP (Graph Manipulating Programs), which is suitable for describing algorithms of graph or network problems. In this paper, we show its implementation, facilities and how to use it. Its main features are:

(1) It has storing area and working area, and it can manage several graphs at the same time;

(2) every command has adequate and reasonable computational time complexity;

(3) it has a part for the interactive (or conversational) use of a graphic display system;

(4) it is suitable for statistical analysis of graphs.

1. まえがき

最近，輸送計画問題など，グラフ構造を持つ種々の問題が，計算機で処理されることが多い．

これらのどの問題においても，妥当な記憶容量を用い妥当な計算時間で処理するようなプログラムを作るには，グラフ構造を記憶するデータ構造に，2，3の共通な工夫が必要である．また，グラフの変形操作（枝の付加，除去等）を実現するプログラムなど，いずれの問題を処理する場合にも必要な，共通のプログラムも多い．従来は，このようなデータ構造の工夫，およびグラフの変形処理用プログラム等が個々の問題で別個に開発され，用いられてきた．

しかし，リスト処理を専門とする言語として LISP が考案されたように，グラフ処理を専門とする言語あるいはプログラムがあればデータ構造などの問題に深く立ち入ることなしにプログラムを作ることができ，プログラム製作の手間および時間の面で非常に有効であろう．

従来からも，その種の言語がいくつか発表されている[1),2),3)]．しかしほとんどの言語は言語理論などの面から研究，開発されたものであり，意味論，文法論などに片寄る傾向がみられる．しかも実際の使用の際問題となる，使用メモリ量，あるいは各操作はどの程度の計算手数のオーダでできるのかといったような問題についてはほとんど触れられていない．

本プログラム GRAMP — Graph Manipulating Programs — は，これらのことを考慮し，下記のような目的を満たすように作られたグラフ処理用プログラムである．

目　的

1. 実用的であること．
2. メモリの使用ができるだけ少ないこと．
3. 各操作が妥当な計算時間でできること．
4. ランダムグラフの発生など，グラフの統計解析が容易であること．
5. CRT を用いて，会話型処理が可能であること．

1の条件を満たすため，取り扱えるグラフは，節点数500個，枝数2,000本程度のものまでとし，複数個のグラフの格納およびそれらに対する基本的処理，変

* Graph Manipulating Programs -GRAMP- by Masaaki TAKIUCHI (Fujitsu, Ltd.), Kazuhiko TAKAMIZAWA, Takao NISHIZEKI, and Nobuji SAITO (Faculty of Engineering, Tohoku University).

** 富士通(株)
*** 東北大学工学部通信工学科

形処理，統計処理などが行えるようになっている．

2および3の条件を満たすため，両方向リストからなるデータ構造を用いている．このことにより各操作の計算手数のオーダは理論的に妥当なものと一致しており，それを前提として書かれた，グラフのアルゴリズムを本プログラムで実行する場合，そのアルゴリズムの計算時間は理論的なものと一致する．

以下において本プログラムの形式，データ構造，各操作命令の説明，およびCRTを介しての会話型処理等について順次述べる．

2. 言語形式，データ構造

2.1 言語形式

言語形式として考えられるのは，(1)従来から使われている言語 (FORTRAN, アセンブラなど) をベースとして作る，(2)グラフ処理の部分だけにプリコンパイラを付け，従来からの言語との混合形式にする，(3)独自のコンパイラを持つ総合的な形式にする，などである．GRAMPにおいては(1)の形式をとっている．すなわち，ベースの言語をFORTRANとし，各命令をサブルーチンの形で作っておき，必要なものをCALL文で呼び出すという形である．このような一番簡単な形式でも，実用上とりたてて困難は感じられず，言語製作の手間，その他の問題を考え合わせると，これで十分と思われる．ベースとしてFORTRANを選んだのは，最も一般的に使われており，アセンブラなどに比べて，計算機の機種によって影響されることが少ないという理由からである．

