Decomposition of the Gozinto's graph with the use of a nesting store

S. Dvořák and B. Kropáč

The Computer Centre, Tesla Rožnov, Rožnov, Czechoslovakia

This paper describes a decomposition algorithm for finite oriented acyclic graphs, which makes use of a nesting store during its operations. The basic operation of the algorithm is partial splitting of the input list describing a graph or any partial graph. The described procedure proved to be very effective.

(Received January 1973)

1. Introduction

A large class of problems in operations research may be formulated in terms of graphs. These graphs naturally represent the relations of partial ordering, which are defined, for example, in sets of production parts or in complexes of interrelated activities. Here we will deal with decomposition of graphs which are characterised by the properties:

1. The graph is finite, oriented and acyclic.
2. There are several terminal nodes (roots) of the graph.

Graphs with these properties are called the Gozinto's graphs.* They do not need to be connected. The operations on these graphs and the way they are kept in computer memories are the objects of considerable attention in literature (e.g. Hu, 1968; Schmidt, 1970), because of the numerous applications of these information structures.

In this paper, we are dealing with the decomposition of Gozinto's graph. By decomposition we mean the selection of partial graphs (subgraphs) which are associated with given sets of graph nodes in the following sense: A partial graph belonging to a given set F of graph nodes consists of all arcs and nodes which are incident with all paths leading to the nodes in F. The solution of this problem is of immediate practical meaning, because in view of file updating, it is advantageous to maintain the whole file together in any type of external memory. On the other hand, the great size of processed files is generally not suitable for processing because it requires a series of additional transfers of data between operational and external computer memories. This disadvantage is especially appreciable for external memories with sequential information access such as magnetic tapes. Some examples of such a situation are:

1. Product cost calculations which are considerably simpler when performed for individual final products or related groups of these products than for the whole production file together. This file is defined by the so-called part-list.
2. Planning and scheduling computations, where the determination of scheduled number x_i of ith part requires also the knowledge of x_j, where j is an arbitrary subscript. This arises, for example, from a relation of the form:
$$x_i = \sum_{j=1}^{N} a_{ij}x_j + y_i\,,\ i = 1, 2, \ldots, N\,,$$
where a_{ij} is the consumption of the ith part in the production of the jth part. When it is impossible to load the vector $\mathbf{x} = (x_1, x_2, \ldots, x_N)^T$ into operational memory, cooperation with external memory becomes inevitable. Processing organised in this way is time consuming unless an external random access mass memory such as a disc is available.
3. Network analysis in some deterministic or indeterministic models (CPM, PERT), especially when the number of nodes of the network is too large, or the network is subject to frequent changes.

Hereinafter we shall denote the Gozinto's graphs as G-graphs.

2. Problem formulation

Let us consider a G-graph $\mathscr{G} = \langle \mathscr{U}, \mathscr{H} \rangle$, where $\mathscr{U}$ is the set of its nodes, $\mathscr{H}$ the set of its arcs. Let $f \in \mathscr{U}$ be a node belonging to this graph. Then by the partial graph $\mathscr{G}_f$ associated with the node f we mean the graph (subgraph) $\mathscr{G}_f = \langle \mathscr{U}_f, \mathscr{H}_f \rangle$ defined by:

(*a*) f is a node of $\mathscr{G}_f$ (i.e. $f \in \mathscr{U}_f$),

(*b*) if g' is a node of $\mathscr{G}_f$ and $g \to g'$ is an arc of $\mathscr{G}$ with initial node g and terminal node g', then g is a node of $\mathscr{G}_f$ and $g \to g'$ is an arc of $\mathscr{G}_f$ (i.e. $g \in \mathscr{U}_f$, $(g \to g') \in \mathscr{H}_f$). The notation $g \to g'$ for an arc is taken from Dvořák and Husička, 1965.

Similarly, when $F = \{f_1, f_2, \ldots, f_k\}$ is a subset of $\mathscr{U}$, then by the partial graph $\mathscr{G}_F$ associated with F we understand the graph $\mathscr{G}_F = \langle \mathscr{U}_F, \mathscr{H}_F \rangle$ defined as follows:

(*a*) every $f_\alpha \in F$ is a node of $\mathscr{G}_F$ (i.e. every $f_\alpha \in \mathscr{U}_F$),

(*b*) if g' is a node of $\mathscr{G}_F$ and $g \to g'$ is an arc of $\mathscr{G}$, then g is a node of $\mathscr{G}_F$ and $g \to g'$ is an arc of $\mathscr{G}_F$ (i.e. $g \in \mathscr{U}_F$, $(g \to g') \in \mathscr{H}_F$).

