THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY

OPTIMAL GRAPH CLUSTERING PROBLEMS WITH APPLICATIONS TO INFORMATION SYSTEM DESIGN

Wan Ping Chiang

CRL-TR-30-84

June 1984

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

ensm

UMR1189

ABSTRACT

OPTIMAL GRAPH CLUSTERING PROBLEMS

WITH APPLICATIONS TO INFORMATION SYSTEM DESIGN

by

Wan Ping Chiang

Chairmen: Toby J. Teorey, William C. Rounds

Information system design problems, including database and software, can often be represented in terms of directed or undirected graphs. Some of these problems typically involve determining how to cut the graph into a set of nonvoid and disjoint subgraphs such that each of the subgraphs satisfies a set of constraints while an objective function defined over the subgraphs is optimized.

This class of information system design problems is analyzed with the following three objectives: First, to formally define this class of problems as an Optimal Graph Clustering (OGC) problem and classify it into a set of subproblems. Second, to find efficient algorithms that give an exact and optimal solution to each problem with nontrivial objective function and constraints. And third, to demonstrate these algorithms' usefulness by formalizing and solving some information system (database and software)

design problems.

Eight classes of digraphs (general digraph, acyclic digraph, out-necklace, out-tree, out-star, in-necklace, in-tree and in-star) and four classes of undirected graphs (general undirected graph, necklace, tree and star) are considered and are used to classify the OGC problem into thirty-five subproblems. These subproblems are shown to be NP-complete problems.

Dynamic programming and enumerative approaches are used to show that eighteen OGC subproblems with a given graph that has n-1 arcs or n arcs can be solved in pseudopolynomial time. It is also determined that in the rest of the NP-complete problems, if the graphs are sparse (i.e., the number of arcs does not exceed the number of vertices greatly) they are solvable by computer.

The class of possible objective functions and constraints applicable to these solutions such that all time complexities remain the same is also defined. All monotonic objective functions and constant time computable constraints are found to be applicable.

The solutions' usefulness is demonstrated by solving three information system design problems: B-tree secondary storage allocation, translation of an integrated schema into a hierarchial schema, and database record clustering.

OPTIMAL GRAPH CLUSTERING PROBLEMS

WITH APPLICATIONS TO INFORMATION SYSTEM DESIGN

by

Wan Ping Chiang

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer and Communication Sciences)
in The University of Michigan
1984

Doctoral Committee:

Associate Professor Toby J. Teorey, Co-chairman
Associate Professor William C. Rounds, Co-chairman

Professor Bernard A. Galler
Associate Professor Vaclav Rajlich

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

This dissertation studies a class of information system (database and software) design problems that has the two following characteristics:

1) The problem can be represented as a graph (directed or undirected).
2) The problem is how to cut the graph into a set of nonvoid and disjoint subgraphs such that each of the subgraphs satisfies a set of constraints while an objective function defined over the subgraphs is optimized.

Some examples of this class of problems are:

- B-tree secondary storage allocation (see Chapter 6);
- translation of database integrated schema to IMS schema (see Chapter 6);
- database record clustering (Schkolnick '77), (Chiang & Teorey '82), (also see Chapter 6);
- computational objects allocation in distributed systems (Jenny '82);
- software system partitioning (Belady & Evangelisti '81); and
- program restructuring (Ferrari '76).

There have been approaches to some of these problems, but they are usually specific to isolated cases or restricted to simple objective function and constraints. This dissertation goes beyond isolated cases to provide a generalized and unified framework for this class of problems and provide a set of efficient and structuralized solutions to them. Efforts are also spent to extend the objective function and constraints. The purpose of this study is thus threefold:

1) To formally define this class of problems as an Optimal Graph Clustering (OGC) problem and classify it into a set of subproblems.
2) To find efficient algorithms that give an exact and optimal solution to each problem with nontrivial objective function and constraints. Heuristic or approximate solutions are not considered.
3) To demonstrate these algorithms' usefulness by formalizing and solving some information system (database and software) design problems.

The first purpose is taken up in this chapter. The definition of the OGC problem and the classification of it into a set of optimal graph clustering subproblems are presented in section 1.1. The complexity issues of the subproblems are studied in section 1.2. Section 1.3 gives a literature survey of the OGC problem and section 1.4 gives the organization of this dissertation.

The graph theory terminology used throughout this

dissertation follows those of Harary ('69) and Aho et al ('74). If a different term/notation is used, it will be followed by the established term/notation in brackets.

1.1 Problem Definition and Classifications

Given a (connected) graph G=(V,E) with vertex set V and arc set E, a clustering K of G is defined as a cutting of G into a set of nonvoid and disjoint (connected) subgraphs by removing some arcs in G. Thus, a clustering K is an assignment function which assigns either 0 or 1 to each arc of E. An assignment with value 0 means the arc is removed (or cut) while a 1 assignment means the arc is not removed (or cut). Each clustering K can be represented as a set of arc assignments X_{ij}, i.e.,

$K = \{X_{ij} \mid X_{ij} = K((v_i,v_j)) \text{ for all } (v_i,v_j) \in E\}$ (see Figure 1-1a),

or a set of clusters it induces, i.e.,

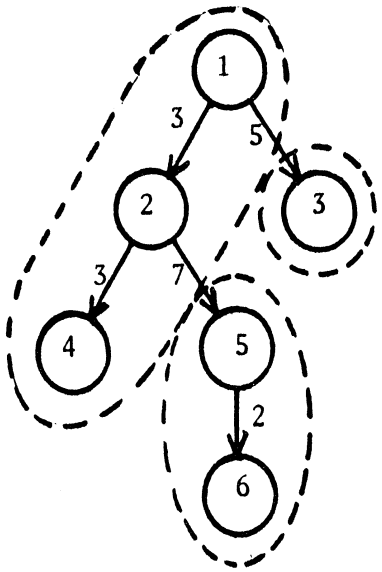
$K = \{C \mid C \text{ is a cluster induced by } K\}$.

Each of these clusters is a (connected) subgraph of G (see Figure 1-1b).

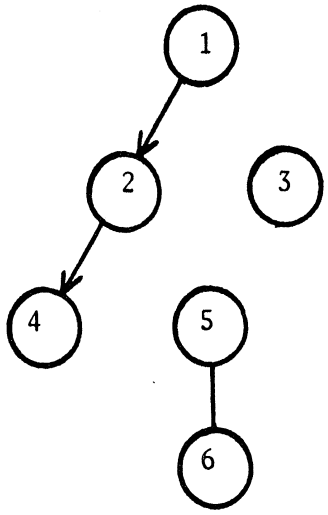
Let $|E|$ represent the number of arcs in G and let PK represent the set of all possible clusterings of G. Thus, PK contains $2^{|E|}$ possible clusterings of G. A clustering of G which satisfies a given set of constraints defined for a problem is called a feasible clustering. Let PK^f be the subset of PK which contains only feasible clusterings.

Based on the above, an Optimal Graph Clustering (OGC) problem can be defined as:

a)



b)



$B = 3$

$\forall v_i \ (w_i=1)$

Figure 1-1a) A clustering with $X_{13}=X_{25}=0$, $X_{12}=X_{24}=X_{56}=1$

Figure 1-1b) Three clusters induced by the clustering

Given: 1) a graph G=(V,E);

2) an objective function *F*;

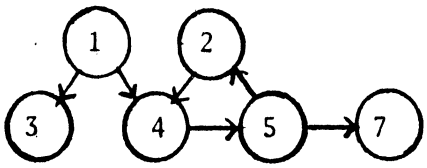
3) two constraints: one limit the size of each cluster and the other requires each cluster to be a (connected) subgraph of G.

Find: a clustering K which satisfies these two constraints and has optimal objective function *F*(K).

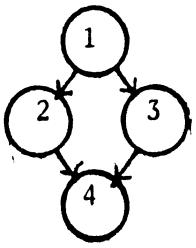
In this dissertation, only twelve classes of graphs are considered. They are:

- <u>(connected)</u> <u>directed</u> <u>graph</u> -- general digraph (G_g), acyclic digraph(G_a), out-necklace(G_{n_o}) [contra functional digraph (Harary '69)], out-tree(G_{t_o}), out-star(G_{s_o}), in-necklace(G_{t_1}) [functional digraph (Harary '69)], in-tree(G_{t_1}), in-star(G_{s_1}) (see Figures 1-2 and 5-1a);
- <u>(connected)</u> <u>undirected</u> <u>graph</u> -- general undirected graph(G_u), necklace(G_n) [unicyclic graph (Harary '69)], tree(G_t), star(G_s) (see Figure 5-1b).

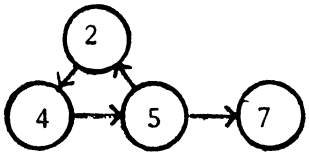
The OGC problem can be classified into many subproblems by using the class of graph given and the class of cluster induced by a clustering. Since an undirected graph can never be obtained by clustering a directed graph and vice versa, the OGC problem can first be classified into two families of problems. One family deals with the directed graph and is called an <u>Optimal Digraph Clustering (ODC)</u> problem. The other deals with the undirected graph and is



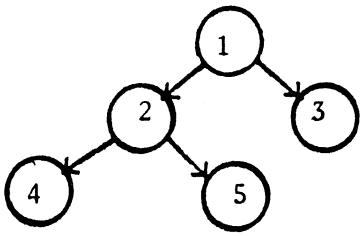
general digraph



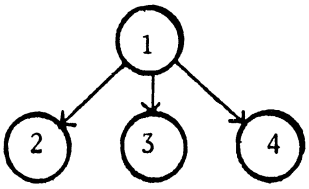
acyclic digraph



out-necklace



out-tree



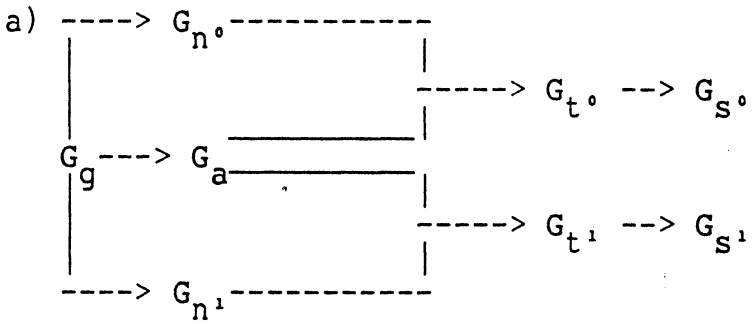
out-star

Figure 1-2 Five classes of digraphs.

called an Optimal Undirected Graph Clustering (OUC) problem.

Both ODC and OUC problems can be refined into a set of subproblems based on whether a class of graph can be obtained by clustering another class of graph. A class of graph B (e.g., out-tree) is a subclass of a class of graph A (e.g., acyclic digraph), denoted A --> B, if B is a class of graph obtainable by clustering A (i.e., B can be a subgraph of A).

Proposition 1.1:

a) 

b) G_u --> G_n --> G_t --> G_s

Proof:

a) (G_g --> G_{n^0}): A necklace is one in which every vertex has indegree 1. Given a general digraph with cycles and q m-indegree vertices v_1, v_2, ..., v_q in it (i.e., those that have in-degree more than 1), to transform an m-indegree vertex v_i into an indegree 1 vertex we need to remove $id(v_i) - 1$ arcs, where $id(v_i)$ is the indegree of v_i. By repeating all combinations of the removal process, we can find at least one occurrence of an out-necklace. If this is not true, then the original given digraph is an acyclic digraph, which is a contradiction. Thus, G_g-->

G_{n^0}. The rest of the subclass relations in a) can be proved similarly. ***Q.E.D.***

b) Similar to a). ***Q.E.D.***

Using Proposition 1.1, the set of subproblems for ODC is: (The first subscript of ODC or OUC is the class of cluster induced; the second is the class of the graph given.)

\digraph \given \ cluster \ induced \	general digraph	acyclic digraph	out(in)-necklace	out(in)-tree	out(in)-star
	G_g	G_a	$G_{n^0}(G_{n^1})$	$G_{t^0}(G_{t^1})$	$G_{s^0}(G_{s^1})$
G_g	ODC_{gg}				
G_a	ODC_{ag}	ODC_{aa}			
G_{n^0}	ODC_{n^0g}		$ODC_{n^0n^0}$		
G_{t^0}	ODC_{t^0g}	ODC_{t^0a}	$ODC_{t^0n^0}$	$ODC_{t^0t^0}$	
G_{s^0}	ODC_{s^0g}	ODC_{s^0a}	$ODC_{s^0n^0}$	$ODC_{s^0t^0}$	$ODC_{s^0s^0}$
G_{n^1}	ODC_{n^1g}		$ODC_{n^1n^1}$		
G_{t^1}	ODC_{t^1g}	ODC_{t^1a}	$ODC_{t^1n^1}$	$ODC_{t^1t^1}$	
G_{s^1}	ODC_{s^1g}	ODC_{s^1a}	$ODC_{s^1n^1}$	$ODC_{s^1t^1}$	$ODC_{s^1s^1}$

The set of subproblems for OUC is:

\graph cluster\given induced \	general graph G_u	necklace G_n	tree G_t	star G_s
G_u	OUC_{uu}			
G_n	OUC_{nu}	OUC_{nn}		
G_t	OUC_{tu}	OUC_{tn}	OUC_{tt}	
G_s	OUC_{su}	OUC_{sn}	OUC_{st}	OUC_{ss}

1.2 Complexity Issues

All the OGC subproblems being discussed in this dissertation are NP-complete problems. To prove that, it is only necessary to show that $ODC_{S^0S^0}$, $ODC_{S^1S^1}$ and OUC_{SS} are. To show OUC_{SS}, $ODC_{S^0S^0}$ and $ODC_{S^1S^1}$ are NP-complete, it is only necessary to show that the integer versions of them with integer objective value function and constraint are. The integer versions of these three problems can be defined as follows:

Given:

1) a star/out-star/in-star;

2) an objective function:

$$F(K) = \sum_{\text{all } \overline{(v_i,v_j)}} f_{ij} \cdot \bar{X}_{ij} \quad \text{where}$$

f_{ij} is a positive integer value associated with each arc (v_i,v_j);

3) two constraints:

3.1) every cluster must be a star/out-star/in-star or its subclass;

3.2) let a vertex weight be a positive integer value w_i associated with each vertex and the cluster weight of a cluster be defined as the sum of all vertex weights in the cluster; every cluster weight must be less than or equal to a positive integer B.

Find: a feasible clustering K such that $F(K)$ is minimized.

Proposition 1.2: All OGC subproblems considered in this

dissertation are NP-complete problems.

Proof: Bertolazzi et al ('80) showed the subset sum problem is polynomial reducible to the integer version of OUC_{SS}. Since the former has been shown to be NP-complete, the latter is also NP-complete. The same technique can also be used to show the integer version of $ODC_{S^0S^0}/ODC_{S^1S^1}$ is NP-complete by replacing the term "tree" with "out-tree/in-tree".

Since the integer version of $OUC_{SS}/ODC_{S^0S^0}/ODC_{S^1S^1}$ is NP-complete, the general version of $OUC_{SS}/ODC_{S^0S^0}/ODC_{S^1S^1}$ that is defined in section 1.1 is also NP-complete by using the proof by restriction technique (Garey & Johnson '79). Again, using proof by restriction, since the rest of the OGC subproblems are mere generalizations of the general version of $OUC_{SS}/ODC_{S^0S^0}/ODC_{S^1S^1}$, they are also NP-complete. ***Q.E.D.***

The fact that all of the OGC subproblems discussed in this dissertation are NP-complete does not mean efficient and practical algorithms cannot be found to solve them. Because they are also number problems, eighteen pseudopolynomial time solutions have been found in this dissertation (see those enclosed in the dashed lines in the two tables above). One advantage of pseudopolynomial time solutions is that they are practical (efficient) in many real life applications. A second advantage is that they can often be turned into fully polynomial time approximate schemes (Garey & Johnson '79).

1.3 Literature Survey

Many studies on the OGC problem originated from diversed application areas. Some of them are listed at the beginning of this chapter. Most of the early approaches formalize the problem too generally (as an OUC_{uu} or ODC_{gg} problem) and are very difficult. Thus, either heuristic algorithms are proposed or problems with restricted objective function and constraints have been solved.

Lukes ('72,'74,'75) used a dynamic programming technique to solve OUC_{uu} and $ODC_{t^\circ t^\circ}$ with restricted objective function and constraints and provided an $O(n \cdot B^2)$ solution for the latter. The n is the number of vertices in the given graph and B is an integer weight bound of the clusters. In this study, Lukes' technique is used with considerable modifications and extensions.

Recently, efficient algorithms have been found for $ODC_{t^\circ t^\circ}$ and OUC_{ss}. They are:

- Bertolazzi et al ('80) developed an $O(n \cdot B)$ algorithm for the integer version of OUC_{ss};
- Johnson and Niemi ('83) found an $O(n^2 \cdot B)$ algorithm for a nonnegative integer version of $ODC_{t^\circ t^\circ}$ where the objective function value, the vertex weight and the bound B are nonnegative integers;
- Schrader ('83) devised an approximations algorithm for $ODC_{t^\circ t^\circ}$;
- Perl and Snir ('83) discussed a generalized solution for the integer version of $ODC_{t^\circ t^\circ}$ -- the

generalization is to allow an additional integer bound on the total objective function value of each cluster.

1.4 The Organization of the Dissertation

This dissertation is organized into seven chapters. In Chapter 1, the motivation, the problem definition and classifications, complexity issues and a literature survey are presented. In Chapters 2 through 4, three subsets of the ODC problem solutions involving only general digraph, acyclic digraph, out-necklace, out-tree and out-star are presented (See Figure 1-2). The discussions are restricted to a simple objective function and two constraints. The division of these ODC problems into three chapters can be formulated as an ODC problem. Its solution is discussed in Appendix A.

In Chapter 5, solutions presented in Chapters 2 to 4 are used to solve the rest of the ODC problems and all of the OUC problems. Also included are: how to deal with disconnected graphs and graphs with parallel arcs or loops; an extension to the class of objective functions and constraints that can be defined for all OGC subproblems so that the time complexity of the corresponding algorithm remains the same; and a technique to lessen the high complexity of algorithms presented in Chapter 4.

In Chapter 6, three information system design problems are presented to demonstrate the usefulness of the solutions. The three applications are: B-tree secondary storage allocation, translation of a database integrated

schema into an IMS schema, and database record clustering.

A summary of results and a future research direction are given in Chapter 7.

CHAPTER 2

OPTIMAL OUT-TREE/OUT-STAR CLUSTERING OF OUT-TREE/OUT-STAR PROBLEMS

This chapter analyzes and solves three ODC problems: $ODC_{t°t°}$, $ODC_{s°t°}$ and $ODC_{s°s°}$ (cluster 1 in Appendix A). Since the algorithm used to solve $ODC_{t°t°}$ is fundamental to the understanding of all solutions to the OGC subproblems, a detailed description of the approach to solving it and its solution are first presented in section 2.1. In section 2.2, a solution for $ODC_{s°t°}$ is discussed. The solution for $ODC_{s°s°}$ is given in section 2.3.

Throughout this chapter and Chapters 3 and 4, two simple constraints and a simple objective function are used. They are defined in terms of:

1) a vertex weight w_r, which is a nonnegative integer, and it is associated with each vertex v_r of V;
2) an arc value f_{ij}, which is a nonnegative real number, and it is associated with each arc (v_r,v_j) of E; and
3) a weight bound B, which is a nonnegative integer.

Also, it is defined here that cluster weight W(C) is the sum of the vertex weight of all vertices in cluster C and a cluster value F(C) is the sum of the arc values of all arcs in cluster C.

2.1 Optimal Out-Tree Clustering of an Out-Tree Problem ($ODC_{t^{\circ}t^{\circ}}$)

The $ODC_{t^{\circ}t^{\circ}}$ problem is to find a set of nonvoid and disjoint clusters that are subgraphs of the given out-tree such that a set of constraints (one of them requires each cluster being an out-tree or its subclass) is satisfied while an objective function is optimized.

A maximization version of $ODC_{t^{\circ}t^{\circ}}$ is defined as:

Given:

1) an out-tree G=(V,E);
2) two constraints on the set of clusters induced by a clustering K are:
   a) cluster graph class constraint: every cluster must be an out-tree or its subclass;
   b) integer cluster weight constraint: every cluster C_p must have a cluster weight less than or equal to B;
3) an objective function F:PK--> $R^{+}\cup\{0\}$ (nonnegative real number) which is defined as the sum of the cluster values of all clusters C_p induced by a clustering K, i.e,

$$F(K) = \sum_{\text{all } C_p \text{ induced by } K} F(C_p)$$

$$= \sum_{\text{all } C_p \text{ induced by } K} \left(\sum_{(i,j)\in C_p} f_{ij} \right)$$

$$= \sum_{X_{ij}=1} f_{ij} = \sum_{(i,j)\in E} X_{ij} \cdot f_{ij}.$$

Find: a feasible clustering K such that $F(K)$ is maximized.

A minimization version of $ODC_{t^{\circ}t^{\circ}}$ can be defined

similarly by changing the objective function to

$$F(K) = \overline{\sum_{(i,j)\in E}} f_{ij} - \overline{\sum_{X_{ij}=1}} f_{ij}$$

$$= \overline{\sum_{X_{ij}=0}} f_{ij} = \overline{\sum_{(i,j)\in E_{t^\circ}}} \bar{X}_{ij} \cdot f_{ij}$$

and the problem is to find a feasible clustering K such that $F(K)$ is minimized.

Obviously, the maximization and minimization versions are equivalent. Hereafter, only the maximization version will be used to define problems and to illustrate their solutions. The solution approach will first be presented in section 2.1.1 and an algorithm to solve $ODC_{t^\circ t^\circ}$ in section 2.1.2.

2.1.1 A Sequential Clustering Approach to $ODC_{t^\circ t^\circ}$

From the definition of a clustering (section 1.1) an enumerative clustering approach can be used to solve $ODC_{t^\circ t^\circ}$ and the rest of the problems being considered in this dissertation. Such an approach consists of three steps:

1) Enumerate all possible clusterings by cutting all possible combinations of one arc, two arcs, ..., and m arcs.
2) Determine a subset of the set of possible clusterings that satisfies all constraints.
3) Find an optimal value clustering among these feasible clusterings.

The first step forms

$$\binom{m}{1} + \binom{m}{2} + \ldots + \binom{m}{m} = 2^m \text{ clusterings}$$

and is therefore impractical. It is proposed here that a sequential clustering approach can solve $ODC_{t^0t^0}$ in pseudopolynomial time.

Given an out-tree G, a sequential clustering approach sequences the clustering process into a finite sequentially ordered stages. At each stage, a larger set of subgraphs of G, which is obtained from the immediate previous stage by adding a vertex or an arc, is considered. Because the set of sugraphs of G considered at one stage contains only one more vertex or arc than its immediate previous stage, the set of all feasible clusterings of one stage can be obtained from the immediate previous stage by simple operations (i.e., operations that take constant time to execute). The main thrust is to use the integer cluster weight constraint and objective function to define an elimination rule such that at most a constant number of feasible clusterings (B) remains at each stage. Thus, the problem can be solved in pseudopolynomial time.

To describe how the clustering process is sequenced and to identify the simple operations needed in the sequential clustering approach, it is necessary to first define some terms and notations.

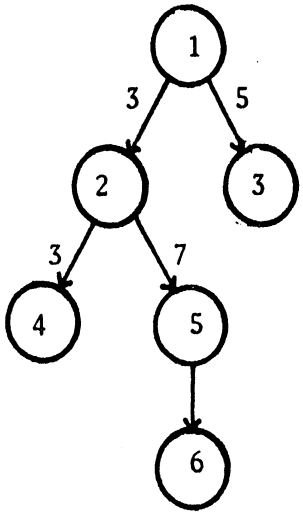
Terms and Notations:

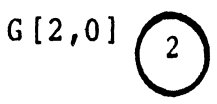
Let the given out-tree G=(V,E) have m arcs and n vertices and let v_{root} be the root of G. For each $v_r \in V$, $G[v_r]$ is any subgraph of G rooted at v_r. $od(v_r)$ is the outdegree of v_r where $od(v_r)=0$ if v_r is a leaf vertex. For each

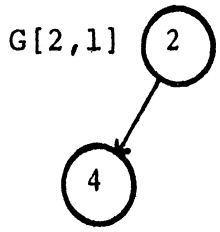
$v_r \in V$ and i, $0 \leq i \leq od(v_r)$, $G[v_r,i]$ is the subgraph of G induced by v_r, by its first i children (from left to right), and by all their descendants (see Figure 2-1). Note that $G[v_{root},od(v_{root})]=G$ and $G[v_r,0]$ is the subgraph of G consisting of only the vertex v_r. A subgraph $G[v_r,i]$ of G rooted at v_r is a complete subgraph if $i = od(v_r)$ and is a partial subgraph if $i < od(v_r)$. Observe that $G[v_r,i+1]$ is a larger subgraph of G than $G[v_r,i]$. It not only contains the partial subgraph $G[v_r,i]$ rooted at v_r but also the complete subgraph rooted at v_j, which is the i+1th child of v_r (denoted by $G[v_j,od(v_j)]$), and the arc (v_r,v_j) connecting the two subgraphs $G[v_r,i]$ and $G[v_j,od(v_j)]$.