2.2 データ構造

計算機内でのグラフのデータ構造を，どのようにするかは，使用メモリの観点からも重要な問題である．グラフ問題でよく使われる隣接行列などの行列タイプのデータ構造は，節点数をnとすると$O(n^2)$程度のメモリを常に必要とし，しかも実際上現われるグラフは，ほとんどスパースなグラフであることを考慮すると，このような行列タイプのデータ構造を用いることは適当でない．

ここでは，各節点に隣接する枝をリストでつなぎ，その先頭番地を各節点に対応するところに入れるという，いわゆる隣接リスト構造をとっている．この場合単にグラフの構造だけを情報として蓄える場合は，各節点についての，出枝の順方向リストだけで十分であるが，グラフの種々の変形を，その妥当な計算手数で行うには，入枝のリストも必要であり，かつ両リストとも，順方向，逆方向の2方向リストにしておく必要がある．

実際の計算機内でのグラフ表現の領域は，大きく格納領域と作業領域に分けられる．格納領域は，複数個のグラフの構造を保存しておく領域であり，処理，変形などの必要なグラフは，一旦作業領域に持ってきてから処理される．このように領域を2つ取ることは，むだのようであるが，このようにしないと，あるグラフを処理，変形した後では，そのグラフの初めの構造は失われることになり，実際の使用の際不便である．ここでは2つの領域に分けたことにより，ある処理を施した後のグラフに新しい名前を付けて再び格納領域に保存するということも可能であり，グラフ相互の比較などが容易に行われる．**Fig. 1**(次頁参照)に格納領域 **Fig. 2**(次頁参照)に作業領域の例を示す．

格納領域は，グラフの構造を保存するためだけの領域であるから，必要のないものは一切省いてある．

GSTART は，各グラフに関する総合的な情報を保存するための配列であり，各グラフの節点数，枝数，有向無向の別および節点リストの先頭番地が格納されている．GPOINT は，各グラフごとの弧立節点および出枝をもつ節点のリストである．また，GSTORE 上の各節点の出枝のリストの先頭番地を含んでいる．従って，まず GSTART で節点リストの先頭番地を知り，そこから GPOINT 上の節点をリストによって1つ1つたどり，各節点について，GSTORE 上のその隣接枝を見ることによって，そのグラフ全体の構造を知ることができる．GLENG は各枝の重み，GNAME には各枝の名前が格納されている．

不要になったグラフを格納領域から消去したような場合，GPOINT, GSTORE 上に使用可能な部分ができる．これらの部分も順方向リストで常に接続しておき，そのリストの先頭番地にそれぞれ AVAILP, AVAILS という変数を割り当てている．次に別なグラフが格納領域に入力される時は AVAILP, AVAILS から節点，枝を入れてゆく．このように使用可能領域も常に管理することにより，メモリの有効利用を計っている．

作業領域は，グラフ処理の各操作の計算手数の面を考えて，格納領域上のグラフに様々な追加情報を付け加えた形になっている．VERTEX には節点に関する情報が入っており，VTOP は節点リストの先頭を表わす変数である．VERTEX が，格納領域上の GPOINT と異なる点は，VERTEX(J) には，節点Jに関

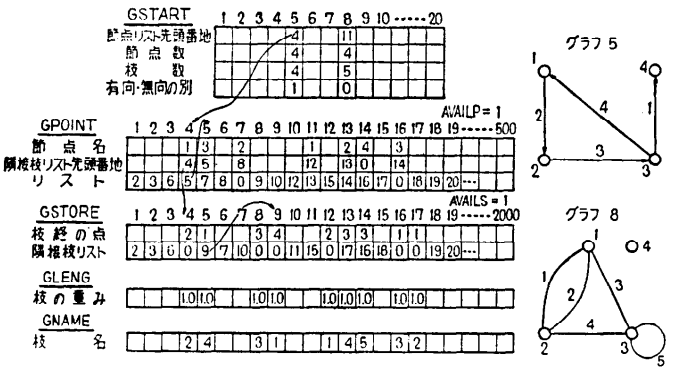


Fig. 1 Data structure of graph storing area.