Therefore
$$\mathscr{G}_F = \mathscr{G}_{f_1} \cup \mathscr{G}_{f_2} \cup \ldots \cup \mathscr{G}_{f_k}$$
is the union of subgraphs associated with the nodes in F. Our task in what follows is to describe an algorithm for solving the following problem: Given a family $\mathbf{F} = \{F_1, F_2, \ldots, F_m\}$ of sets F_α of G-graph nodes determine all partial graphs $\mathscr{G}_{F_1}$, $\mathscr{G}_{F_2}, \ldots, \mathscr{G}_{F_m}$. This task will be referred to as the decomposition of the original graph $\mathscr{G}$.

The situations described in the introduction indicate the meaning of decomposition algorithms. Some of them use topological enumeration of graph vertices, i.e. every initial node of an arc is denoted by a lower natural number than the terminal node of that arc. Here it is necessary to carry out topological enumeration before the actual decomposition, and when the decomposition is over, we have to pass back to the original node numbering. An algorithm for topological enumeration has been described, for example, by Dvořák and Vaculín (1969).

In this paper, an essentially different form of the decomposition algorithm is presented. This algorithm makes use of the nesting store (stack, push-down storage, last-in-first-out list). This information structure serves in a natural form for storage of data that cannot be used immediately but have to be stored for further processing. This situation arises also in a lot of algorithms for information structures which can be represented by these graphs (e.g. the syntactic trees generated during translation of phrase-structure programming languages). An example of G-graph and its partial graph associated with nodes 1 and 2 is given in **Fig. 1.**

*This name originated from Vaszonyi (Vaszonyi, 1962). The 'mathematician' Zepartzat Gozinto of course did not exist. Vaszonyi derived its name from the pronunciation of the English phrase 'the part that goes into', that characterises the first usage of these graphs.

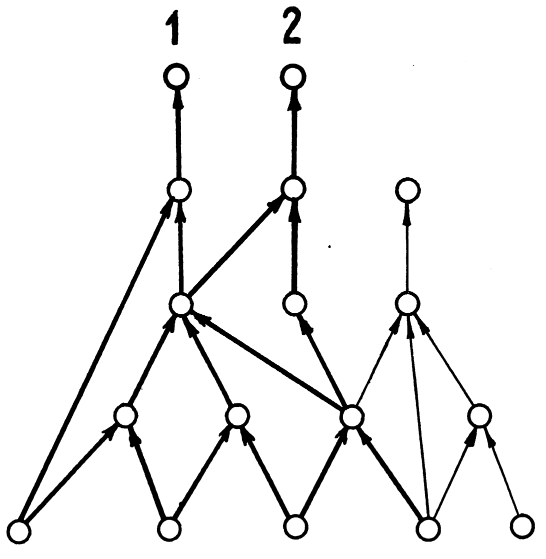

Fig. 1 *G*-graph and its partial graph associated with nodes 1, 2.

3. Graph definition

Let us assume *G*-graph $\mathscr{G}$ is given by the ordered list $L_{\mathscr{G}}$ of its arcs $i_\alpha \to j_\alpha$, $1 \leq \alpha \leq L$, where L is the number of arcs of $\mathscr{G}$. Let N be the number of nodes of $\mathscr{G}$. Without loss of generality we can assume:

