DEGREE CONSTRAINED SUBGRAPHS OF LINEAR GRAPHS

Robert John Urquhart

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in the
University of Michigan
1967

Doctoral Committee:

Associate Professor Eugene L. Lawler, Chairman
Associate Professor Bruce W. Arden
Professor Harvey L. Garner
Associate Professor Arch W. Naylor
Professor Norman R. Scott

ACKNOWLEDGMENTS

I would like to express my gratitude and appreciation to Professors Bruce Arden, Harvey Garner, Eugene Lawler, Arch Naylor, and Norman Scott, for serving on my doctoral committee. Particular thanks is due to my committee chairman Dr. Lawler, for initially suggesting the topic, and for the time spent in subsequent discussions. Also I wish to thank Lee White for the many discussions we have had concerning this area of research.

I would like to thank IBM Corporation for their financial support through the IBM Fellowship Program, and also to express my appreciation to many friends at IBM who were responsible, in various ways, for my selection to this program.

My wife Ruth deserves special thanks for her patience and encouragement, and also for the typing of many rough drafts. Acknowledgment is also due to Karen Sue Meneghini for the final typing, and Suat Söylerkaya for his drafting efforts.

This work was supported in part by Air Force contract AF 30(602)-3546.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONT'D)

LIST OF FIGURES

LIST OF FIGURES (CONT'D)

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 Introduction

Many problems in engineering can be formulated as graphical problems. The purpose of this dissertation is to investigate and develop algorithms for a class of combinatorial problems on linear graphs. The approach taken is characterized by the use of linear programming theory. The classical transportation problem was first solved by a linear programming approach by Dantzig [5]. Kuhn [15], and Fulkerson [3] used similar techniques to solve the assignment and flow problems respectively.

All the above problems can be stated as linear programs and could be solved by the simplex algorithm of linear programming. The solutions obtained would be integers due to the fact that the constraint matrices have the unimodular property. The special structure of the transportation and flow problems allowed the development of special algorithms which are more efficient than the simplex algorithm.

The matching problem on a graph is to find a maximum subset of edges, such that no two edges are incident to the same vertex. This problem can be stated as an integer linear program. Its constraint matrix does not have the unimodular property, therefore noninteger solutions may be found if the simplex algorithm is used. Balinski [1] has shown that $\frac{1}{2}$ is the only noninteger solution possible for the matching problem with the integer requirement removed.

Tutte [22] and Berge [2] have studied the matching problem

and characterized maximum solutions, but neither suggested an efficient algorithm. In 1965 Edmonds [9] developed an algorithm for the matching problem which used a dual linear programming approach similar to Kuhn's algorithm for the assignment problem. Edmonds' work is particularly significant in that it is the first instance of an efficient solution to an integer linear programming problem whose constraint matrix does not have the unimodular property. The purpose of this dissertation is to develop efficient algorithms for a series of generalizations of the matching problems, which are called degree constrained subgraph problems.

All of the optimization problems discussed here are finite and thus enumeration would always find the optimum solution. But with increasing problem size, enumeration is not always practical. We would like to find algorithms which are "efficient" in some sense.

A rigorous definition of "efficient" would lead to an extensive discussion of the foundations of algorithm theory [7]. Edmonds developed the concept of efficiency based on asymptotic growth of complexity versus problem size. For graphical problems the number of vertices or edges is a good measure of size. A reasonable measure of complexity is the time required to find a solution on a particular computer. We can now ask:

(1) what is the most complex problem of a given size,

and,

(2) as size increases can the asymptotic behavior of the complexity be determined?

When it can, we have a useful measure of the relative efficiency of the algorithm.

Most problems of engineering interest have factorial ($n!$) exponential (2^n) or algebraic (n^k) growth. Edmonds shows that the matching algorithm grows as n^4 (algebraically) where n is the number of vertices of the graph. The best algorithms for the general covering problem and for the general integer programming problem grow exponentially. The shortest path algorithm (n^3) is an example of an algorithm which grows algebraically. The degree constrained subgraph algorithms developed in this dissertation have algebraic growth.

This concept of growth is not always meaningful for two reasons.

(1) Although no good bound on growth can be computed an algorithm may in practice always be very efficient. The prime example of this is the simplex algorithm which nearly always grows algebraically in complexity for increasing constraints or variables. The best theoretical bound for the simplex algorithm is exponential.

(2) The size at which the asymptotic behavior becomes evident may be beyond the useful range of the algorithm.

Nevertheless, this concept of algorithm efficiency is helpful in evaluating the useful range of an algorithm. For all practical purposes the difference between algebraic and exponential growth is more important than the difference between finite and infinite growth.

1.2 Definitions

This section will present notation and the definition of many of the concepts used throughout this dissertation.

A graph $G = (V,E)$ is a set of vertices V, and a set of edges E. Each edge e_{ij} in E is a set of two vertices (v_i, v_j).

Edge e_{ij} is said to be incident to vertices v_i and v_j.

A subgraph H of a graph $G = (V,E)$ is a set of vertices V_H and edges E_H such that:

(1) $V_H \subset V$ and,

(2) $E_H \subset \{e_{ij} | v_i \epsilon V_H, v_j \epsilon V_H\}$

Often we will define a subgraph as a set of edges. In this case the vertices of the subgraph are all vertices of the original graph. If S is a subset of the vertices of a graph G, then S(G) is the subgraph consisting of the vertices of S and all edges of G with both vertices in S. G-S is the subgraph consisting of all edges with neither vertex in S.

A weighted graph (G,c) is a graph with a real number c_{ij} associated with each edge e_{ij}.

A path in a graph is a sequence of edges $\{e_{12}, e_{23}, \cdots e_{n-1, n}\}$ such that consecutive edges in the sequence have a common vertex. An edge can occur only once in a given path.

The vertex of the first edge in the path P, which is not in common with the second edge, is called the initial vertex of path P. The vertex of the last edge, which is not in common with the next to the last edge, is called the final vertex. If the initial and final vertices of a path are identical then the path is a cycle.

A vertex of the path which is neither the initial nor final vertex is called an intermediate vertex.

The degree of a vertex v_i relative to a graph (subgraph) is the number of edges of the graph (subgraph) to which v_i is incident.

The degree of v_i will be denoted by $d(i/G)$, where G may be a graph, subgraph, or set of edges.

A simple path is a path for which:

(1) The degree of the initial and final vertices relative to the path is exactly one.

(2) The degree of each intermediate vertex relative to the path is exactly two.

A circuit is a path for which all vertices have degree two relative to the path.

A connected subgraph is a subgraph for which a path exists between each pair of vertices of the subgraph. A maximal connected subgraph is called a component. A connected graph is a graph with only one component.

A bipartite graph is a graph which contains no circuits that have an odd number of edges. Another characterization of a bipartite graph is that the vertices of the graph can be partitioned into two sets V_1 and V_2, such that every edge of G has one end in V_1, and one end in V_2.

The incidence matrix A of a graph is defined as,

$A_{k(ij)} = 1$ if edge e_{ij} is incident to vertex k.

$A_{k(ij)} = 0$ otherwise.

A matching of a graph G is a subset M of the edges of G such that at most one edge of M is incident to any vertex.

An upper degree constrained subgraph (UDCS) relative to a vector d, is a subset D of the edges of G such that at most d_i edges of D are incident to vertex v_i.

An upper and lower degree constrained subgraph (ULDCS) relative to vectors d^1 and d^2 is a set of edges D such that the degree of v_i relative to D is between d_i^1 and d_i^2.

A d-factor is a subset of edges D such that the degree of v_i relative to D is exactly d_i.

A multiple degree constrained subgraph (MDCS) relative to a vector d, is a set of edges D, and a set of integers x, where x_{ij} is associated with edge e_{ij}, such that $\sum_{\forall_j \ni e_{ij} \in D} x_{ij} \leq d_i$.

A matrix has the unimodular property if each minor has value 0 or ± 1. The incidence matrix of a bipartite graph has the unimodular property.

The cardinality matching problem is to find a matching which has the maximum number of edges. The cardinality UDCS, factor, ULDCS, and MDCS problems are defined similarly. The weighted matching problem is to find a matching whose edges have the maximum total weight. The weighted UDCS, factor, ULDCS and MDCS problems are defined similarly.

A vector which consists of all ones will be denoted by ϵ. The dimension of ϵ will always be clear from context.

The sum of the components of an integer valued vector x is denoted by $|x|$. If x is a $\{0,1\}$ vector $|x|$ is the number of ones in x. If A is a set, $|A|$ will also denote the number of elements in the set.

1.3 Summary of Thesis

Algorithms, theoretical characterization, and practical applications of a series of generalizations of the matching problem are

presented in this thesis. The contents of each chapter is summarized below.

(1) The solution to the matching problem, as developed by Edmonds, is presented in Chapter 2. Also an alternate algorithm for matching is discussed. This algorithm was programmed to gain computational experience.

(2) In Chapter 3, the solution of the upper degree constrained subgraph (UDCS) problem is characterized and an algorithm is developed to find such a solution.

(3) An algorithm for the factors problem is presented in Chapter 4, and a relationship is shown between this solution and previous theoretical work of Tutte. Although characterization of this problem has been known for some time, this is the first efficient algorithm which has been described for its solution.

(4) The upper and lower degree constrained subgraph (ULDCS) problem is discussed in Chapter 5. This problem contains the UDCS, matching, assignment and many other combinatorial problems as special cases. The interrelation between all of these problems is discussed in this chapter.

(5) Chapter 6 discusses the multiple degree constained subgraph (MDCS) problem. This differs from all the previous problems in that an edge in the solution subgraph may be used more than once in satisfying the degree constraints. The MDCS problem is shown to be equivalent to the matching problem, and this equivalence is used to develop an algorithm.

(6) Chapter 7 discusses theoretical and engineering application for the previously described results. These results are applicable to parallel processing, optimum routing, optimum system design and dynamic scheduling of resources. Further areas of research are also suggested in this chapter.

(7) Appendix A contains a listing of the matching program and presents the computational experience which has been obtained.

(8) Appendix B summarizes the linear programming results which are used in this thesis.

(9) Appendix C presents algorithms for the Chinese Postman's Problem and for the detection of a negative cycle in a weighted graph.

CHAPTER II

MATCHING

2.1 Introduction

The cardinality matching and weighted matching problems have been solved by Edmonds [9,10]. This chapter will characterize solutions of the matching problem and describe the algorithms which find these solutions. Section 2.2 describes the cardinality matching algorithm developed by Edmonds, and also describes an alternate algorithm for which a computer problem was developed. Appendix A gives the computer program and the computational experience with that program. Section 2.3 discusses the weighted matching problem and describes the algorithm Edmonds developed for this problem. In the solution of these problems much attention must be given to particular subgraphs which will be called blossoms. In section 2.4 blossoms are discussed. Blossoms are closely related to the existence of noninteger solutions. A proof that the weighted matching algorithm achieves the optimum solution is given in section 2.5. Finally, section 2.6 contains a geometric interpretation of augmenting paths in terms of convex polyhedra, and also suggests an algorithm which automatically finds optimum k-cardinality matchings on the way to finding the maximum weighted matching.

2.2 Cardinality Matching

This section presents the cardinality matching algorithm of Edmonds with some minor changes, having to do with the method of treating

blossoms. The viewpoint taken here will lead to a significant improvement in proving optimality of the weighted matching algorithm in section 2.5. In order to discuss matching, several definitions will be required.

The following definitions are with reference to a given matching M. If no edge of M is incident to vertex v_i, then v_i is an exposed vertex. An alternating path is a path whose edges are alternately in M and not in M.

Definition 2.2.1

An augmenting path is an alternating path between two exposed vertices.

$M \oplus P$, where M and P are sets, is the symmetric difference of M and P (ie. $M \oplus P = (M - P) \cup (P - M)$). In order to express the exchanging of the role of matched and unmatched edges along an alternating path P, the symmetric difference notation is used. Alternating paths were first studied by Peterson [19]. Berge [2] characterized maximum cardinality matching by the use of augmenting paths.

Theorem 2.2.1 (Berge)

A matching M is not of maximum cardinality if and only if there exists an augmenting path between two exposed vertices.

Proof:

Suppose there exist an augmenting path P. Form $M^1 = M \oplus P$. It is clear that M^1 is a matching and that the cardinality of M^1 is greater than that of M, therefore the "if" part of the theorem is shown.

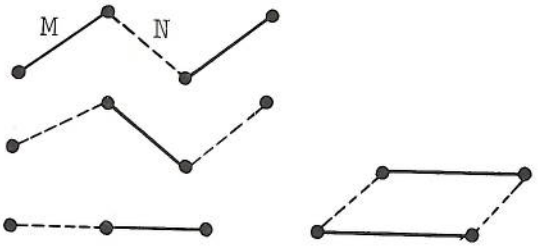
Suppose M is not a maximum cardinality matching, and let N be any maximum cardinality matching. Examine the connected components of $M \oplus N$. It is not difficult to see that each component of $M \oplus N$ is an

alternating path. Three types of paths are possible (Figure 2.2.1).

Type 1 paths

Type 2 paths

Type 3 paths



Matching Augmenting Paths

Figure 2.2.1

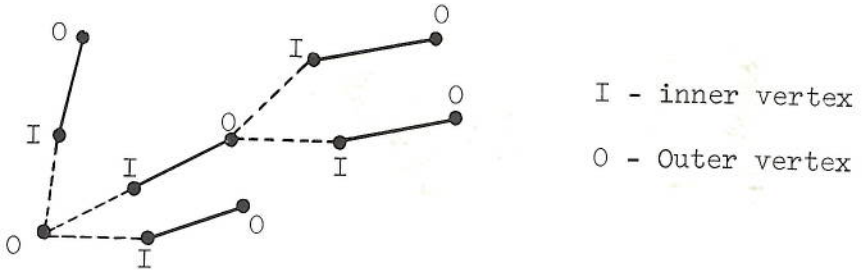
Since $|N| > |M|$ at least one of the components must have more edges of N than M, and hence that component is a type 2 path. But this demonstrates the existence of an augmenting path between two vertices exposed relative to M.