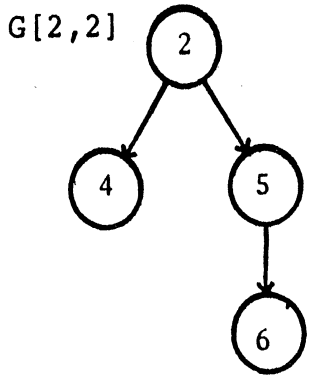
The clustering process is sequenced in stages by visiting the vertices of G in postorder traversal sequence (Aho et al '74). At each vertex v_r, $G[v_r,0]$ is first considered and then $G[v_r,1]$, $G[v_r,2]$, ..., $G[v_r,od(v_r)]$. Since there are n vertices and m arcs, there are m+n stages. Figure 2-2 shows all eleven stages of the clustering process of the out-tree that has six vertices and five arcs given in Figure 2-1. Note that each vertex v_r is involved in $od(v_r)+1$ stages.

In Figure 2-2, because the vertices are visited in postorder traversal sequence, at each stage of the clustering process, the graph considered are a set of subgraphs of G (not necessarily a single connected subgraph). For example, at stage 6, there are two









B = 3

$\forall v_i$ (w_i=1)

Figure 2-1 An out-tree and its G[v_r,i]'s.

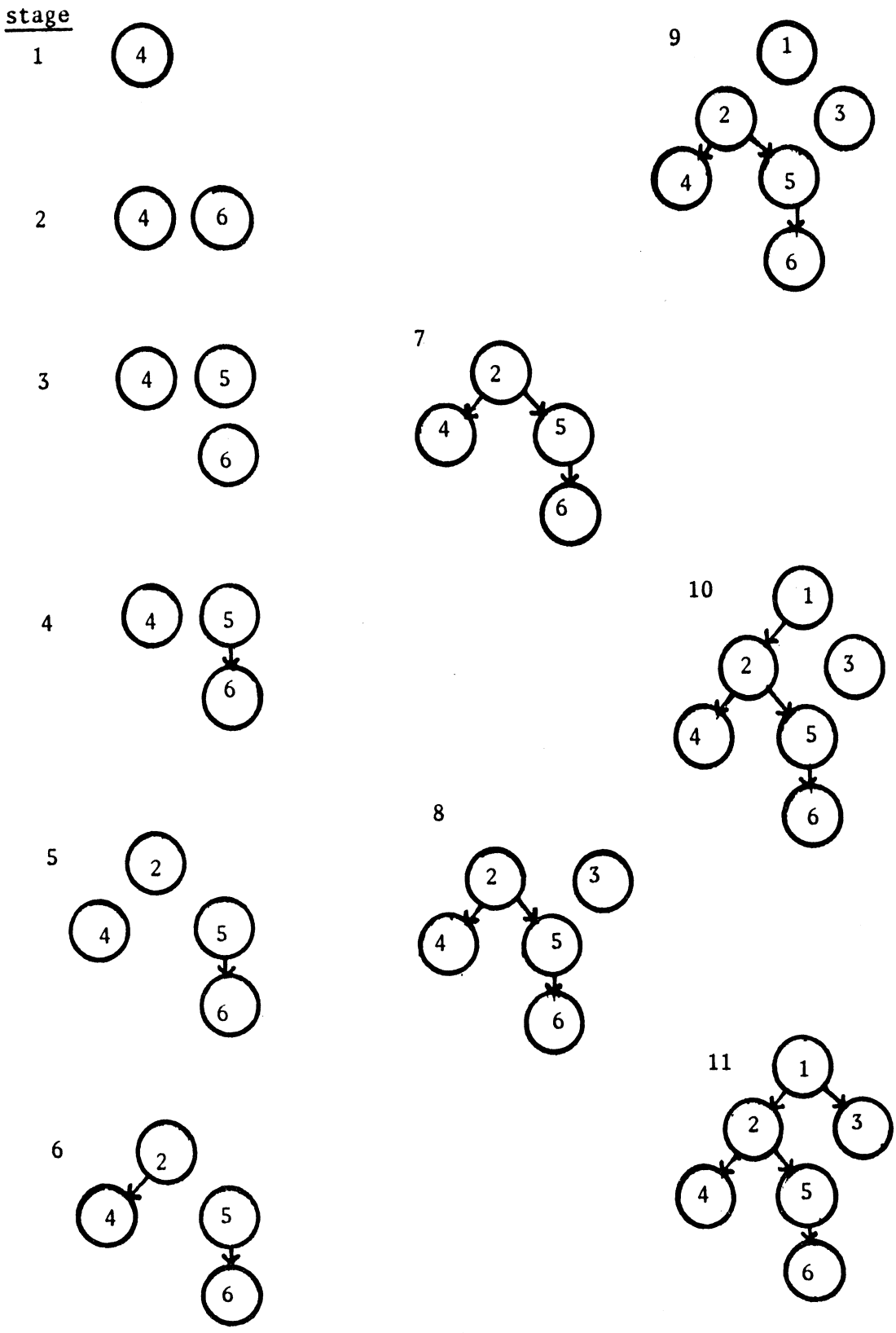


Figure 2-2 The eleven stages of the clustering process of the out-tree in Figure 2-1.

subgraphs: {V = {2,4}, E = {(2,4)}} and {V = {5,6}, E = {(5,6)}}. Except for maybe one, each subgraph in the set is a complete subgraph of G. For example, at stage 6, subgraph {V = {5,6}, E = {(5,6)}} is a complete subgraph rooted at 5 while {V = {2,4}, E = {(2,4)}} is a partial subgraph rooted at 2. At stage 7, there is only one subgraph rooted at 2 and it is a complete subgraph. There are two patterns of changes in the set of subgraphs considered from stage to stage:

1) If all subgraphs in stage k are complete subgraphs, then in stage k+1 the set of subgraphs considered will contain a new vertex as a separate subgraph. This stage k+1 is called a new vertex stage (e.g., stages 1,2,3,5,8,and 9 in Figure 2-2).

2) If one of the subgraphs in stage k is a partial subgraph, then in stage k+1 the set of subgraphs considered will contain a new arc which connects the root of this partial subgraph with the root of a complete subgraph existing in the set of subgraphs considered in stage k. This stage k+1 is called a new arc stage (e.g., stages 4,6,7,10,and 11 in Figure 2-2).

Since the set of subgraphs of G considered at each new stage only contains one more new vertex or arc than the immediate previous stage, there is no need to form all possible clusterings from scratch for each new stage. This means that if the set of all possible clustering PK_k of stage k is

known, the set of all possible clusterings PK_{k+1} of the next stage k+1 can be obtained by some simple operations on PK_k.

The set of all possible clustering PK_k formed at stage k can be represented by s subsets of clusterings if there are s disjoint subgraphs considered at that stage. Each subset of clusterings represents the set of all possible clusterings for each subgraph. Let each clustering be represented by the set of clusters it induces. For example, in Figure 2-2, {V={4},E=Ø},{{V={5,6},E=Ø}, {V={5,6},E={(5,6)}} is all possible clusterings of the two subgraphs in stage 4. To obtain all possible clusterings of stage 5, a new vertex stage, an operation which "inserts" vertex 2 as a separate cluster into the set of possible clusterings is needed. Thus, the resulting clusterings of stage 5 is {V={2},E=Ø},{{4},E=Ø},{{V={5,6},E={(5,6)},{V={5,6},E=Ø}}. Consider stage 6, a new arc stage, since vertices 2 and 4 are connected by arc (2,4), there is a possibility that {V={2},E=Ø} and {V={4},E=Ø} could be "connected" together to form one cluster after clustering (i.e, assignment X_{24}=1) or not be connected together but simply "put" next to each other (i.e., assignment X_{24}=0). Thus, the new set of possible clusterings in stage 6 is {{V={2,4},E={(2,4)}},{V={2,4},E=Ø}}, {{V={5,6},E={(5,6)}},{V={5,6},E=Ø}}.

It is now obvious that the necessary operations needed in the sequential clustering approach are an insertion operation in the new vertex stage and connect/put operations

in the new arc stage. To formalize these operations, let $\{G[v_1,od(v_1)], G[v_2,od(v_2)], \ldots, G[v_j,od(v_j)], \ldots, G[v_r,i]\}$ be the set of mutually disjoint subgraphs rooted at $v_1,v_2,\ldots,v_j,\ldots,v_r$ that are being considered in stage k, with $G[v_j,od(v_j)]$ as the i+1th complete child subgraph of v_r. Let PK_k be the set of all possible clusterings formed at stage k. PK_k includes $PK(G[v_1,od(v_1)], PK(G[v_2,od(v_2)]), \ldots, PK(G[v_j,od(v_j)], \ldots,$ and $PK(G[v_r,i])$ because there are r disjoint subgraphs considered at stage k. The operations needed in the two types of stages are:

For new vertex stage:

At stage k+1, the set of subgraphs considered is $\{G[v_1,od(v_1)], G[v_2,od(v_2)], \ldots, G[v_j,od(v_j)], \ldots, G[v_r,i], G[v_p,0]\}$ where $G[v_p,0]$ represents the new vertex v_p that is introduced. There is only one way to obtain a clustering of a single vertex, which is a cluster containing only this vertex. Thus, the new set of all possible clusterings can be obtained by inserting $\{\{V=\{v_p\},E=\emptyset\}\}$ into PK_k, i.e.,

INSERT operation:

$$PK_{k+1} = PK_k \cup \{\{V=\{v_p\},E=\emptyset\}\}.$$

For new arc stage:

At stage k+1, the set of subgraphs considered is $\{G[v_1,od(v_1)], G[v_2,od(v_2)], \ldots, G[v_{j-1},od(v_{j-1})], G[v_{j+1}, od(v_{j+1})], \ldots, G[v_{r-1},od(v_{r-1})], G[v_r,i+1]\}$ where $G[v_r,i+1]$ is a subgraph formed by connecting $G[v_j,od(v_j)]$ and $G[v_r,i]$

that exist in stage k with an arc (v_r,v_j). Let the set of possible clusterings of $G[v_j,od(v_j)]$ and $G[v_r,i]$ be denoted by $PK[v_j,od(v_j)]$ and $PK[v_r,i]$ respectively. Thus,

$$PK_{k+1} = PK_k - PK[v_r,i] - PK[v_j,od(v_j)] \cup PK[v_r,i+1].$$

Let $K[v_r,i]$ and $K[v_j,od(v_j)]$ be any two clusterings of $PK[v_r,i]$ and $PK[v_j,od(v_j)]$ respectively. Let $K[v_r,i] = \{C_1,C_2,\ldots,C_q\}$, and $K[v_j,od(v_j)] = \{C_1',C_2',\ldots,C_p'\}$, and assume C_1 and C_1' are clusters that contain the root of $G[v_r,i]$ and $G[v_j,od(v_j)]$ respectively.

The set of all possible clusterings for $PK[v_r,i+1]$ can be obtained from all possible combinations of $K[v_r,i]$ in $PK[v_r,i]$ and $K[v_j,od(v_j)]$ in $PK[v_j,od(v_j)]$ for assignment $X_{ij}=$ 0 (denoted by $PK^0[v_r,i+1]$) and 1 (denoted by $PK^1[v_r,i+1]$) on arc (v_r,v_j), i.e.,

$PK[v_r,i+1] = PK^0[v_r,i+1] \cup PK^1[v_r,i+1]$;

$PK^0[v_r,i+1] = \{K^0[v_r,i+1]$ is formed using PUT for every pair of $K[v_r,i]$ and $K[v_j,od(v_j)]\}$;

$PK^1[v_r,i+1] = \{K^1[v_r,i+1]$ is formed using CONNECT for every pair of $K[v_r,i]$ and $K[v_j,od(v_j)]\}$, where

PUT operation:

$K^0[v_r,i+1] = K[v_r,i] \cup K[v_j,od(v_j)]$;

CONNECT operation:

$K^1[v_r,i+1] = \{C_2,C_3,\ldots,C_p,C_2',C_3',\ldots,C_q'\} \cup \{C_1"\}$

where $C_1"$ is equal to the connecting of clusters C_1 and C_1' by arc (v_r,v_j).

To obtain all feasible clusterings instead of just all possible clusterings, the out-tree cluster and cluster

weight constraints need to be enforced in each stage. This can be easily done because:

1) Any cluster formed in the sequential clustering approach is a subgraph of an out-tree and is also an out-tree itself. Thus, the out-tree cluster constraint is automatically enforced in the process.

2) For each pair of clusterings from $PK[v_r,i]$ and $PK[v_j,od(i+1)]$, only the CONNECT operation form a new cluster $C_1"$ which has a new higher cluster weight value. To make sure every cluster formed is less than B at each new arc stage, the CONNECT operation can be made <u>conditional</u> as follows:

   IF $W(C_1) + W(C_1') \leq B$ THEN

   $K^1[v_r,i+1] = \{C_2,C_3,\ldots,C_p,C_2',C_3',\ldots,C_q'\} \cup \{C_1"\}$

Thus, the set of all feasible clusterings formed at stage k+1 (denoted PK^f_{k+1}) can be easily obtained by applying either INSERT or PUT/conditional CONNECT operations on PK^f_k.

2.1.2 $ODC_{t^\circ t^\circ}$-1 Algorithm

For computational purpose, let each feasible clustering K of a subgraph be represented by a triplet (F(K),W(K),CF(K)) that contains the following three components:

1) A <u>value component</u> F(K) which stores the sum of the cluster values of all the clusters induced by the clustering.

2) A <u>weight component</u> W(K) which stores the cluster

weight of the cluster that contains the root of the subgraph.

3) A <u>configuration component</u> CF(K) which stores the clusters induced by this clustering of the subgraph and each cluster is represented by the vertices in it enclosed by "< >".

At each stage of the clustering process, if there are s subgraphs considered at a stage, there are s sets of feasible triplets formed. Each of these s sets of feasible triplets represents all feasible clusterings of a subgraph considered at that stage.

Notation:

Let PK_k^f and PK_{k+1}^f now represent all feasible triplets produced at stages k and k+1, and let $PK^f[v_r,i+1]$, $PK^f[v_r,i]$, and $PK^f[v_j,od(v_j)]$ now represent all feasible triplets of the subgraph $G[v_r,i+1]$, $G[v_r,i]$ and $G[v_j,od(v_j)]$ respectively. Let $K[v_r,i+1]$, $K[v_r,i]$ and $K[v_j,od(v_j)]$ be a feasible triplet in $PK^f[v_r,i+1]$, $PK^f[v_r,i]$ and $PK^f[v_j,od(v_j)]$ respectively.

The three operations can now be redefined in terms of triplet formation:

INSERT operation:

$$PK_{k+1}^f = PK_k^f \cup \{(0,w_p,<v_p>)\}.$$

PUT operation:

form $K^\circ[v_r,i+1]$ with

$F(K[v_r,i+1]) = F(K[v_r,i]) + F(K[v_j,od(v_j)])$,

$W(K[v_r,i+1]) = W(K[v_r,i])$, and

$CF(K[v_r,i+1]) = CF(K[v_r,i]) \cup CF(K[v_j,od(v_j)])$.

Conditional CONNECT operation:

IF $W(C_1) + W(C_1') \leq B$ THEN

form $K^1[v_r,i+1]$ with

$F(K[v_r,i+1]) = F(K[v_r,i]) + F(K[v_j,od(v_j)]) + f_{rj}$,

$W(K[v_r,i+1]) = W(C_1) + W(C_1')$, and

$CF(K[v_r,i+1]) = \{C_2,C_3,\ldots,C_p,C_2', C_3',\ldots,C_q'\} \cup \{C_1 \cup C_1'\}$.

For example, let {(0,1,<4>)},{(0,1,<5><6>),(2,2,<5,6>)} be the set of all feasible triplets formed at stage 4 in Figure 2-2. The feasible triplet sets formed at stage 5 using the INSERT operation are {(0,1,<2>)},{(0,1,<4>)},{(0,1,<5><6>),(2,2,<5,6>)}. The feasible triplet sets formed at stages 6 and 7 using the PUT and conditional CONNECT are: (Assume B=3 and $\forall v_p$ $(w_p=1)$.)

stage 6:

{(0,1,<2><4>),(3,2,<2,4>)},

{(0,1,<5><6>),(2,2,<5,6>)};

stage 7:

{(0,1,<2><4><5><6>),(2,1,<2><4><5,6>),

(3,2,<2,4><5><6>),(5,2,<2,4><5,6>),(7,2,<2,5><4><6>),

(9,3,<2,5,6><4>),(10,3,<2,4,5><6>)}.

In the above example, every new vertex stage forms one additional new triplet using INSERT while every new arc

stage can double the number of triplets formed in the immediate previous stage using PUT/conditional CONNECT. Thus, unless reduction of the number of feasible triplets can be done at the end of each new arc stage, the total computation time can still be exponential. Fortunately, some triplets formed at each new arc stage can be eliminated because they can never be extended to the optimal clustering of the out-tree.

The idea is to define a dominance relation on the set of new feasible triplets formed at each new arc stage. Let K_1 and K_2 be any two feasible triplets formed at the same new arc stage. K_1 dominates K_2 if $F(K_1) \geq F(K_2)$ and $W(K_1) \leq W(K_2)$.

Proposition 2.1: The dominated triplet K_2 cannot be extended to the optimal clustering of the out-tree.

Proof: (See Proposition 4.4 in Chapter 4 where a generalized proof is given.) ***Q.E.D.***

Proposition 2.1 implies that at the end of each new arc stage, all dominated triplets can be eliminated. The elimination can be done in two steps by decomposing $W(K_1) \leq W(K_2)$ into

$W(K_1) = W(K_2)$ and $W(K_1) < W(K_2)$:

step 1) $W(K_1) = W(K_2)$ and $F(K_1) \geq F(K_2)$:

For triplets that have the same weight, retain the one with the highest value and eliminate the rest. Thus, only one triplet remains for each integer weight value.

step 2) $W(K_1) < W(K_2)$ and $F(K_1) \geq F(K_2)$:

For every remaining triplet K_2, if it has higher weight than another triplet K_1 but have lower or equal value, then eliminate K_2.

Assume that every vertex has vertex weight 1. Since the weight of a triplet which represents a feasible clustering of a subgraph is an integer less than or equal to B, the maximum number of triplets remaining for a subgraph after each stage is B. If the vertex has weight higher than 1, and suppose Z is the maximum number of vertices that can be put in a cluster with cluster weight less than or equal to B, then the maximum number of triplets remaining for a subgraph after each stage is Z (<B).

A complete algorithm called $ODC_{t^\circ t^\circ}$-1 can now be specified:

ALGORITHM: $ODC_{t^\circ t^\circ}$-1

INPUT: an out-tree G;

OUTPUT: an optimal out-tree clustering of G and its value;

PROCESS:

I) FOR each v_r visited in postorder traversal sequence DO:

1) FOR j=0 to $od(v_r)$ DO:

1.1) IF j=0 THEN form a new PK^f_{k+1} from PK^f_k using INSERT; ELSE

1.2) form a new PK^f_{k+1} from PK^f_k using PUT and conditional CONNECT;

eliminate all dominated triplets;

END;

II) find the optimal value clustering in PK^{f}_{m+n} and RETURN the solution;

END.

Proposition 2.2: The running time of algorithm $ODC_{t^{\circ}t^{\circ}}$-1 is $O(n \cdot Z^2)$ (or $O(n \cdot B^2)$ if every vertex has vertex weight 1).

Proof: Every new vertex stage takes constant time while every new arc stage takes $2 \cdot Z^2$ time because of the PUT and conditional CONNECT. Thus, step I) takes $O(n \cdot Z^2)$ time. Step II) takes O(Z) time. The time complexity of $ODC_{t^{\circ}t^{\circ}}$ is therefore $O(n \cdot Z^2)$. ***Q.E.D.***

Example: Applying $ODC_{t^{\circ}t^{\circ}}$-1 to the example given in Figure 2-2, the following 24 triplets are formed during the clustering process:

formed triplets	formed by INSERT (stage #)	formed by P/C (stage #)	elimi. by P/C (stage #)	elim. by Dominate rel. (stage #)
(0,1,<4>)	1		6	
(0,1,<6>)	2		4	
(0,1,<5>)	3		4	
(0,1,<5><6>)		4	7	
(2,2,<5,6>)		4	7	
(0,1,<2>)	5		6	
(0,1,<2><4>)		6	7	
(3,2,<2,4>)		6	7	
(0,1,<2><4><5><6>)		7		7
(2,1,<2><4><5,6>)		7	10	
(7,2,<2,5><4><6>)		7	10	
(9,3,<2,5,6><4>)		7		7
(3,2,<2,4><5><6>)		7		7
(5,2,<2,4><5,6>)		7		7
(10,3,<2,4,5><6>)		7	10	

(0,1,<3>)	8		11	
(0,1,<1>)	9		10	
(2,1,<1><2><4><5,6>)		10		10
(7,1,<1><2,5><4><6>)		10		10
(10,1,<1><2,4,5><6>)		10	11	
(5,2,<1,2><4><5,6>)		10		10
(10,3,<1,2,5><4><6>)		10		10
(10,1,<1><2,4,5><6><3>)		11		
(15,2,<1,3><2,4,5><6>)		11		

The optimal solution is (15,2,<1,3><2,4,5><6>).

Further improvements on $ODC_{t°t°}$-1 can be made, but since the time complexity will not change, we will not discuss them here.

2.2 Optimal Out-Star Clustering of an Out-Star Problem ($ODC_{s°s°}$)

The $ODC_{s°s°}$ problem is to find a set of <u>bounded</u> out-star clusters, or its subclass clusters (i.e, those that have cluster weight less than B), from an out-star such that the total value of the set is optimized. A formal definition for the problem is the same as that of $ODC_{t°t°}$ except that the given digraph is an out-star and each cluster C_p must be an out-star or its subclass.

Since an out-star is a special case of out-tree, all previous results of $ODC_{t°t°}$ are applicable here. Thus, an algorithm $ODC_{s°s°}$-1, which is identical to $ODC_{t°t°}$-1, can be used to solve $ODC_{s°s°}$ and even have a lower time complexity.

Proposition 2.3: The application of $ODC_{s°s°}$-1 yields a time complexity of $O(n \cdot Z)$.

Proof: The time complexity is $O(n \cdot Z)$ instead of $O(n \cdot Z^2)$ because each $PK^f[v_j, od(v_j)]$ has only one triplet in it

instead of Z triplets. Thus, each new arc stage takes O(n·Z) time. *Q.E.D.*

Example: Applying the $ODC_{t°t°}$-1 to the out-star shown in Figure 2-3 yields the optimal clustering solution as (8,3,<1,3,4><2>).

2.3 Optimal Out-Star Clustering of an Out-Tree Problem ($ODC_{s°t°}$)

The $ODC_{s°t°}$ problem is to find a set of bounded out-star clusters, or its subclass clusters, from an out-tree such that the total value of the set is maximized. A formal definition for this problem is the same as that of $ODC_{t°t°}$ except that each cluster C_p must be an out-star or its subclass.

The constraint that each cluster must be an out-star or its subclass can be enforced in many ways (e.g., incorporate the constraint as an additional condition for applying conditional CONNECT). The following proposition suggests the most effective way to enforce the constraint while lowering the time complexity of the solution algorithm.

Proposition 2.4: Only two triplets need to be retained in each $PK^f[v_j,od(v_j)]$. One has a cluster containing v_j alone, the other has the maximum value.

Proof: The constraint that each cluster must be an out-star or its subclass means that if a vertex v_j is connected to its parent vertex v_r in a cluster by the conditional CONNECT

Figure 2-3 An example out-star and its optimal clustering result.

operation (i.e., assignment X_{rj}=1), then v_j cannot have any child v_k connected to it (i.e., every assignement X_{jk} must be 0) and vice versa (otherwise the formed cluster is not an out-star or its subclass). Thus, not every pair of triplets in $PK^f[v_r,i]$ and $PK^f[v_j,od(v_j)]$ can be used by conditional CONNECT to form a new triplet. As a matter of fact, only a triplet in $PK^f[v_j,od(v_j)]$ which contains v_j alone as a cluster can be used to form a new triplet with every triplet in $PK^f[v_i,i]$.

Similarly, given a triplet $K[v_r,i]$ in $PK^f[v_r,i]$, there is no need to form Z triplets from $K[v_r,i]$ and every triplet in $PK^f[v_j,od(v_j)]$ using PUT. This is because all these Z formed triplets have the same weight and thus only the one with the highest value remains at the end of this stage. This implies that only the triplet in $PK^f[v_j,od(v_j)]$ with the maximum value is necessary to form a new cluster with every triplet in $PK^f[v_r,i]$ using PUT.