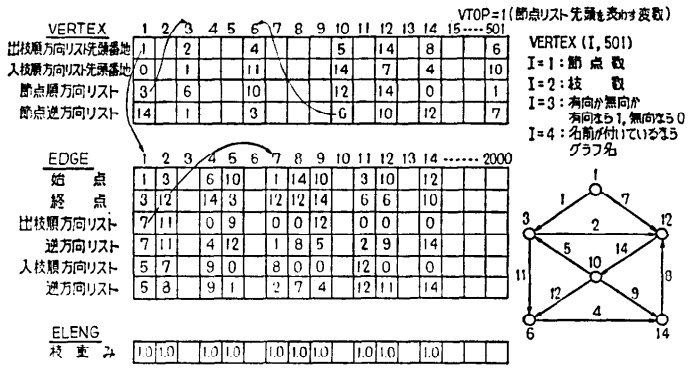


Fig. 2 Data structure of graph processing area.

する情報が入っていることである．このような対応付けを行わないと，ある節点を見つけるのに，その都度節点リストを先頭から見てゆく必要があり，計算手数の面で不都合である． EDGE には枝に関する情報が入っており，出枝，入枝の両方向リストよりなる．枝の格納の仕方は，VERTEX の場合と同じ理由から，枝Ｉは EDGE[I] に入れている．さらに本来なら無向グラフの場合，節点 v_i, v_j に枝がある場合は，それぞれ (v_i, v_j), (v_j, v_i) として同じ名前の枝が２本 EDGE 上に出てくるはずであるが，メモリの有効利用の点から，どちらか１本しか入れていない．すなわち無向グラフも有効グラフのような形で入れている．このようにしても実際グラフを取り扱う時に，枝の向きを無視すれば，何ら差し支えない．すなわち無方向グラフの場合，節点Ｊの隣接リストとしては，入枝リストも見て，その場合 EDGE 上の始点，終点を逆転させて出枝リストとして考えるわけである．この操作は EDGE が出枝，入枝両方のリストを持つために可能になるのであって，出枝リストしかないような場合は，無向グラフに関してこのような取り扱い方はできない．このことによって GRAMP においては，無向，有向を問わず，枝数2,000本までのグラフが取り扱える．

3. 操 作 命 令

3.1 命令の種類

GRAMP は，現在24種類のサブルーチンを持つ．それらは，その機能によって，次の５つの命令群に分けられる．

(1) グラフの管理に関する命令
(2) グラフの基本的変形命令
(3) 応用的処理命令
(4) 統計処理用命令
(5) 会話型処理用命令

Table 1 (次頁参照)に，上の(1)～(5)の各命令群ごとに，含まれるサブルーチンとその機能および手数のオーダを示す．

一般にグラフ処理言語に，どの程度の機能を持たせるかは，いまだ議論の余地がある．GRAMP を作るにあたっては(1)，(2)のように，グラフを取り扱うえで，基本的に必要不可欠あるいは，あった方が便利なものはすべて備える，(3)のように，ある特定の問題に対する解を求めるような命令に関しては，その最適あるいは最適に近いアルゴリズムがわかっているものについては備えてしまうという方針をとった．もちろん，(3)のような命令群まで備えなくてはならないかという疑問はあるが，別にあっても悪影響は及ぼさないであろうという考えから備えているわけである．また，(3)の命令群に関しては，備えるとすればいったい何をその出力として返すのかという問題も生じたが，使用経験に基づいて，例えば CONECT, GRANDM 命令については **Table 2** (次頁参照)，**Table 3** (次頁参照)のような出力形式にしている．(1)，(2)の命令に関しては，まだこの他にも，必要とされる命令があるかも知れないが，これからの使用過程においてそういったものが発見されれば，組み入れていく考えである．(2)の，グラフの基本的変形命