1. The nodes of $\mathscr{G}$ are denoted by the natural numbers $1, 2, \ldots, N$. If not so, we shall carry out such numbering before processing, e.g. by sorting and dichotomic search.
2. The list $L_{\mathscr{G}}$ is arranged in such a way, that for every node i firstly listed arcs are the arcs $i \to i'_\alpha$, where i is the initial node, and after them arcs $i''_\alpha \to i$, where i is the terminal node, that is to say, the list $L_{\mathscr{G}}$ is arranged as follows:

$$\begin{array}{l}
\vdots \\
\left.\begin{array}{l} i \to i'_1 \\ \vdots \\ i \to i'_2 \\ \vdots \\ i \to i'_p \end{array}\right\} \begin{array}{c}\text{for every node } i\text{:} \\ \text{at the beginning} \\ \text{arcs with } i \text{ as initial node,}\end{array} \\
\vdots \\
\left.\begin{array}{l} i''_1 \to i \\ \vdots \\ i''_2 \to i \\ \vdots \\ i''_q \to i \end{array}\right\} \begin{array}{c}\text{followed by} \\ \text{arcs with } i \\ \text{as terminal node.}\end{array} \\
\vdots
\end{array}$$

If the list $L_{\mathscr{G}}$ is ordered in accordance with the described rule, we call it a (topologically) top-down ordered list. If the original list $L^0_{\mathscr{G}}$ should not be top-down ordered, it may be rearranged to be so by the following method:

1. For every node i we determine how many times it occurs in $L^0_{\mathscr{G}}$ as the initial (lower) node. Let N_i be this number.
2. The arc $i \to j$ may be rewritten into the top-down ordered list $L_{\mathscr{G}}$ if and only if $N_j = 0$. Therefore, it is sufficient to go through the list $L^0_{\mathscr{G}}$ in a cyclic manner. When for current record $i \to j$ N_j equals zero, the arc $i \to j$ is transferred as the next one into the ordered list $L_{\mathscr{G}}$ and is deleted from $L^0_{\mathscr{G}}$. N_i is then decreased by 1 and we pass to the next record in the reduced list $L^0_{\mathscr{G}}$. When $N_j \neq 0$ we pass to the next record $i' \to j'$ in $L^0_{\mathscr{G}}$ immediately. We proceed in this way until the list $L^0_{\mathscr{G}}$ is empty.

Therefore, the assumptions given above are actually no limitations of generality. For the sake of simplicity let us assume that every group of nodes, whose associated partial graph is to be taken out, consists of only one node. The extension to arbitrary number of nodes in the group follows easily from the further description of the decomposition method.

4. The decomposition algorithm

Let there be given m nodes $f_1, f_2, \ldots, f_m$ whose associated partial graphs $\mathscr{G}_{f_1}, \mathscr{G}_{f_2}, \ldots, \mathscr{G}_{f_m}$ in the *G*-graph $\mathscr{G}$ are to be determined. Let us assume for definiteness magnetic tape files. Let A be the unit, which bears the input file $L_{\mathscr{G}}$ corresponding to the graph $\mathscr{G}$. Let B and C be magnetic tape handlers, which will be used as working devices, and let V be the output magnetic tape unit. The algorithm may be easily modified to deal with a greater number of magnetic tape units, but this will not be described here.

Let P denote the unit with an input file and let Q and R be working units. Thus we have P = A, Q = B and R = C at the beginning. The principle of the presented decomposition algorithm consists of the partial decomposition ('halving') of the current input list located on P and the recording of the partial lists into which the original list has been split. The newly created partial lists are written on Q and R respectively as the next data files. This partial decomposition is repeated for every input list, i.e. for the original input list and every created partial list. These lists are in turn selected with the use of stack mechanism.

More precisely, two lists associated with sets

$$D = \{f_d, f_{d+1}, \ldots, f_s\} \quad \text{(the 'lower' set)}$$

and

$$H = \{f_{s+1}, f_{s+2}, \ldots, f_h\} \quad \text{(the 'upper' set)}$$

of graph nodes are created during the partial decomposition (splitting) of the input list on P. This list is assumed to correspond to the set

$$D \cup H = \{f_d, f_{d+1}, \ldots, f_h\}$$

of graph nodes. Here $s = [(d + h - 1)/2]$; $[x]$ is the integral part of x. We have of course $d = 1$, $h = m$ at the beginning, hence s is a centre of the interval $[d, h]$ chosen so as the cardinality (i.e. the number of elements) of the lower set D does not exceed that of the upper set H. Now several situations have to be distinguished:

1. If $d < s$, then neither of the sets D and H consists of just one element. In that case the list belonging to D and also the list belonging to H must be split further. Hence both lists are written as next data files on working tapes so that on tape R(Q) the list associated with $H(D)$ is written. Information about content and storing of lists associated with H and D are written into stack, respectively. For every set it is sufficient to keep in the stack only the subscript of the first and last nodes and the unit, where the corresponding list has been recorded.
2. If $d = s$, then D contains only one node. Therefore, the list associated with D can be immediately written on the output unit and hence we shall take Q = V. Only information about the list associated with H will be recorded into the stack.
3. If $d > s$, then D is an empty set and H contains only one node. Therefore, the list associated with H can be written immediately on the output unit. In this case the output unit is R and we have R = V. No new information will be recorded into the stack.