From both arguments above, it can be concluded that there is only two triplets needed to be retained for each $PK^f[v_j,od(v_j)]$. ***Q.E.D.***

An algorithm $ODC_{s\circ t\circ}$-1 for $ODC_{s\circ t\circ}$ can thus be easily modified from $ODC_{t\circ t\circ}$ as follows:

ALGORITHM: $ODC_{s\circ t\circ}$-1

INPUT: an out-tree G;

OUTPUT: an optimal out-star clustering of G and its value;

PROCESS:

I) FOR each v_r visited in postorder traversal sequence

DO:

1) FOR j=0 to od(v_r) DO:

1.1) same as 1.1) of $ODC_{t°t°}$-1;

1.2) same as 1.2) of $ODC_{t°t°}$-1;

END;

2) eliminate all triplets except two: the triplet obtaining a cluster that contains v_r alone and the triplet with the highest value;

END;

II) same as II) of ODC(t°t°)-1;

END.

Proposition 2.5: The running time of algorithm $ODC_{s°t°}$-1 is O(n·Z).

Proof: Immediate from that every $PK^f[v_j,od(v_j)]$ contains only two triplets. ***Q.E.D.***

Example: Applying $ODC_{s°t°}$-1 to the out-tree shown in Figure 2-1, the optimal clustering solution is (15,2,<1,3><2,4,5><6>).

CHAPTER 3

OPTIMAL OUT-TREE/OUT-STAR/OUT-NECKLACE CLUSTERING OF GENERAL DIGRAPH/ACYCLIC DIGRAPH/NECKLACE PROBLEMS

This chapter uses the cluster graph class constraint and $ODC_{t\circ t\circ}/ODC_{s\circ t\circ}/ODC_{s\circ n\circ}$ to analyze and solve eight ODC problems: $ODC_{t\circ g}$, $ODC_{t\circ a}$, $ODC_{t\circ n\circ}$, $ODC_{s\circ g}$, $ODC_{s\circ a}$, $ODC_{s\circ n\circ}$, $ODC_{n\circ n\circ}$ and $ODC_{n\circ g}$ (clusters 2, 3, 4, 5 and 6 in Appendix A). Since the solution of the first three problems can be directly applied to the rest, an analysis and solution of the first three problems are given first in sections 3.1 and 3.2. The solution to the next three problems are given in section 3.3. In section 3.4, solutions to the last two are given.

3.1 The Structures of the Set of Possible Clusterings that Satisfy the Cluster Graph Class Constraint

Let K_i and K_j be any two clusterings of a digraph G. $K_i \leq K_j$ if every cluster induced by K_i is a subgraph of some cluster induced by K_j. For example, in Figure 3-1, each K_3, K_4, K_5, K_8, K_9, K_{10} and $K_{12} \leq K_1$. Note that if $K_i \leq K_j$, then K_i can be obtained from K_j by removing some arcs in K_j. The $\leq$ relation is reflexive, transitive and anti-symmetric. It induces a partial ordering on the set of all possible

clusterings PK of G. Let [PK,≤] denote the partly ordered set (poset) that consists of a set PK and a partial ordering relation ≤ on PK.

For any two elements K_i and K_j in PK, an element K_k in PK is called the greatest lower bound, or meet of K_i and K_j, denoted by $K_i \cdot K_j$ if

$K_k \leq K_i$, $K_k \leq K_j$, and

$\forall$ K_l in PK such that $K_l \leq K_i$ and $K_l \leq K_j$ imply $K_l \leq K_k$.

Dually, an element $K_m \in$ PK is called the least upper bound, or join of K_i and K_j, denoted by $K_i + K_j$, if

$K_i \leq K_m$ and $K_j \leq K_m$ and

$\forall$ K_l in PK such that $K_i \leq K_l$ and $K_j \leq K_l$ imply $K_m \leq K_l$.

A lattice is a partly ordered set which has a least upper bound and a greatest lower bound for every pair of elememts. Let x and y be any of the five digraph classes. Define the set of all x clusterings of a specific y as the subset of all possible clusterings of y which satisfies the constraint that each of these clusterings induces cluster of type x or it subclass.

Proposition 3.1: The set of all out-tree clusterings of an out-tree (and out-star clusterings of an out-star) together with the ≤ relation form a lattice.

Proof: This is immediate from the fact that every subgraph of out-tree/out-star is an out-tree/out-star. Thus, every possible clustering is an out-tree clustering and the set forms a lattice. ***Q.E.D.***

But the set of out-tree clusterings of a general

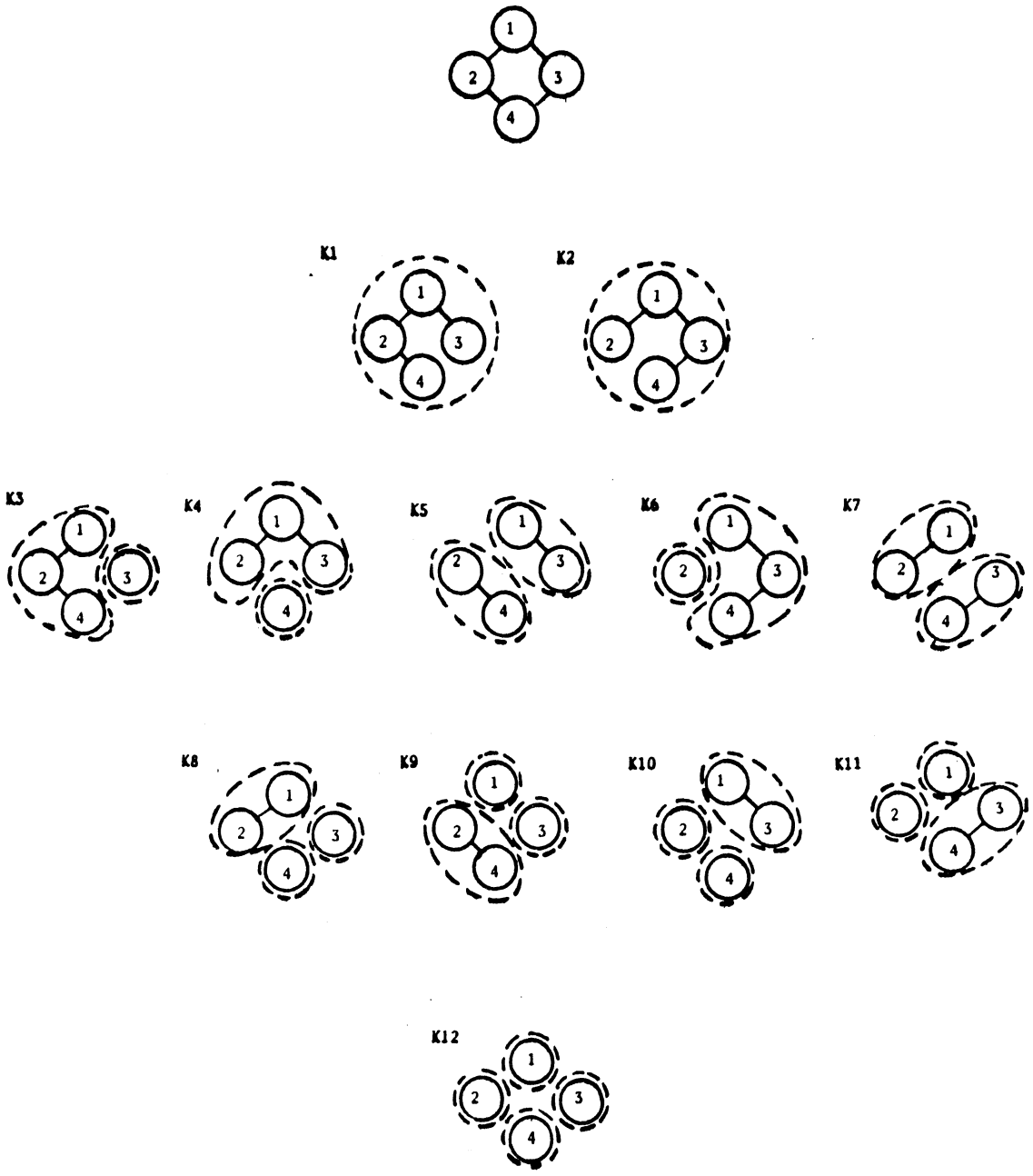


Figure 3-1 All possible out-tree clusterings of an acyclic digraph.

digraph, acyclic digraph or out-necklace is not a lattice because it does not always have a least upper bound for each pair of elements. For example, in Figure 3-1, $K_1 \cdot K_2 = K_4$ and $K_3 + K_{10} = K_1$, but $K_3 + K_7$ is undefined.

A poset [PK,≤] is a join-semilattice (meet-semilattice) if for any two elements K_i and K_j in the poset PK, $K_i + K_j$ ($K_i \cdot K_j$) exists.

Proposition 3.2: The set of all out-tree clusterings of a general digraph (acyclic digraph, out-necklace) together with the ≤ relation form a meet-semilattice.

Proof: For each cluster C_i induced by K_i and C_j by K_j, let $C = C_i \cap C_j$ then form three clusters (where ∩ represents graph intersection):

1) $C_i - C$.
2) $C_j - C$.
3) C.

The set of all clusters so formed for each pair of clusters from K_i and K_j, denoted by K_k, is also a clustering of PK because each cluster is an out-tree or it subclass. Obviously, $K_k \leq K_i$ and $K_k \leq K_j$.

Now, assume that there is a K_l in PK such that $K_l \leq K_i$, $K_l \leq K_j$ and $K_k \leq K_l$. Let $C = C_i \cap C_j \neq \emptyset$ be an induced cluster in K_l. By $K_k \leq K_l$, C must also be a subgraph of some C' induced by K_l. Assume $C - C' \neq \emptyset$, then C' cannot be a subgraph of both K_i and K_j. Thus, C' is not a subgraph of both K_i and K_j. This contradicts the assumption that $K_l \leq K_i$ and $K_l \leq K_j$. Therefore, $K_l \leq K_k$. This means that $\forall K_l$ in PK such

that $K_1 \leq K_i$ and $K_1 \leq K_j$ imply $K_1 \leq K_k$. Thus, the set of all out-tree clusterings of a general digraph/acyclic digraph/out-necklace together with $\leq$ form a meet-semilattice. ***Q.E.D.***

An element $K_i \in PK$ is maximal when there is no other element K_j in PK from which $K_i \leq K_j$ (a minimal element is similarly defined). For a lattice, there is always one minimal element and one maximal element. In a meet-semilattice, we note that there is always one minimal element (e.g., K_{12} in Figure 3-1) and many maximal elements (e.g., K_1 and K_2 in Figure 3-1).

Proposition 3.3: The poset formed by the set of out-tree /out-star clusterings of an out-tree/out-star and the $\leq$ relation has only one maximal out-tree/out-star clustering.

Proof: Immediate from the set of all out-tree/out-star clusterings of an out-tree/out-star which together with the $\leq$ relation form a lattice (from Proposition 3.1). ***Q.E.D.***

Proposition 3.4: The poset formed by the set of out-tree clusterings of an out-necklace and the $\leq$ relation has K maximal out-tree clusterings if the number of arcs in the only cycle in the out-necklace is K.

Proof: Each vertex in an out-necklace has indegree 1, while each vertex in an out-tree also has indegree 1 except for the root which has indegree 0. Thus, a maximal out-tree clustering can be obtained by removing a minimum of one arc from an out-necklace. But not all sets of clusterings generated by removing one arc from the out-necklace are

out-tree clusterings. It is only those that are formed by removing an arc in the cycle of the out-necklace are out-tree clusterings. Thus, if there are K arcs in the out-necklace, there are K maximal out-tree clusterings. ***Q.E.D.***

Proposition 3.5: The poset formed by the set of out-tree clusterings of an acyclic digraph and the ≤ relation has

$$L = \prod_{i=1}^{p} a_i \text{ maximal out-tree clusterings}$$

if the acyclic digraph has p m-indegree vertices where each has indegree a_i.

Proof: Let the indegree of the set of m-indegree vertices in the given acyclic digraph be $a_1, a_2, \ldots, a_p$. To form a maximal out-tree clustering from the acyclic digraph requires the removal of the "redundant" arcs that exist in each of the m-indegree vertices so that every m-indegree vertex can be reduced to have indegree 1. There are

$$\binom{a_1}{a_1-1} \cdot \binom{a_2}{a_2-1} \cdot \ldots \cdot \binom{a_p}{a_p-1}$$

$$= a_1 \cdot a_2 \cdot \ldots \cdot a_p \text{ combinations.}$$

Thus, there are $\prod_{i=1}^{p} a_i$ maximal out-tree clusterings. ***Q.E.D.***

Proposition 3.6: The poset formed by the set of out-necklace clusterings of a general digraph and the ≤ relation has at most

$$L = \prod_{i=1}^{p} a_i \text{ maximal out-necklace clusterings}$$

if the general digraph has p m-indegree vertices where each has indegree a_i.

Proof: Similar to the proof given for Proposition 3.5. *Q.E.D.*

3.2 Optimal Out-Tree Clustering of General Digraph, Acyclic Digraph and Out-Necklace Problems ($ODC_{t\circ g}$, $ODC_{t\circ a}$, $ODC_{t\circ n\circ}$)

The $ODC_{t\circ g}$, $ODC_{t\circ a}$ and $ODC_{t\circ n\circ}$ problems are to find a set of bounded out-tree clusters, or its subclass clusters, from a general digraph, acyclic digraph and out-necklace respectively such that the total value of the set is optimized.

Again, the enumerative clustering approach is too expensive to use. A better approach is to use the maximal out-tree clustering concept and the $ODC_{t\circ t\circ}$-1 discussed in section 2.1. A maximal out-tree clustering induces a set of largest out-tree clusters that can be obtained from an acyclic digraph or out-necklace by removing minimal number of arcs from the m-indegree vertices. Since any possible out-tree clustering, including the optimal one, ≤ one of the maximal out-tree clustering (or can be obtained by additional arc removals of the induced clusters), the optimal solution can thus be found in one of the maximal out-tree clusterings. We can do that by using $ODC_{t\circ t\circ}$-1. The optimal out-tree clustering of an acyclic digraph or out-necklace is therefore the maximumm value one among the set of optimal out-tree clusterings obtained from all maximal out-tree clusterings. An algorithm for these two problems can be formulated as follows:

ALGORITHM: *$ODC_{t\circ a}$-1/$ODC_{t\circ n\circ}$-1*

INPUT: an acyclic digraph/out-necklace G;

OUTPUT: an optimal out-tree clustering and its value;

PROCESS:

1) FOR each maximal out-tree clustering of G which induces one or more out-tree clusters DO:
   1.1) apply algorithm $ODC_{t\circ t\circ}$-1 to obtain an optimal solution for each out-tree cluster;
   1.2) IF the sum of the optimal out-tree clustering values of these out-tree clusters is the largest value so far, keep the solution;

   END;
2) RETURN the last solution kept;

END.

An algorithm for $ODC_{t\circ g}$, named $ODC_{t\circ g}$-1, is similar to algorithm $ODC_{t\circ a}$-1 except that instead of applying $ODC_{t\circ t\circ}$-1 in step 1.1, $ODC_{t\circ n\circ}$-1 is used and all occurrences of maximal out-tree clusterings are changed to maximal out-necklace clusterings.

Proposition 3.7: The $ODC_{t\circ n\circ}$-1, $ODC_{t\circ a}$-1 and $ODCC_{t\circ g}$-1 algorithms find

1) the optimal out-tree clustering of out-necklace in $O(n\cdot Z^2\cdot K)$ time;
2) the optimal out-tree clustering of acyclic digraph in $O(n\cdot Z^2\cdot L)$ time; and
3) the optimal out-tree clustering of general digraph in $O(n\cdot Z^2\cdot \bar{K}\cdot L)$ time.

***Proof*:**

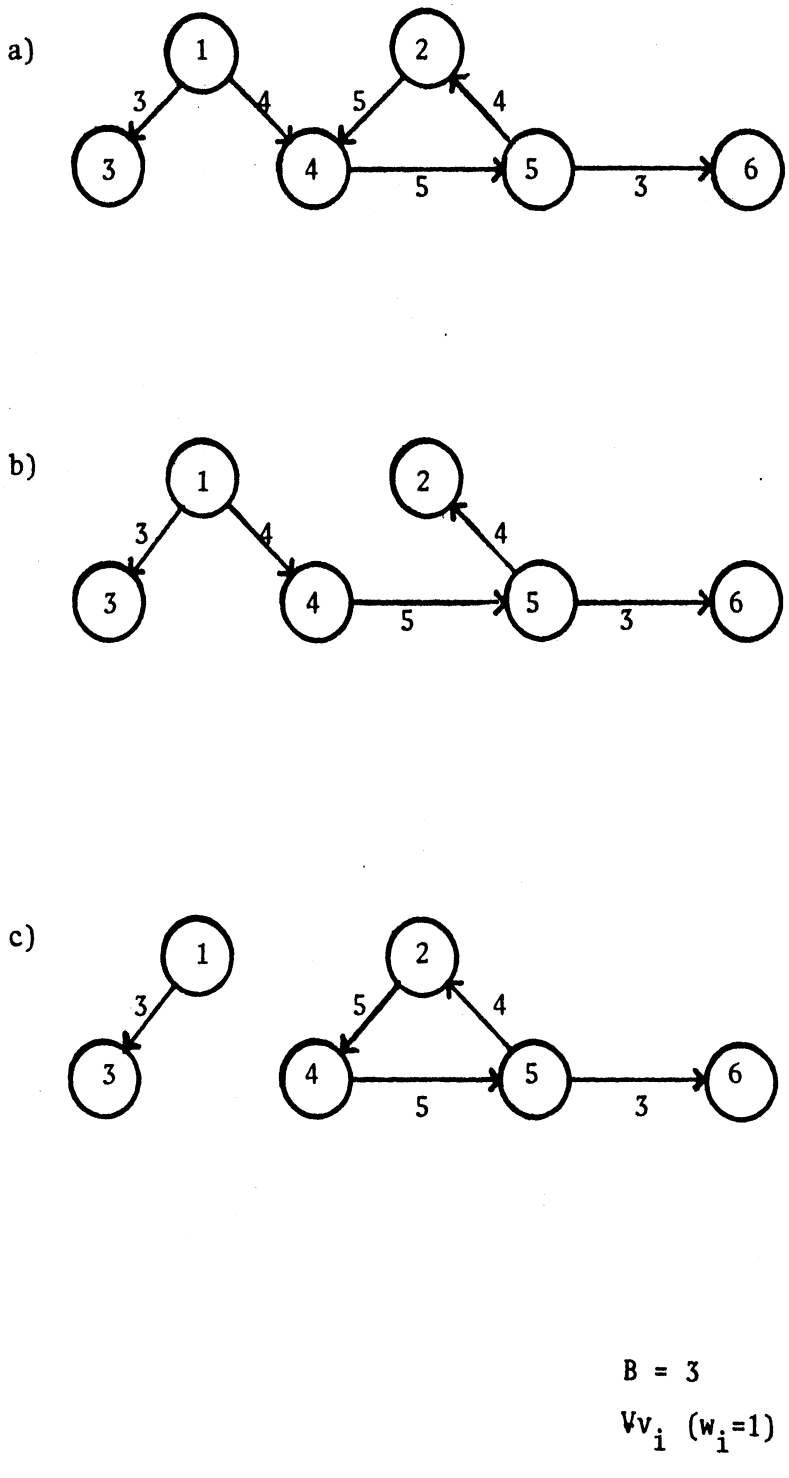
1) In step 1 of $ODC_{t\circ n\circ}$-1, a given maximal out-tree clustering induces only one out-tree cluster. Thus, steps 1.1 and 1.2 take $O(n\cdot Z^2)$ by Proposition 2.2. Since there are K maximal out-tree clusterings to search for (see Proposition 3.4) in the step, the time complexity is $O(n\cdot Z^2\cdot K)$.

2) In step 1 of $ODC_{t\circ a}$-1, let a given maximal out-tree clustering contain w out-trees with $n_1,n_2,\ldots,n_w$ vertices respectively in them. Steps 1.1 and 1.2 take $O(n_1\cdot Z^2)+O(n_2\cdot Z^2)+\ldots+O(n_w\cdot Z^2) = O(n\cdot Z^2)$ time to find an optimal solution. Since there are L maximal out-tree clusterings to search for (see Proposition 3.5) in step 1, the time complexity is $O(n\cdot Z^2\cdot L)$.

3) In step 1 of $ODC_{t\circ g}$-1, let a given maximal necklace clustering contain w necklaces with $n_1,n_2,\ldots,n_w$ vertices respectively in them. Steps 1.1 and 1.2 take $O(n_1\cdot Z^2\cdot K_1)+O(n_2\cdot Z^2\cdot K_2)+\ldots+O(n_w\cdot Z^2\cdot K_w) = O(n\cdot Z^2\cdot \bar{K})$ where $\bar{K}$ is the maximum of all K_i, $1\le i\le w$. Since there are L maximal out-necklace clusterings to search for, the time complexity is $O(n\cdot Z^2\cdot \bar{K}\cdot L)$. ***Q.E.D.***

***Example*:** The general digraph shown in Figure 3-2a contains the two maximal out-necklace clusterings that are in Figures 3-2b and c.

Applying $ODC_{t\circ g}$-1 to the general digraph shown in Figure 3-2a, we find the optimal solution in Figure 3-2b which has a total value of 16 and 2 clusters:



$B = 3$

$\forall v_i \ (w_i=1)$

Figure 3-2 A general digraph and its two maximal out-necklace clusterings.

Cluster 1 contains vertices 1 and 3.

Cluster 2 contains vertices 2, 4, 5 and 6.

3.3 Optimal Out-Star Clustering of General Digraph, Acyclic Digraph and Out-Necklace Problems ($ODC_{s\circ g}$, $ODC_{s\circ a}$, $ODC_{s\circ n\circ}$)

The $ODC_{s\circ g}$, $ODC_{s\circ a}$ and $ODC_{s\circ n\circ}$ problems are to find a set of bounded out-star clusters, or its subclass clusters, from a general digraph, acyclic digraph, and out-necklace respectively such that the total value of this set is optimized. An approach to solving $ODC_{s\circ a}$ and $ODC_{s\circ n\circ}$ can be formulated similarly to the approach presented in section 3.2, which is to use the maximal out-tree clustering concept and the efficient $ODC_{s\circ t\circ}$ algorithm. This is based on the fact that an out-star is a subclass of an out-tree. Since any optimal out-star clustering of the given digraph (acyclic digraph or out-necklace) $\leq$ one of the maximal out-tree clusterings, the optimal solution can be found in one of them. $ODC_{s\circ t\circ}$-1 will obtain an optimal out-star clustering for each maximal out-tree clustering and the clustering with the maximum value is then the optimal out-star clustering of the given digraph. Thus, two algorithms, $ODC_{s\circ a}$-1 and $ODC_{s\circ n\circ}$-1, can be formulated for $ODC_{s\circ a}$ and $ODC_{s\circ n\circ}$ as follows:

ALGORITHM: $ODC_{s\circ a}$-1/$ODC_{s\circ n\circ}$-1

INPUT: an acyclic digraph/out-necklace G;

OUTPUT: an optimal out-star clustering and its value;

PROCESS:

1) FOR each maximal out-tree clustering of G which induces one or more out-tree clusters DO:
   1.1) apply algorithm $ODC_{s\circ t\circ}$-1 to obtain an optimal solution for each out-tree cluster;
   1.2) IF the sum of the optimal out-star clustering values of these out-tree clusters is the largest value so far, keep the solution;
   END;
2) RETURN the last solution kept;

END.

An algorithm for $ODC_{s\circ g}$, named $ODC_{s\circ g}$-1, is similar to $ODC_{s\circ a}$-1 except that instead of applying $ODC_{s\circ t}$-1 in step 1.1, $ODC_{s\circ n\circ}$-1 is needed and all occurrences of maximal out-tree clusterings are replaced by maximal out-necklace clusterings.

Proposition 3.8: The $ODC_{s\circ n\circ}$-1, $ODC_{s\circ a}$-1 and $ODC_{s\circ g}$-1 algorithms find:

1) the optimal out-star clustering of out-necklace in $O(n\cdot Z\cdot K)$ time;
2) the optimal out-star clustering of acyclic digraph in $O(n\cdot Z\cdot L)$ time; and
3) the optimal out-star clustering of general digraph in $O(n\cdot Z\cdot \bar{K}\cdot L)$ time.

Proof: Similar to the proof given for Proposition 3.7. ***Q.E.D.***

Example: Applying $ODC_{s\circ g}$ to Figure 3-2a, we obtained the

optimal solution for Figure 3-2b, which has total value 14 and contains 2 out-stars:

Star 1 contains vertices 1, 3 and 4.

Star 2 contains vertices 2, 5 and 6.

3.4 Optimal Out-Necklace Clustering of Out-Necklace and General Digraph Problems ($ODC_{n°n°}$, $ODC_{n°g}$)

The $ODC_{n°n°}$ and $ODC_{n°g}$ problems are to find a set of bounded out-necklace clusters, or its subclass clusters, from a given out-necklace and general digraph respectively, such that the total value of this set is optimized. The $ODC_{n°n°}$ problem is first solved in section 3.4.1 and the solution is then used to solve $ODC_{n°g}$ in section 3.4.2.

3.4.1 $ODC_{n°n°}$ Problem

An unique property of an out-necklace is that it has only one cycle in it. To find a set of out-necklace clusters from it, observe that if the weight of the cycle contained in a given out-necklace is less than B, then the optimal out-necklace clustering will always contain the cycle in one of the induced clusters. If the cycle is overweight, then the optimal out-necklace will never contain the cycle in one of the clusters. Thus the $ODC_{n°n°}$ problem can be divided into two subproblems where each deals with one of these two possibilities.