An alternating tree (Figure 2.2.2) relative to a given matching is a tree with the added conditions that:

(1) exactly one vertex of the tree is exposed. This is called the root of the tree.

(2) All paths from the root are alternating paths, and

(3) all maximal paths from the root are of even length.



Alternating Tree

Figure 2.2.2

An inner vertex of an alternating tree is a vertex which is at the end of a path of odd length from the root. An outer vertex of an alternating tree is a vertex which is at the end of a path of even length from the root. The degree of each inner vertex is exactly two while the degree of an outer vertex may be one or more. The number of outer vertices is always one more than the number of inner vertices, and the number of matched edges in the tree is equal to the number of inner vertices. An external vertex is a vertex which is not in the tree.

An augmenting tree is an alternating tree with one additional edge e_{12} from an outer vertex v_1 of the tree to some exposed vertex v_2 not in the tree. This edge uniquely determines an augmenting path from v_2 to the root.

A simple blossom B with respect to a given matching M is a circuit consisting of $2r + 1$ edges and $2r + 1$ vertices, where r of the edges are in M. Thus there is one vertex of the simple blossom which is exposed relative to $M \cap B$.

When a simple blossom is found in the matching algorithm it is shrunk to a simple vertex. By shrinking simple blossoms in this manner the concept of an alternating tree can be preserved in the resulting graph.

As a consequence of several such shrinkings a vertex v_k in the resulting graph may be the image of several simple blossoms from previous graphs. If so, the preimage of v_k will be some odd set of vertices S_k in the original graph. The vertices S_k along with all

edges with both vertices in S_k, will be called a blossom.

A blossom which is not contained within some other blossom will be called an outermost blossom. These blossoms will be particularly important in the weighted matching algorithm discussed in section 2.3. A discussion of the structure of blossoms will be given in section 2.4.

A blossomed tree is an alternating tree with one additional edge e_{12} which joins two outer vertices of the tree. Let P_1 be the path from v_1 to the root v_r, and P_2 be the path from v_2 to v_r. Then $(P_1 \oplus P_2) \cup e_{12}$ is a blossom.

A Hungarian tree is an alternating tree for which any edge incident to an outer vertex is also incident to another vertex of the tree.

The cardinality matching algorithm is initiated by specifying an initial matching, possibly empty, and proceeds by systematically searching for augmenting paths.

Trees rooted at an exposed vertex are grown by alternately adding unmatched and matched edges until the tree either

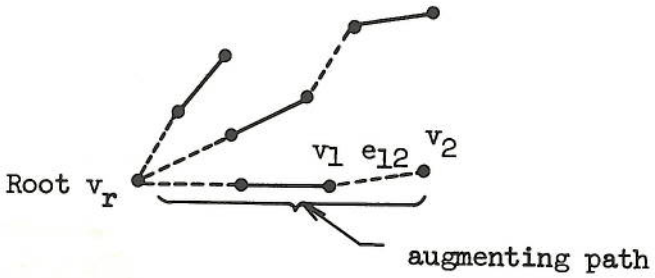
(1) augments,

(2) blossoms, or

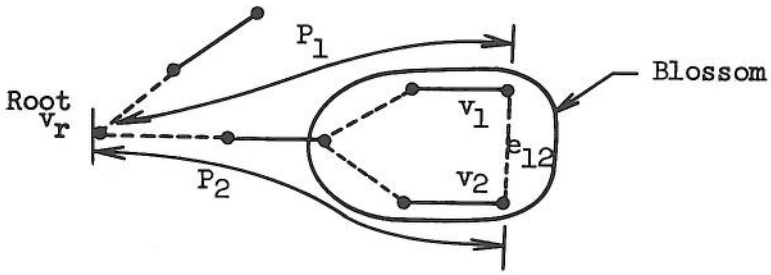
(3) becomes Hungarian.

See Figure 2.2.3 for examples of each of the above conditions, and Figure 2.2.4 for a flow diagram of the algorithm.

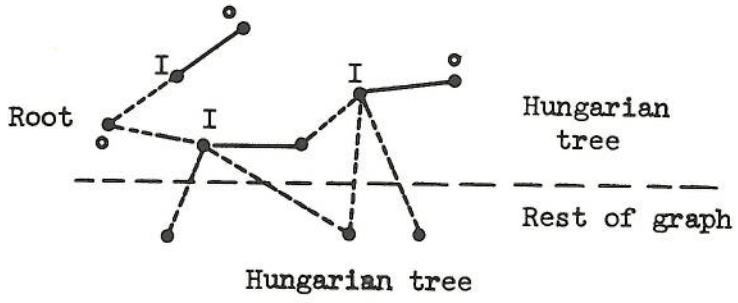
If the tree augments, the cardinality of the matching can be increased by one by tracing the augmenting path back to the root, and then interchanging the role of matched and unmatched edges along that path. The tree is discarded following augmentation and a new tree



Augmenting tree

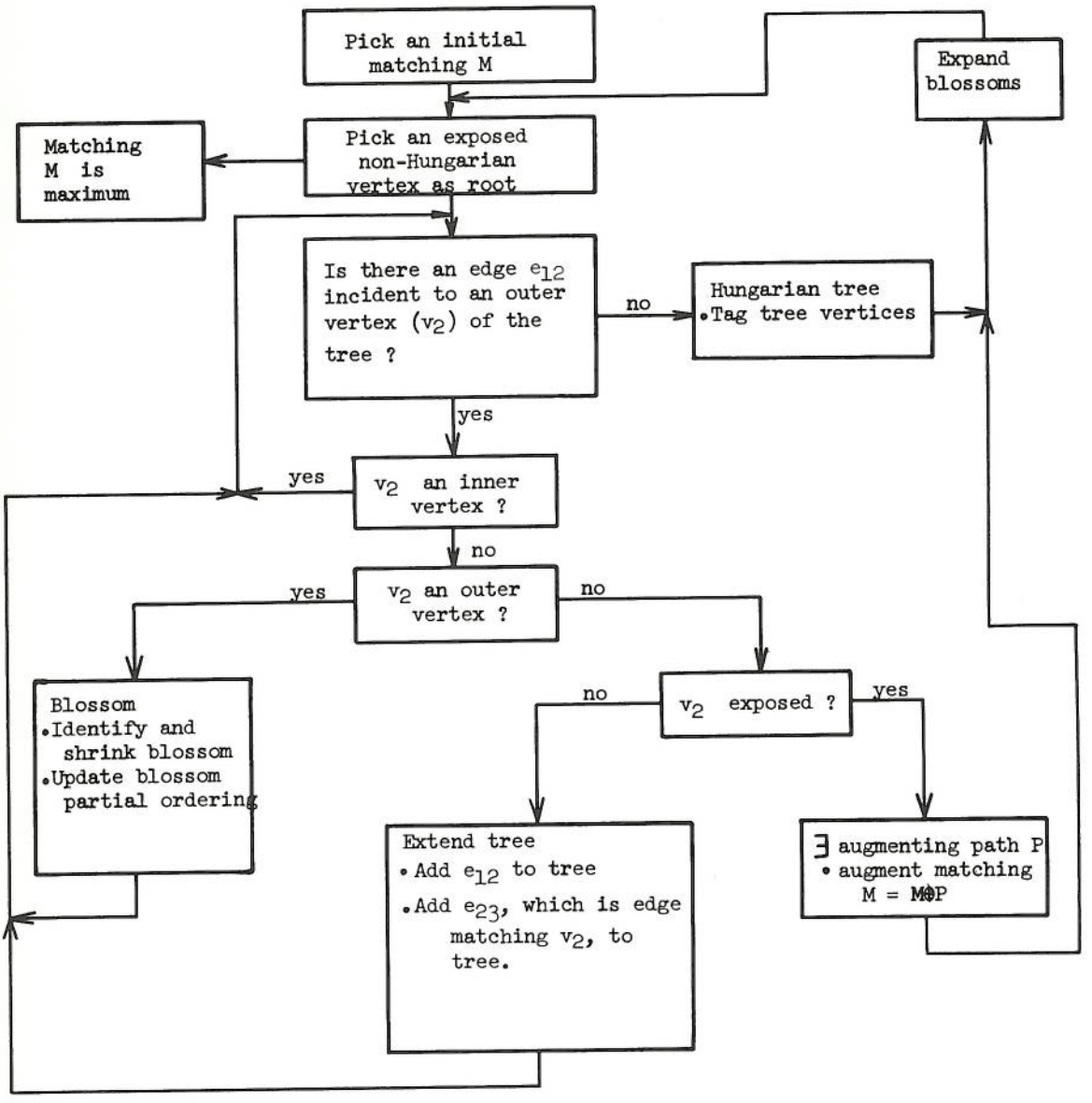


Blossomed tree



Matching Trees

Figure 2.2.3



Cardinality Matching Algorithm

Figure 2.2.4

begins at some remaining exposed vertex.

If the tree blossoms, the resulting blossom is identified by backtracing the two paths to the root. The blossom is then shrunk, the tree is retained and further search for an augmenting path continues.

If the tree becomes Hungarian then the maximum matching of this tree is independent of the remaining graph. This fact is proved in Theorem 2.2.2 below.

The overall strategy of the algorithm is to either successively find augmenting paths, or to find Hungarian trees which can be eliminated from further consideration. Since each Hungarian tree eliminates one exposed vertex and each augmentation eliminates two exposed vertices, we can conclude that fewer than n trees, where n is the number of vertices of the graph, will have to be grown. Edmonds computes an asymptotic upper bound on the growth of computation time with the number of nodes as n^4. This is arrived at as follows:

(1) Fewer than n trees are grown.

(2) The maximum number of edges in a tree increases linearly with n.

(3) Each tree requires backtracing paths to identify blossoms and augmenting paths. The length of these paths grows linearly with n.

(4) Finally the search for an edge incident to a given vertex grows linearly with n.

The above operations are independent and thus the overall algorithm grows as n^4.

The results of a more detailed growth analysis of this program are shown in Figure 2.2.5. The program grows as n^4 only in one area, the expansion of blossoms. There is one operation within the expand routine which requires searching for a vertex v_1 in one blossom B, which is incident to a different blossom. This operation grows as n^2 and may be repeated n times per tree. Since there are n trees this results in n^4 growth. The computation experience in Appendix A shows a growth of less than n^3. This is probably due to a low coefficient on the one critical area. n^4 growth would not be apparent until n became much larger.

The following theorem proves that when a Hungarian tree forms relative to a particular matching M and exposed vertex v, the entire tree can be removed from further consideration without affecting the overall maximum matching.

Theorem 2.2.2

If J is a Hungarian tree in graph G, with matching M_J and M_{G-J} is any maximum matching of G-J, then

$M_J \cup M_{G-J}$ is a maximum matching of G.

Proof

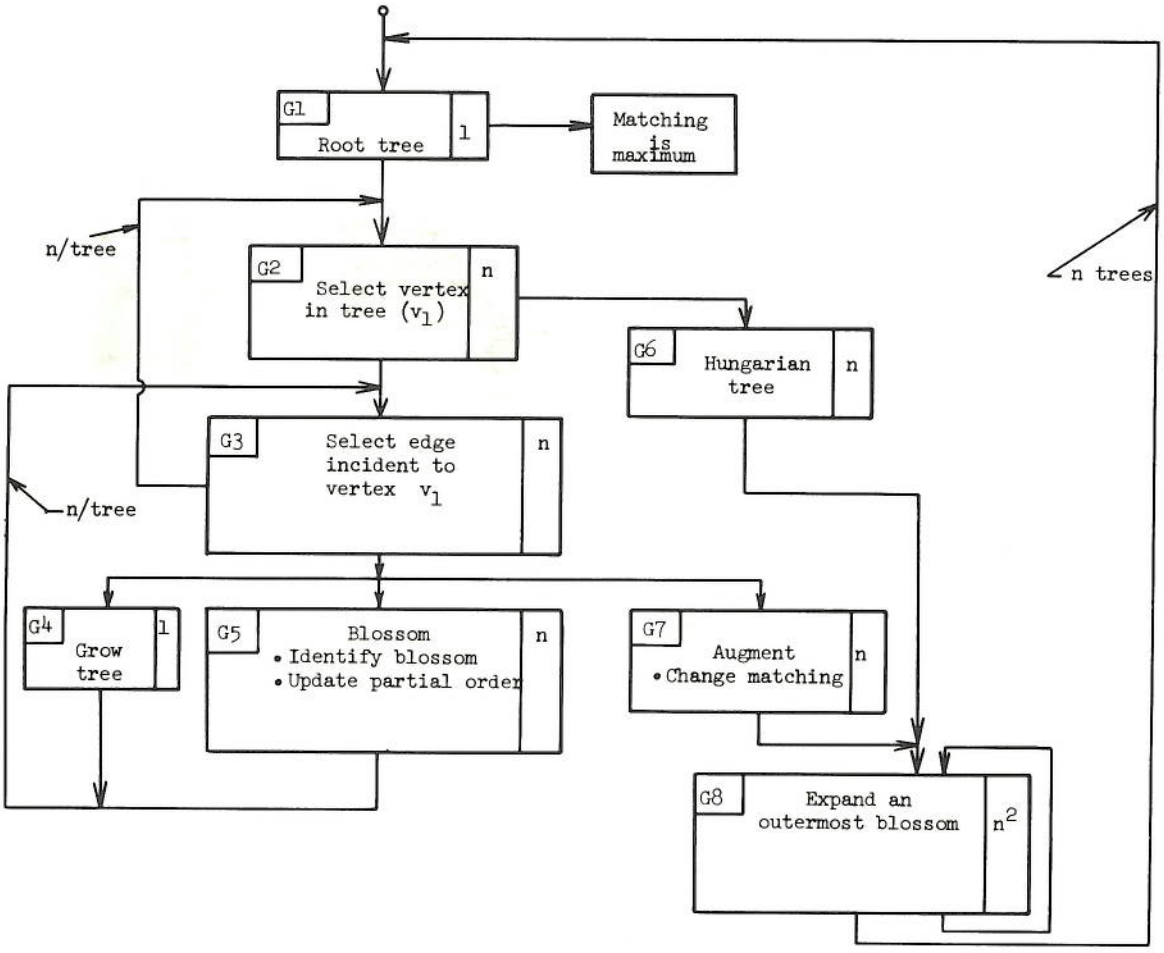
If J is a Hungarian tree, we partition the edges of G into three sets relative to V_J, the edge set of J.