Table 1 GRAMP Subroutines.
n: the number of vertices of the graph
e: the number of edges of the graph
$O(\cdot)$: order function

	サブルーチン名	機能	手数のオーダ
(1)	CLEAR	格納領域を初期化する.	
	GINPUT	外部からグラフを格納領域へ読み込む.	$O(\max(n,e))$
	LOAD (NOG)	グラフ名NOGのグラフを格納領域から作業領域へ持ってくる. NOG＝0なら空のグラフが入る.	$O(\max(n,e))$
	STORE (NOG)	作業領域に入っているグラフにNOGという名前を付けて格納領域に返す.	$O(\max(n,e))$
	KILL(NOG)	グラフ名NOGのグラフを格納領域から消去する.	$O(\max(n,e))$
(2)	VADD (L)	節点名Lの節点を付け加える.	$O(1)$
	EADD (L, M, PL, NAME)	始点L, 終点M, 枝長PL, 枝名NAMEの枝を付け加える.	$O(1)$
	EDELI (NAME)	名前NAMEの枝を消去する.	$O(1)$
	VDEL (L)	名前Lの節点を消去する.	$O(d_L)$ d_L:節点Lの次数
	ESHORT (NAME)	名前NAMEの枝の両端点を短絡し, その場合できる自己ループを除去する.	$O(d_M)$ e_{NAME}＝(L, M)
	VSHORT (L, M)	節点LとMを短絡する. 新しくできる節点名はLとする.	$O(d_M)$
(3)	CONECT	グラフの連結性の判定および各連結成分を求める.	$O(\max(n,e))$
	BICON	グラフの非可分性の判定および各非可分成分を求める.	$O(\max(n,e))$
	STCON	有向グラフの強連結性の判定および各強連結成分を求める.	$O(\max(n,e))$
	MICST	グラフの最小コスト木を求める.	$O(e \log e)$
	SPATH	指定された1つの節点から, グラフの他のすべての節点までの最短経路およびその長さを求める.	$O(n^2)$
	MFLOW	2つの節点間の最大フローを求める.	$O(n^2 \cdot e)$
	MATCH	2部グラフの最大マッチングを求める.	$O(n^{1/2} \cdot e)$
	VCNT	グラフの節点連結度を求める.	$O(n^{1/2} \cdot e^2)$
	PLANAR	グラフの平面性を判定する.	$O(n)$
(4)	GRANDM	各種のランダムグラフを発生させる.	$O(\max(n^2, e \log e))$
(5)	DIALOG	対話型処理を行う.	
	DISPLY	作業領域上のグラフをCRT上に自動表示する. ただし, $n \leq 50$ $e \leq 100$	$O(e)$
	DISPIN	CRTより対話型でグラフを手動入力して, 作業領域上にリストおよび座標値などを入れる.	

Table 2 An explanation of arguments (1).

CONECT (NOC, KOTAE, IOPT)

1. IOPT＝1
   グラフが連結かどうかを判定する. NOC＝0 なら非連結グラフ, NOC＝1なら連結グラフ.
2. IOPT＝2
   各連結成分の節点数, 枝数が2次元配列 KOTAE に入る. NOC は連結成分数を表わす.
3. IOPT＝3
   2の動作の他に, 各連結成分に, それぞれ名前を付け, 別個のグラフとして格納領域に返す.

Table 3 An explanation of arguments (2).

GRANDM (NV, NE, NXZERO, NDRCT, NCNCT, NOPT)

NV: 節点数 NE: 枝数 NXZERO: 発生のための初期値. サブルーチンを1回呼ぶごとに異なった値が自動的に入る.

NDRCT＝{0: 無向ランダムグラフ / 1: 有向ランダムグラフ}

NCNCT＝{0: ランダムグラフ / 1: 連結ランダムグラフ}

NOPT ＝{1: 自己ループ, 並列枝共になし / 2: 自己ループなし, 並列枝あり / 3: 自己ループ, 並列枝共にあり}

NDRCT, NCNCT, NOPT の組み合わせにより12種類のランダムグラフを発生.