The $ODC_{t°t°}$-1 algorithm can be used to solve the first subproblem if we observe that by aggregating all vertices in the cycle into one vertex and replacing all the arcs going

out of the vertices in this cycle by arcs going out of this newly formed vertex, the out-necklace becomes an out-tree with the newly formed vertex as the root. This newly formed root can then be assigned a weight equal to the total weight of all vertices in the cycle and the total value of the cycle can be added on later to the cluster induced by the optimal clustering that contains the root.

The second subproblem can be easily solved because to find an optimal out-necklace clustering which contains no cycle in any of the induced clusters is to find an optimal out-tree clustering of the out-necklace. Thus, the $ODC_{t\circ n\circ}$-1 algorithm can be used to solve this problem.

An algorithm for $ODC_{n\circ n\circ}$ can be formulated as follows:

ALGORITHM: $ODC_{n\circ n\circ}$-1

INPUT: an out-necklace;

OUTPUT: an optimal out-necklace clustering and its value;

PROCESS:

1) IF the total weight of the K vertices on the cycle is less than B, THEN
   1.1) transform the out-necklace into an out tree as described above;
   1.2) apply the $ODC_{t\circ t\circ}$-1 algorithm to find an optimal solution;
   1.3) replace the root in the configuration of the optimal solution obtained in 1.2) by all K vertices in the cycle;
   1.4) add to the value of the optimal solution obtained

in 1.2) the total value of the cycle;

1.5) RETURN the configuration and its value obtained in 1.3) and 1.4);

2) ELSE

2.1) apply the $ODC_{t \circ n \circ}$-1 algorithm to find an optimal solution;

2.2) RETURN the configuration and its value;

END.

Proposition 3.9: The $ODC_{n \circ n \circ}$-1 algorithm finds the optimal solution in $O(n \cdot Z^2 \cdot K)$ time.

Proof: Step 1) takes $O(n \cdot Z^2)$ time from Proposition 2.2. Step 2) takes $O(n \cdot Z^2 \cdot K)$ time from Proposition 3.7. Thus, the time complexity is $O(n \cdot Z^2 \cdot K)$. ***Q.E.D.***

Example: The out-necklace shown in Figure 3-3a contains a cycle with weight $3 \leq B=4$. It is transformed into an out-tree as shown in Figure 3-3b. Applying the $ODC_{t \circ t \circ}$-1 algorithm obtains an optimal solution which has value 14 and contains 2 clusters:

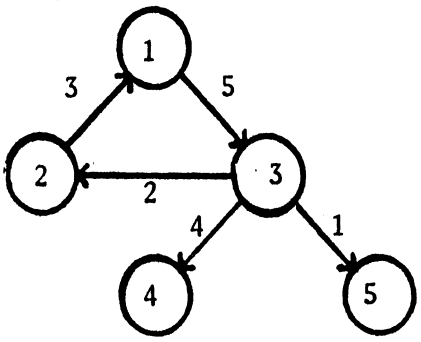
Cluster 1 is an out-necklace with vertices 1, 2, 3 and 4.

Cluster 2 is an out-necklace with vertex 5.

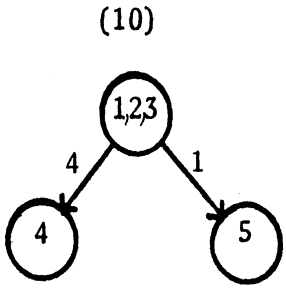
3.4.2 $ODC_{n \circ g}$ Problem

We tackle this problem by using the same idea used in solving $ODC_{t \circ g}$. Since the optimal out-necklace clustering of a general digraph $\leq$ one of the maximal out-necklace clusterings, the optimal solution can be found among this maximal set. The approach is to use $ODC_{n \circ n \circ}$ to obtain an

a)



b)



B = 4

Vv_i (w_i=1)

Figure 3-3 An example out-necklace and its optimal clustering result.

optimal out-necklace clustering for each maximal out-necklace clustering. From this set of optimal out-necklace clusterings, the one clustering with the maximum value is then the optimal out-necklace clustering for the given digraph.

An algorithm for $ODC_{n\circ g}$ can be formulated as follows:

ALGORITHM: $ODC_{n\circ g}$***-1***

INPUT: a general digraph G;

OUTPUT: an optimal out-necklace clustering and its value;

PROCESS:

1) FOR each maximal out-necklace clustering of G which induces a set of out-necklace clusters DO:
   1.1) apply algorithm $ODC_{n\circ n\circ}$-1 to obtain an optimal solution for each out-necklace cluster;
   1.2) IF the sum of the optimal out-necklace clustering values of the set of out-necklace clusters is the largest value so far, THEN keep the solution;
2) RETURN the last solution kept;

END.

Proposition 3.10: The $ODC_{n\circ g}$-1 algorithm finds the optimal solution in $O(n\cdot Z^2\cdot \bar{K}\cdot L)$ time.

Proof: Similar to the proof given for 3) of Proposition 3.7. ***Q.E.D.***

Example: By applying the $ODC_{n\circ g}$-1 algorithm to the general digraph shown in Figure 3-2a, we obtain the optimal solution in Figure 3-2c. This optimal solution has value 20 and

contains 2 clusters:

Cluster 1 is an out-necklace with vertices 1 and 3.

Cluster 2 is an out-necklace with vertices 2, 4, 5 and 6.

CHAPTER 4

OPTIMAL ACYCLIC DIGRAPH/GENERAL DIGRAPH CLUSTERING OF ACYCLIC DIGRAPH/GENERAL DIGRAPH PROBLEMS

This chapter generalizes the sequential clustering approach presented in section 2.1 to analyze and solve ODC_{aa}, ODC_{gg} and ODC_{ag} (cluster 7 in Appendix A). ODC_{aa} is solved in section 4.1, ODC_{gg} in section 4.2 and ODC_{ag} in section 4.3.

4.1 Optimal Acyclic Digraph Clustering of an Acyclic Digraph Problem (ODC_{aa})

The ODC_{aa} problem is to find a set of bounded acyclic digraph clusters, or its subclass clusters, from a given acyclic digraph such that the total value of the set is optimized. A formal definition for the problem can be obtained by replacing all occurrences of an out-tree by an acyclic digraph in the formal definition of $ODC_{t^0t^0}$ that is given in section 2.1. The solution approach will first be presented in section 4.1.1 and then an algorithm to solve the problem in section 4.1.2.

4.1.1 A Sequential Clustering Approach to ODC_{aa}

Section 2.1 in Chapter 2 has presented a sequential clustering approach which obtains the optimal solution of an

out-tree through a sequence of clustering increasingly larger set of subgraphs such that all feasible clusterings of each set can be constructed from those of the immediate previous set using simple operations. This approach can be applied to ODC_{aa} as well except that the given acyclic digraph must be restructured into a tree-like structure so that the same approach can be successfully applied. This tree-like structure, $T=(V,E^0 \cup E^1)$, has the same vertex set V as the original digraph G=(V,E) but divides the arc set E into a tree arc set E^0 (e.g., solid lines in the second figure in Figure 4-1) and a co-tree arc set E^1 (e.g., dash lines in the same figure).

An algorithm for constructing such a tree-like structure from an acyclic digraph with a designated root v can be devised by modifying the depth-first search algorithm (Aho et al '74) for undirected graphs. The only modification is to ignore the arc direction of the acyclic digraph in the restructuring process. Thus, instead of parent child relation, an undirectional relation is-related-to is defined on the vertex set V of G such that a vertex v_j is-related-to v_r if either $(v_r,v_j) \in E$ or $(v_j,v_r) \in E$. The restructuring algorithm named RESTRUCTURE-1 is as follows:

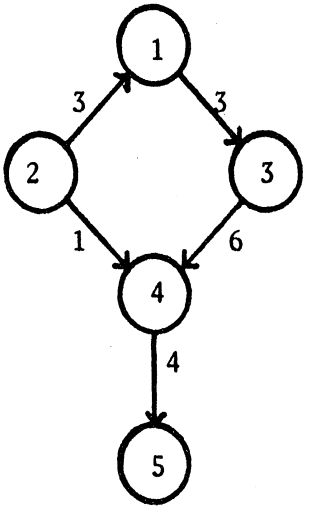
ALGORITHM: RESTRUCTURE-1(G,v)

INPUT: an acyclic digraph G=(V,E) with a designated root v;

OUTPUT: $T=(V,E^0 \cup E^1)$;

PROCESS:

ALGORITHM: Search-1(v_r)



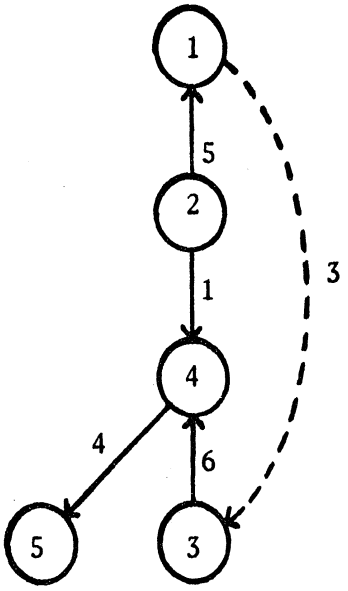


Figure 4-1 An acyclic digraph and its tree-like structure.

PROCESS:

1) mark v_r as "visited";

2) FOR each vertex v_j which is-related-to v_r DO:

2.1) IF v_j is not visited THEN add the arc (i.e., (v_r,v_j) or $(v_j,v_r) \in E$) to E^0;
CALL Search-1(v_j);

2.2) ELSE add the arc (i.e., (v_r,v_j) or $(v_j,v_r) \in E$) to E^1;

END;

I) CALL Search-1(v);

II) RETURN T;

END.

Proposition 4.1: The run time of algorithm RESTRUCTURE-1 is O(Max(m,n)) where m is the number of arcs and n is the number of vertices in G.

Proof: Immediate from the SEARCH algorithm of Aho et al ('74). ***Q.E.D.***

An important feature of $T = (V, E^0 \cup E^1)$ is that its vertex set V together with its tree arc set E^0 form a tree (if we ignore the direction of each tree arc). The m+n stage sequential clustering process can thus be based on this tree just like the one for $ODC_{t^0t^0}$-1 (see Figure 4-2). But a complication in the clustering process is that all those co-tree arcs should also be considered. This creates a need to consider forming a set of clusterings with the "bounded nonchild descendant" in addition to the "child" vertices at each stage because of the additional

possibilities of cluster formation induced by those co-tree arcs. For example, vertices 1,2 and 3 in Figure 4-1 could form a cluster. At the stages when vertex 2 is considered (e.g., stage 7 in Figure 4-2), a set of clusterings should be formed not only with vertex 4 but also with vertex 3. To formalize this idea, some new terms and additional processes are needed.

If there is a path (ignore the arc direction) from the root of T to v_r and a path from v_r to v_j in the tree formed by V and E^o, then v_r is the ancestor of v_j and v_j is the descendant of v_r. In the case where v_r is-related-to v_j in the tree by a tree arc, then v_r is also called the parent of v_j and v_j the child of v_r.

A vertex v_j is called a neighbor of a vertex v_r if v_j satisfies one of the following two rules:

1) v_j is a child of v_r.
2) v_j is a nonchild descendent of v_r and v_j lies on a path $v_r, x_1, x_2, \ldots, x_p, v_j$ $(p \geq 0)$ such that if p=0, then v_r is-related-to v_j by a co-tree arc. If p>0, then all x's are ancestors of v_r and x_p is-related-to v_j by a co-tree arc (e.g., in Figure 4-1, vertex 3 is a nonchild descendant of 2 and lies on the path 2,1,3.).

Also, v_j is a bounded nonchild neighbor of v_r if both of the following two rules are true:

1) v_j is a neighbor of v_r.
2) v_j is a nonchild descendant and the path

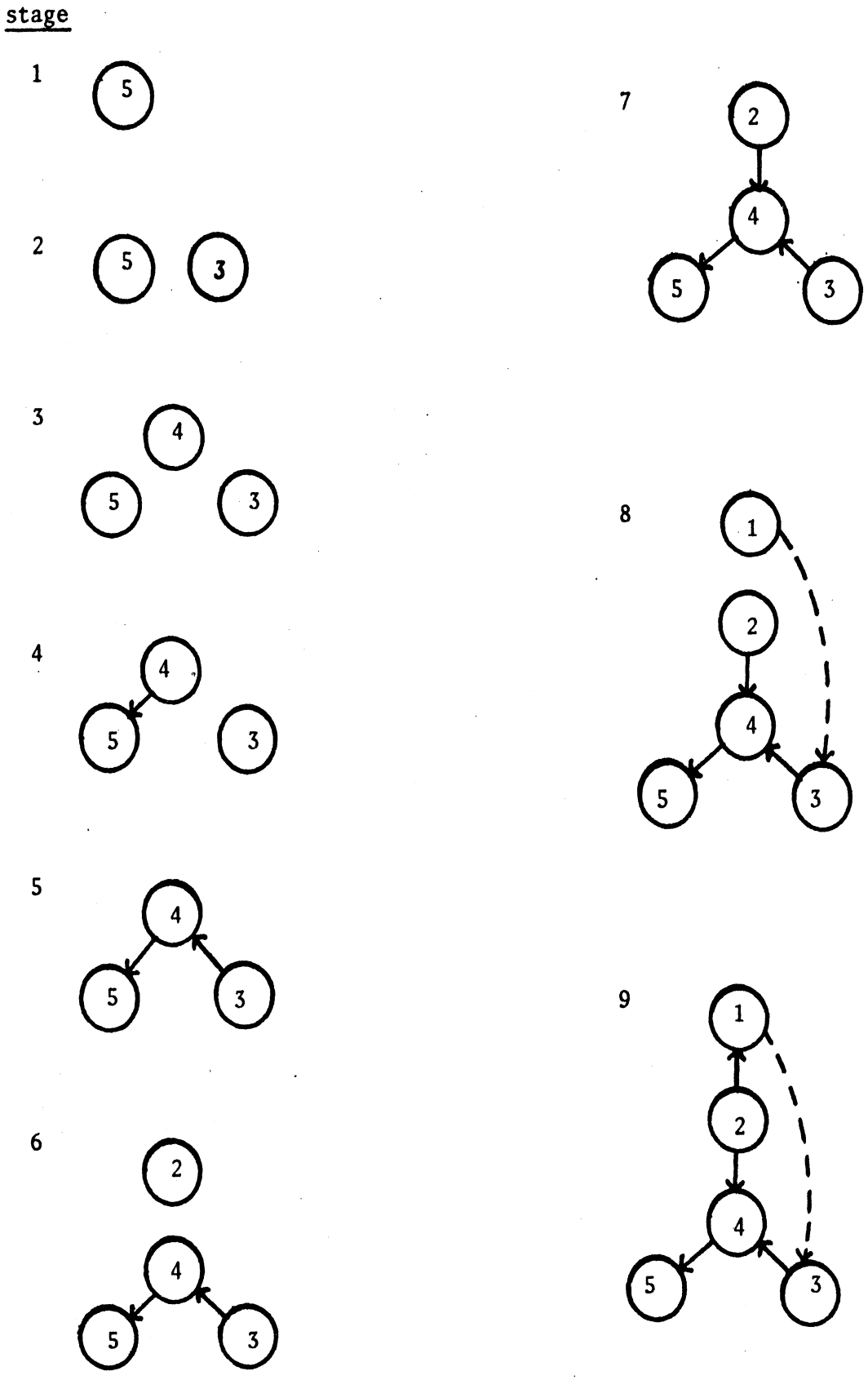


Figure 4-2 The eleven stages of the clustering process.

$v_r, x_1, x_2, \ldots, x_p, v_j$ is bounded, i.e., $W(v_r, x_1, x_2, \ldots, x_p, v_j) \leq B$ (e.g., in Figure 4-1, vertex 3 is a bounded nonchild neighbor of vertex 2 if B=3 and $\forall v_q \in V(w_q=1)$).

To find the set of neighbors and bounded nonchild neighbors for each vertex, two functions-- DF-NO and LOW -- are needed. DF-NO assigns depth-first search numbers to vertices in the order they are visited in the construction of $T=(V,E^0 \cup E^1)$ from a given digraph $G=(V,E)$. For each vertex v_r in V, LOW: V-->Z^+ can be defined as:

$LOW(v_r)$= MIN({DF-NO(v_r)} ∪
{DF-NO(v_k)|there exists a co-tree arc (i.e., (v_k, v_j) or (v_j, v_k)) in E^1 such that v_j is a descendant of v_r, and v_k is an ancestor of v_r in the tree formed by V and E^0}).

The DF-NO for each vertex in V can be easily obtained during the construction of T for the given digraph. The LOW value can also be easily obtained because the definition of LOW can be reformulated recursively as:

$LOW(v_r)$=MIN({DF-NO(v_r)}
∪ {LOW(v_j)|v_j is a child of v_r}
∪ {DF-NO(v_k)|v_r is-related-to v_k by a co-tree arc and v_k is an ancestor of v_r}).

An algorithm called RESTRUCTURE-2, which is modified from Aho et al's ('74) SEARCHB for finding T, LOW and DF-NO values for each vertex, is as follows:

ALGORITHM: RESTRUCTURE-2(G,v)

INPUT: an acyclic digraph G=(V,E) with a designated root v;

OUTPUT: a $T=(V,E^{0} \cup E^{1})$ with LOW and DF-NO values for each vertex;

PROCESS:

ALGORITHM: Search-2(v_r)

PROCESS:

1) mark v_r as "visited";

DF-NO(v_r) = df-count;

LOW(v_r) = DF-NO(v_r);

df-count = df-count + 1;

2) FOR each vertex v_j that is-related-to v_r DO:

2.1) IF v_j is not visited DO:

2.1.1) parent(v_j) = v_r;

2.1.2) CALL Search-2(v_j);

2.1.3) LOW(v_r) = MIN(LOW(v_r),LOW(v_j));

2.2) ELSE IF v_j is not parent(v_r) THEN

LOW(v_r) = MIN(LOW(v_r), DF-NO(v_j));

END;

I) df-count = 1;

II) CALL Search-2(v);

III) RETURN T with LOW and DF-NO values for each vertex;

END.

Proposition 4.2: The run time of algorithm RESTRUCTURE-2 is the same as that for RESTRUCTURE-1.

Proof: Immediate from algorithm SEARCHB from Aho et al ('74). ***Q.E.D.***

Using the DF-NO and LOW values of each vertex v_r, a

neighbor list $L_{nb}[v_r]$ and a bounded nonchild neighbor list $L_{bn}[v_r]$ of the vertex can be defined for each subgraph rooted at v_r:

$$L_{nb}[v_r] = \{v_j | v_j \in ch(v_r)\} \cup \{v_k | v_k \in L_{nb}[v_j] \text{ s.t. } LOW(v_k) \leq DF\text{-}NO(v_r)\};$$

$$L_{bn}[v_r] = \{v_m | v_m \notin ch[v_r] \text{ and } v_m \in L_{nb}[v_r] \text{ and } LOW(v_m) \leq DF\text{-}NO(v_r) \text{ and } W(v_r, x_1, x_2, \ldots, x_p, v_m) \leq B\}.$$

An algorithm for finding $T=(V, E^0 \cup E^1)$ with a neighbor list $L_{nb}[v_r]$ and a bounded nonchild neighbor list $L_{bn}[v_r]$ for each vertex v_r can thus be formulated as follows:

ALGORITHM: RESTRUCTURE-3(G,v)

INPUT: an acyclic digraph G with a designated root v;

OUTPUT: a $T=(V, E^0 \cup E^1)$ with a neighbor list $L_{nb}[v_r]$ and a bounded nonchild neighbor list $L_{bn}[v_r]$ for each vertex v_r;

PROCESS:

ALGORITHM: Search-3(v_r)

PROCESS:

1) IF v_r has no children THEN RETURN $L_{nb}[v_r]=L_{bn}[v_r]=\emptyset$;
   ELSE
2) FOR each vertex $v_j \in ch(v_r)$ DO:
   2.1) CALL Search-3(v_j) to obtain $L_{nb}[v_j]$ and $L_{bn}[v_j]$;
   2.2) $L_{nb}[v_r] = L_{nb}[v_r] \cup \{v_j\} \cup \{v_k | v_k \in L_{nb}[v_j] \text{ s.t. } LOW(v_k) \leq DF\text{-}NO(v_r)\}$;
   2.3) FOR each $v_m \in L_{nb}[v_j]$ DO:
      2.2.1) IF $w_m + w_r \leq B$ and $LOW(v_m) \leq DF\text{-}NO(v_r)$

THEN

$v_s = v_r$;

PATH-WEIGHT = $w_m + w_r$;

WHILE there is no co-tree arc between

v_s and v_m DO:

v_s = parent(v_s);

PATH-WEIGHT = PATH-WEIGHT + w_s;

END;

2.3.2) IF PATH-WEIGHT ≤ B THEN

$L_{bn}[v_r] = L_{bn}[v_r] \cup \{v_m\}$;

END;

END;

I) CALL RESTRUCTURE-2(G,v);

II) CALL Search-3(v);

III) RETURN T with the two lists, $L_{nb}[v_r]$ and $L_{bn}[v_r]$, that are associated with each vertex v_r;

END.

Proposition 4.3: The time complexity of RESTRUCTURE-3 is $O(n^2 \cdot d(T))$ where d(T) is the maximum depth of the constructed T.

Proof: Step I) takes O(max(m,n)) time from Proposition 4.2; II) takes $O(n^2 \cdot d(T))$ time; III) takes O(n) time. ***Q.E.D.***

Given T, $L_{nb}[v_r]$ and $L_{bn}[v_r]$, the clustering process can be sequenced in the m+n stages just like $ODC_{t_0t_0}$-1 (see Figure 4-2). The INSERT, PUT, and conditional CONNECT operations are defined exactly the same way as that defined in section 2.1.1. But at each new arc stage, in addition to

the PUT and conditional CONNECT operations, a new conditional COMBINE operation forms a set of feasible clusterings (denoted by $PK^2[v_r,i+1]$) for each pair of feasible clusterings $K[v_r,i]$ in $PK^f[v_r,i]$ and $K[v_k]$ in $PK^f[v_j,od(v_j)]$, for each v_k which is a bounded nonchild neighbor of v_r. Thus, the set of feasible clusterings formed at each new arc stage now is:

$$PK^f_{k+1} = PK^f_k - PK^f[v_r,i] - PK^f[v_j,od(v_j)] \cup PK^f[v_r,i+1]$$

where

$PK^f[v_r,i+1] = PK^0[v_r,i+1] \cup PK^1[v_r,i+1] \cup PK^2[v_r,i+1]$.

$PK^0[v_r,i+1]$ and $PK^1[v_r,i+1]$ are the same as those defined in section 2.1.1.

$PK^2[v_r,i+1] = \{K^2[v_r,i+1] | K^2[v_r,i+1]$ is formed using COMBINE operation for each pair of $K[v_r,i]$ and $K[v_k,od(v_k)]$ and for each $v_k \in L_{bn}[v_r]\}$.

For each bounded nonchild neighbor v_k of v_r, i.e., $v_k \in L_{nb}[v_r]$, the COMBINE operation forms a set of feasible clusterings for each pair of feasible clusterings: $K[v_r,i]$ in $PK^f[v_r,i]$ and $K[v_k]$ in $PK^f[v_j,od(v_j)]$. However, there is a consideration which depends on whether there is an arc between v_r and v_k or not:

Case 1:

If there is a co-tree arc between v_r and v_k, then the new clustering is formed in exactly the same way as that formed by the conditional CONNECT operation defined above using $K[v_r,i]$ and $K[v_k]$.

Case 2:

If there is no co-tree arc between v_r and v_k, then the new clustering is formed by merging the cluster C_l with the cluster C_l' into one new cluster C_l'' with no arc connecting them. Include this new cluster with all the other clusters in both clusterings to form a new clustering. This new clustering contains:

1) all clusters except C_l in $K[v_r,i]$;

2) all clusters except C_l' in $K[v_k]$; and

3) C_l''.

Note that C_l'' contains two disconnected subgraphs of G. (This is different from the cluster formed by PUT and CONNECT which contains one connected subgraph.) The reason C_l'' is formed in this way is that in a future stage, these two disconnected subgraphs can be connected together through a co-tree arc which relates an ancestor v_m of v_r with v_k.