(1) E_J, the edges with both ends in V_J.

(2) $E_{J\bar{J}}$, the edges with exactly one end in V_J.

(3) E_{G-J}, the edges with neither end in V_J.

Let N be any matching of G, then N can be partitioned relative to the edge partition above into,



Function	Function Growth	No. of Times Executed	Total Growth
G1	1	n	n
G2	n	n^2	n^3
G3	n	n^2	n^3
G4	1	n^2	n^2
G5	n	n^2	n^3
G6	n	n	n^2
G7	n	n	n^2
G8	n^2	n^2	n^4

Matching Algorithm Growth

Figure 2.2.5

$$N = N_J \cup N_{J\bar{J}} \cup N_{G-J}.$$

(1) $|M_{G-J}| > |N_{G-J}|$,

since M_{G-J} is a maximum matching of G-J.

Each edge in $N_{J\bar{J}}$ touches exactly one inner vertex of J by the definition of a Hungarian tree. Let $I(J)$ be the number of inner vertices of J. The removal of m inner vertices from the alternating tree J breaks it into $m + 1$ disjoint trees with a total of $I(J) - m$ inner vertices.

Since the cardinality of a maximum matching on an alternating tree is equal to the number of inner vertices we have,

$$I(J) - |N_{J\bar{J}}| > |N_J|, \text{ or}$$

(2) $|M_J| = I(J) \geq |N_J| + |N_{J\bar{J}}|$.

Combining (1) and (2) we have $|M| \geq |N|$, but since N is an arbitrary matching we conclude M is maximum.

2.3 Weighted Matching

The weighted matching problem on a graph G can be stated as an integer program as follows:

(1) max cx (=W)

subject to:

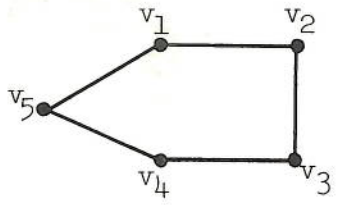
(2) $Ax \leq \epsilon$

(3) $x_{ij} \epsilon \{0,1\}$,

where c is a vector of edge weights of the graph and x is a $\{0,1\}$ vector. $x_{ij} = 1$ if and only if edge e_{ij} is in the matching. A is the incidence matrix of the graph G.

The removal of the explicit integer constraints will result in an ordinary linear program which may have noninteger solutions. The sources of noninteger solutions are the odd cycles of the graph. For example, consider the following graph with all edge weights equal to one.

Example 2.3.1



The maximum solution without the integer constraint is $x_{12} = x_{23} = x_{34} = x_{45} = x_{51} = .5$ which has weight 2.5, while the maximum integer matching is clearly 2.

If the graph contains no odd cycles then it is a bipartite graph, and the matching problem reduces to the classical assignment problem. Efficient solutions for this problem are well known [15].

Edmonds [10] found that the following set of linear constraints were sufficient to guarantee integrality. For every subset of $2r + 1$ vertices there are r or fewer matched edges with both ends in the subset. This then leads to a linear program, without explicit integrality constraints, whose associated constraint polyhedron has only integer vertices. The linear program is:

(4) max cx

subject to:

(5) $Ax \leq \epsilon$

(6) $Rx \leq r$

(7) $x \geq 0$.

R is the "odd set of vertices" versus edge incidence matrix of the graph G, and r is a vector of integers. Associated with each odd set of vertices S_k ($|S_k| = 2r_k + 1$), there is an equation in (6) of the form $R_k x \leq r_k$. R_k is a $\{0,1\}$ edge vector where $R_{k(ij)} = 1$ if and only if edge e_{ij} has both endpoints in S_k. R_k will also be used to denote the set of all edges with both ends in S_k. It is easily seen that any integer matching satisfies (6). What is not clear is whether these constraints are sufficient to guarantee integrality. This proof will be given in section 2.5.

The linear programming notation and results used here are contained in Appendix B. Considering (4), (5), (6), and (7), as a primal linear program, we obtain the corresponding dual linear program below:

(8) min $\epsilon y + rz$

subject to:

(9) $A^T y + R^T z \geq c$

(10) $y \geq 0$, $z \geq 0$.

y_i is the dual variable associated with vertex v_i, and z_k is the dual variable associated with the odd set of vertices S_k. A typical dual equation is,

(11) $y_1 + y_2 + \sum_{e_{12} \epsilon R_k} z_k \geq c_{12}$.

Duality theory of linear programming states that,

(12) $W = \max cx = \min (\epsilon y + rz) = U$ when the extrema exist,

and further,

(13) $W \leq U$,

for all feasible x, and feasible y, z. Thus if feasible solutions to both the primal and dual linear programs can be found such that $U = W$, then these solutions are necessarily optimum.

The algorithm is motivated by the orthogonality conditions of linear programming. These conditions are essentially that:

a) if a primal variable is strictly positive then the associated dual equation is satisfied with equality, and

b) if a dual variable is strictly positive then the associated primal equation is satisfied with equality.

The orthogonality conditions for this pair of linear programs are:

(14) $x_{ij} > 0 \Rightarrow y_i + y_j + \sum_{e_{ij} \in R_k} z_k = c_{ij}$.

(15) $y_i > 0 \Rightarrow A_i x = 1$. That is, vertex v_i is not exposed.

(16) $z_k > 0 \Rightarrow R_k x = r_k$. That is, an odd set of vertices has a nonzero dual variable only if it is maximally matched with edges of R_k. Notice that a blossom satisfies one of the $R_k x \leq r_k$ constraints with equality.

If there exist feasible solutions to both the primal and dual linear programs which also satisfy the orthogonality conditions, then these solutions are also optimal. The algorithm starts with and maintains feasibility and orthogonality except for condition (15).

A feasible primal is the matching with no edges. This also satisfies (14) initially. A feasible dual is obtained by letting

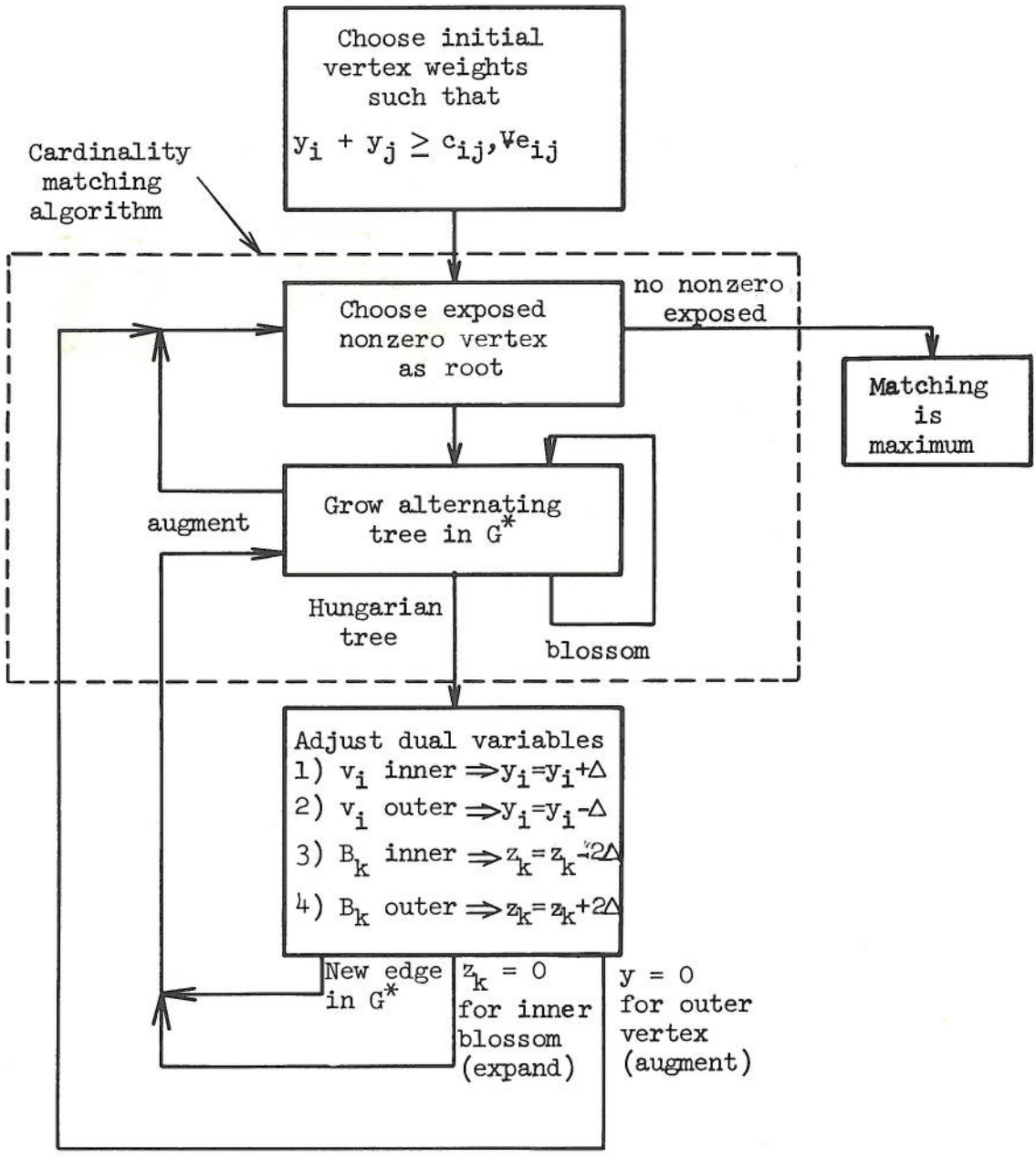
$z_k = 0$ for all odd sets of vertices, and by choosing any y variables which are large enough to satisfy $y_i + y_j \geq c_{ij}$, for all edges. Since $z = 0$ (16) is satisfied, and thus (15) is the only orthogonality condition which is not satisfied initially. We will describe a procedure in which every time a tree is grown the number of vertices violating (15) is reduced. Thus fewer than n trees are required.

The algorithm (Figure 2.3.1) is based upon the primal and dual linear programs with integrality constraints as formulated above. The algorithm is a generalization of the Hungarian method used by Kuhn [15], to solve the assignment problem. Initial values of the dual variables y_i are chosen sufficiently high to satisfy,

$$y_i + y_j \geq c_{ij}, \ \forall e_{ij} \epsilon G.$$

A restricted primal problem is induced by this selection of y variables. Only edges of G, which are not totally within some blossom, and for which $y_i + y_j = c_{ij}$, are used to grow trees. This defines a subgraph which is called G*.

The cardinality matching algorithm described in section 2.2 is used to grow an alternating tree rooted at a nonzero exposed vertex. A succession of blossoms may form, at which time the blossoms are shrunk and the vertices of the blossom are identified as outer. If a cardinality augment occurs, it is used to augment the matching and the tree is discarded. This is called a strong augment since both the cardinality and the weight of the solution is increased. Each time the tree blossoms, the number of vertices which are classified as outer increases, and each time edges are added to the tree, the number



Weighted Matching Algorithm

Figure 2.3.1

of vertices remaining out of the tree is decreased. Eventually either an augment occurs, or the tree cannot be extended, in which case the tree is called Hungarian.

At this time we leave the tree growing phase, and make a uniform change Δ in the dual variables. The procedure of computing and applying this change to the dual variables is called the Hungarian process. The following changes are made:

(1) If v_i is an outer vertex, y_i is lower by Δ.

(2) If v_i is an inner vertex, y_i is raised by Δ.

(3) If the blossom B_k is an outermost blossom of the tree and it is classified as an outer vertex, then the associated z_k variable is increased by 2Δ.

(4) If the blossom B_k is an outermost blossom of the tree and it is classified as an inner vertex, then the associated z_k variable is decreased by 2Δ.

There are 3 conditions which limit the value of Δ:

(1) The raising of the y variable for outer vertices of the tree can cause a new edge to enter G^*. The value of Δ defined by this condition is,

$$\min_{e_{ij} \in E_1} \{c_{ij} - y_i - y_j\}.$$

E_1 is the set of all edges which have one endpoint which is an outer vertex, and one endpoint which is not in the tree. Having brought this new edge into G^*, we return to the cardinality algorithm to search for further tree growth.

(2) y_i becomes zero for some outer vertex. This outer vertex may be the root of the tree in which case we have clearly reduced the number of vertices violating (15). If an outer vertex v_1, other than the root, becomes zero, we change the matching along the even path between v_1 and the root. This operation is called a weak augment since the cardinality of the matching is unchanged. This results in the root becoming matched and thus it now satisfies (15). However, the vertex v_1 becomes exposed. But since $y_1 = 0$, v_1 still satisfies (15), and in this case also we have reduced the number of vertices violating (15).