Having performed these steps we have to take information from the last record made in the stack (top of the stack) and perform partial decomposition using this information. This process continues until we reach an empty stack in a certain step. In this case the given decomposition is finished and lists associated with $f_1, f_2, \ldots, f_m$ are written in that order on the output unit V.

In the previous part, the decomposition algorithm has been globally described. Now we are going to describe the partial decomposition of the list L_Γ, which corresponds to the G-graph Γ, into two lists L_{Γ_D} and L_{Γ_H}. In this decomposition, we have to determine for every arc $i \to j$ in Γ whether it belongs:

(*a*) neither to Γ_D nor to Γ_H,
(*b*) to Γ_D and does not belong to Γ_H,
(*c*) to Γ_H and does not belong to Γ_D,
(*d*) to Γ_D and also to Γ_H.

At this point let us define a 'two-bit' register REG(i) for every node $i (1 \leqq i \leqq N)$, where the first (second) bit $\mathsf{REG}_D(i)$ ($\mathsf{REG}_H(i)$) is of logical value 1, if the node i belongs to the partial graph associated with $D(H)$ and it is 0 otherwise. Let us put in accordance with the definition of the partial graph associated with D

$$\mathsf{REG}_D(i) = 1 \text{ for } i \in D \text{ and } \mathsf{REG}_D(i) = 0 \text{ for } i \notin D$$

at the beginning. $\mathsf{REG}_H(i)$ is defined similarly and

$$\mathsf{REG}(i) = \langle \mathsf{REG}_D(i), \mathsf{REG}_H(i) \rangle .$$

Now we take the current arc $i \to j$ contained in L_Γ. REG(j) = 0 implies the arc $i \to j$ belongs to neither of the partial graphs and, therefore, we shall pass to the next arc. If $\mathsf{REG}_M(j) = 1$ then the node j belongs to the graph associated with $M(M = D, H)$. Therefore, also node i and arc $i \to j$ belong to this graph and we shall write this arc in the corresponding list. Then we fill the array REG for node i in accordance with partial graph definition using the term

$$\mathsf{REG}(i) = \mathsf{REG}(i) \vee \mathsf{REG}(j) ,$$

where $\vee$ denotes the logical disjunction (sum). Then we pass to the next record in L_Γ. We proceed in this way until the list L_Γ is exhausted. It is noted here that the top-down ordering of L_Γ ensures, that:

(*a*) the register REG(i) defines exactly the incidence of node i first time it is used,
(*b*) the top-down ordering is retained in the lists L_{Γ_D} and L_{Γ_H}.

The details of this procedure are given by the FORTRAN subroutine in the Appendix.

5. Algorithm implementation

The subroutine DECOMP which performs the decomposition of the given G-graph in the manner described above, calls a subroutine RWDF and a function DISJ. RWDF(I) rewinds the tape I to the beginning of the last file written on it, DISJ(I, J) carries out the disjunction of arguments I and J. EOFTST(P, J) is machine-oriented routine, which checks the end of the file on device P. This routine can be used as FORTRAN function (then its value is negative if end of file is not reached) or subroutine (then J contains 1 at the end of the file and 2 otherwise). For the sake of simplicity the REG registers are word-oriented. The statements of the program given in the appendix are written in accordance with FORTRAN-IV (ANSI) specifications.

In practice, the presented algorithm can be advantageously modified, for instance

(*a*) the working file can be defined in operational memory and only on overflow of the assigned memory range will a tape be used (reduction to 25% of original time);
(*b*) when the input list corresponds to one node only it may be copied on the output tape immediately in all but the first steps. The partial decomposition in this case may be avoided.