4.1.2 ODC_{aa}-1 Algorithm

Let a feasible clustering be represented by a triplet with three components. The value component of a triplet is defined in the same way as that defined in section 2.1.2. The weight component now is a list of s entries enclosed in ( ) if there are s induced clusters. Each entry stores the cluster weight of the corresponding cluster. This is necessary because at each stage the triplets are formed not only from those of the child but also bounded nonchild

neighbors. Since the latter can be in any cluster induced by $K[v_j,od(v_j)]$, it creates a need to keep track of all cluster weights. We will use the notation $W(C[v_r,i])$ and $W(C[v_r,i+1])$ to denote the cluster weight of the induced cluster containing v_r in $K[v_r,i]$ and $K[v_r,i+1]$ respectively. $W(C[v_k])$ will denote the cluster weight of the induced cluster containing nonneighbor v_k in $K[v_j,od(v_j)]$. The configuration component is also modified to indicate not only vertices in each cluster but also the arcs in the cluster. This is necessary because the cluster is no longer an out-tree and therefore a set of vertices cannot uniquely represent the cluster. For example, in Figure 4-1, suppose B=4 and vertices 1,2,3, and 4 form a cluster. The notation <1,2,3,4> does not tell which of the following 4 acyclic digraphs the cluster represents:

V = {1,2,3,4} E = {(2,1)(2,4)(3,4)(1,3)}

V = {1,2,3,4} E = {(2,1)(2,4)(3,4)}

V = {1,2,3,4} E = {(2,4)(3,4)(1,3)}

V = {1,2,3,4} E = {(2,1)(3,4)(1,3)}

Let the configuration component CF(K) that is used in Chapter 2 now be $CF^0(K)$ and the newly added representation of the arcs be $CF^1(K)$. Thus, $CF(K) = CF^0(K) \cup CF^1(K)$.

In terms of triplet formation, the four operations can now be defined as:

INSERT: same as that defined in section 2.1.2.

PUT: same as that defined in section 2.1.2.

conditional CONNECT:

IF $W(C[v_r,i]) + W(C[v_j,od(v_j)]) \leq B$ THEN

form $K^2[v_r,i+1]$

with

$F(K^2[v_r,i+1]) = F(K[v_r,i]) + F(K[v_j,od(v_j)]) + f_{rj}$,

$W(C[v_r,i+1]) = W(C[v_r,i]) + W(C[v_j,od(v_j)])$,

$CF(K^2[v_r,i+1]) = CF^0(K^2[v_r,i+1]) \cup CF^1(K^2[v_r,i+1])$

where

$CF^0(K^2[v_r,i+1]) = \{C_2,C_3,\ldots,C_p,C_2',C_3',\ldots,C_p'\}$
$\cup \{C_1 \cup C_1'\}$,

$CF^1(K^2[v_r,i+1]) = CF^1(K[v_r,i]) \cup CF^1(K[v_j,od(v_j)])$
$\cup \{(v_r,v_j) \text{ or } (v_j,v_r)\}$.

conditional COMBINE:

IF there exists a co-tree arc between v_r and v_k THEN:

form a $K^2[v_r,i+1]$ using conditional CONNECT;

ELSE

form a $K^2[v_r,i+1]$ with

$F(K[v_r,i+1)]) = F(K[v_r,i]) + F(K[v_k])$,

$W(C[v_r,i+1]) = W(C[v_r,i]) + W(C[v_k])$, and

$CF(K[v_r,i+1]) = CF^0(K[v_r,i]) \cup CF^1(K[v_k])$ where

$CF^0(K[v_r,i+1]) = \{C_2,C_3,\ldots,C_p,C_2',C_3',\ldots,C_q'\}$
$\cup \{C_1 \cup C_1'\}$,

$CF^1(K[v_r,i+1]) = CF^1(K[v_r,i]) \cup CF^1(K[v_k])$.

END.

For example, in Figure 4-2, let {{(0,(1,1,1),<3><4><5>) (6,(2,1),<3,4><5>(3,4)) (4,(2,1),<4,5><3>(4,5)) (10,(3),<3,4,5>(3,4)(4,5)} {<2>}} be the set of feasible triplets formed at stage 6. At stage 7, the set of feasible

triplets formed using the respective operations are:

PUT: (0,(1,1,1,1),<2><3><4><5>)

(6,(1,2,1),<2><3,4><5>(3,4))

(4,(1,2,1),<2><4,5><3>(4,5))

(0,(1,3),<2><3,4,5>(3,4)(4,5))

CONNECT: (1,(2,1,1),<2,4><3><5>(2,4))

(7,(3,1),<2,3,4><5>(2,3)(3,4))

(5,(3,1),<2,4,5><3>(2,4)(4,5))

COMBINE (vertex 3 is a bounded nonchild neighbor of 2):

(0,(2,1,1),<2,3><4><5>)

(4,(2,2),<4,5><2,3>(4,5))

A dominance relation can still be defined over the set of triplets formed at each new arc stage even though it is now more restricted (or not as powerful as the one defined in Chapter 2).

Let v_r be the parent of vertex v_j in the tree induced by V and E°. Suppose $K_1[v_j]$ and $K_2[v_j]$ are any two feasible triplets formed in a new arc stage. A cluster C_1 induced by $K_1[v_j]$ is cluster-similar to a cluster C_2 induced by $K_2[v_j]$ if both clusters contain the same set of neigbhors in $L_{nb}[v_r]$. $K_2[v_j]$ is triplet-similar to $K_1[v_j]$ if for every cluster C_2 induced by $K_2[v_j]$, which contains one or more neighbors in $L_{nb}[v_r]$, there is a C_1 induced by $K_1[v_j]$ which is cluster-similar to C_2 and vice versa.

Define $K_1[v_j]$ dominates $K_2[v_j]$ if

1) $K_2[v_j]$ is triplet-similar to $K_1[v_j]$;

2) for every pair of C_1 and C_2 induced by $K_1[v_j]$ and

$K_2[v_j]$ respectively, if C_1 is cluster-similar to C_2, then $W(C_2) \geq W(C_1)$;

3) $F(K_1[v_j]) \geq F(K_2[v_j])$.

Proposition 4.4: The dominated triplet $K_2[v_j]$ defined above cannot be extended to the optimal clustering of the acyclic digraph.

Proof: Let the n vertices visited in postorder sequence be denoted by $v_1, v_2, \ldots, v_n$. A triplet formed in visiting v_j, denoted by $K[v_j]$, can be represented by a sequence of pairs (Lukes '75):

$$K[v_j] = [v_1, < >], [v_2, C_2], \ldots, [v_j, C_j]$$

where the first entry of each pair represents the vertex visited and the second entry represents the cluster to which the vertex is added on.

Let $K[v_r]$ and $K[v_n]$ be two feasible triplets formed in the ith and nth stages respectively. We define a derivation of a feasible triplet $K[v_n]$ from $K[v_r]$ (v_n is the root of the tree induced by V and E^o) as the sequence:

$$[v_{r+1}, C_{r+1}], [v_{r+2}, C_{r+2}], \ldots, [v_n, C_n].$$

Let $K[v_{k-1}]'$ be triplet-similar to $K[v_{k-1}]"$ and both are formed in the same new arc stage. Let $K[v_{k-1}]'$ have equal to or higher value than $K[v_{k-1}]"$. Assume that there exists a triplet $K[v_n]"$ derived from $K[v_{k-1}]"$ that has a higher value than any feasible triplet derived from $K[v_{k-1}]'$ for root n. We now show this is impossible.

Let a derivation of $K[v_n]"$ from $K[v_{k-1}]"$ be $[v_k, C_k"], [v_{k+1}, C_{k+1}"], \ldots, [v_n, C_n"]$. Since $K[v_{k-1}]'$ is

triplet-similar to $K[v_{k-1}]$", a triplet $K[v_n]'$ can be formed from $K[v_{k-1}]'$ with the derivation $[v_k,C_k'],[v_{k+1},C_{k+1}'],\ldots,[v_n,C_n']$ such that for all $m=k,k+1,\ldots,n$, C_m' is cluster-similar to C_m". This triplet $t[v_n]'$ is a feasible triplet because every cluster C_m' in it contains the same neighbors as C_m" and has equal to or lower cluster weight than C_m". Since there are no co-tree arcs between the set of nonneighbors and the set of vertices that are ancestors of v_{k-1}, the values of triplets formed at $v_k,v_{k+1},\ldots,v_n$ are independent of all the nonneighbors that appear in a cluster together with neighbors. Thus, the difference in cluster values between each pair of C_m' and C_m" is the sum of the values of those (tree or co-tree) arcs that exist among the nonneighbors in C_m' and C_m". Since $K[v_{k-1}]'$ has equal to or higher value then $K[v_{k-1}]$", for $m = k,k+1,\ldots,n$, the value of C_m' is equal to or higher than C_m". Thus, $K[v_n]'$ has equal to or higher value than $K[v_n]$". This is a contradiction to the assumption made above. ***Q.E.D.***

Given the definition of the four operations and the dominace relation, an algorithm for ODC_{aa} can be outlined as follows:

ALGORITHM: ODC_{aa}-1

INPUT: an acyclic digraph G;

OUTPUT: an optimal acyclic digraph clustering of G and its value;

PROCESS:

I) choose an arbitrary vertex v as root and CALL RESTRUCTURE-3(G,v) to build a T of G with lists $L_{nb}[v_r]$ and $L_{bn}[v_r]$ for each vertex v_r;

II) FOR each v_r visited in postorder traversal sequence DO:

1) FOR i=0 to $od(v_r)$ DO:

1.1) IF i=0 THEN form a new PK^f_{k+1} from PK^f_k using INSERT operation; ELSE DO:

1.2) form a new PK^f_{k+1} from PK^f_k using PUT, conditional CONNECT, and conditional COMBINE; eliminate all dominated triplets;

END;

III) find the triplet in PK^f_{m+n} that has the maximum value and RETURN the solution;

END.

Proposition 4.5: Let v_p be the parent of v_r. Each new arc stage forms no more than $B^{n[p,r]} \cdot n[p,r]! \cdot B^{n[r,j]} \cdot n[r,j]! \cdot (n[r,j] + 1)$ new feasible triplets where

$n[p,r] = |L_{nb}[v_p] \cap V[v_r,i]|$ and

$n[r,j] = |L_{nb}[v_r] \cap V[v_j,od(v_j)]|$.

Proof: See Appendix B. ***Q.E.D.***

Note: when the given digraph is an out-tree, n[p,r]=1 and n[r,j]=1 in the ODC_{aa}-1. Each new arc stage forms no more than $2 \cdot B^2$ new feasible triplets and have only B triplets remaining after elimination.

Example: Applying the ODC_{aa} algorithm to the acyclic digraph shown in Figure 4-1 will yield the following result:

triplet formed	triplet formed by I/CO/P/COM. (stage#)	triplet elim. by CO/P/COM. (stage#)	triplet elim. by domi. (stage#)
(0,(1),<5>)	1	4	
(0,(1),<3>)	2	5	
(0,(1),<4>)	3	4	
(0,(1,1),<4><5>)	4	5	
(4,(2),<4,5>(4,5))	4	5	
(0,(1,1,1),<3><4><5>)	5	7	
(6,(2,1),<3,4><5>(3,4))	5	7	
(4,(2,1),<4,5><3>(4,5))	5	7	
(10,(3),<3,4,5>(3,4)(4,5))	5	7	
(0,(1),<2>)	6	7	
(0,(1,1,1,1),<2><3><4><5>)	7	9	7
(1,(2,1,1),<2,4><3><5>(2,4))	7	9	7
(0,(2,1,1),<2,3><4><5>)	7	9	7
(6,(1,2,1),<2><3,4><5>(3,4))	7	9	
(7,(3,1),<2,3,4><5>(2,4)(3,4))	7	9	
(4,(1,2,1),<2><4,5><3>(4,5))	7	9	
(5,(3,1),<2,4,5><3>(2,4)(4,5))	7	9	
(4,(2,2),<4,5><2,3>(4,5))	7	9	
(10,(1,3),<2><3,4,5>(3,4)(4,5))	7	9	
(0,(1),<1>)	8	9	
(6,(1,1,2,1),<1><2><3,4><5>(3,4))	9		9
(11,(2,2,1),<1,2><3,4><5>(2,1)(3,4))	9		9
(9,(3,1,1),<1,3,4><2><5>(1,3)(3,4))	9		9
(7,(1,3,1),<1><2,3,4><5>(2,4)(3,4))	9		9
(4,(1,1,2,1),<1><2><4,5><3>(4,5))	9		9
(9,(2,2,1),<1,2><4,5><3>(2,1)(4,5))	9		9
(7,(1,2,2),<2><4,5><1,3>(1,3)(4,5))	9		9
(5,(1,3,1),<1><2,4,5><3>(2,4)(4,5))	9		9
(8,(2,3),<1,3><2,4,5>(1,3)(2,4)(4,5))	9		9
(4,(1,2,2),<1><4,5><2,3>(4,5))	9		9
(12,(3,2),<1,2,3><4,5>(2,1)(1,3)(4,5))	9		9
(10,(1,1,3),<1><2><3,4,5>(3,4)(4,5))	9		
(15,(2,3),<1,2><3,4,5>(2,1)(3,4)(4,5))	9		

The optimal clustering solution is (15,(2,3),<1,2>,<3,4,5>(2,1)(3,4)(4,5)).

4.2 Optimal General Digraph Clustering of a General Digraph Problem (ODC_{gg})

The ODC_{gg} problem is to find a set of bounded general

digraph clusters, or its subclass clusters, from a general digraph such that the total value of this set is optimized. ODC_{aa}-1 can be used to solve ODC_{gg} with one modification. The need for the modification comes from the observation that if two vertices v_r and v_j form a cycle (i.e., both (v_r,v_j) and (v_j,v_r) are in E), then in T, not only is there a tree arc (e.g., (v_r,v_j)) between v_r and v_j, but there is also a co-tree arc (e.g., (v_j,v_r)) between them. In ODC_{aa}-1, this co-tree arc does not exist and has not been taken into consideration in forming a set of feasible triplets for each subtree root v_r. To include this situation, there are <u>four</u> <u>cases</u> to consider based on the assignments to these two arcs:

1) X_{rj}=0 and X_{jr}=0. This has been handled by the PUT operation in step 1.2 of ODC_{aa}-1.1.
2) X_{rj}=1 and X_{jr}=0. This has been handled by the CONNECT operation in step 1.2 of ODC_{aa}-1.1.
3) X_{rj}=0 and X_{jr}=1. This needs to be considered.
4) X_{rj}=1 and X_{jr}=1. This needs to be considered.

Since the three triplets formed for cases 2, 3 and 4 will always have the same weight and the same set of neighbors, using Proposition 4.4, the two triplets formed from cases 2 and 3 will always be dominated triplets because the one formed for case 4 will have the highest value. Thus, only cases 1 and 4 need to be taken into consideration. The modification that needs to be done so that ODC_{aa}-1 can be applied to ODC_{gg} is to modify the value

and configuration calculation of the CONNECT operation in step 1.2 of ODC_{aa}-1 to:

$$F(K[v_r,i+1])=F(K[v_r,i])+F(K[v_j,od(v_j)])+f_{rj}+f_{jr},$$

$$CF^1(K[v_r,i+1]) = CF^1(K[v_r,i]) \cup CF^1(K[v_j,od(v_j)] \cup \{(r,j)\} \cup \{(j,r)\}.$$

4.3 Optimal Acyclic Digraph Clustering of a General Digraph Problem (ODC_{ag})

The ODC_{ag} problem is to find a set of bounded acyclic digraph clusters, or its subclass clusters, from a given general digraph such that the total value of the set is optimized. To solve this problem, an algorithm ODC_{ag}-1 can be devised that is similar to the ODC_{aa}-1 algorithm except for two modifications.

The first modification is to consider the possibility of two vertices v_r and v_j forming a cycle. Unlike ODC_{gg} where only cases 1 and 4 are taken into consideration, cases 1, 2 and 3 are needed here. Case 4 is never allowed because it introduces cycles into the resulting cluster. Thus, one more conditional CONNECT operation must be added to step 1.2 of ODC_{aa}-1 so that cases 2 and 3 are both covered.

The second modification is to add one more elimination rule before the dominance eliminations (step 1.2) of ODC_{aa}-1. The added rule is that for any cluster in a newly formed triplet, if the cluster is not acyclic then eliminate the triplet. The algorithm for testing whether a cluster (digraph) is acyclic is a simple procedure (Aho et al '83). It is obvious that the optimal solution will then contain

only acyclic digraph clusters.

CHAPTER 5

EXTENSIONS

In the previous three chapters, the solutions to the fourteen ODC problems that are related to the classes of general digraph, acyclic digraph, out-necklace, out-tree and out-star have been presented. This chapter considers three extensions to those solutions. They are:

1) extensions of classes of graphs (directed or undirected);
2) monotonic objective function and additional constraints for all problems; and
3) divide-and-conquer technique for ODC_{aa}, ODC_{gg} and ODC_{ag}.

Each extension will be discussed in the following three sections in their respective order.

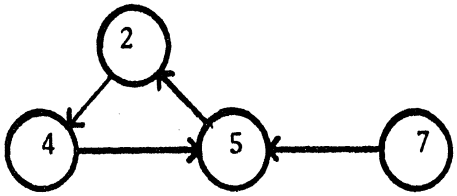
5.1 Extensions of Classes of Graphs

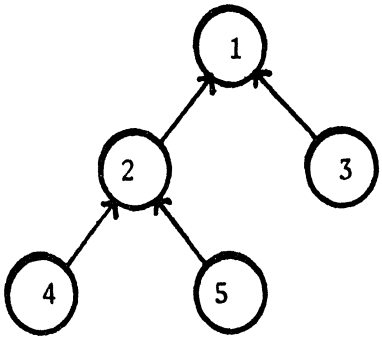
In some application areas, the converse digraphs of out-necklace, out-tree, and out-star, which are <u>in-necklace</u> $G_{n^{-1}}$, <u>in-tree</u> $G_{t^{-1}}$ and <u>in-star</u> $G_{s^{-1}}$ respectively (see Figure 5-1a), are more appropriate problem representations. Using the principle of directional duality (Harary '69), which states that "for each theorem about digraphs there is a

corresponding theorem obtained replacing every concept by its converse", the solution given in Chapters 2 and 3 can be modified to solve all ODC problems that are related to in-tree, in-star and in-necklace (i.e., change all "indegree" to "outdegree" and "m-indegree vertices" to "m-outdegree vertices"). The modifications are trivial and need not be discussed.

Undirected graphs, i.e., general undirected graph G_u, necklace G_n, tree G_t and star G_s (see Figure 5-1b), can also be solved in practically the same way. That is, the solutions given in Chapters 2,3 and 4 can be used to solve them with slight or no modification at all. Obviously, ODC_{aa}-1 can be used to solve OUC_{uu}. To solve OUC_{nn}, OUC_{tn}, OUC_{sn}, OUC_{tt}, OUC_{st} and OUC_{ss}, the direction can be imposed uniquely on the given necklace, tree and star. These problems can then be solved by using $ODC_{n\circ n\circ}$-1, $ODC_{t\circ n\circ}$-1, $ODC_{s\circ n\circ}$-1, $ODC_{t\circ t\circ}$-1, $ODC_{s\circ t\circ}$-1 and $ODC_{s\circ s\circ}$-1. The solutions to OUC_{nu}, OUC_{tu} and OUC_{su} can be solved using a similar algorithm to ODC_{ag}-1 by changing the acyclic test to necklace, tree and star tests. These three tests can be computed in constant time at each stage.

Another extension is to allow the given graph (directed or undirected graph) to be a disconnected graph. The optimal clustering of each connected graph in the disconnected graph is found independently. The final solution is the union of the optimal solution obtained from clustering each connected graph.





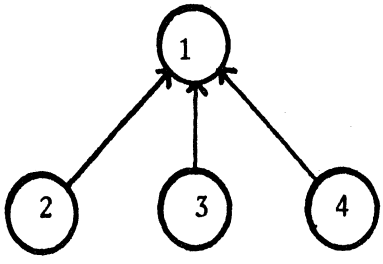
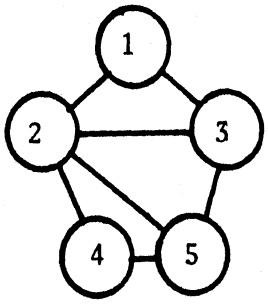
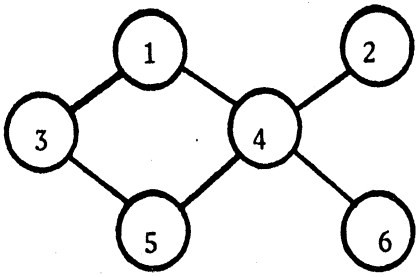
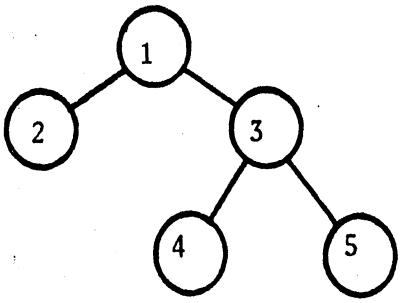


Figure 5-1a In-necklace, in-tree and in-star.







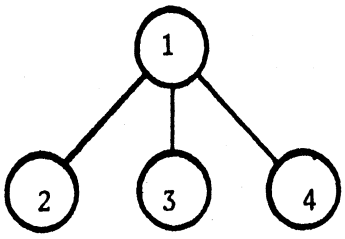


Figure 5-1b General undirected graph, necklace, tree and star.

In a case where the given graph contains parallel arcs or loops, our approach is still applicable if the graph can first be reduced to one without these parallel arcs and loops. For example, in many practical situation, the loop can simply be eliminated and each group of parallel arcs can be replaced by one arc with value equal to the sum of the arc values in the group (Chiang & Teorey '82).

5.2 Extensions of Objective Function and Constraint

So far, a simple objective function and two constraints (defined in Chapter 2) have been used for presenting the solutions of all the OGC subproblems. This section considers extensions to the class of objective function and constraints such that all previous algorithms will still be applicable and have the same time complexity.

A basic approach common to all the solutions of the OGC subproblems discussed so far is that they all use ODC_{aa}-1 (or its special case $ODC_{t^0t^0}$-1) without modifications in their computing of objective function value. Thus, any objective function applicable to ODC_{aa}-1 is also applicable to the rest of the OGC subproblems. This class of objective function is the class of monotonic objective functions which can be defined using the notation that is used in the proof for Proposition 4.4.

Let the n vertices in the out-tree visited in postorder traversal sequence be denoted by $v_1, v_2, \ldots, v_n$. A triplet formed for $K[v_j]$ can be represented by a sequence of pairs:

$$K[v_j] = [v_1, < >], [v_2, C_2], \ldots, [v_j, C_j].$$

Let $K_1[v_j] = (F(K_1[v_j]),W(K_1[v_j]),CF(K_1[v_j]))$ and $K_2[v_j] = (F(K_2[v_j]),W(K_2[v_j]),CF(K_2[v_j]))$ be two triplets formed in the same stage.

Let a <u>derivation</u> of a triplet $K_1[v_n]$ from a triplet $K_1[v_j]$ be the sequence of pairs: $[v_{j+1}, C_{j+1}']$, $[v_{j+2},C_{j+2}']$, ..., $[v_n,C_n']$. An <u>induced derivation</u> of a triplet $K_2[v_n]$ from $K_2[v_j]$ can be defined as the sequence $[v_{j+1},C_{j+1}"]$, $[v_{j+2},C_{j+2}"]$, ..., $[v_n,C_n"]$ such that C_i' is cluster-similar to $C_i"$ for $j+1 \le i \le n$. An objective function is <u>monotonic</u> if the following is true:

1) $F(K_1[v_j]) \le F(K_1[v_n])$.

2) $F(K_1[v_j]) \ge F(K_2[v_j]) \Longrightarrow F(K_1[v_n]) \ge F(K_2[v_n])$.

Three example monotonic objective functions are as follows:

1) $F_1(K) = \overline{\sum_{(i,j)\in E}} a\cdot X_{ij}\cdot f_{ij} + b\cdot X_{ij}\cdot \overline{f}_{ij}$

   where a and b are nonnegative real constants.

2) $F_2(K) = \overline{\sum_{v\in V}} a\cdot f(v)\cdot p(v)$

   where a is a positive real constant and f is any nonnegative real constant function which assigns a nonnegative real number to each vertex in V, i.e., $f:V\rightarrow R^+ \cup \{0\}$. Let $v_1,v_2,\ldots,v$ be a path from root v_1 to a vertex v. The integer function p assigns an integer number to each vertex in V as follows:

   $p(v_1) = 1$,

   $p(v_{r+1}) = p(v_r) + \overline{X}_{r,r+1}\cdot$

3) $F_3(K) = a\cdot F_1(K) + b\cdot F_2(K)$.

To show that a monotonic objective function is applicable to

ODC_{aa}-1, we only need to prove it is applicable to Proposition 4.4.

Proposition 5.1: Proposition 4.4 is still true when the objective function is a monotonic objective function.

Proof: Suppose $K_1[v_j]$ and $K_2[v_j]$ are feasible triplets formed in the same new arc stage and $K_1[v_j]$ dominates $K_2[v_j]$, i.e.,

1) $K_2[v_j]$ is triplet-similar to $K_1[v_j]$;
2) for every pair of C_1 and C_2 induced by $K_1[v_j]$ and $K_2[v_j]$ respectively, $W(C_2) \geq W(C_1)$;
3) $F(K_1[v_j]) \geq F(K_2[v_j])$.

Let $K_2[v_n]$ be a feasible triplet derived from $K_2[v_j]$. Then by 1), $K_1[v_n]$ can be derived from $K_1[v_j]$ with the induced derivation sequence. By 2), for each pair of $C_1 \in K_1[v_j]$ and $C_2 \in K_2[v_j]$, $W(C_2) \geq W(C_1)$. Thus, $K_1[v_n]$ is a feasible triplet. Further more, by 3) and the definition of a monotonic funtion,

$F(K_2[v_j]) \leq F(K_1[v_j])$ implies $F(K_2[v_n]) \leq F(K_1[v_n])$.