(3) A blossom must be expanded. The net amount that an outermost blossom B_k is adjusted downward is $\frac{1}{2} z_k$. The value of z_k is increased initially while it is an outer vertex. Following an augment this blossom may become an inner vertex of a subsequent tree which may reduce the value of z_k. Since z_k can never be allowed to go negative by (10), the minimum value of z_k for these type blossoms limits the maximum value of Δ. If the maximum value of Δ is determined by this condition, then after adjusting the dual variable as described, the appropriate blossom is expanded.

After n or fewer occurrences of steps 1 and 3, either a strong augment or step 2 above will occur. Thus each time a tree is rooted the

number of vertices violating (15) is reduced. Therefore after fewer than n trees, the orthogonality conditions are all satisfied, and the solution found is optimum. The various states of a vertex, and transitions between states which can occur during the algorithm, are shown in Figure 2.3.2.

The concept of an augmenting path which was introduced for the cardinality problem can be extended to the weighted matching problem. The weight $w(M)$ of a matching M is the sum of the edge weights of M.

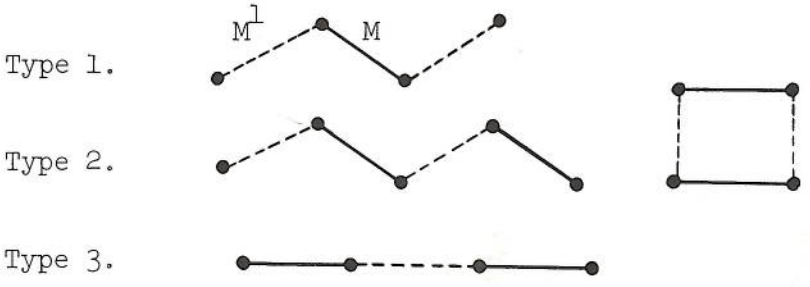
Definition 2.3.1

A weighted augmenting path P relative to a weighted graph G and a matching M is an alternating path such that:

(1) $M^1 = M \oplus P$ is a matching, and

(2) $w(M^1) \geq w(M)$.

The above definition leads to three different types of paths (Figure 2.3.3).



Weighted Augmenting Paths

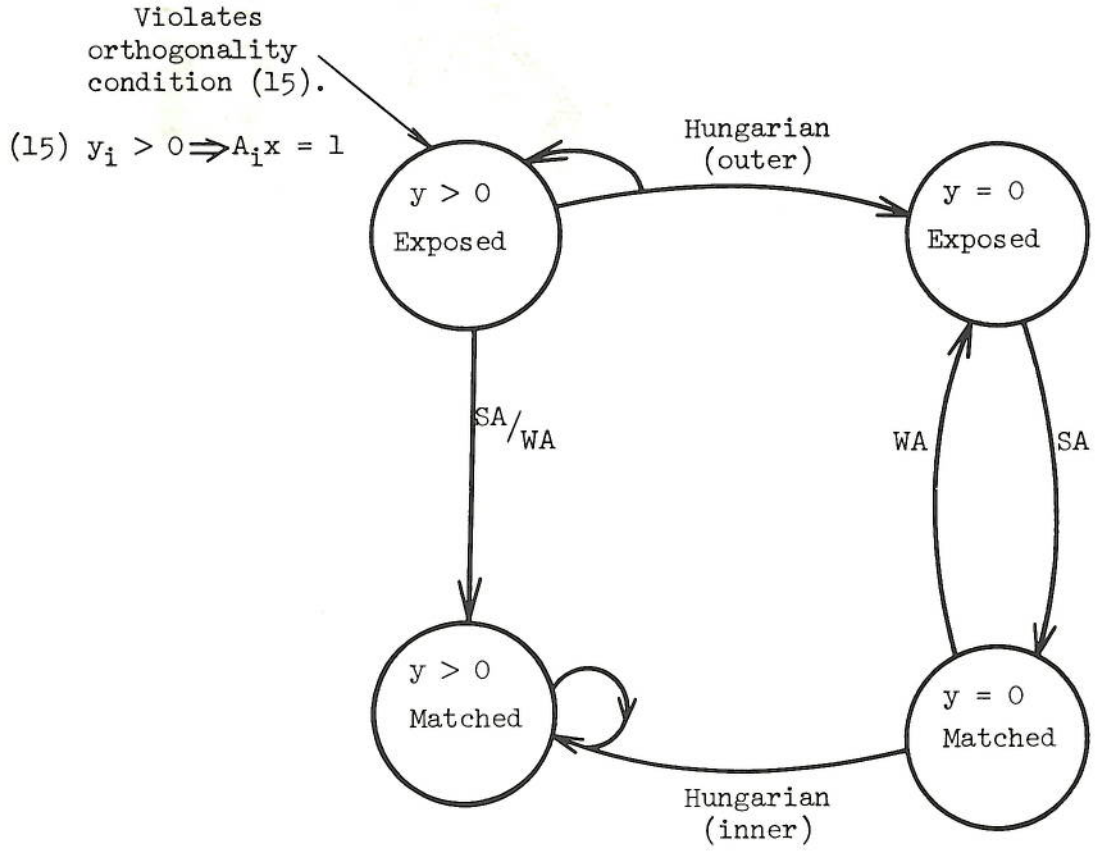
Figure 2.3.3

Theorem 2.3.1

Given a weighted graph G, a matching M is not of maximum

SA - Strong augment

WA - Weak augment



Vertex State Transition Diagram

(Weighted Matching)

Figure 2.3.2.

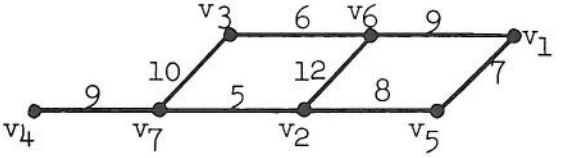
weight if and only if there exists a weighted augmenting path.

Proof

If a path P, satisfying Definition 2.3.1, exists then form $M^1 = M \oplus P$. From the definition $w(M^1) > w(M)$. Now suppose M is not maximum. Then there exists some maximum matching N such that $w(N) > w(M)$. Consider the symmetric difference of M and N. As in the cardinality case it can be seen that each component of $M \oplus N$ is an alternating path. But since $w(N) > w(M)$, there exists at least one component P_1 such that $w(N \cap P_1) > w(M \cap P_1)$. Thus the alternating path P_1 satisfies the conditions of the definition and the theorem is proved.

Example 2.3.2

Find the maximum matching of the graph G shown below.



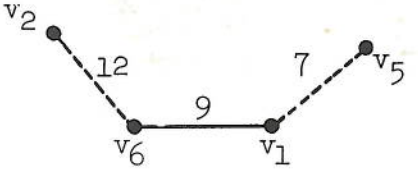
A valid set of initial vertex weights is $y_i = 6$, for all v_i.

Tree 1

(1) Root at v_1 → Tree is Hungarian.

(2) Lower y_1 to 3 → Grow e_{16} → Augment, $M_1 = \{e_{16}\}$.

Tree 2

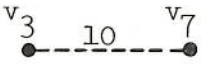


(1) Root at v_2.

(2) Grow e_{26} and e_{61} → Tree is Hungarian.

(3) Lower y_2 and y_1 by 2
Raise y_6 by 2 } → Grow e_{15} → Augment, $M_2 = \{e_{26}, e_{15}\}$.

Tree 3



(1) Root at v_3 → Tree is Hungarian.

(2) Lower y_2 by 2 → Grow e_{37} → Augment, $M_3 = \{e_{15}, e_{26}, e_{37}\}$.

Tree 4

(1) Root at v_4

(2) Lower y_4 by 3 → Grow e_{47}, e_{37} → Tree is Hungarian.

Lower y_4 and y_3 by 3
Raise y_7 by 3
} → $y_4 = 0$.

The orthogonality conditions are now satisfied. As a further check we can compute $U = \epsilon y + rz$, and $W = cx$.

$$W = cx = (x_{15}, x_{26}, x_{37}) \quad (7, 12, 10) = 29$$

$$U = \epsilon y + rz = (1 + 8 + 1 + 0 + 6 + 4 + 9) = 29$$

2.4 Matching Blossoms

A blossom is a set of vertices and edges which satisfy a primal integrality constraint with equality (i.e., $R_k x = r_k$). It has already been shown that odd length simple closed paths lead to noninteger vertices of the constraint polyhedon specified by:

$$Ax \leq \epsilon$$

$$x \geq 0.$$

When blossoms are detected in the algorithm, Edmonds shrinks them to a single vertex. This is a very convenient notation, in terms of retaining the alternating tree structure.

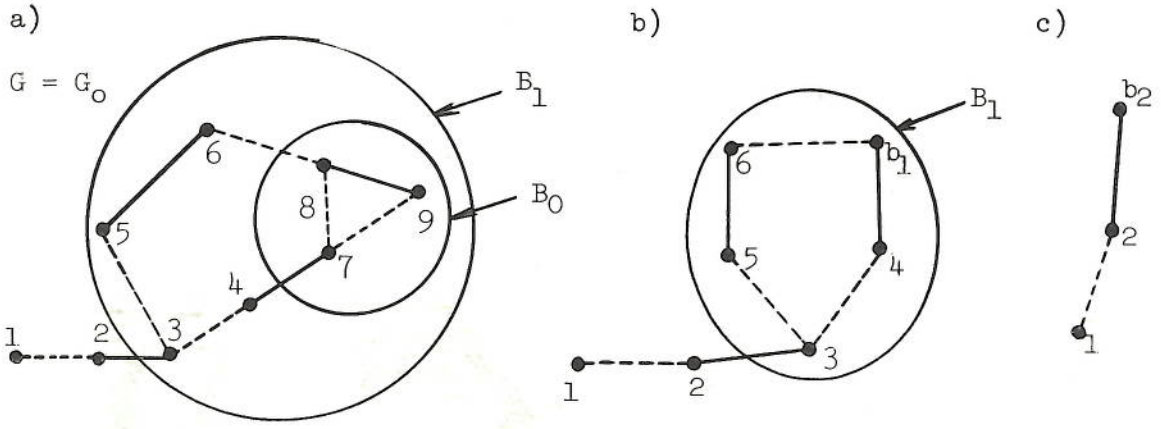
Notice that by our definition of inner and outer vertices, each vertex within a blossom is both inner and outer. We can ignore the inner interpretation for now, but later when we discuss degree constrained subgraphs this interpretation will be very important.

Edmonds suggests shrinking the vertices of a blossom to a single vertex, creating another graph. The same result can be obtained by identifying which vertices are in a particular blossom, classifying them all as outer vertices, and continuing to look for tree growth from edges of the original graph. Thus the vertices of the blossom remain distinct. Edges between vertices of the same blossom are ignored in subsequent growth.

We will always talk of shrinking blossoms to a single vertex, thereby creating a new graph. That is, when the first blossom B_0 forms, we say B_0 is shrunk to b_1 forming graph G_1 etc. We will not actually form G_1 but rather tag the vertices of B_0 with the label b_1 and continue to use edges of $G_0 = G$ for subsequent growth.

After a blossom B_0 has been formed and shrunk, another edge may be found which is incident to both an outer vertex v_1 of the tree and some vertex v_2 within B_0. This forms another blossom B_1 which contains all the vertices of B_0 along with other vertices.

An example of one blossom being contained within another blossom is shown in Figure 2.4.1. Figure 2.4.1.a) depicts the blossoms as set of vertices in the original graph, while Figures 2.4.1.b) and c) show the effect of shrinking blossoms as they occur.



$S_o = \{7, 8, 9\}$

$S_1 = \{3, 4, 5, 6, 7, 8, 9\}$

$HP_o = \{e_{78}, e_{89}, e_{79}\}$

$HP_1 = \{e_{34}, e_{4b_1}, e_{b_1 6}, e_{65}, e_{53}\}$

Matching Blossoms

Figure 2.4.1

Assume that edge e_{68} was the last edge added in forming B_1. Notice vertex v_5 which was an inner vertex prior to the adding of edge e_{68} is now an outer vertex. The even length path P from v_5 to v_1 is:

$$P = \{e_{56}, e_{68}, e_{89}, e_{97}, e_{74}, e_{43}, e_{32}, e_{21}\}.$$

In order to find the even length path back to the root from any vertex within a blossom B_k certain information is required about the structure of the tree when that blossom formed. It is sufficient to remember the circuit HP_k associated with each blossom B_k.

Notice that each subsequent blossom B_k is either disjoint from a particular previous blossom B_ℓ or else B_k properly contains B_ℓ. This induces a natural partial ordering on the blossoms where,

$$B_k > B_\ell \iff S_k \supset S_\ell .$$

This partial ordering is important when blossoms are subsequently expanded, to find the matching on G.

A vertex v_k which is the image of some blossom can be treated as a simple vertex as far as growth and augmenting are concerned. But eventually the vertex must be expanded to obtain the actual matching on $G_o = G$. The following theorem characterizes the freedom allowable in matching a blossom.

Theorem 2.4.2

Let S_k be the odd vertex set in G of blossom B_k. If $v_1 \epsilon S_k$ then there exists a maximum matching M of $S_k(G)$ which leaves v_1 exposed.

Proof

The blossom is expanded in any manner consistent with the partial ordering. That is, B_i is expanded before B_j if and only if $S_i \supset S_j$ or $S_i \cap S_j = \emptyset$.

At the first step B_k is expanded by replacing B_k with the circuit HP_k. One of the vertices of HP_k is either v_1 or contains v_1. That vertex is left exposed and the rest of HP_k is maximally matched. But now we are at the same situation relative to v_1 so that eventually v_1 is exposed in $G_o = G$. At each step the vertices of HP_i, which did not contain v_1, became matched.