令の計算手数は, 理論的に妥当なものと一致させている. このことにより, これらの命令を使ってアルゴリズムを作る場合, その全体としての計算手数は, 容易に計算でき, どの程度の演算時間を必要とするかなどの情報を知ることができる.

3.2 外部からのグラフの読み込み

GINPUT ルーチンを使って外部からグラフを計算機の格納領域に入力する具体的な方法について説明する. 各グラフは, データとしてカードによって読み込まれるが, まずあるグラフの名前, 節点数, 枝数, 有向無向の別が GSTART 上に入力される. 次にそのグラフについての枝集合を入力する. すなわち, 枝の始点, 終点, 枝の重み, 枝名を1本づつの枝に関して入力し, 始点がまだ入っていない場合は, GPOINT 上の AVAILP の番地にその節点を入れ, そのグラフの節点リストと接続する. さらに枝は GSTORE 上の AVAILS の番地に入れられる. 既に節点が存在する場合は, AVAILS に枝を入れた後, その節点に関する隣接リストにその枝を加える. 節点, 枝を入れた後, AVAILS 変数を変えておくのは勿論である. ここで GPOINT 上に節点が既に存在するかどうかの判定のためには, 配列 POINT を使う. POINT は始めその値はすべて0にしておく. GSTORE 上に, 節点Iに関するリストが作られたら, その先頭番地を POINT (I) に格納する. 従ってある節点Jを考えると, POINT (J)＝0 なら GPOINT 上にまだJは入っ

てないし，POINT(J)=K なら，GPOINT 上に格納され，かつ GSTORE 上のK番地にその隣接枝のリストの先頭番地があるとわかる．弧立節点の入力は，便宜上，その弧立節点を始点とし，終点その他をすべて0としたカードにより行う．1つのグラフの入力が終ったら，同じように次のグラフについて入力を行うが，グラフ入力の区切りには始点が0，終点が0以外のカードを入力するものとする．また，グラフの入力の終了の合図として，始点，終点共0のカードを読み込むものとする．**Fig. 3**に，外部からのグラフ入力の例を示す．

また，**Fig. 4**, **Fig. 5**(次頁参照)に，GRAMP で書いた実際のプログラムの例を示す．

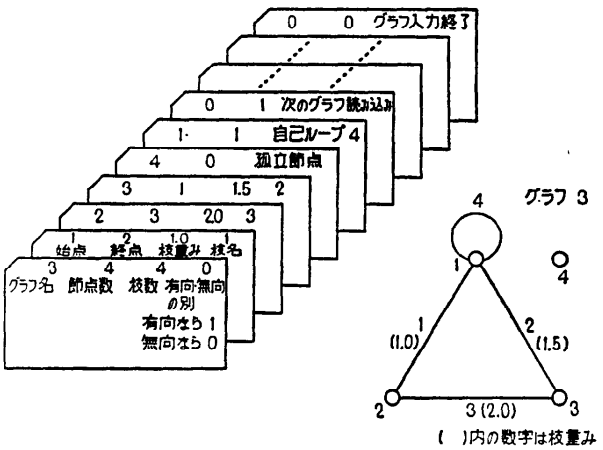


Fig. 3 Input of graph data.