Hence, $K_2[v_j]$ can be eliminated. ***Q.E.D.***

Proposition 5.2: Proposition 2.1 is still true when the objective function is a monotonic objective function.

Proof: Immediate from Proposition 5.1. ***Q.E.D.***

To extend the set of constraints applicable to ODC_{aa}-1 and others, observe that the two simple constraints used in previous chapters are mandatory. This is because the cluster graph class constraint is used in the problem

classification (see section 1.1) while the integer cluster weight constraint is used in all algorithms to bound the computation time (e.g., in section 2.1.2, only $2 \cdot B^2$ new triplets are formed in each new arc stage).

But the class of formula for calculating the cluster weight and additional constraints can be imposed on all OGC subproblem solutions without increasing the time complexity if they are constant time computable constraints. A constraint is constant time computable if it can be defined in recursive formula such that it can be enforced in each stage of the clustering process in constant time and is independent of the calculation of the objective function value. Some examples of such constant time computable constraints are:

1) Let $d(v_1,v_2)$ be the number of arcs on the unique path from a vertex v_1 in a cluster C_r to a descendant v_2:

$$C_1 : \operatorname{Max}_{v_2 \in C_r}(d(v_r,v_2)) \leq Y$$

where v_r is the root of C_r and Y is any integer.

2) $C_2 : \overline{\sum_{v_i \in C}} \; w_i \leq B$

where w_i is the weight of vertex v_i.

3) Let $C[v_i]$ be an out-tree cluster rooted at v_i. The length $L(C[v_i])$ is defined as

$$L(C[v_i]) = s_i + \overline{\sum_{\forall j(X_{ij}=1)}} \lceil n_j/n_i \cdot L(C[v_j]) \rceil$$

where n_i and n_j are two integers associated with vertices v_i and v_j respectively, s_i is the length of vertex v_i and $L(C[v_j])$ is the length of the complete

subgraph in cluster $C[v_i]$ rooted at v_j:

C_3 : $L(C[v_i]) \leq B$.

All these three types of constraints can obviously be defined in recursive formula, which can be computed in constant time at each stage, and have nothing to do with the calculationn of the objective function value.

5.3 Divide-and-Conquer Technique for ODC_{aa}, ODC_{gg} and ODC_{ag}

This section applies Lukes' divide-and-conquer idea (Lukes '75) to lessen the high time complexity of ODC_{aa}, ODC_{gg} and ODC_{ag} with the simple objective function and constraints. For each of these problems, we divide the given digraph into a set of covers (nonvoid and nondisjoint subgraphs), find a solution to each of these covers independently and then recombine their clustering solutions into an optimal clustering solution. Before we present the algorithm in sections 5.3.2 and 5.3.3, some terminology is first defined in section 5.3.1.

5.3.1 Cutpoint and Cover

A cutpoint of a digraph G=(V,E) is defined as a vertex in V such that removing the vertex from V results in two or more subgraphs of G (e.g., vertices 5,6,8,10 and in Figure 5-2a). A digraph which has a cutpoint is called separable, and one which has none is called nonseparable. Let V'$\subseteq$V and E'$\subseteq$E. The induced subgraph G'(V',E') is called a (maximum biconnected) cover if G' is nonseparable and if for every larger V", V'$\subseteq$V"$\subseteq$V, the induced subgraph G"(V",E") is

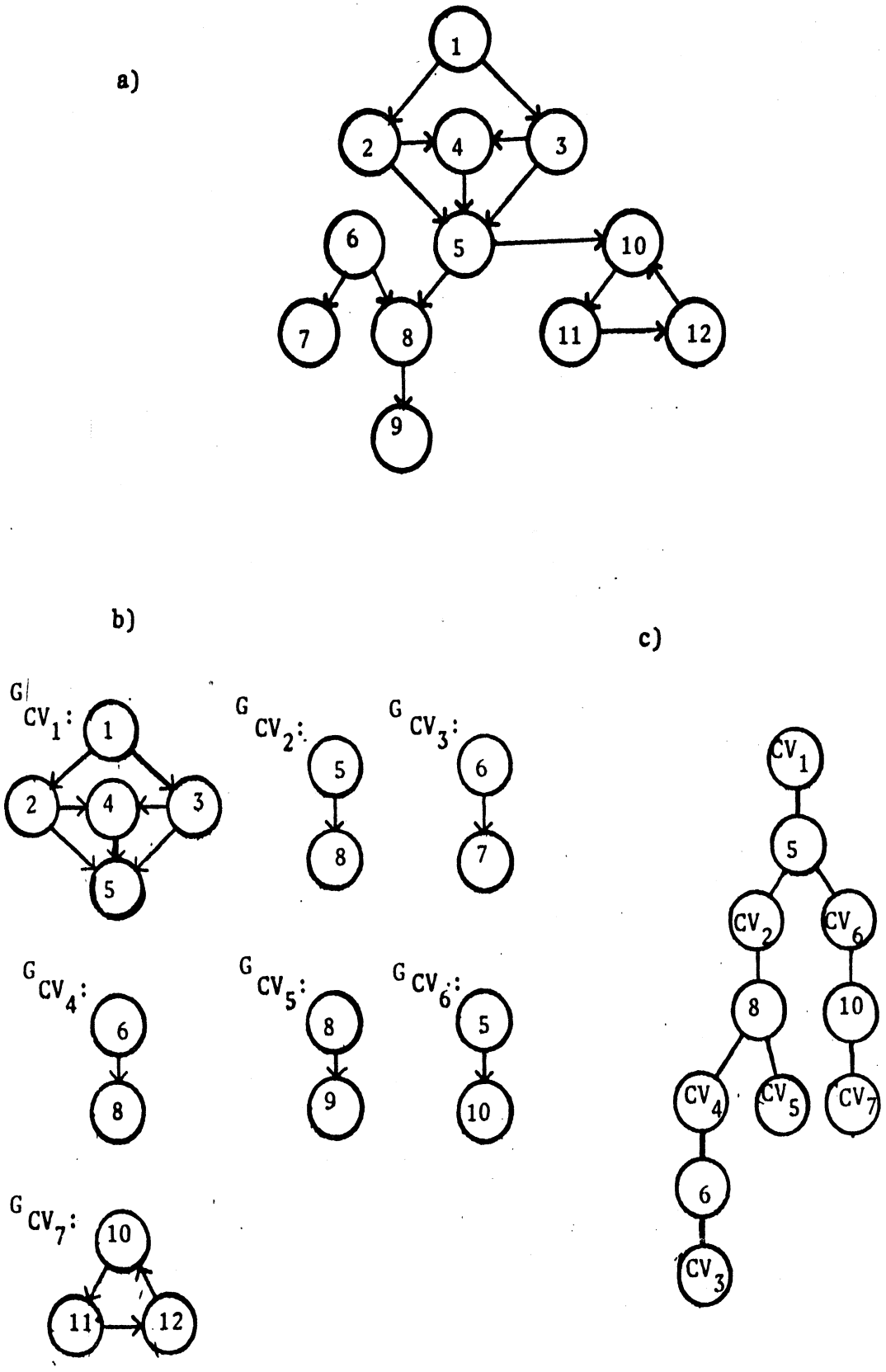


Figure 5-2 A general digraph and its cover graph.

separable (e.g., G_{CV_1} to G_{CV_7} in Figure 5-2b).

A cover (undirected) graph CG = (V,E) (e.g., Figure 5-2c) can be constructed from the set of covers and cutpoints of G. The vertex set V of CG is a set of vertices where each vertex represents either a cover (called cover vertex CV_j) or a cutpoint (called cutpoint vertex cp_i), i.e.,

$$V = \{ cp_1, cp_2, \ldots, cp_r, CV_1, CV_2, \ldots, CV_t \}.$$

The arc set E of CG is a set of undirected arcs (cp_i,CV_j) where the cutpoint cp_i is a vertex in the cover which CV_j represents, i.e.,

$$E = \{ (cp_i,CV_j) \mid cp_i \text{ is a vertex in the cover } G_{CV_j} \}.$$

Proposition 5.3: The cover (undirected) graph CG=(V,E) constructed from G have the following properties:

a) Each cover represented by a cover vertex in CG is nonseparable.

b) For any two covers represented by two cover vertices in CG, their intersection contains at most one vertex.

c) cp is a cutpoint in CG if and only if cp is the intersection of some two covers in G.

d) CG is a tree.

Proof: a), b) and c) are immediate from Aho et al ('74). d) is immediate from b). ***Q.E.D.***

If a digraph G has more than one cover, the following proposition shows that it is valid to find the optimal solution of G by clustering each cover independently and then combine these clusterings to generate an optimal

solution of G.

Proposition 5.4: Let a digraph G have s covers, where s>1, and K be an optimal clustering of G and induces t clusters, K= { C_1, C_2, ..., C_t }. The optimal solution K of G can always be decomposed into s subsets of clusters where each subset is a feasible clustering of one cover and the sum of the value of these s subsets is equal to the optimal clustering value.

Proof: The set of clusters induced by K can be divided into two sets. One set (KN) contains those clusters that have no cutpoints, while the other set (KC) does. Thus, $F(K) = F(KN) + F(KC)$.

Let <u>in-the-same-cover</u> be a relation defined on the set of clusters induced by K. Any two clusters C_1 and C_2 induced by K are in-the-same-cover if they contain only vertices of the same cover.

Since the set KN is a set of clusters that contain no cutpoints, each cluster in KN contains vertices in one cover only. This is because if a cluster contains vertices that belong to more than one cover, the cluster must also contain the cutpoints connecting these covers. Thus, the set KN can easily be partitioned into s disjoint subsets, KN_1, KN_2, ..., KN_s, by using the in-the-same-cover equivalent relation while the total value remains the same, i.e.,

$$\sum_{i=1}^{s} F(KN_i) = F(KN).$$

Since the set KC is a set of clusters that contain

cutpoints, each cluster in KC could contain vertices of one or more covers. For each cluster C in KC that contains vertices that belong to m of the s covers, C can be partitioned into m smaller disjoint clusters $\{C_1, C_2, \ldots, C_m\}$ such that

1) $C_1 \cup C_2 \cup \ldots \cup C_m = C$, and
2) for each cutpoint cp in C, cp is assigned to only one cover.

The partition of each C into m smaller clusters obviously do not increase the total value of the cluster, i.e.,

$$\sum_{j=1}^{m} F(C_j) = F(C).$$

Since the intention is to decompose K into s subsets of clusters where each subset is a feasible clustering of one cover, each cutpoint, which is assigned to only one cover, should also be included as a single vertex cluster in all covers that contain it. This newly formed set of clusters does not increase the value of the set KC because only the single vertex clusters, which each has no value, are added to other clusters. Each cluster now contains vertices belonging to only one cover. It can now easily be divided into s subsets, KC_1, KC_2, ..., KC_s, and satisifies

$$\sum_{i=1}^{s} F(KC_i) = F(KC).$$

Thus, the original optimal clustering K can be transformed into s subsets of clusters without changing the total value:

$$\{\{KN_1 \cup KC_1\}, \{KN_2 \cup KC_2\}, \ldots, \{KN_s \cup KC_s\}\}$$

and each $\{KN_i \cup KC_i\}$ is a feasible clustering of cover i.

Q.E.D.

From the proof of Proposition 5.3, the optimal solution of a digraph G can be obtained by the reverse process. Thus, the divide-and-conquer process consists of two steps:

1) Divide G into a set of covers.
2) Find and combine the clusterings of each cover to form the optimal solution.

5.3.2 Divide G into a Set of Covers

The process of finding a set of covers for a digraph is similar to the process of finding a set of biconnected components of an undirected graph. A biconnected component of an undirected graph G is a (maximum) subgraph of G which has no cutpoints. An efficient algorithm for finding biconnected components has been presented in Aho et al ('74). We modifiy it to solve the cover finding problem here.

The basic idea of the cover finding algorithm is to use the property that the removal of a nonleaf vertex of a tree splits a tree into two or more subgraphs. Thus, the algorithm first constructs a (depth-first) tree-like structure from the given digraph and then decides which vertices are cutpoints. The decision rules are based on the following proposition:

Proposition 5.5: Let G=(V,E) be a given digraph. Let $T=(V,E^0 \cup E^1)$ be a tree-like structure for G. Vertex v_r is a cutpoint of G if either of the following is true for v_r:

1) v_r is the root and has more than one child.

2) v_r is not a root nor a leaf vertex, and for some child v_j of v_r, there is no co-tree arc between any descendant of v_j (including v_j itself) and a proper ancestor of v_r.

Proof: Immediate from Aho et al ('74). ***Q.E.D.***

Using DF-NO(v_r) and LOW(v_r) that have been presented in Chapter 4, a vertex v_r is a cutpoint if

1) v_r is not a leaf vertex;

2) either one of the following is true:

2.1) v_r is not a root and LOW(v_j) $\geq$ DF-NO(v_r) where v_j is a child of v_r;

2.2) v_r is a root and has more than one child.

The following COVER-FINDING algorithm uses the concept of DF-NO(v_r) and LOW(v_r) to find all covers. The algorithm is similar to Aho et al's ('74) finding biconnected components algorithm.

ALGORITHM: COVER-FINDING-1

INPUT: an acyclic/general digraph G=(V,E);

OUTPUT: a list of the arcs for each cover of G and a set of cutpoints;

PROCESS:

ALGORITHM: Cover-search(v_r)

PROCESS:

1) mark v_r as "visited";

DF-NO(v_r) = df-count;

LOW(v_r) = DF-NO(v_r);

2) FOR each vertex v_j which is-related-to v_r DO:

2.1) PUSH the tree arc (i.e.,(v_j,v_r) or (v_r,v_j)) into STACK if it is not already there;

2.2) IF v_j is not visited DO:

2.2.1) parent(v_j) = v_r;

2.2.2) Cover-search(v_j);

2.2.3) IF LOW(v_j) ≥ DF-NO(v_r) DO:

cutpoint-set = cutpoint-set ∪ {v_r};

POP from STACK all arcs up to and including the arc containing v_j (i.e., (v_j,v_r) or (v_r,v_j));

include the arcs in cover(cover-count);

cover-count = cover-count + 1;

LOW(v_r) = MIN(LOW(v_r),LOW(v_j));

2.3) ELSE IF v_j is not parent(v_r) THEN

LOW(v_r) = MIN(LOW(v_r), DF-NO(v_j));

END;

I) Df-count = 1;

cover-count = 1;

II) choose an arbitrary vertex v as root of T to be built and CALL Cover-search(v);

III) RETURN the list of arcs of each cover and the set of cutpoints;

END.

Proposition 5.6: The run time of algorithm COVER-FINDING-1 is O(m) where m is the number of arcs in G.

Proof: Immediate from Aho et al's finding biconnected

cutpoints algorithm ('74).

5.3.3 Find and Combine the Clusterings of Each Cover to Form the Optimal Solution

Using the list of arcs of each cover and the set of cutpoints, a cover graph with a cover vertex as root can be easily constructed. The cover graph which is a tree can be used to order a sequential sequence of clusterings and combinations of covers similar to $ODC_{t^{\circ}t^{\circ}}$-1.

The vertices of the constructed tree is still visited in postorder traversal sequence. But for each vertex v_r, the $od(v_r)+1$ stages induced by visiting $G[v_r,0]$, $G[v_r,1]$, ..., $G[v_r,od(v_r)]$ in $ODC_{t^{\circ}t^{\circ}}$-1 is now aggregated into one stage. Thus, the sequential clustering sequence of the cover graph consists of n stages. Figure 5-3 shows the eleven stages of the cover graph that is in Figure 5-2.

The n stages can be classified into two types by whether the vertex considered is a cutpoint or cover vertex. The two types are <u>cutpoint stage</u> and <u>cover stage</u>. If the vertex considered at a certain stage is a cutpoint vertex cp_j, then that stage is a cutpoint stage. All the children of the cutpoint vertex cp_j are cover vertices. The operation needed at this type of stage is simply to collect all the feasible clusterings formed so far by each subtree of the tree rooted at these cover vertices. The set of feasible clustering collected is called a <u>cutpoint list</u> of cp_j, denoted by $CL[cp_j]$.

If the vertex considered at a stage is a cover vertex

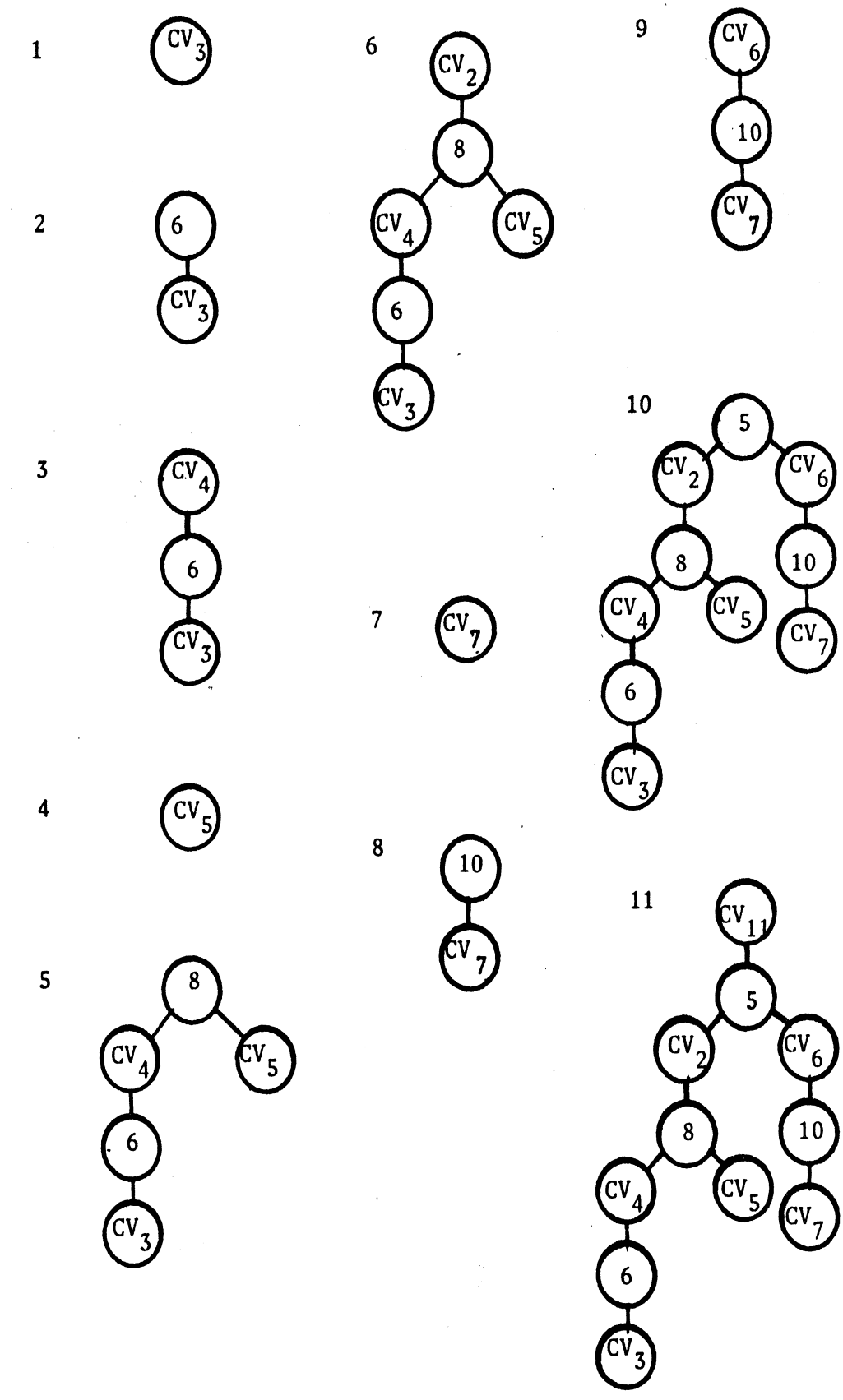


Figure 5-3 Eleven stages of the clustering process of cover graph given in Figure 5-2.

(CV), then it is a cover stage. The only parent vertex of CV is a cutpoint vertex (cp_i) and all the children of CV are cutpoint vertices (cp_j). The operation needed at this type of stage is to form a set of feasible clusterings from the corresponding cover G_{CV} which CV represents and from all the children cutpoint lists $CL(cp_j)$. Depending on whether it is an ODC_{aa}, ODC_{gg} or ODC_{ag} problem, a corresponding algorithm of ODC_{aa}-1, ODC_{gg}-1 or ODC_{ag}-1 can be modified to suit this purpose. The modifications are as follows:

1) Since the set of feasible clusterings formed for the cover vertex CV will be combined with its ancestor cover vertex, which shares the same cutpoint vertex cp_i with CV in the cover graph, only the set of all feasible clusterings with respect to this cutpoint needs to be formed. Step I) of ODC_{aa}, ODC_{gg} and ODC_{ag} can be modified to select cp_i as the root of the tree-like structure T to be constructed. The set of feasible clusterings produced at the end is the set desired.

2) Suppose v_r is a vertex in the cover G_{CV} and v_r is a cutpoint cp_j in CG. To obtain a set of all feasible clusterings for vertex v_r in CV, the vertex should be composed with the corresponding cutpoint list of $CL[cp_j]$ at the end of the stage that considers v_r. The operation is called a COMPOSE operation which is defined as:

   COMPOSE operation:

FOR each pair of feasible triplets $K[v_r]$ and $K[cp_j]$ from $PK^f[v_r]$ and $CL[cp_j]$ respectively, form a triplet with

$$F(K[v]) = F(K[v_r]) + F(K[cp_j]),$$

$$W(K[v]) = W(K[v_r]) + W(K[cp_j]) - w_r,$$

$$CF(K[v]) = CF^0(K[v]) \cup CF^1(K[v]) \text{ where}$$

$$CF^0(K[v]) = \{C_2, C_3, \ldots, C_p, C_2', C_3', \ldots, C_q'\} \cup \{C_1 \cup C_1' - \{v_r\}\},$$

$$CF^1(K[v]) = CF^1(K[v_r]) \cup CF^1(K[cp_j]).$$

The second step of the divide-and-conquer process for finding and combining clusterings of each cover to form the optimal solution can now be described in an algorithm:

ALGORITHM: FIND-GLOBAL-SOLUTION

INPUT: a list of arcs for each cover and a set of cutpoints;

OUTPUT: an optimal solution and its value;

PROCESS:

I) choose an arbitrary cover vertex CV as root and construct a cover graph CG using the list of arcs for each cover and the set of cutpoints;

II) FOR each vertex in the cover graph traversed in postorder sequence DO:

IF the vertex is a cover vertex CV THEN

1) let cp_i be the parent vetex of CV in the cover graph; choose cp_i as the root and CALL RESTRUCTURE-3(G_{CV}, cp_i) to build a tree-like structure T with lists $L_{nb}[v_r]$ and $L_{bn}[v_r]$ for

each vertex;

2) FOR each v_r in the cover G_{CV} traversed in postorder sequence DO:

2.1) FOR j=0 to od(v_r) DO:

2.1.1) same as 1.1 of ODC_{aa}, ODC_{gg} or ODC_{ag};

2.1.2) same as 1.2 of ODC_{aa}, ODC_{gg} or ODC_{ag};

END;

2.2) IF v_r is a cutpoint in the cover graph THEN

2.2.1) COMPOSE $PK^f[v_r,od(v_r)]$ produced in 2.1 with the cutpoint list $CL[v_r]$ to form a new set of $PK^f[v_r,od(v_r)]$;

2.2.2) eliminate all dominated triplets in $PK^f[v_r,od(v_r)]$ formed in 2.2.1;

END;

ELSE IF the vertex is a cutpiont cp_i THEN DO:

FOR each child v_j of cp_i DO:

$CL[cp_i] = CL[cp_i] \cup PK^f[v_j]$;

END;

END;

III) find the triplet in $PK^f[CV,od(CV)]$ that has the maximun value and RETURN it;

END.

This algorithm reduces the number of feasible triplets formed at each stage of ODC_{aa}-1, ODC_{gg}-1 or ODC_{ag}-1 to a number directly proportional to the number of feasible triplets formed if each cover were clustered independently. This algorithm can also be applied to OUC_{uu}, OUC_{nu}, OUC_{tu}

and OUC_{su} as well with the simple objective function and two constraints given in section 2.1.

CHAPTER 6

SOME INFORMATION SYSTEM DESIGN APPLICATIONS

This chapter considers three information system design problems that can be formalized and solved as OGC problems. To show that an information system design problem can be solved by using an OGC solution algorithm, it is necessary to define the objective function and constraints, decide which ODC problem it is and prove the constraints and objective function to be applicable. The objective function of a design problem can usually be formulated in many ways. Only one way is shown here to illustrate the solution approach. The three information system design problems to be discussed are:

1) B-tree secondary storage allocation problem (section 6.1);
2) translation of integrated schema to IMS schema problem (section 6.2); and
3) database record clustering problem (section 6.3).