If a vertex, which is matched by the above construction, is a simple vertex we are finished with it. However, if it is the image of a blossom B_i then the matched edge defines a vertex $v_j \epsilon S_i$ which

must be left exposed relative to $S_i(G)$. Thus we return to the same situation as the hypothesis, but with fewer blossoms. Since the number of blossoms is finite the process eventually terminates with only the originally specified vertex v_1 exposed.

2.5 Proof by Duality Theory

The basis of the proof that the algorithm achieves a maximum weighted matching is the following theorem from linear programming (see Appendix B). If a feasible primal and a feasible dual solution can be found which together satisfy the orthogonality conditions then the solutions are optimal.

Thus we must show:

$$\left.\begin{array}{ll} (1) & x \geq 0 \\ (2) & Ax \leq \epsilon \\ (3) & Rx \leq r \end{array}\right\} \quad \text{Primal Feasibility}$$

$$\left.\begin{array}{ll} (4) & y \geq 0 \\ (5) & z \geq 0 \\ (6) & y_i + y_j + \sum_{e_{ij} \epsilon R_k} z_k \geq c_{ij} \end{array}\right\} \quad \text{Dual Feasibility}$$

$$\left.\begin{array}{ll} (7) & x_{ij} > 0 \Longrightarrow y_i + y_j + \sum_{e_{ij} \epsilon R_k} z_k = c_{ij} \\ (8) & y_i > 0 \Longrightarrow A_i x = 1 \quad (\text{vertex } v_i \text{ is not exposed}) \\ (9) & z_k > 0 \Longrightarrow R_k x = r_k \quad (S_k \text{ is maximally matched by } R_k) \end{array}\right\} \quad \text{Orthogonality}$$

Since the solution we obtained by the algorithm was a matching, conditions (1), (2), and (3) are immediate. Since the algorithm

always stops lowering vertex weights when an outer vertex y variable goes to zero, and all inner vertex weights are only raised, (4) is always satisfied. z_k variables are always nonnegative in the algorithm so that (5) is always satisfied.

Our convergence argument was based on elimination of exposed vertices for which $y_i > 0$, thus (8) is satisfied. The only way in which a nonzero z_k variable occurs is when a blossom forms. But then $R_k x = r_k$ and condition (9) follows. Finally then, the proof reduces to showing (6) and (7).

y_i is the value of the dual variable associated with vertex v_i during the algorithm. Let $\bar{y}_i$ be the final value of y_i when the algorithm terminates. Similarly let $\bar{z}_k$ be the final value of z_k for each blossom. We must show that,

$$(10)\quad \bar{y}_i + \bar{y}_j + \sum_{e_{ij} \epsilon R_k} \bar{z}_k \geq c_{ij}.$$

Initially, $y_i + y_j \geq c_{ij}$ for all edges. Further, this relationship is clearly maintained by the algorithm as long as both vertices of an edge are not contained within the same blossom. Let y_i^{ℓ} be the value of y_i when the ℓ^{th} blossom forms. Then,

$$(11)\quad y_i^{\ell} + y_j^{\ell} \geq c_{ij}.$$

After edge e_{ij} is within a blossom B_k (i.e., $e_{ij} \epsilon R_k$), both y_i and y_j are lowered the same amount. That amount being the summation of $\frac{1}{2} \bar{z}_k$, for all blossoms B_k which totally contain edge e_{ij}. Therefore, if B_ℓ is the first blossom which contains edge e_{ij};

$$(12)\quad y_i = y_i^{\ell} - \frac{1}{2} \sum_{e_{ij} \epsilon R_k} \bar{z}_k$$

$$(13)\quad \bar{y}_j = y_j^{\ell} - \frac{1}{2} \sum_{e_{ij} \in R_k} \bar{z}_k .$$

Combining (11), (12) and (13) we have,

$$(14)\quad \bar{y}_i + \bar{y}_j + \sum_{e_{ij} \in R_k} \bar{z}_k \geq c_{ij}, \text{ and (6) is proved.}$$

If edge e_{ij} is in the matching, it entered it by being in some tree which resulted in an augmentation. But in order to be grown into a tree it must be that,

$$y_i + y_j = c_{ij}.$$

Further a matched edge which is not within a blossom, but is in a tree, is always between an inner and outer vertex. Thus, the equality above is maintained. If edge e_{ij} goes into a blossom, then the proof of (6) can be used with the equality holding instead of the unequality. Thus (7) is true, and the algorithm has been shown to obtain an optimal solution.

With this result it can now be shown that the convex polyhedron defined by the constraints:

$$Ax \leq \epsilon,$$

$$Rx \leq r, \text{ and}$$

$$x \geq 0,$$

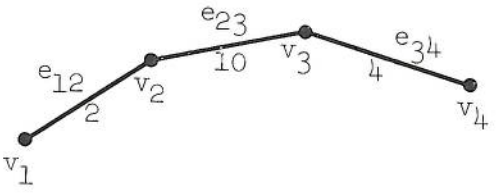
has only integer vertices. First of all, every vertex of a convex polyhedron is the unique maximum for some choice of the objective function. Also for every choice of objective function the maximum cost is assumed at some vertex. The edge weights define the objective function in this case and they were arbitrarily chosen. Thus since the algorithm found an integer matching solution for arbitrary edge weights, it can be

concluded that the polyhedron has only integer vertices.

2.6 Geometric Interpretation

There is a very interesting connection between augmenting paths and edges of the convex polyhedron defined by the matching problem. Let us consider a very simple matching problem for which the polyhedron can be drawn in three spaces.

Example 2.6.1



The best matching M_1 on G is e_{23} which has weight 10. The best 2-edge matching M_2 on G is $\{e_{12}, e_{34}\}$ which has weight 6. If 5 is added to each edge of G then $w(M_1) > w(M_2)$ (i.e., $w(M_1) = 14$, $w(M_2) = 15$). In general an optimum k-cardinality match can be made to be the optimum weighted matching regardless of cardinality, by application of an appropriate uniform change to the weight of each edge.

Let us write each constraint for this problem.

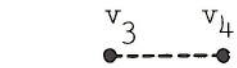
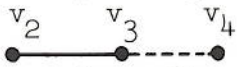
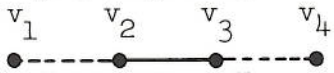
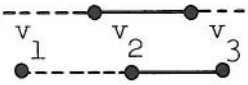
vertex constraints	edge constraints
$x_{12} \leq 1$	$x_1 \geq 0$
$x_{12} + x_{23} \leq 1$	$x_2 \geq 0$
$x_{23} + x_{34} \leq 1$	$x_3 \geq 0$
$x_{34} \leq 1$	

The associated cost function is,

$$W = 2x_{12} + 10\,x_{23} + 4\,x_{34}.$$

These constraints result in the polyhedron shown in Figure 2.6.1.

Each edge of this polyhedron corresponds to an alternating path of G relative to some matching (Table 2.6.1).

Edge of Polyhedron	Alternating path
0,0,0 to 0,0,1	
0,1,0 to 0,0,1	
0,1,0 to 1,0,1	
0,1,0 to 1,0,0	
1,0,0 to 1,0,1	

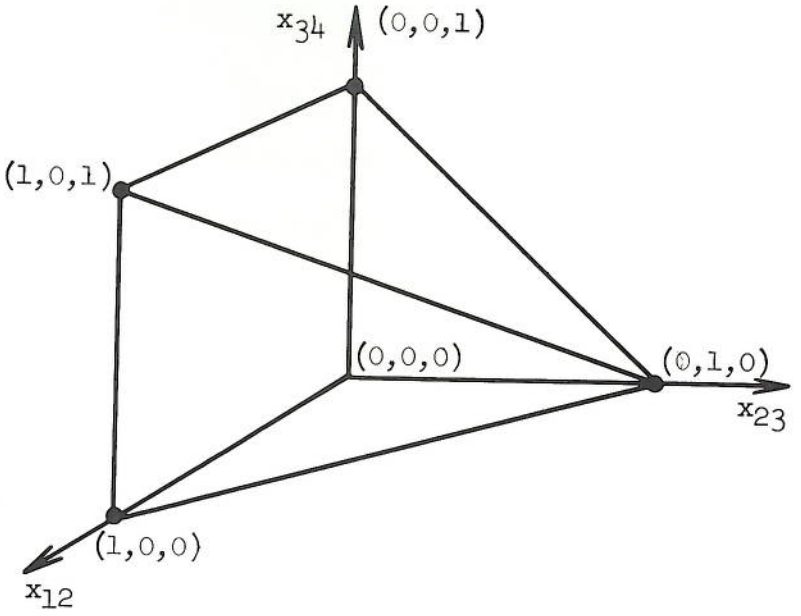
Correspondence Between Edges of Polyhedron and Alternating Paths

Table 2.6.1

Each vertex of the polyhedron is specified by $n(n = 3)$ independent constraints satisfied with equality, and each edge of the polyhedron is specified by $n-1$ $(n-1 = 2)$ independent constraints satisfied with equality.

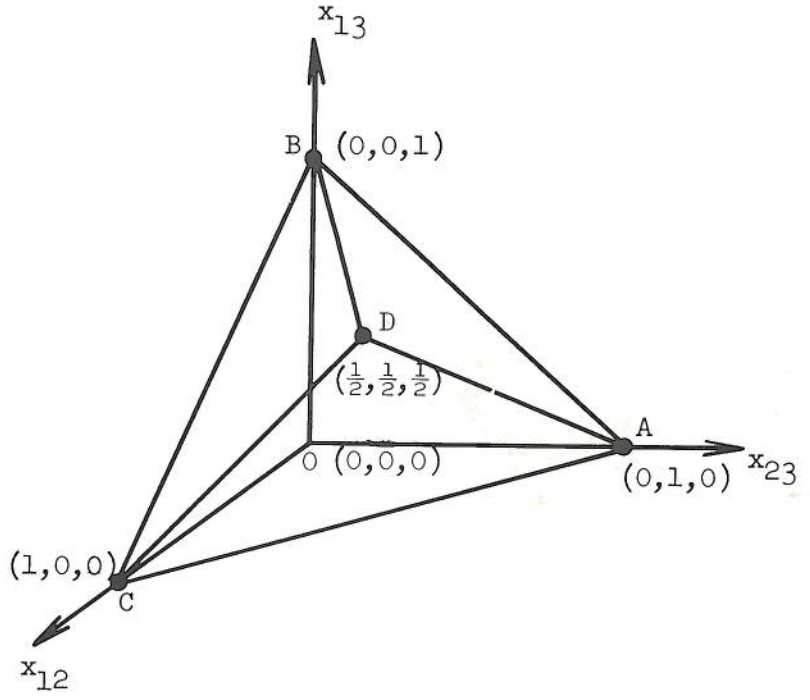
The edge weights of G determine a cost plane. In this case $W = 2x_{12} + 10x_{23} + 4x_{34}$, where W is the weight of the matching if x is a feasible solution. In other words the maximum cost is the largest value of W such that the cost plane intersects the polyhedron.

If we add λ to all edges of the graph then the cost plane becomes



Polyhedron for Example 2.6.1

Figure 2.6.1



Polyhedron for Example 2.6.2

Figure 2.6.2

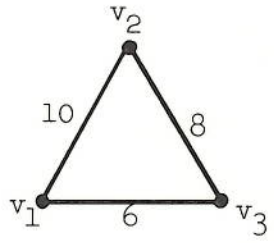
$$W = (2 + \lambda)x_{12} + (10 + \lambda)x_{23} + (4 + \lambda)x_{34}$$

whose normal vector is $(2 + \lambda, 10 + \lambda, 4 + \lambda)$. This addition of λ to the cost in effect tilts the cost plane. For very large λ it is clear that the cost plane approaches the cardinality cost plane, and thus the maximum weighted matching will also have maximum cardinality.

For this example with $\lambda = 4$ the polyhedron edge between $(0,1,0)$ and $(1,0,1)$ lays in the cost plane. For λ slightly greater than 4 the cost is maximized at the $(1,0,1)$ vertex. The critical value of λ, in this case 4, is also the value of the associated augmenting path.

Notice that for weak augments the cost plane tilts in such a way that the weight of the path does not change, since just as many edges are in the matching as are out of the matching.

Example 2.6.2



vertex constraints	edge constraints
$x_{12} + x_{23} \leq 1$	$x_{12} \geq 0$
$x_{23} + x_{13} \leq 1$	$x_{23} \geq 0$
$x_{13} + x_{12} \leq 1$	$x_{13} \geq 0$

The polyhedron associated with these constraints is shown in Figure 2.6.2.

In this example the optimum solution is at the noninteger vertex $D = (.5, .5, .5)$, and has cost $(.5, .5, .5)\,(10, 8, 6) = 12$. Edmonds' constraint $x_{12} + x_{23} + x_{13} \leq 1$ eliminates this vertex and causes the maximum to occur at c with cost 10. If $c_{12} > c_{23} + c_{13}$, this integrality constraint would not be necessary.

The above interpretation of augmenting paths together with the discussion of variable cardinality solutions as a function of the parameter λ leads to an alternate algorithm to the weighted matching problem. In summary the algorithm is:

(1) Begin with all vertex weights equal and sufficiently high to satisfy $y_i + y_j \geq c_{ij}$, $\forall\ e_{ij} \in G$.

(2) Proceed as in the weighted matching algorithm except that all exposed vertices are considered to be outer vertices. Thus the weight of exposed vertices are always equal.

This algorithm has the very interesting feature that at any step if there are k edges in the matching then those k edges constitute the optimum k cardinality matching. Further, no weak augments will ever occur.

CHAPTER III

UPPER DEGREE CONSTRAINED SUBGRAPHS (UDCS)

3.1 Introduction

The UDCS problem [3], [10] can be stated as an integer program as follows:

(1) max cx

subject to:

(2) $Ax \leq d$

(3) $x_{ij} \in \{0,1\}$.