When we have to choose the partial graphs associated with a groups of nodes, we assume that the groups indices are recorded in the NLIST array and that ADR(2*J − 1) (ADR(2*J)) contains the index of the beginning (ending) node of the Jth group of nodes, which are themselves recorded in array NODE consecutively for the first, second, . . . group. The only change in the presented algorithm for this case is in the initial setting of the registers. The DO-loop statements, which perform this setting, are to be changed as follows:

```
          J       = NLIST(I)
          J1      = ADR(2*J − 1)
          J2      = ADR(2*J)
          DO n J  = J1, J2
          K       = NODE(J)
        n REG(K)  = 1 or DISJ(2, REG(K)) .
```

6. Effectiveness of the algorithm

The presented algorithm has been tested in many cases and proved to be superior to the algorithms mentioned in Section 2 for two reasons:

1. It can deal with a large number of groups associated partial graphs of which we are seeking.
2. It can cope with a large number of nodes in the initial graph $\mathscr{G}$, because for registration of node incidence only two bits are sufficient.

The effectiveness of the algorithm is also indicated by an estimation of the reading and writing operations. In fact

$$\begin{aligned}\rho(1, 2, \ldots, m) = {} & R(V_1 \cup V_2 \cup \ldots \cup V_m) \\ & + W(V_1 \cup \ldots \cup V_{[m/2]}) + W(V_{[m/2]+1} \cup \ldots \cup V_m) \\ & + \rho(1, 2, \ldots, [m/2]) + \rho([m/2] + 1, \ldots, m) ,\end{aligned}$$

where

$\rho(\alpha, \beta, \ldots)$ denotes the total number of reading and writing operations during the decomposition of the list associated with groups $\alpha, \beta, \ldots$,

$R(V \cup V' \cup \ldots)$ denotes the number of reading operations of the list $V \cup V' \cup \ldots$,

$W(V \cup V' \cup \ldots)$ denotes the same number but for writing operations.

If the initial volume of reading is $|V_0|$, where V_0 is the initial input list, then the foregoing equation yields approximately

$$\begin{aligned}\rho(1, 2, \ldots, m) & \approx |V_0| + 2\left(2\frac{m}{2}\bar{V} + 4\frac{m}{4}\bar{V} + \ldots\right) \\ & \approx |V_0| + 2\bar{V}m \, lg_2 m ,\end{aligned}$$

if $\bar{V}$ is taken to be the average number of records in the partial graph of a node or a group of nodes.

The modified algorithm (which uses part of the operational memory and does not perform unnecessary decompositions) has been programmed in FORTRAN-IV (ANSI) for the GE-427 computer. Time needed to take out about 150 partial graphs corresponding to individual final products was 2·1 min. with 80 kc tapes and 3·85 μs memory cycle.

Appendix

The decomposition procedure described above corresponds to the following FORTRAN-subroutine:

```
      SUBROUTINE DECOMP(NLIST, M,A,B, C,V)
      INTEGER NLIST(1),M,A,B,C,V
C
C     DECOMPOSITION ALGORITHM FOR THREE TAPE
C       UNITS
C     NLIST  LIST OF NODES TO WHICH PARTIAL GRAPHS
C            ARE TO BE CHOSEN
C     M      THE RANGE OF THE USED PART OF NLIST
C     A,B,C  WORKING UNITS (A CONTAINS THE INITIAL
C            INPUT)
C     V      OUTPUT UNIT
C
      INTEGER STACK(60),D,H,S,SP,P,Q,R,NS,VS,REG(1000)
      INTEGER DISJ
      INTEGER I,J,L
      REWIND A
      REWIND B
      REWIND C
      REWIND V
      P = A
      H = M
      D = 1
      SP = 0
      GO TO 40
   10 IF (SP) 240,240,20
   20 P = STACK(SP)
      H = STACK(SP-1)
      D = STACK(SP-2)
      SP = SP - 3
      CALL RWDF(P)
   30 IF (P.EQ.A) GO TO 40
      Q = A
      IF (P.EQ.B) GO TO 50
      R = B
      GO TO 60
   40 Q = B
   50 R = C
   60 S = (D+H-1)/2
   70 DO 80 I = 1,1000
   80 REG(I) = 0
      IF (S-D) 90,100,100
   90 R = V
      GO TO 140
  100 STACK(SP+1) = S+1
      STACK(SP+2) = H
      STACK(SP+3) = R
      SP = SP+3
      IF (S.GT.D) GO TO 110
      Q = V
      GO TO 120
  110 STACK(SP+1) = D
      STACK(SP+2) = S
      STACK(SP+3) = Q
      SP = SP+3
  120 DO 130 I = D,S
      J = NLIST(I)
  130 REG(J) = 1
  140 S = S+1
      DO 150 I = S,H
      J = NLIST(I)
  150 REG(J) = DISJ(2,REG(J))
  160 READ (P) NS,VS
      IF (EOFTST(P,J)) 170,230,230
  170 L = REG(VS)
      IF (L) 160,160,180
  180 GO TO (190,210,200),L
  190 WRITE (Q) NS,VS
      GO TO 220
  200 WRITE (Q) NS,VS
  210 WRITE (R) NS,VS
  220 REG(NS) = DISJ(REG(NS),L)
      GO TO 160
  230 IF (H.GT.D) END FILE Q
      END FILE R
      CALL RWDF(P)
      GO TO 10
  240 END FILE V
      REWIND V
      RETURN
      END
```