6.1 B-Tree Secondary Storage Allocation

Computer based information systems are critical to the management of large public or private organizations. Such systems typically operate on databases containing billions

of characters and serves varying user communities' information needs (March '83). In these systems, indexes (or directories) are built upon the data files (which are stored on secondary storage) to facilitate random or sequential accesses. Because of the large amount of data files, the indexes (or directories) are large files in their own right and are stored in secondary storage. In the present practice of Data Base Management Systems (DBMS) and Information Retrival systems, B-tree is one of the most popular techniques for organizing index files (Comer '79).

The idea of B-tree is related to binary search tree and m-ary search tree. A binary search tree is a tree which contains either no vertices at all, or every vertex in the tree contains a key and two children where

1) all keys in the left subtree are less (alphabetically or numerically) than the key in the root;
2) all keys in the right subtree are greater than the key in the root; and
3) the left and right subtrees are also binary search trees (Horowitz & Sahni '76).

A m-ary search tree is a generalization of a binary search tree in which each vertex in it has at most m-1 keys and m children. If t_1 and t_2 are two children of some vertex, and t_1 is to the left of t_2, then the keys in the subtree rooted at t_1 are all less than the keys in the subtree rooted at t_2.

A B-tree of order m is a special type of balanced m-ary

tree with the following properties:

1) The root is either a leaf or has at least two children.
2) Each vertex, except for the root and the leaf vertices, has between $\lceil m/2 \rceil$ and m children.
3) Each path from the root to a leaf vertex has the same length (Aho et al '83).

Figure 6-1 shows a B-tree of order 5.

To retrieve a record in the data file with a given key value, the B-tree is used instead of searching all record sequentially in the data file. The process always starts by searching from the root to the leaf vertex which contains the key value and a pointer to the record with that key value in the data file. If each vertex in the B-tree of order m is stored in a separate (physical) block and there are a total of n different records (or keys) in the data file, each such retrieval needs no more than $\log_{\lceil m/2 \rceil} n+1/2$ physical block accesses (PBAs) to the index file. Since very often the length of a vertex in the B-tree is much less than the secondary storage block length (which is typically 2000 to 4000 bytes), many vertices can be stored in one block. This gives a chance of lowering the $\log_{\lceil m/2 \rceil} n+1/2$ PBAs by storing more than one vertex in a block. But observe that only storing a vertex with its parent in the same block can reduce the number of PBAs. Thus, for each vertex there is only one decision to be made, i.e, whether to store the vertex with its parent in the same block or not. This is

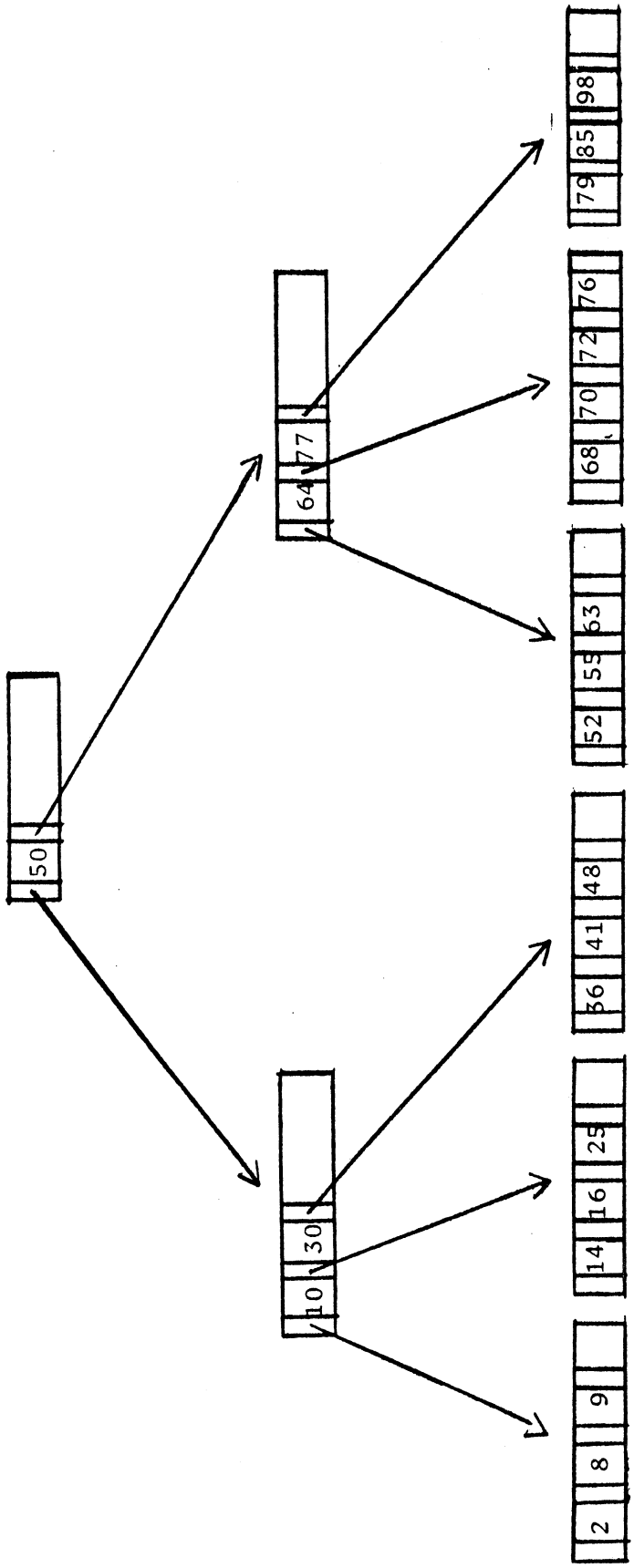


Figure 6-1 A B-tree of order 5.

equivalent to assigning a 0 or 1 on each arc and thus an OGC problem. The decision can be based on the usage pattern of the environment which is the access frequencies to each key in the B-tree. The higher the frequency a key is accessed the more ancestors this key should be stored in the same block with. Since a B-tree can be represented as an out-tree, the problem then is to find a set of out-tree clusters or its subclass clusters such that the total expected number of PBAs for accessing every key by the environment are minimized.

The B-tree secondary storage allocation problem can be formally defined as an $ODC_{t°t°}$ problem:

Given:

1) an out-tree G=(V,E) which represents the B-tree of order m;
2) two constraints: one requires each cluster be an out-tree or its subclass; another limits the length $L(C_i)$ of each cluster C_i induced by a clustering K to a block length B, i.e.,

   $\forall C_i \in K \quad L(C_i) = \overline{\sum_{v \in C_i}} s(v) \leq B$ where

   s is a constant function which assigns to each vertex an intger value equal to the size of the vertex, which is: m · size of pointer + (m-1) · size of key;
3) an objective function which is defined as

   $$F(K) = \overline{\sum_{v \in V}} f(v) \cdot PBA(v)$$

   where f(v) is the frequency of accesses to v; PBA(v)

is the number of physical block accesses needed to access v from the root. Let $r=v_1,v_2,\ldots,v_m=v$ be a path for root r to a vertex v. The PBA function can be defined recursively as

$PBA(v_1) = 1;$

$PBA(v_{i+1}) = PBA(v_i) + \bar{X}_{v_i,v_{i+1}}$;

($\bar{X}_{v_i,v_{i+1}}$ is the inverse of the assignment on the arc (v_i,v_{i+1})).

Find: A feasible clustering K such that $F(K)$ is minimized.

Proposition 6.1: The objective function defined above is monotonic and the second constraint is constant time computable.

Proof: Immediate from F_2 and C_2 of section 5.2. ***Q.E.D.***

6.2 Translation of Integrated Schema to IMS Schema

Database design is the process of synthesizing conceptual and target DBMS schema (logical or physical) that satisfies the information manipulation needs of a community of users. A stepwise database design process which consists of the following stages has widely been accepted (Teorey & Fry '82):

Stage 1) Requirements formulation and analysis.

Stage 2) Conceptual design.

Stage 3) Implementation design.

Stage 4) Physical design.

This section considers a principal issue in an

implementation design, which is how to map an integrated schema produced by the conceptual design stage to an IMS schema.

Many design tools as surveyed by Chen et al ('80) have been developed to aid each stage of the design process. Recently, a new database design system called View Integration System (VIS) (Chiang et al '83) has been built to support the first three stages of a database design process. The VIS integrates many related tools to support a bottom-up logical database methodology. One of the tools is for translation of an integrated schema to an IMS schema which can be based on several solutions given in this dissertation and will be discussed in this section.

An integrated schema synthesized from user views by VIS consists of a set of 3NF record types (or groups) interrelated by relationships which are essentially of type 1:m (1:1 is a special case of it). Depending on the structure of the integrated schema, it can be represented as a general digraph , acyclic digraph, out-necklace, out-tree, or out-star with each vertex representing a record type and each arc representing a relationship.

The primary task of translating the integrated schema to the IMS schema is to transform the integrated schema into a schema that satisfies the rules of IMS. An IMS schema is made up of one or more physical databases and logical database descriptions.

A physical database description is made up of

descriptions for a set of segments interrelated by a set of relationships in a hierarchical tree structure. The concept of segments is equivalent to the concept of record types and each hierarchical parent to child relationship is also of type 1:m. Thus, each physical database description can be represnted as an out-tree with each vertex representing a segment and each arc representing a 1:m relationship. But a set of out-trees cannot represent those additional arcs that form m-indegree vertices or cycles that could exist in a general digraph, acyclic digraph or out-necklace. The logical database descriptions are used to supplement these missing information.

A logical database description is composed of descriptions of segments from one or more physical data bases interrelated by 1:m relationships in a hierarchical tree structure. The idea of forming a logical database description is to establish a relationship between any point in a physical database and a target segment of another or the same physical database. The target segment is called the logical parent segment. This relationship is provided by establishing a link field as a new data field created as part of an existing segment or part of a newly created virtual segment which contains the link field (Cardenas '79).

The links in the logical or physical database are unidirectional, proceeding from parent to child and not vice versa. But each child can be specified to have a pointer to

its parent. In the following discussion, it is assumed that this is the case. This allows a link to be bidirectional and makes for a simpler representation for m-indegree vertices and cycles.

Figure 6-2a shows two alternatives of translating an integrated schema into its corresponding IMS physical databases and logical database descriptions in pictorial form. The solid boxes, solid arcs and dash boxes represent segments, relationships and virtual segments respectively in physical database descriptions. The dash line represents a relationship in logical database descriptions. Figure 6-2b is another example.

Several restrictions are imposed on the hierarchial tree structure of logical or physical database descriptions. The relevant one is that a maximum of 255 segments types may be included and up to 15 segment types are allowed in any one hierarchical path.

Again, to decide which alternative is better, the usage pattern of the environment must be estimated or collected. The usage pattern used in Figure 6-2a is the frequency of transitions between two vertices in the integrated schema.

Obviously, if the frequencies of transitions from segment STUDENT to GRADE (i.e., 300 times/day), as well as COURSE and GRADE (i.e., 100 times/day) shown in the integrated schema in Figure 6-2a are specified, the first translation shown in Figure 6-2a is a better choice. This is because if the occurrences of each segment type are

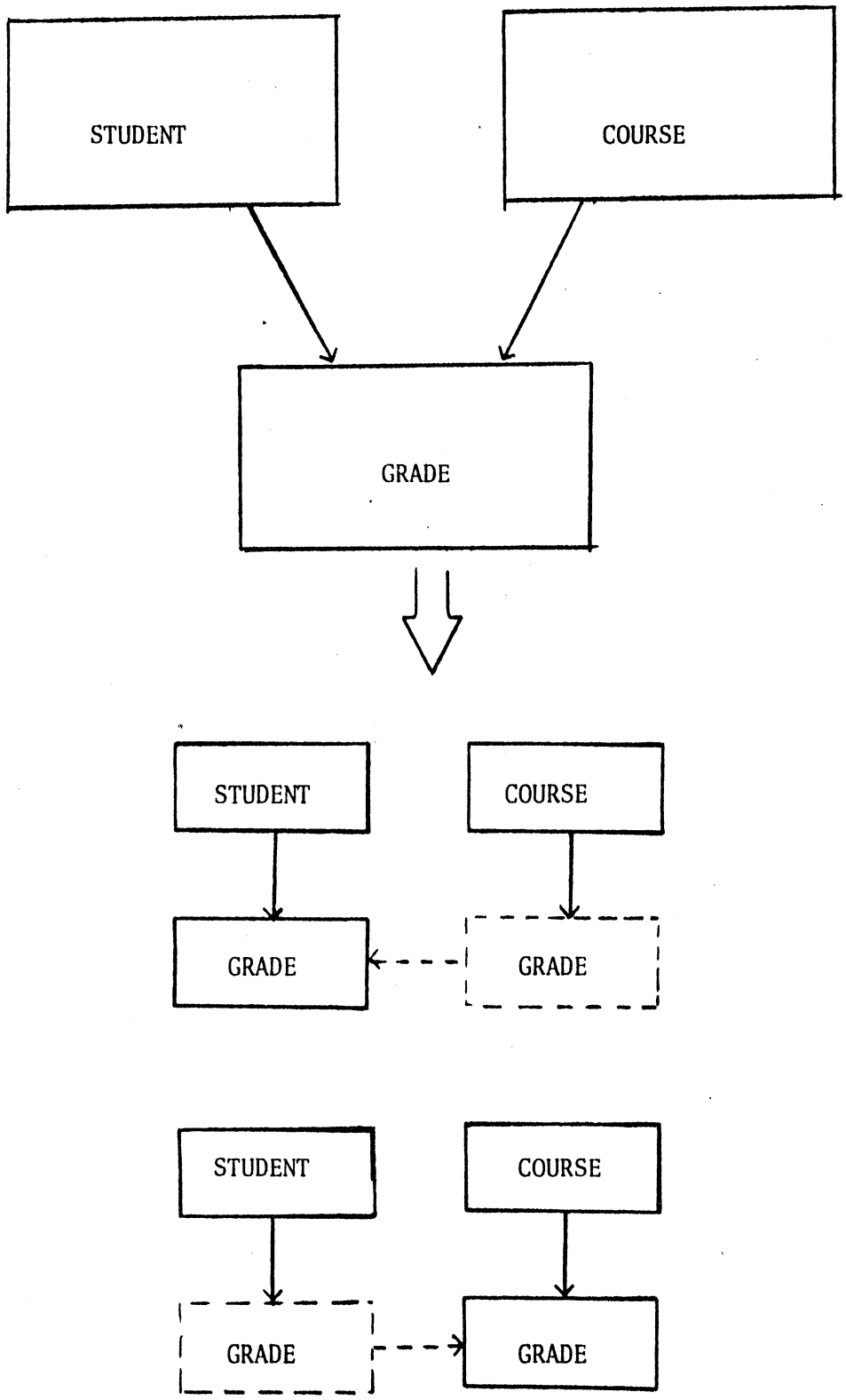


Figure 6-2a Translating an integrated schema into its corresponding IMS physical and logical databases: example 1.

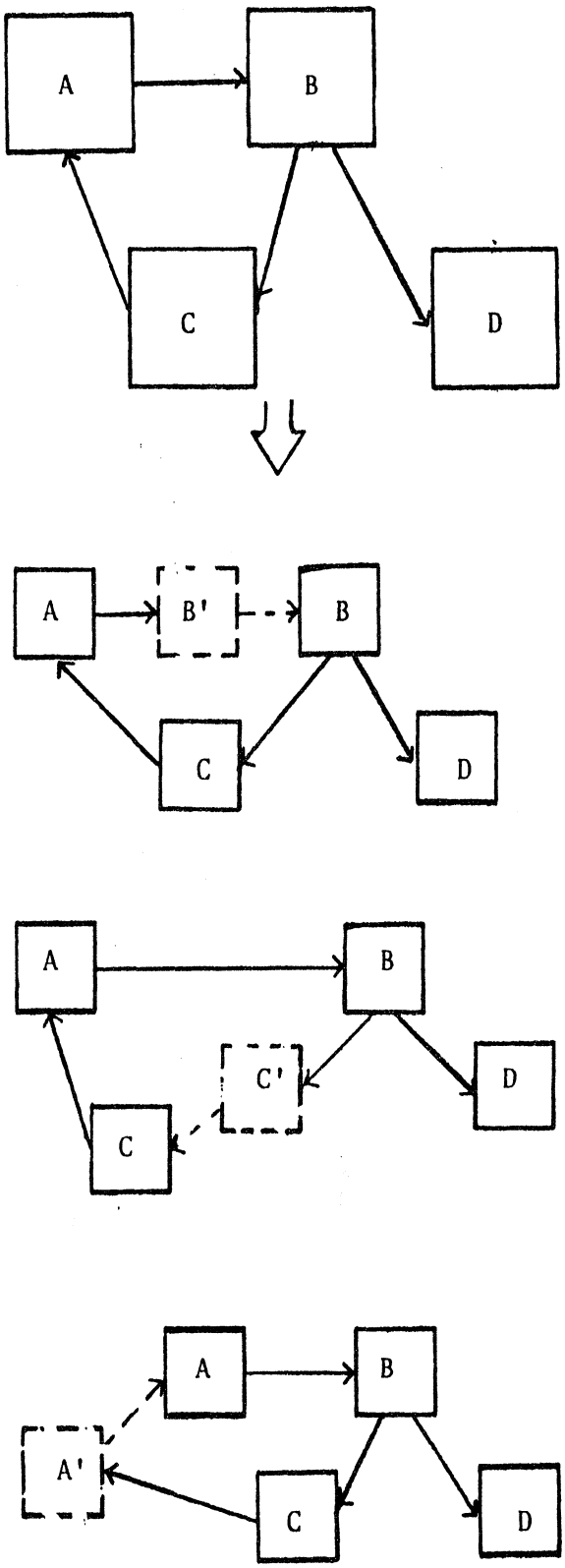


Figure 6-2b Translating an integrated schema into its corresponding IMS physical and logical databases: example 2.

stored in different physical blocks, the total number of PBAs of the schema for accessing GRADE from STUDENT and accessing GRADE from COURSE together is less than that of the second translation:

Total block access of first translation in Figure 6-2a:

$300 \cdot 1 + 100 \cdot 2 = 500$.

Total block access of second translation in Figure 6-2a:

$300 \cdot 2 + 100 \cdot 1 = 700$.

This translation problem can be formalized clearly as $ODC_{t \circ g}$, $ODC_{t \circ n \circ}$, $ODC_{t \circ a}$, $ODC_{t \circ t \circ}$, or $ODC_{s \circ s \circ}$, depending on the class of digraph the integrated schema belongs to. The translation problem can be formalized as follows:

Given:

1) a digraph $G=(V,E)$ which represents the integrated schema;
2) three constraints: one requires each cluster be an out-tree or its subclass; one limits the size $S(C_i)$ of each cluster C_i of a clustering K to 255, i.e.,

$$\forall C_i \in K \quad S(C_i) = \text{number of vertices in } C_i \leq 255;$$

one limits the depth $D(C_i)$ of each cluster C_i of a clustering K to 15, i.e.,

$$\forall C_i \in K \quad D(C_i) = \operatorname{Max}_{v \in C_i} (d(v)) \leq 15$$

where d(v) assigns an integer that is equal to the length of the unique path from root of C_i to v;

3) an objective function which is defined as

$$F(K) = \overline{\sum_{X_{ij}=1}} f_{ij} + 2 \cdot \overline{\sum_{X_{ij}=0}} f_{ij}$$

$$= \sum_{(i,j)\in E} (2 \cdot X_{ij} + \overline{X}_{ij}) \cdot f_{ij}$$

where f_{ij} is the transition frequency between vertices v_i and v_j.

Find: A feasible clustering K such that $F(K)$ is minimized.

Proposition 6.2: The objective function defined above is monotonic and the second and third constraints are constant time computable.

Proof: This is immediate from F_1 and C_2 and C_1 in section 5.2. ***Q.E.D.***

6.3 Database Record Clustering

An important cost factor of a physical database design depends on the secondary storage search time that is required by the typical operational workload. The major component of that search time is the expected number of physical block accesses (PBAs). Thus, one important consideration in the initial physical database design, or a later reorganization of the database, is to map the logical database structure into a secondary storage in such a way so as to minimize PBAs (Chiang & Teorey '82). One way to minimize the expected PBAs is to make use of the database usage pattern to group occurrences of those record types that are logically related and are frequently accessed together as clusters and store them into the same block so that inter-block accesses can be minimized.

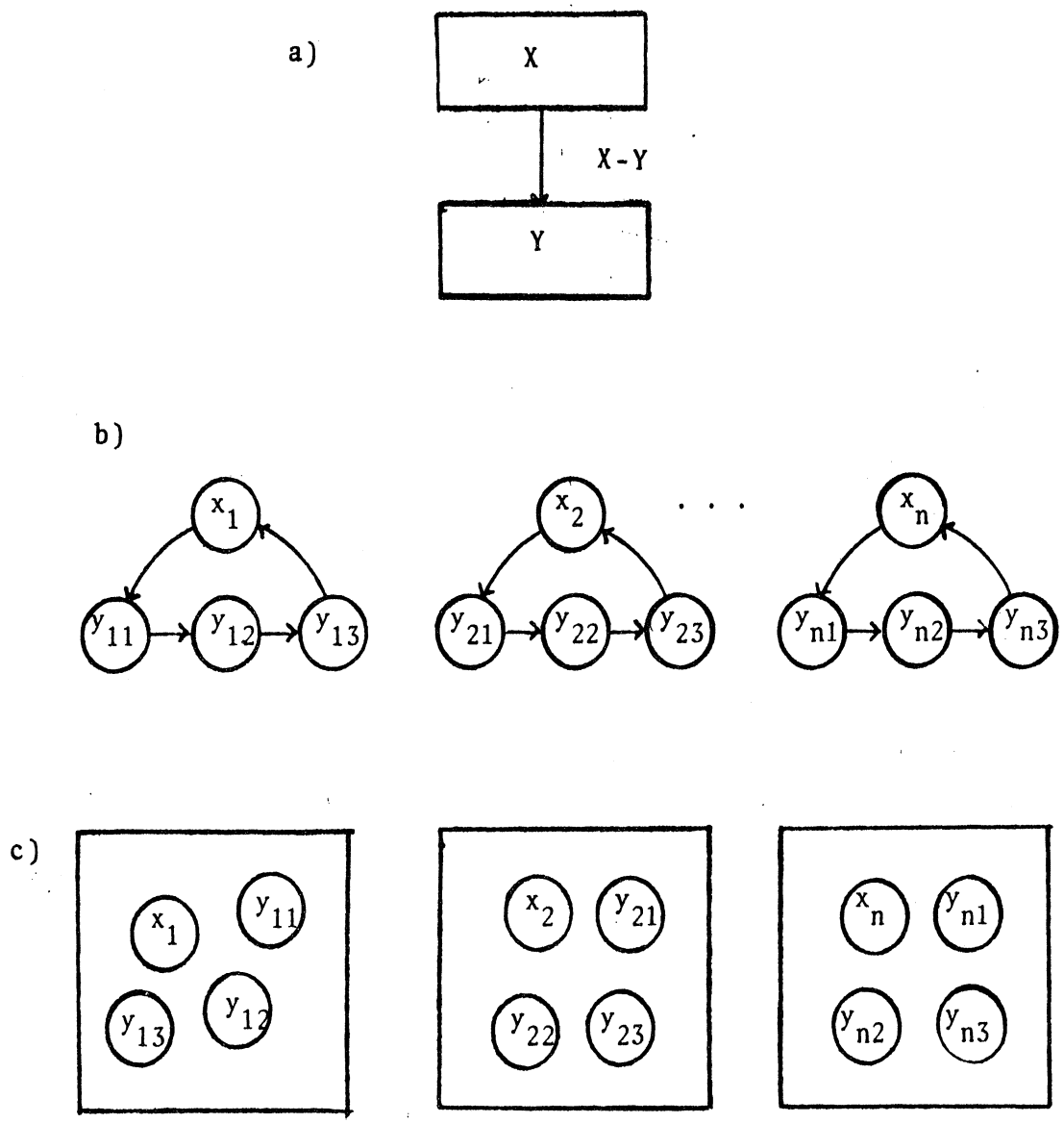
Many DBMS' (e.g., IDMS, IMS, SYSTEM R) allow users to

specify such a request using a statement like CLUSTERED VIA SET NEAR OWNER. For example, in Figure 6-3a, X is the parent record type of child record type Y. The occurrences of X and Y are physically linked together like Figure 6-3b. Assume occurrences of X and Y are frequently accessed together. A user can specify CLUSTERED Y VIA X-Y NEAR OWNER X which will result in the placement of occurrences of X and Y by the DBMS as shown in Figure 6-3c. This placement has the advantage that any time x_i is read into memory, y_{i1}, y_{i2} and y_{i3} are also in memory. This reduces PBAs.

There are several restrictions to the way a database designer can specify the grouping of occurrences statement. They are:

1) each specification must be via a set type;
2) each record type can only be clustered with one of its parent;
3) a record type may not be CLUSTERED VIA a SET type of which it is both member and owner type; and
4) the specification cannot form a cycle.