The removal of the explicit integer constraints would result in a linear program which might have noninteger solutions. Edmonds [10] found a set of linear constraints which are sufficient to guarantee integer solutions to this problem. This chapter will demonstrate the sufficiency of these added constraints and develop an algorithm which will exhibit an optimum integer solution.

The linear program which has only integer solutions is:

(4) max cx

subject to:

(5) $Ax \leq d$

(6) $Tx \leq r$

(7) $Ix \leq \epsilon$

(8) $x \geq 0$

The associated dual linear program is:

(9) min $(dy + rz + \epsilon w)$

subject to:

(10) $A^T y + R^T z + Iw \geq c$

(11) $y \geq 0,\ z \geq 0,\ w \geq 0$

d is a vector of degree constraints, where d_i is the maximum number of edges which can be incident to vertex v_i in any solution. T is an incidence matrix of subsets of vertices versus edges. A set of vertices and edges which satisfy one of the equations of (6), with equality (i.e., $T_k x = r_k$), will be called a blossom. The exact specification of T and further characterization of blossoms is given in section 3.6. I is the identity matrix, hence (7) restricts each x_{ij} variable to be less than 1.

y_i is a dual variable associated with vertex v_i, z_k is the dual variable associated with the k^{th} primal constraint of (6), and w_{ij} is the dual variable associated with one of the primal constraints of (7).

The constraints in (6) are only necessary when odd cycles exist in the graph. Even when odd cycles occur they may not lead to noninteger solutions. Since much of the complexity of the algorithm arises from the presence of blossoms, the following order of presentation will be used:

(1) Characterization of optimum UDCS solutions.

(2) Description of the cardinality algorithm for a bipartite graph (no odd cycles).

(3) Description of the algorithm for a weighted bipartite graph.

(4) Characterization of blossoms.

(5) Description of the cardinality algorithm for a general graph.

(6) Description of the algorithm for a general weighted graph.

(7) Proof that the algorithm results in an optimum solution.

3.2 Characterization of UDCS Solutions

The concept of an augmenting path, which was used in the solution and characterization of the matching problem, is also applicable to the upper degree constrained subgraph (UDCS) problem. Let D be a subset of the set of edges of a graph satisfying the given degree constraints d, that is, $d(i/D) \leq d_i$, for all vertices.

Definition 3.2.1

P is a UDCS cardinality augmenting path, relative to a feasible solution D, means that:

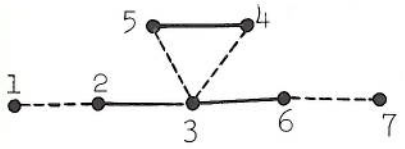
(1) The edges of P are alternately in D and $\overline{D}$.

(2) $D' = D \oplus P$ is a feasible solution.

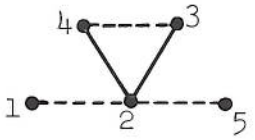
(3) $|D'| > |D|$.

(4) P is minimal with respect to (1), (2), and (3).

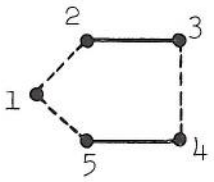
UDCS augmenting paths are not simple paths as were matching augmenting paths. For example the following paths (Figure 3.2.1) satisfy Definition 3.2.1.



$P_1 = \{e_{12}, e_{23}, e_{34}, e_{45}, e_{53}, e_{36}, e_{67}\}$



$P_2 = \{e_{12}, e_{23}, e_{34}, e_{42}, e_{25}\}$



$P_3 = \{e_{12}, e_{23}, e_{34}, e_{45}, e_{51}\}$

——● denotes edge in solution D

●----● denotes edge not in D

UDCS Augmenting Paths

Figure 3.2.1

The main result of this section will be the proof of the following theorem.

Theorem 3.2.1

D is not a maximum cardinality UDCS solution if and only if there exists a UDCS cardinality augmenting path.

Proof

If such a path P exists then $D' = D \oplus P$ is a feasible solution of higher cardinality. Thus D could not have been maximum.

The "only if" is considerably more difficult. We will construct a path satisfying (1), (2), (3), and (4) of the definition of a UDCS cardinality augmenting path. If D is not maximum then there exists an F such that $|D| < |F|$. From all such solutions, choose one such that $|D \oplus F|$ is minimal.

constraint d_i at terminal vertex v_i (i.e., $d_i \geq d(i/D')$).

Case (a) Since the path has the maximum number of edges, it must be that $d(i/F) \geq d(i/D) + 1$, or else the path could be extended. Since F is feasible, $d_i \geq d(i/F)$. Thus,

$$d_i \geq d(i/D) + 1 = d(i/D').$$

Case (b) $d_i \geq d(i/D) > d(i/D')$.

Case (c) As in case (a), the maximality of the number of edges insures that,

$$d(i/F) \geq d(i/D) + 2.$$

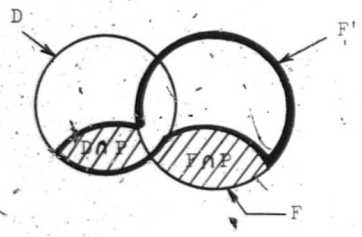
Since $d_i \geq d(i/F)$, we have,

$d_i \geq d(i/F) \geq d(i/D) + 2 \geq d(i/D')$, or finally, $d_i \geq d(i/D')$.

Case (d) $d_i \geq d(i/D) > d(i/D') \Longrightarrow d_i \geq d(i/D')$.

Thus $D' = D \oplus P$ is a feasible solution. Similarly we can show that $F' = F \oplus P$ is a feasible solution

Now suppose that $|D \cap P| \geq |F \cap P|$. See Figure 3.2.2.



Venn Diagram for Theorem 3.2.1

Figure 3.2.2

$$|F'| = |F| - |F \cap P| + |D \cap P|.$$

Therefore $|F'| \geq |F| > |D|$. Now by inspection of Figure 3.2.2,

$$|F' \oplus D| = |F \oplus D| - |D \cap P| - |F \cap P|.$$

Thus, F' has a smaller symmetric difference with D than F, which contradicts the assumed minimality of F with respect to this property. Thus, $|D \cap P| < |F \cap P|$. This implies that $|D'| > |D|$, and property (3) is proved.

Property (4) is true, for otherwise we would violate the assumed minimality of $|D \oplus F|$. Q.E.D.

So far, we have only assumed that,

$$D \cap P \subset D\text{-}F, \text{ and } F \cap P \subset F\text{-}D.$$

But it is obvious that,

$$D \cap P = D\text{-}F, \text{ and } F \cap P = F\text{-}D.$$

For otherwise $|D' \oplus D| < |F \oplus D|$, which again would contradict the assumed minimality of $|D \oplus F|$. Thus, in fact $D' = F$.

The symmetric difference of D and F consists of exactly one UDCS cardinality augmenting path. A similar theorem for the weighted UDCS problem can be shown. First we need an additional definition.

Definition 3.2.2

A path P is a UDCS augmenting path relative to a given UDCS solution D, if P satisfies:

(1) Edges of P are alternately in D and $\overline{D}$.

(2) $D' = D \oplus P$ is a feasible solution.

(3) The edge weights are such that $w(D') > w(D)$. That is,

D' is a larger weight solution to the given degree constrained problem.

(4) P is minimal with respect to (1), (2), and (3).

Theorem 3.2.2

D is not a maximum weight UDCS solution if and only if there exists a UDCS augmenting path P.

Proof

Again the "if" is obvious. By using the path P, form $D' = D \oplus P$ The definition of P insures the D' is feasible and also that $w(D') > w(D)$. The proof of the "only if" part is quite similar to the proof of the previous theorem, and will not be given in entirety. Since D is not maximum there exist an F such that $w(F) > w(D)$. From all such F select one for which $w(D \oplus F)$ is minimum. The rest of the proof proceeds from this point as in Theorem 3.2.1.

3.3 UDCS Hungarian Trees

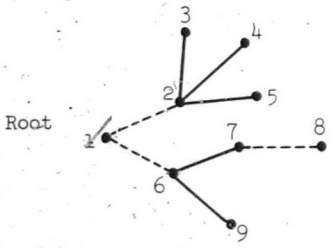
The concept of a Hungarian tree which was useful in proving convergence in the cardinality matching problem generalizes to the UDCS case. For the definitions below we assume a given degree constraint vector d and a feasible UDCS solution D.

A vertex v_i is called saturated if $d(i/D) = d_i$. A vertex v_i is called unsaturated if $d(i/D) < d_i$.

The concepts of alternating trees, and inner and outer vertices used in matching are valid for the UDCS case; but some extra complexity arises due to the greater generality of this problem.

An alternating UDCS tree is a tree rooted at a vertex which is unsaturated. Also, the unique path in the tree from the root to any other vertex in the tree is an alternating path beginning with an edge not in D, and all maximal paths from the root are of even length.

A UDCS augmenting tree is a UDCS alternating tree with one additional edge which forms a UDCS augmenting path. Figure 3.3.1 shows a UDCS augmenting tree. If edge e_{78} is removed the tree becomes an alternating UDCS tree.



$\{e_{16}, e_{67}, e_{78}\}$ is the odd length path.

UDCS Augmenting Tree

Figure 3.3.1

The inner vertices of an alternating UDCS tree are those vertices at the end of paths of odd length from the root.

The outer vertices of an alternating UDCS tree are those vertices at the end of paths of even length from the root.

A UDCS Hungarian tree J is an alternating UDCS tree with the following properties:

(1) All inner vertices are saturated.

(2) All edges in $\overline{D} \cap \overline{J}$ which have one end incident to an outer vertex, have the other end incident to an inner vertex of the tree.

(3) All edges in $D \cap \overline{J}$ which have one end incident to an inner vertex, have the other end incident to an outer vertex of the tree.

An alternate characterization of a UDCS Hungarian tree is that it is an alternating UDCS tree that can not be extended to include any other edges. This characterization will be used in the following sections to justify the termination of further tree growth when such a tree is found.

Theorem 3.3.1 below shows that for bipartite graphs a maximum UDCS solution on a Hungarian tree is independent of the rest of the graph. A similar result for the case of general graphs is obtained as a by-product of the weighted UDCS algorithm for general graphs.

Lemma 3.3.1

Let G be a bipartite graph, J a UDCS Hungarian tree and $I(J)$ the set of inner vertices of J. Then any maximum UDCS solution of $J(G)$ has cardinality $\sum_{v_i \in I(J)} d_i$.

Proof

First of all,

$$|J(G) \cap D| = \sum_{v_i \in I(J)} d_i,$$

since all inner vertices are saturated by edges of D, and by property (3) of Hungarian trees, every edge in D incident to an inner vertex is in $J(G)$.

Since there do not exist edges between two outer vertices, nor edges between two inner vertices, when G is a bipartite graph, it follows that each edge in $J(G)$ is incident to exactly one inner vertex.

Thus any solution of cardinality N has exactly N incidences to inner vertices, and the cardinality of a UDCS solution on $J(G)$ is bounded above by $\sum_{v_i \in I(J)} d_i$. But since D^* actually achieves this bound, the lemma is proved.

We partition the edges of G relative to J into four sets:

(1) E_J, the edge set of J.

(2) $E_{J(G)-J}$, the edges in $J(G)$ which are not in J.

(3) $E_{J\bar{J}}$, the edges with exactly one end in $J(G)$.

(4) E_{G-J}, the edges with neither end in $J(G)$.

Using this partition define D_J as

$$D_J = D \cap \{E_J \cup E_{J\bar{J}}\}.$$

Let D_{G-J} be any maximum UDCS solution of $G-J$ which is consistent with D_J. That is,

$$d(i/D_{G-J}) + d(i/D_J) \leq d_i, \text{ for all vertices.}$$

Theorem 3.3.1

$D_J \cup D_{G-J}$ is a maximum UDCS solution of G relative to d.

Proof

Let F be any solution on G relative to d. Partition F as follows,

(1) $F = F_J \cup F_{J(G)-J} \cup F_{J\bar{J}} \cup F_{G-J}$,

where

$$F_J \subset E_J,\ F_{J(G)-J} \subset E_{J(G)-J},\ F_{J\bar{J}} \subset E_{J\bar{J}},\ \text{and}\ F_{G-J} \subset E_{G-J}.$$

$\Delta = |D \cap E_{J\bar{J}} \cap F|$ is the number of edges which were in $D \cap E_{J\bar{J}}$, and are not used in F. Since the removal of each of these edges may allow one additional edge in the maximal solution on $G-J$, we have,

$$(2)\quad |D_{G-J}| + \Delta \geq |F_{G-J}|.$$

Lemma 3.3.1 shows that each edge used in $\bar{D} \cap E_{J\bar{J}}$ reduces the maximum solution on $J(G)$ by one, thus

$$(3)\quad |D \cap E_J| \geq |F_J| + |F_{J(G)-J}| + |F_{J\bar{J}} \cap \bar{D}|.$$

But it is easily seen that removing edges of $D \cap E_{J\bar{J}}$ does not increase the maximum solution on $J(G)$, since the maximum solution on a Hungarian tree is determined by the degree of the inner vertices. Hence by adding $|D \cap E_{J\bar{J}}|$ to both sides of (3) and using the definition of Δ we arrive at,

$$(4)\quad |D_J| \geq |F_J| + |F_{J(G)-J}| + |F_{J\bar{J}}| + \Delta.$$

Combining (2) and (4) we have,

$$|D_J| + |D_{G-J}| \geq |F|.$$

Since F is an arbitrary solution, the theorem is proved.

3.4 Cardinality UDCS (Bipartite Graph)

The cardinality UDCS algorithm on a bipartite graph systematically searches for cardinality augmenting paths as described in section 3.2. The bipartite structure allows a discussion of the basic features of the general UDCS algorithm without the complications caused by odd cycles.