References

Berge, C. (1962). *The Theory of Graphs and its Applications*, London: Methuen.

Dvořák, S., and Husička, M. (1965). A matrix model of production plan, *Statistika a demografie*, V, pp. 171-189.

Dvořák, S., and Vaculín, J. (1969). A choice of partial graphs from oriented acyclic graph, *Ekonomicko-matematický obzor*, Vol. 5, No. 4, pp. 454-464.

Hu, T. C. (1968). A Decomposition Algorithm for Shortest Paths in a Network, *Operations Research*, Vol. 16, No. 1, pp. 91-102.

Schmidt, W. P. (1970). Grundlagen, Verfahren und Möglichkeiten der Datenorganisation für die Teilebedarfsermittlung (Stücklisten-auflösung), Teil III, *Elektronische Datenverarbeitung*, Vol. 12, No. 4, pp. 253-260.

Vaszonyi, A. (1962). *Die Planungsrechnung in Wirtschaft und Industrie*, Wien und München: Springer-Verlag.

Book review

Interactive Computing in BASIC, by P. C. Sanderson, 1973; 161 pages. (*Butterworths*, £4·00 hard cover, £2·00 paperback)

In a London bookshop I recently counted no less than ten different textbooks on programming in BASIC. That BASIC has attracted such attention from authors is perhaps the greatest possible tribute to this rapidly-growing language, which is surely the most commonly used in interactive computing.

Mr. Sanderson's is the latest then in an already long line of books which deal with this subject. In many ways it is no better nor worse than the others, and any criticisms I make of it could be equally well applied elsewhere.

The book can be divided into three main sections: firstly an introductory section which deals with computing, flowcharting, programming languages, etc. in a fairly standard manner; secondly the main body of the text which introduces BASIC; and finally a chapter that deals with the conversion of BASIC programs to FORTRAN.

My first criticism of this book is that, like many others, it has a definite numerical bias. This seems unfortunate since BASIC is a true general-purpose language, many implementations having powerful facilities for non-numeric work. However string-handling and character manipulation receive only a cursory treatment here. Another fault seems to be that many of the examples given in the text are trivial, thus there is a definite shortage of genuine case-studies which are surely necessary for beginners. Also some parts of BASIC are given very sketchy treatment (e.g. The BASIC editor and command language), while other essential parts are omitted (e.g. file-handling). Finally I feel that the chapter introducing FORTRAN is unnecessary; BASIC is arguably a more powerful and sophisticated language than FORTRAN, and anyone with a comprehensive implementation of BASIC at their disposal need have little recourse to FORTRAN. In any case, one chapter is insufficient to state the case for FORTRAN.

The main difficulty for anyone attempting a book such as this, must be the problem of dialects. So many different versions of BASIC are now implemented that it must be tempting for authors to gloss over those areas where there is the greatest variation. Mr. Sanderson does at least include a brief chapter highlighting the differences between several implementations; however a proposed standard for BASIC has now been published, and wise authors will base their texts on this in future.

M. I. Jackson (Hatfield)