The first restriction implies that the database record clustering problem is a clustering problem. The second, third and fourth restrictions imply that each cluster must be an out-tree or its subclass. Thus, the database record clustering problem is an $ODC_{s°s°}$, $ODC_{t°t°}$, $ODC_{t°a}$, $ODC_{t°n°}$, or $ODC_{t°g}$ problem if the given logical database can be represented as an out-star, out-tree, out-necklace, acyclic digraph or general digraph respectively.



A rectangle represents a record type.

A square represents a page.

A circle represents an occurrence.

Figure 6-3 Placements of record occurrences using CLUSTERED VIA SET NEAR OWNER.

The database record clustering problem can be formalized as follows:

Given:

1) a digraph G=(V,E) which represents the database schema;
2) two constraints: one requires each cluster to be an out-tree or its subclass; one limits the length $L(C[v_i])$ of each cluster $C[v_i]$ rooted at v_i to the block length B:

$$L(C[v_i]) = s_i + \sum_{\forall j(X_{ij}=1)} \lceil n_j/n_i \cdot L(C[v_j]) \rceil \leq B$$

where n_i and n_j are the expected number of occurrences of vertices v_i and v_j, s_i is the length of vertices v_i and $L(C[v_j])$ is the length of the complete subgraph in $C[v_i]$ rooted at v_j;

3) an objective function which is defined as

$$F(K) = \sum_{X_{ij}=0} f_{ij}$$

where f_{ij} is the ~~same function defined in section 6.2.~~ transition frequency between vertices v_i and v_j.

Find: A feasible clustering K such that $F(K)$ is minimized.

Proposition 6.3: The objective function defined above is monotonic and the second constraint is constant time computable.

Proof: Immediate from F_1 and C_3 of section 5.2. ***Q.E.D.***

Example: Let the database schema given by Teorey and Fry ('80) be represented by an acyclic digraph shown in Figure

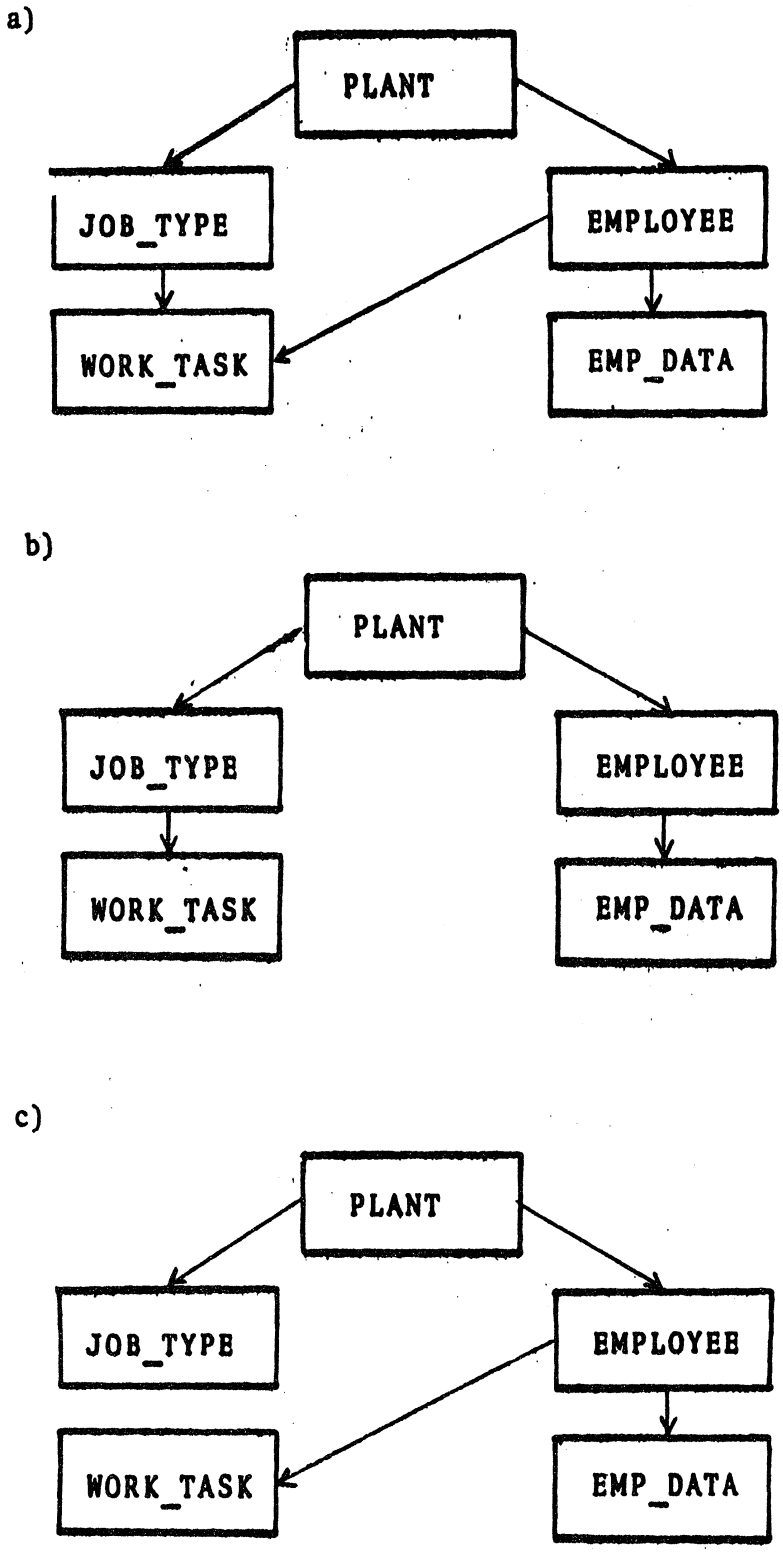


Figure 6-4 A database schema and its two maximal out-tree clusterings.

6-4a. The record length (s_i) and the expected number of occurrences (n_i) of each record type v_i is shown in the following table:

record type	record length	expected no. of occurences
PLANT	60	50
JOB_TYPE	26	1K
WORK_TASK	13	200K
EMPLOYEE	80	100K
EMP_DATA	36	100K

The usage information which is characterized by the frequency and the number of logical record accesses (LRAs) to the database schema by each application is defined in the following table:

application	frequency	LRAs
T1	100/day	1 PLANT+1 EMPLOYEE+1 EMP_DATA
T2	100/day	1 PLANT+1 EMPLOYEE+1 EMP_DATA
T3	200k/day	1 EMPLOYEE+ 1 WORK_TASK+ 1 JOB_TYPE
T4	120k/year	1 EMPLOYEE
R1	Quarterly	100k EMPLOYEE +100k EMP_DATA
R2	Annually	50 PLANT + 100k EMPLOYEE
R3	Monthly	50 PLANT
R4	Monthly	50 PLANT + 50·20 JOB_TYPE
R5	Daily	50 PLANT + 100k EMPLOYEE + 200k WORK_TASK + 200k JOB_TYPE
R6	Monthly	50 PLANT+100k EMPLOYEE+100k EMP_DATA
R7	1000/day	1 EMPLOYEE

Let f_{ij} be the transition frequency between v_i and v_j. A table showing f_{ij} can be constructed using the frequency and LRA of each application as follows:

(We assume: 1 year=240 working days, 1 quarter=60 working days, 1 month= 20 working days.)

(v_i,v_j)	f_{ij}	from

PLANT, EMPLOYEE	0	
EMPLOYEE, EMP_DATA	100+100+100k/60	
	+100k/20=6.86k	T_1, T_2, R_1, R_6
PLANT, JOB_TYPE	1k/20=0.05k	R_4
JOB_TYPE, WORK_TASK	200k+200k=400k	T_3, R_5
EMPLOYEE,WORK_TASK	200k+200k=400k	T_3, R_5

Let the block size B be 2,000. Since the given database schema can be represented as an acyclic digraph and the desired clusters are out-tree or its subclass, the $ODC_{t\circ a}$ presented in Chapter 3 is applied here. The following traces the results produced by each major steps:

step	action performed
1	Obtain the first maximal out-tree clustering (see Figure 6-4b).
1.1	Apply $ODC_{t\circ t\circ}$-1 to obtain 3 clusters: cluster 1 contains: PLANT, JOB_TYPE cluster 2 contains: WORK_TASK cluster 3 contains: EMPLOYEE, EMP_DATA with value $F(K)$= 0 + 400k + 400k = 800k.
1.2	Keep the solution (because this is the first result).
1	Obtain the second maximal out-tree clustering (see Figure 6-4c).
1.1	Apply $ODC_{t\circ t\circ}$-1 to obtain 2 clusters: cluster 1 contains: PLANT, JOB_TYPE cluster 2 contains: EMPLOYEE, WORK_TASK EMP_DATA

with value $F(K)= 0 + 400k = 400k$.

1.2 Keep the solution (because 400k is less than 800k).

2 RETURN the kept solution with an $F(K)=400k$. ***Q.E.D.***

Note: Without clustering, the number of PBAs is 0 + 6.86k + 0.05k + 400k + 400k = 806.91k, while with clustering, it is 400k. This is a savings of (806.91-400)/806.91 = 50%.

CHAPTER 7

SUMMARY AND FUTURE RESEARCH

7.1 Summary of Results

In this dissertation, we have investigated the problem of cutting a graph into clusters such that a set of constraints on the clusters is satisfied while an objective function is optimized.

In Chapter 1, eight classes of digraphs and four classes of undirected graphs are considered and used to divide the Optimal Graph Clustering (OGC) problem into a set of thirty-five subproblems.

Chapter 2 describes a sequential approach which obtains optimal solutions for $ODC_{t\circ t\circ}$, $ODC_{s\circ t\circ}$ and $ODC_{s\circ s\circ}$ in $O(n*Z^2)$, $O(n*Z)$ and $O(n*Z)$ time respectively. The set of constraints considered are: a) every cluster must be an out-tree/out-star or its subclass; b) every cluster must have cluster weight less than or equal to B. The objective function is to maximize the sum of all cluster values.

In Chapter 3, the maximal out-tree/out-star clustering concept and the $ODC_{t\circ t\circ}$-1/$ODC_{s\circ t\circ}$-1 are used to find solutions to $ODC_{t\circ t\circ}$, $ODC_{t\circ a}$, $ODC_{t\circ n\circ}$, $ODC_{s\circ g}$, $ODC_{s\circ a}$ and $ODC_{s\circ n\circ}$. The solution to $ODC_{t\circ t\circ}$ and $ODC_{t\circ n\circ}$ are then used to solve $ODC_{n\circ n\circ}$, which in turn is used to solve $ODC_{n\circ g}$.

Only $ODC_{t°n°}$ $ODC_{s°n°}$ and $ODC_{n°n°}$ have pseudopolynomial time solutions, i.e., $O(n*Z^2*K)$, $O(n*Z*K)$ and $O(n*Z^2*K)$. Others are exponential with respect to the number of m-indegree vertices. Thus, if the given digraph is a sparse graph its ODC solution is still computable. The objective function and the set of constraints used are the same as that in Chapter 2.

Chapter 4 describes a generalized sequential approach of Chapter 2 to solve ODC_{aa}, ODC_{gg} and ODC_{ag}. The same set of constraints and the objective function of Chapter 2 are used. The time complexity is totally dependent on the number of neighbors of each vertex. An upper bound on the number of new triplets formed by each stage is:

$$B^{n[p,r]} * n[p,r]! * B^{n[r,j]} * n[r,j]! * (n[r,j]+1).$$

In Chapter 5, the solutions of Chapters 2 to 4 are used to solve the rest of the OGC subproblems. A solution approach for disconnected graph or graph with parallel arcs or loops are also discussed. Next, the class of objective functions and constraints are extended to the class of monotonic objective function and constant time computable constraints for all solutions presented in the dissertation without changing the time complexity. A divide-and-conquer technique is then presented to lessen the high complexity of the ODC_{aa}-1, ODC_{gg}-1 and ODC_{ag}-1 algorithms. The technique reduces the number of feasible triplets formed at each stage to a number directly proportional to the number of feasible triplets formed if each cover were clustered independently.

Chapter 6 illustrates the usefulness of the solutions by applying them to three information system design problems.

7.2 Future Research

An immediate future research should be to extend the applicable objective functions and constraints with respect to the divide-and-conquer technique discussed in section 5.3. This technique can substantially cut down the computational steps of all strong NP-complete problems.

Since only eight classes of digraphs and four classes of undirected graphs have been considered in this dissertation, a natural extention is to consider other classes of graphs such as planar graph, etc. For example, the solution to planar graph is useful for solving the schema or program structure layout problems.

Instead of the problem of finding a set of (disjoint, nonvoid) clusters, another extension to the Optimal Graph Clustering problem is the finding of a set of (nondisjoint, nonvoid) covers which satisfies a set of constraints while an objective function is optimized. The solution of this class of problem can be used to decompose database schema, programs, and queries for distributed processing.

APPENDICES

APPENDIX A

THE BREAKUP OF THE FOURTEEN ODC PROBLEMS AS AN ODC PROBLEM

The writing of a dissertation is also a design problem. It is to present the research results in a structured way for easy comprehension. A structured presentation typically includes a sequence of chapters roughly of equal length and covers the following topics:

1) Introduction to the problem.
2) Solutions to the problem.
3) Extensions to the solution.
4) Applications of the solution.
5) Conclusion and future direction.

This dissertation follows such a structure closely; however, the solutions to the ODC problems are exceptionally long. To maintain approximately equal length chapters, it is necessary to break up the solutions into three separate chapters. This breakup can be formalized as an Optimal Digraph Clustering problem. Using this problem as an example, the purpose and general content of this dissertation can be illustrated.

Figure A-1 shows a digraph which represents the breakup problem. The digraph is an out-tree which has 14 vertices, each representing an ODC problem solution. Each arc on the

digraph represents one of the following two cases:

1) The head of the arc uses a modified version of the tail in its solution. This applies to arcs $ODC_{t^{\circ}t^{\circ}}$ --> $ODC_{s^{\circ}t^{\circ}}$ and $ODC_{t^{\circ}t^{\circ}}$ --> ODC_{gg}.
2) The head of the arc uses an exact version of the tail in its solution. This applies to the rest of the arcs.

The values associated with the arcs in the first case indicate how similar the two vertices on the arc are, while the values in the second case indicate how importantly the tail vertex is used in the head vertex. The value rating is from 1 to 5 which gives a sense of how two vertices on an arc are related to each other. The weight appearing at the upper right corner of each vertex represents the number of pages that is planned for each solution.

Let us suppose each cluster size should not exceed 20 pages. And let the optimization function be defined as the finding of a set of hierarchical related problems such that their closeness is maximized. Thus, this division problem is an instance of $ODC_{t^{\circ}t^{\circ}}$.

A solution to this problem is given by the circles in Figure A-1 indicating 7 clusters. Clusters 1 and 7 both have weights 20, so each will be presented in one chapter. The rest of the five clusters weights' do not add up to 20, therefore, they will be presented together in one chapter.

As to the lineup of the three chapers, it is discovered that the solution of $ODC_{t^{\circ}t^{\circ}}$ is fundamental to the rest of

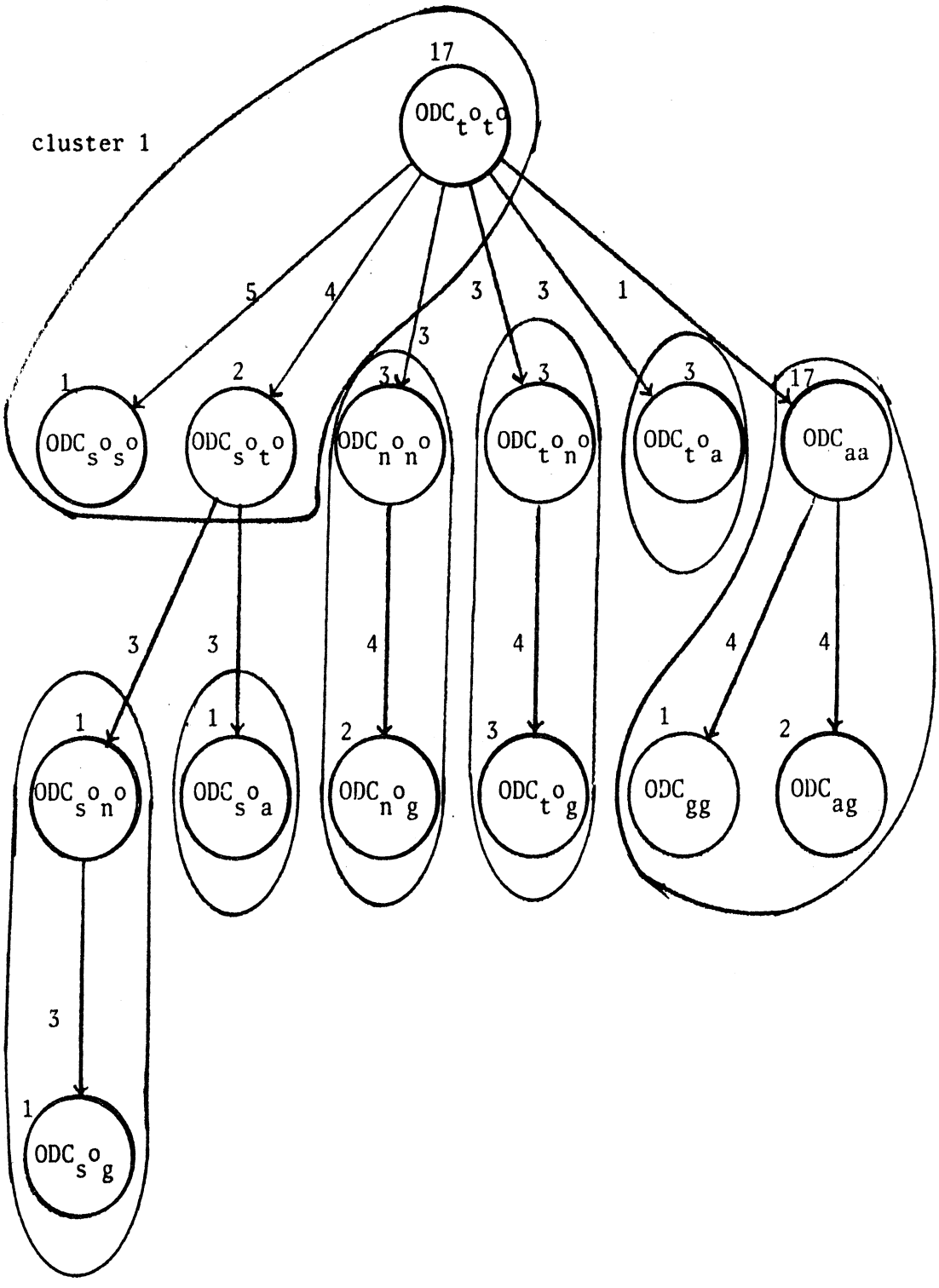


Figure A-1 A digraph representing the ODC breakup problem.

the fourteen problems, therefore, the solutions to the problems of cluster 1 in Figure A-1 are presented first (Chapter 2). The solutions to clusters 2 to 6 are then presented in the next chapter (Chapter 3) because they are problems that use directly the results of cluster 1. Problems in cluster 7 are more difficult and are therefore left to the last chapter (Chapter 4).

APPENDEX B

PROOF OF PROPOSITION 4.7

The number of steps taken by each type 2 stage is directly proportional to the number of triplets formed by step 1.2, which can be estimated by deciding:

1) the number of feasible triplets in $PK^f[v_j,od(v_j)]$;
2) the number of feasible triplets in $PK^f[v_r,i]$; and
3) the number of triplets formed by PUT, conditional CONNECT and conditinal COMBINE.

Let v_p be the parent of v_r. Let $n[r,j]$ be the number of neighbors of v_r in $V[v_j,od(v_j)]$, i.e., $n[r,j] = |L_{nb}[v_r] \cap V[v_j,od(v_j)]|$, and $n[p,r]$ be the number of neighbors of v_p in $G[v_r,i]$, i.e., $n[p,r] = |L_{nb}[v_p] \cap V[v_r,i]|$.

The set of feasible triplets in $PK^f[v_j,od(v_j)]$ can be partitioned into disjoint blocks by using the triplet-similar relation. Let PK_1 be one of the blocks induced by the equivalence relation. Since there are $n[r,j]$ neighbors, these neighbors can be distributed into no more than $n[r,j]$ distinct clusters. Any given cluster can assume a weight that varies from 1 to B because of the nonneighbors. Thus, the number of clusterings in PK_1 is no greater than $B^{n[r,j]}$ or $Z^{n[r,j]}$ if the vertex weight is greater than 1.

The upper bound on the number of blocks induced by the

triplet-similar relation is the number of ways in which n[r,j] distinct objects can be distributed in i nondistinct cells, where i varies from 1 to n[r,j]. Lukes ('75) showed this number to be smaller than n[r,j]!. Thus, the upper bound of the number of feasible triplets in $PK^f[v_j,od(v_j)]$ is $B^{n[r,j]} \cdot n[r,j]!$.

Similarly, the upper bound of the number of feasible clusterings in $PK^f[v_r,i]$ is $B^{n[p,r]} \cdot n[p,r]!$. An upper bound on the number of triplets formed by PUT is:

$$B^{n[p,r]} \cdot n[p,r]! \cdot B^{n[r,j]} \cdot n[r,j]!.$$

An upper bound on the number of triplets formed by conditional CONNECT and COMBINE is:

$$B^{n[p,r]} \cdot n[p,r]! \cdot (B-1) \cdot B^{n[r,j]-1} \cdot n[r,j]! \cdot n[r,j].$$

An upper bound on the number of triplets formed by PUT, conditional CONNECT and conditional COMBINE is:

$$B^{n[p,r]} \cdot n[p,r]! \cdot B^{n[r,j]} \cdot n[r,j]! \cdot (n[r,j]+1).$$

BIBLIOGRAPHY

BIBLIOGRAPHY

Aho, A.V. et al. The Design and Analysis of Computer Algorithms. Reading, Mass: Addison-Wesley; 1974.

Aho, A.V. et al. Data Structures and Algorithms. Reading, Mass: Addison-Wesley; 1983.

Belady, L.A. and Evangelisti, C.J. System Partitioning and Its Measure. The Journal of Systems and Software , Vol.2; 1981.

Bertolazzi, P. et al. Analysis of a Class of Graph Partitioning Problems. 18th Annual Allerton Conference on Communication, Control, and Computing; 1980.

Cardenas, A.F. Data Base Management Systems. Boston, Mass: Allyn and Bacon, Inc.; 1979.

Chen, P.P. et al. Survey of State-of-the-Art Logical Database Design Tools. Graduate School of Management, Univ. of California, Los Angeles; 1981.

Chiang, Wan P. and Teorey, Toby J. A Method for Database Record Clustering. Proceedings of the Conference on Information System; December 13-15, 1982: p. 57-73.

Chiang, Wan P. et al. Data Modeling with PSL/PSA: The View Integration System (VIS) (Preliminary Draft). Ann Arbor, Mich: ISDOS Inc.; 1983.

Comer, Douglas. The Ubiquitous B-Tree. Computing Surveys, Vol.11, No.2; June 1979.

Ferrari, Domenico. The Improvement of Program Behavior. Computer, Vol.9, No.11; 1976.

Garey, M.R. and Johnson, D.S. Computers and Intractability, a Guide to the Theory of NP-Completeness. San Francisco, Calif: Freeman; 1979.

Harary, F. Graph Theory. Reading, Mass: Addison-Wesley; 1969.

Horowitz, E. and Sahni, S. Fundamentals of Data Structures. Woodland Hills, Calif.: Computer Science Press,

Inc.; 1976.

Jenny, C.J. Placing Files and Processes in Distributed Systems: A General Method Considering Resources with Limited Capacity. IBM Research Report, RZ 1157 (41911). Yorktown Heights, N.Y., Zurich, Switzerland: IBM Research Div.; July 1982.

Johnson, D.S and Niemi, K.A. One Knapsacks, Partitions, and a New Dynamic Programming Technique for Trees. Math. Operation Res., Vol.8; 1983.

Lukes, J.A. Combinatorial Solutions to Partitioning Problems. Ph.D. Thesis, Stanford Univ., Calif; 1972.

Lukes, J.A. Efficient Algorithm for the Partitioning of Trees. IBM Journal of Res. Develop., Vol.18; 1974.

Lukes, J.A. Combinatorial Solution to the Partitioning of General Graphs. IBM Journal of Res. Develop., Vol.19; 1975.

March, Salvatore T. Techniques for Structuring Database Records. Computing Surveys, Vol.15, No.1; March 1983.

Perl, Yehoshua and Snir, Marc. Circuit Partitioning with Size and Connection Constraints. Networks, Vol.13; 1983.

Schkolnick, M. A Clustering Algorithm for Hierarchical Structures. ACM Trans. Database Syst., Vol.2, No.1; 1977.

Schrader, Rainer. Approximations to Clustering and Subgraph Problems on Trees. Discrete Applied Mathematics, Vol.6; 1983.

Teorey, T.J. and Fry, J.P. The Logical Record Access Approach to Database Design. ACM Computering Suveys, Vol.12, No.2; June 1980: p. 179-211.

Teorey, T.J. and Fry, J.P. Design of Database Structures. Englewood Cliffs, N.J.: Prentice-Hall; 1982.