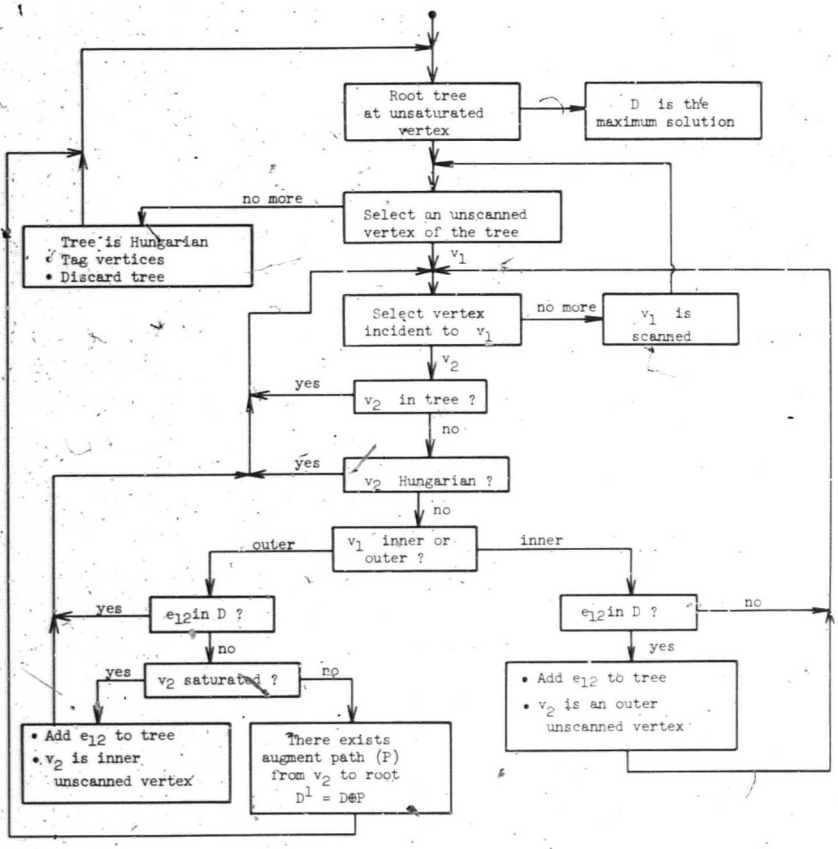
The algorithm (Figure 3.4.1) begins by selecting any feasible solution D, (e.g., D = $\emptyset$) and rooting an alternating tree at any unscanned unsaturated vertex. A scanned vertex is a vertex of the tree for which all incidences have been checked and no further growth is possible from that vertex.

Having rooted a tree, the algorithm searches for an edge which will enlarge the tree by bringing in another vertex. If a vertex v_1 in the tree is incident to a vertex v_2 not in the tree which is also not a vertex of some previous Hungarian tree, then further growth is possible. If v_1 is an outer vertex, and e_{12} is in the present solution D, then one of two things happen.

(1) If v_2 is unsaturated, the unique alternating path from v_2 back through the tree to the root is a UDCS augmenting path. The solution is changed by interchanging edges in D with edges not in D along the path. A new tree is rooted at some remaining unsaturated vertex. All vertices become unscanned and the tree is discarded.

(2) If v_2 is saturated, e_{12} is added to the tree and v_2 is classified as an unscanned inner vertex of the tree.

If v_1 is instead an inner vertex, and if e_{12} is in D, then e_{12} is added to the tree, and v_2 is classified as an outer unscanned vertex. In all cases, the incidences to v_1 are continually examined until either an augmenting path occurs, or v_1 becomes a scanned vertex. This particular method is important in determining an upper bound on the asymptotic growth of the algorithm. Essentially for



Maximum Cardinality UDCS Algorithm (Bipartite Graph)

Figure 3.4.1

any one tree each edge of the graph is examined at most twice, that is, once from each vertex incident to that edge. When no further vertices can be added to the tree (equivalently all tree vertices are scanned), the present tree J is Hungarian. By the results of section 3.3, the solution on the Hungarian tree J and on edges between the tree and the remaining graph $G-J$ can be fixed. The tree vertices are tagged so that none of these edges will enter subsequent trees.

The following definitions are with respect to a given UDCS solution D and degree constraint d.

The slack u_i of vertex v_i is, $u_i = d_i - d(i/D)$.

The slack U_D of a UDCS solution D is the total slack of all vertices which have not been involved in Hungarian trees.

Since vertices in Hungarian trees do not have to be used in order to find an optimum solution, the algorithm is finished when $U_D = 0$. Each tree, whether it results in an augmenting path or in a Hungarian tree, reduces the slack by at least one, therefore fewer than $|d|$ trees are required.

The amount that a Hungarian tree reduces the slack is equal to the sum of the slack u_i of outer vertices of the tree. This could be as low as one. An augment always reduces slack by exactly two.

Since each edge is examined at most twice per tree and fewer than $|d|$ trees are required to find the optimum solution, the algorithm grows asymptotically as $|d|n^2$.

3.5. Weighted UDCS (Bipartite Graph)

The algorithm for finding a maximum weight UDCS on a bipartite

graph will now be developed. This algorithm is motivated by the primal-dual linear programs, so let us restate them without the odd cycle constraints:

Primal:

(1) max cx

subject to:

(2) $Ax \leq d$

(3) $Ix \leq \epsilon$

(4) $x \geq 0$

Dual:

(5) min $dy + \epsilon w$

subject to:

(6) $A^T y + Iw \geq c$

(7) $y \geq 0$

(8) $w \geq 0$

The orthogonality conditions of linear programming are:

(9) $y_i > 0 \Longrightarrow A_i x = d_i$

(10) $w_{ij} > 0 \Longrightarrow x_{ij} = 1$

(11) $x_{ij} > 0 \Longrightarrow y_i + y_j + w_{ij} = c_{ij}$.

The proof of the optimality of the algorithm relies upon the linear programming theorem (Theorem B.3) relating orthogonality, feasibility, and optimality. If a feasible solution can be shown to be orthogonal, then it is optimal.

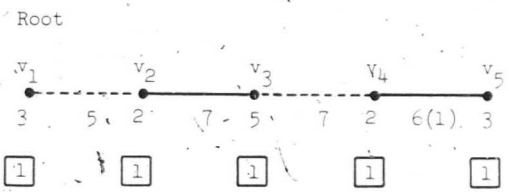
The primal constraint matrix given by (2) and (3) has the unimodular property; thus all vertices of the associated polyhedron will

be integers, and this particular problem could be solved by the simplex method. However, the purpose here is to introduce the structure of the algorithm for which the simplex method can not be used.

Basically the algorithm (Figure 3.5.1) uses the cardinality algorithm as a subroutine. A restricted subgraph G^* is defined where an edge is in the subgraph if and only if $y_i + y_j = c_{ij}$. The cardinality algorithm is used to optimize this restricted problem by growing a tree and looking for an augment. If an augment is found it is used and another tree is rooted in G^*. If during this operation a tree becomes Hungarian, the cardinality algorithm is terminated and the dual variables are adjusted. The adjustment either brings a new edge into G^* or reduces y_i to zero for some vertex.

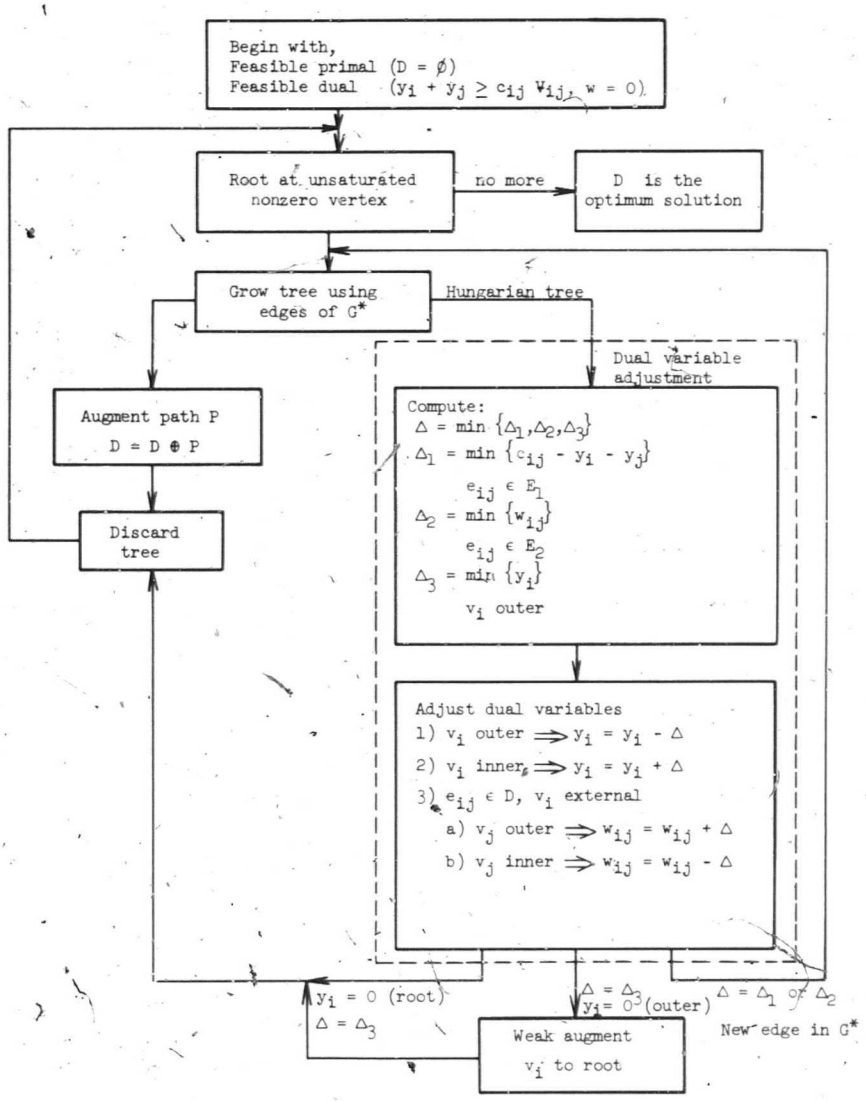
There are a few changes required in growing a tree for the weighted case, as opposed to the cardinality case. The major change is the restriction to edges for which $y_i + y_j = c_{ij}$. One result of this is that an inner vertex of the tree may not have an edge incident to it which is both in the solution and in the tree.

Example 3.5.1



Notation

(1) The number contained in the box near each vertex is the degree constraint.



Weighted UDCS Algorithm (Bipartite Graph)

Figure 3.5.1

(2) The number not in the box near each vertex is the present value of y_i.

(3) The number on each edge is the edge weight.

(4) ●-----● means not in solution.

●——● means in solution.

(5) The number in parentheses after the edge weight c_{ij} is the value of w_{ij}.

In this example v_1 is the root, and the tree would be grown out to v_4, with v_1 and v_3 being inner vertices. Edge e_{45} will not be added to the tree since $y_4 + y_5 \neq c_{45}$. Thus, this tree is Hungarian and the dual variables must be adjusted in order to continue.

A vertex v_i which is unsaturated and for which $y_i > 0$ is called nonorthogonal.

The nonorthogonality U_D of a UDCS solution D, is equal to the sum of the slacks of nonorthogonal vertices. That is,

$$U_D = \sum_{v_i \text{ nonorthogonal}} (d_i - d(i/D)).$$

Notice that if $U_D = 0$, conditipn (9) is satisfied.

The vertices can be classified into the following four types relative to being saturated or unsaturated and y_i being equal or greater than zero:

(1) v_i unsaturated, $y_i > 0$ (nonorthogonal)

(2) v_i unsaturated, $y_i = 0$

(3) v_i saturated, $y_i > 0$

(4) v_i saturated, $y_i = 0$

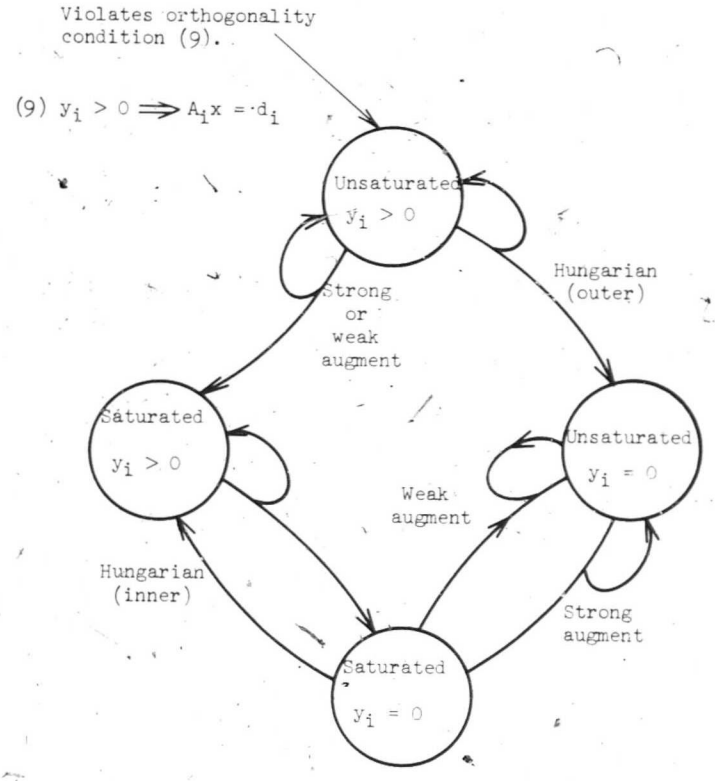
Figure 3.5.2 shows which transitions between vertex states are possible. Once a vertex leaves the nonorthogonal state it can never return.