4. 会話型処理

グラフの規模が小さい場合は，グラフの変形などの操作は，CRT の画面上にグラフを表示し，それを見ながら操作処理を行うのが，直観的にもわかりやすく有効である．また，CRT を通して GRAMP 全体の操作，呼び出しなどができれば，非常に使いやすいものになると思われる．そのため GRAMP は，会話型処理も含めた形でシステムを構成してある．**Fig. 6**(次頁参照)に会話型処理の全体の流れを示す．

GRAMP において会話型処理を行う場合はまず，DIALOG という命令を呼び出す．DIALOG は会話型処理全体を管理する命令であり，会話者が CRT に付属する Function-Key を押すことにより GRAMP の各命令に分岐する．規模の小さいグラフに関しては，DISPLY 命令によって，それを実際に画面上に表示させて処理をすることも可能である．DISPIN 命令は，会話者が，画面を見ながらグラフを，作業領域に入力する場合に適用される．この場合グラフの各節点を1点，1点カーソルで座標を決めながら入力するわけであるから，かなり手間はかかる．しかし各点の配置は人間が決定するわけであるから，表示されるグラフが非常に見やすくなるという利点がある．**Fig. 7**（次頁参照）に手動入力グラフに対して，その最小コスト木を求めた実例を示す．

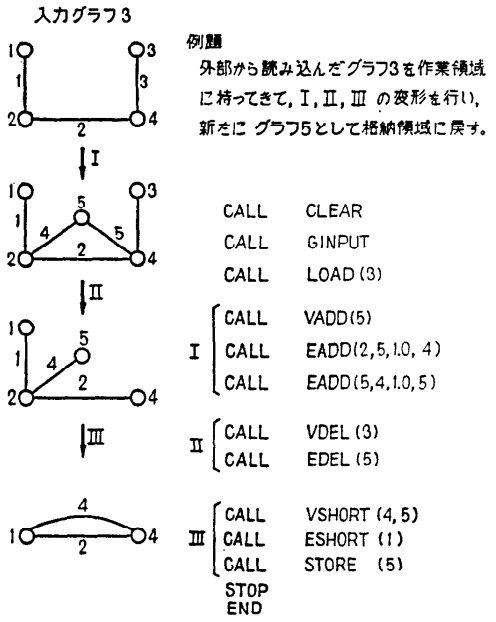


Fig. 4 An example of GRAMP program(1).

4.1 グラフ自動表示アルゴリズム

DISPLY 命令は前に述べたように，作業領域上のグラフを画面上に，自動的に節点の配置を決めて表示するが，その際問題となるのは，リスト構造で表わされたグラフをいかにしてグラフらしく画面上に表現するかということである．

そのためには

（1） 画面全体に均等に節点が配置されている．

（2） 枝の交差ができるだけ少ない．

ことなどが必要であろう．

節点の配置法として考えられる簡単な方法としては各節点を円周上に配置する，あるいは画面を格子状に

例題 節点数 100，枝数 150 の連結な無向ランダムグラフを 100 個発生させ，そのうち何個が非可分グラフであるかを調べる．

```
INTEGER*4  NXZERO
NUMBER= 0
NXZERO=31578249
DO 1  I=1,100
CALL GRANDM (100,150,NXZERO, 0 , 1 , 1)
CALL BICON (NOC, KOTAE,  1 )
IF (NOC. EQ. 1 ) NUMBER=NUMBER+ 1
1 CONTINUE
WRITE ( 6 , 2 ) NUMBER
2 FORMAT ('1', 10X, 'NUMBER=', I 3 )
STOP
END
```

Fig. 5 An example of GRAMP program (2).

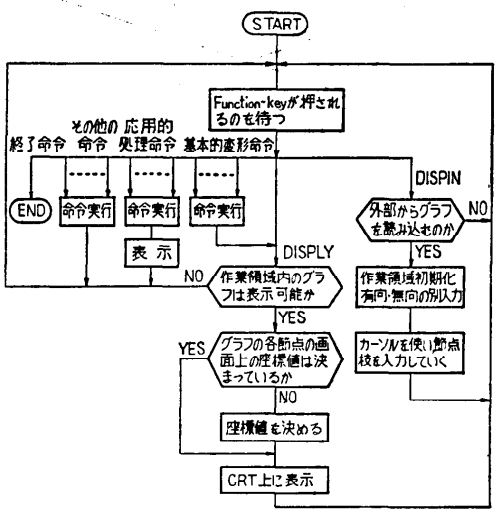


Fig. 6 A flowchart of the interactive processing of graphs.

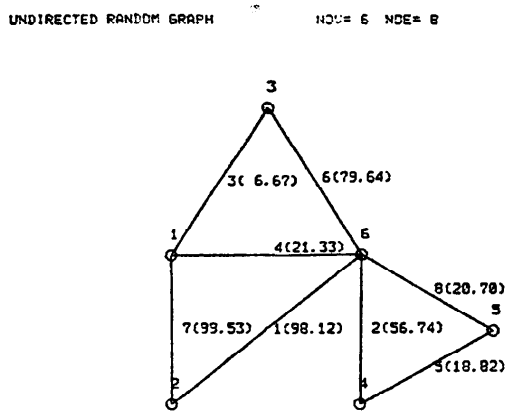
分割し，その格子点に節点を配置するなどが考えられる．しかしこれらの方法では，上に述べた２つの条件を共に満足させることは困難である．以下において，GRAMP においてとっている方法を説明する．ただし簡単のため，表示しようとするグラフは連結であり，自己ループ，並列枝はないとしてある．また節点名は，１から50まで，枝名は，１から100までの整数と限定してある．

グラフ自動表示アルゴリズム

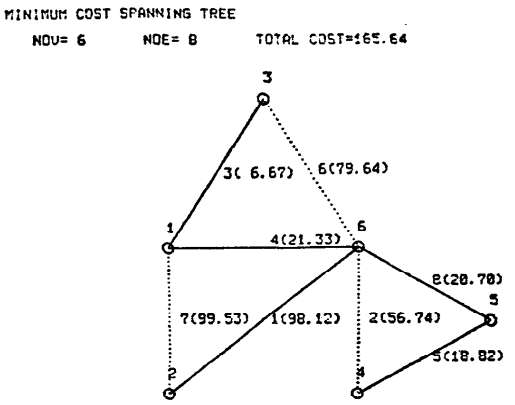
（1） 与えられたグラフの木を求める．

（2） (1)で求めた木の中心を求める．

（3） 木の中心から，各節点までの深さを求める．かつ各深さに存在する節点数を求める．



(a) Display of the graph.



(b) The result of MICST subroutine.

Fig. 7

（4） 画面の高さを深さに応じて分割し各節点の y 座標を求める．

（5） 画面の横幅を各深さのレベルにある節点数に応じてそれぞれ分割し，各節点の x 座標を決める．

（6） 枝を付け加える．

以上のアルゴリズムによって，画面上にグラフを自動表示させるわけであるが，同じ深さのレベルにある節点間に枝を引く時，各節点の y 座標が等しいため直線で結ぶとまずい場合がある．それを避けるため，同じ深さのレベルにある節点間に枝を結ぶ時は，すぐ隣り合った節点間のみを直線で結び，そうでない場合は２次曲線で結んでいる．

このようなアルゴリズムを前に述べた，サブルーチン GRANDM で発生させたランダムグラフについて

実験してみたが，やはり最後は人間が，節点の位置の移動などにより，見づらい部分を外部から修正してやるという必要が起こる場合が多い．しかしこの方法は，グラフの木を利用した表示法であり，木である限り完全に画面上に枝が交差することなしに表示できるため，グラフそのものが木に近いような場合，および木を取り扱う問題のような場合は有効である．**Fig. 8** にランダムグラフに対する自動表示例およびその修正例を示す．なお，各枝のコストは，必要に応じて表示できるようになっている (Fig. 7(a))．

会話型処理に関して必要とされる領域は，前に述べた作業領域の他に，各節点の座標を入れる領域および枝が2次曲線なら，その曲線の方程式の係数を入れておく領域などが必要である．

5. むすび

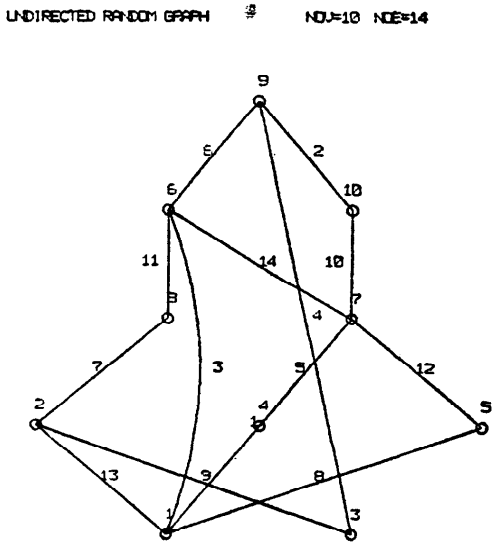
グラフ処理用プログラム GRAMP について，そのデータ構造，処理機能，計算手数のオーダ，会話型処理等について述べた．GRAMP 全体の大きさは，命令部で 50 k 語，データ部で 55 k 語程度である*．

GRAMP は現在，グラフに関する様々なアルゴリズムを記述するのに用いられたり，ランダムグラフの統計解析などに応用されている[6]．

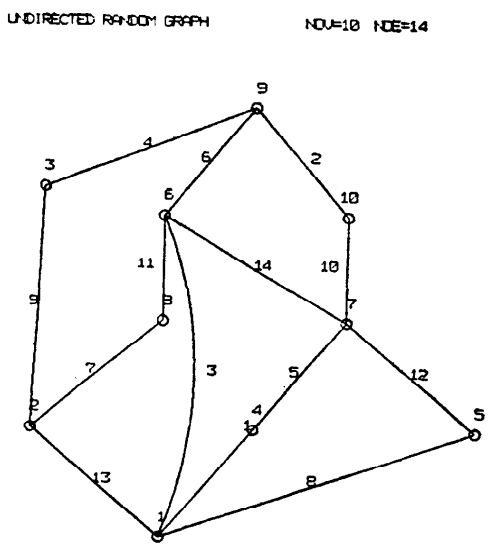
また，会話型処理をシステムの中に組み入れたことにより，規模の小さいグラフについては，実際に目で見ながら処理が可能であるので，直観的な理解ができる．

しかし実用上問題となる，エラー発見およびその対策をある程度備えてはいるが，系統だったものではなくこの方面においては，改善の余地がある．その他諸操作命令については，ここで説明した他にどのような命令が必要とされるか，あるいは応用分野についてもその範囲を広げてゆくなどの問題があるが，これらについては，これからの使用過程において解決されてゆくものと考えられる．

会話型処理については，グラフ自動表示アルゴリズムの改良，処理能力の拡大などの問題が残されている．



(a) The random graph on display.



(b) Modification of (a).

Fig. 8

参考文献

1) S. Berkowitz: GIRL Graph Information Retrieval Language Design of Syntax, Software Engineering, pp. 119-139, Academic Press, New York (1971).
2) T. W. Pratt etc.: A language extension for graph processing and its formal semantics, CACM, 14, pp. 460-467 (July 1971).
3) 杉藤，真野，鳥居：グラフ処理用2次元言語 GML とその機能，信学論 (D)，Vol. J59-D, pp. 597--604 (1976-09).
4) 伊理，中森：算法の最近の進歩，信学誌，Vol. 158, pp. 433-445 (1975-09).
5) A. V. Aho, J. E. Hopcroft, and J. D. Ullman:

* 1語＝2バイト，使用計算機は FACOM 230-38S.

The Design and Analysis of Computer Algorihmst, pp. 172-223, Addison-Wesley, Reading (1974).
6) 高見沢, 滝内, 西関, 斎藤: ランダムグラフの統計解析, 信学会回路とシステム研資, CST76-122 (1976-12).
7) 滝内, 高見沢, 西関, 斎藤: グラフ処理言語-GRAMP-, 信学会回路とシステム研資, CST 76-117 (1976-12).

(昭和52年5月18日受付)
(昭和52年11月10日再受付)