The algorithm begins with feasible primal and dual solutions. $D = \emptyset$ is a feasible primal solution. By choosing y_i large enough so that $y_i + y_j \geq c_{ij}$ for all e_{ij}, the dual is initially feasible. w is initially zero for all edges so that (10) is initially satisfied. Since $D = \emptyset$, $s_{ij} = 0$ for all edges and thus (11) is satisfied. Therefore only condition (9) is not satisfied.

A tree is rooted at any nonorthogonal vertex v_i. Edges in G^* are used to extend the tree. If there are no nonorthogonal vertices, the algorithm is finished and the solution D is optimum. The tree continues to grow until there is either a cardinality augmenting path in the tree or the tree can not be extended any further (Hungarian tree).

The cardinality augmenting path P is called a strong augment since both the cardinality and weight of the solution $D^1 = D \oplus P$ is higher. An augmenting path which only increases the weight is called a weak augment. The strong augment reduces the slack at the root and therefore U_D is reduced by at least one. Following this augment, the tree is discarded and another tree is rooted.

If a tree becomes Hungarian, the dual variables y and w are adjusted in such a way to maintain a feasible dual. The inner vertices are raised and the outer vertices are lowered a uniform amount Δ. For an edge e_{ij} in D with one external vertex and one inner(outer) vertex, s_{ij} is decreased(increased) by Δ.



Vertex State Transitions During UDCS Algorithm

Figure 3.5.2

There are three different conditions which limit the amount of dual variable adjustment:

(1) Let E_1 be the set of edges not in D that are between an outer and an external vertex. Then $\Delta_1 = \min_{e_{ij} \in E_1} \{c_{ij} - y_i - y_j\}$ is the amount of adjustment which will bring a new edge into G^* and keep all other edges of E_1 feasible.

(2) Let E_2 be the set of edges in D and between an inner and an external vertex. Then $\Delta_2 = \min \{w_{ij}\}$, is the amount of adjustment which will bring one edge of E_2 into G^* and keep all other edges of E_2 satisfying (11)

(3) $\Delta_3 = \min_{v_i \text{ outer}} \{y_i\}$ is the maximum adjustment which keeps each $y_i \geq 0$.

The adjustment made to the inner and outer vertices is

$$\Delta = \min \{\Delta_1, \Delta_2, \Delta_3\}.$$

As a result of these adjustments one of the following three events occurs:

(1) If the value of Δ was determined by either Δ_1 or Δ_2, then a new edge is available for growth in G^*.

(2) If $y_i = 0$ for the root, it is no longer a nonorthogonal vertex and U_D is thus reduced. The tree is discarded and a new tree rooted.

(3) If $y_i = 0$ for some outer vertex other than the root, then the even path P from v_i to the root is a weak augmenting path. By forming $D^1 = D \oplus P$, the slack of the

root is reduced and the slack of v_i is increased. But since $y_i = 0$, the nonorthogonality of D^1 is less than that of D. The tree is discarded and a new tree rooted.

In summary, after adding at most n edges to a tree, it eventually results in either a strong augment, a weak augment, or the weight of the root going to zero. In each of these cases the nonorthogonality is decreased by at least one. Therefore after fewer than $|d|$ trees, condition (9) is satisfied and the primal and dual solutions are orthogonal and hence optimal (Theorem B.3).

When the algorithm terminates, all vertices satisfy (9), hence for each vertex v_i, either $y_i = 0$ or v_i is saturated. Also when $x_{ij} > 0$, $y_i + y_j + w_{ij} = c_{ij}$. Therefore,

$$W = cx = \sum_{x_{ij} > 0} (y_i + y_j + w_{ij})\, x_{ij}$$

$$= \sum_{\forall i} d_i y_i + \sum_{\forall ij} w_{ij} = dy + \epsilon w = U .$$

From Theorem B.1 of Appendix B, $U \geq W$ when both primal and dual are feasible. But since $U = W$, both U and W are optimum.

We have now presented algorithms for both the cardinality and weighted UDCS problems on a bipartite graph. Before discussing similar algorithms for the case of general graphs, we will consider UDCS blossoms.

3.6 UDCS Blossoms

In the description of the matching problem the concept of a

blossom was used. A blossom was shown to be closely related to the added constraint equations which converted the integer program into a strictly linear program. A generalization of the same idea is required for the UDCS problem.

Recall from section 3.1 that the constraints added to replace the explicit integrality constraints were:

$$Tr \leq r$$

$$Ix \leq e$$

A blossom B_k has been defined as a set of edges and vertices which satisfies one of the $T_k x \leq r_k$ constraints with equality. Associated with each constraint is a set of vertices S_k, and a set of edges T_k. T_k will also be used to symbolize a $\{0,1\}$ vector where $T_{k(ij)} = 1$ if and only if edge e_{ij} is in the set T_k. The set T_k consists of two disjoint sets:

(1) R_k, the set of edges with both ends in S_k, and

(2) F_k, a subset of the edges with exactly one end in S_k.

The integer r_k is defined by

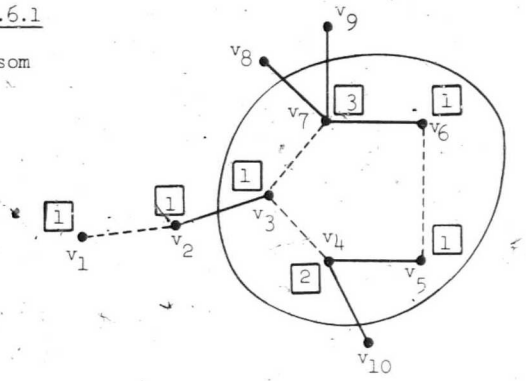
$$2r_k + 1 = \sum_{v_i \in S_k} d_i + |F_k|.$$

These constraints are closely related to the matching integrality constraints with the extra degree requirements of the UDCS problem taken into consideration. A circuit through an odd set of vertices is essential in order to cause nonintegrality. Potential UDCS blossoms can be detected during the tree growth when an edge is added between two outer vertices, or between two inner vertices, since this

guarantees the existence of an odd circuit.

Example 3.6.1

UDCS Blossom



The blossom consists of vertices

$$S_k = \{v_3, v_4, v_5, v_6, v_7\}$$

and edges,

$$T_k = \{e_{37}, e_{67}, e_{56}, e_{45}, e_{34}, e_{4,10}, e_{78}, e_{79}\}.$$

Maximum integer solution = 6.

Maximum solution = 6.5

$$x_{12} = x_{78} = x_{79} = x_{4,10} = 1$$

$$x_{34} = x_{45} = x_{56} = x_{67} = x_{73} = .5$$

Each vertex within a blossom becomes both an inner and outer vertex in the sense that it is at the end of either an odd or even path back to the root depending on which direction around the circuit is used. In Example 3.6.1 the odd path from v_5 to the root is $\{e_{56}, e_{67}, e_{73}, e_{23}, e_{12}\}$, while the even path is $\{e_{45}, e_{34}, e_{23}, e_{12}\}$.

In matching, the interpretation of vertices in blossoms as inner vertices was unimportant since inner vertices could have only one edge of the matching incident to it. This restriction is not true in the UDCS case, and the inner vertex interpretation is very important in both:

(1) the detection of blossoms, and

(2) the continued growth of the tree after detecting a blossom.

If any vertex, except the root, in the circuit of a potential blossom is unsaturated, we use the inner vertex interpretation, and trace the odd length path back to the root to identify an augmenting path. If the root is in the circuit and its slack is greater than one an augmenting path also exists. When no augmenting paths exist we have identified a blossom.

A blossom is formed by adding an edge between vertices v_1 and v_2 of the tree. The paths P_1 from v_1 to the root, and P_2 from v_2 to the root, are then found. $\{P_1 \oplus P_2 \cup e_{12}\}$ defines the circuit through the vertices of the blossom. The tip of the blossom is the vertex furthest from the root which is in both P_1 and P_2. There are three types of blossoms relative to the edges which are in F_k:

(1) When the tip is the root the blossom is called a root blossom. F_k then consists of all edges which are in the present solution D and have exactly one end in S_k.

(2) When the tip is an outer vertex other than the root the blossom is called a light blossom. In this case F_k

consists of all edges which are in D and have exactly one end in S_k except for one edge. That is the edge which is incident to the tip and also in $P_1 \cap P_2$.

(3) When the tip is an inner vertex the blossom is called a heavy blossom. F_k, in this case, consists of all edges which are in D and have exactly one end in S_k plus one other edge which is not in D. That is the edge which is incident to the tip and in $P_1 \cap P_2$.

Figure 3.6.1 shows an example of each type blossom and its associated F_k set. By a degree counting argument, it can be seen that the requirement that each vertex of a blossom be saturated, along with the edge sets defined for each type blossom, results in the satisfaction of one of the $T_k x \leq r_k$ constraints with equality.

For example, let B_k be a light blossom. Let D_1 be the number of edges of D which have both ends in S_k, and D_2 be the number of edges of D which have exactly one end in S_k.

All vertices of a light blossom must be saturated, thus

$$2D_1 + D_2 = \sum_{v_i \in S_k} d_i .$$

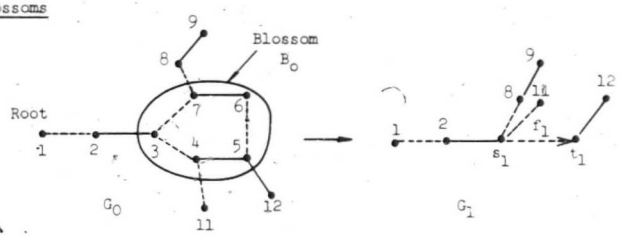
By the specification of which edges are in F_k we have

$$|F_k| + 1 = D_2 .$$

Combining the above we obtain,

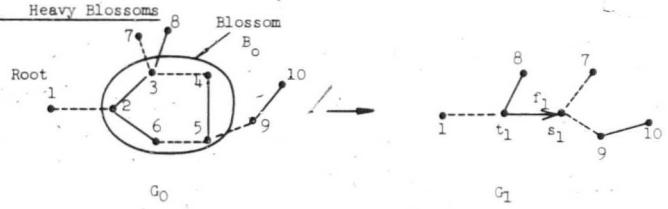
$$2(D_1 + D_2) = \sum_{v_i \in S_k} d_i + |F_k| - 1 .$$

Case 1 Light Blossoms



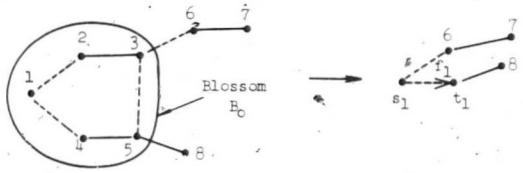
$F_0 = e_{5,12}$

Case 2 Heavy Blossoms



$F_0 = \{e_{12}, e_{38}\}$

Case 3 Root Blossoms



$F_0 = \{e_{58}\}$

UDCS Blossoms

Figure 3.6.1

But $T_k x = D_1 + D_2 + 1$ for light blossoms, thus

$$T_k x = \frac{\Sigma\, d_i + |F_k| - 1}{2} = r_k.$$

Similarly, the equivalence of the saturated vertex viewpoint and equality of a $T_k x \leq r_k$ constraint can be shown for root and heavy blossoms.

When blossoms occurred, in the case of matching, we classified all vertices in the blossom as outer. Then all outer vertices were examined in an attempt to extend the alternating tree. It was convenient in describing this process to consider all vertices of the blossom as a single vertex. A similar representation will be used for the UDCS blossoms. Since vertices within the blossoms are both inner and outer, the representation shown in Figure 3.6.1 is used.

The representation of the blossom as an outer vertex is s_{k+1}, while t_{k+1} is the representation of the blossom as an inner vertex. The dotted or solid line, f_{k+1} between s_{k+1} and t_{k+1} denotes the odd path from any vertex in the blossom back to the tip of the blossom. When a blossom B_k is formed, all edges in F_k are attached in G_{k+1} to the generalized inner vertex t_{k+1}, and all other edges are attached to s_{k+1} (Figure 3.6.1).

It will be convenient when considering the weighted UDCS problem to consider the vertices of a blossom to be distinct. Each vertex v_i becomes two vertices v_i and $v_{i'}$. v_i is the outer vertex representation and $v_{i'}$ is the inner vertex representation. When a blossom first forms, $y_i = y_{i'}$. When v_i becomes involved in a

Hungarian tree as an outer vertex y_i is lowered. At the same time $y_{i'}$ is raised the same amount. If B_k is the first blossom to contain vertex v_i, then z_k is defined as, $z_k = y_{i'} - y_i$.

When a blossom B_k is detected, it is shrunk to form a new graph G_{k+1} with the vertices of the blossom replaced with two vertices s_{k+1} and t_{k+1}. Several questions as how to proceed arise:

(1) How do blossoms affect the further growth of the tree?

(2) How do blossoms affect the formation of later blossoms?

(3) How are blossoms affected by augments?

(4) Are blossoms expanded after each tree is discarded, and if not, how can they enter subsequent trees?

(5) When must blossoms be expanded?

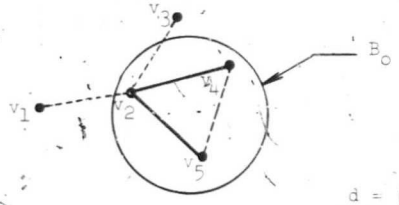
(1) Continued growth of tree

The tree can generally be continued in G_{k+1} just as if a blossom were a single edge between the vertices s_{k+1} and t_{k+1}. However, since all vertices in the blossom are now both inner and outer vertices, an edge incident to an inner vertex, which was previously ignored, may now cause further growth.

The following example demonstrates how this could occur.

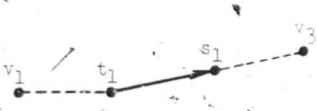
Example 3.6.2

a.)



d = (1, 2, 1, 1, 1)

b.)

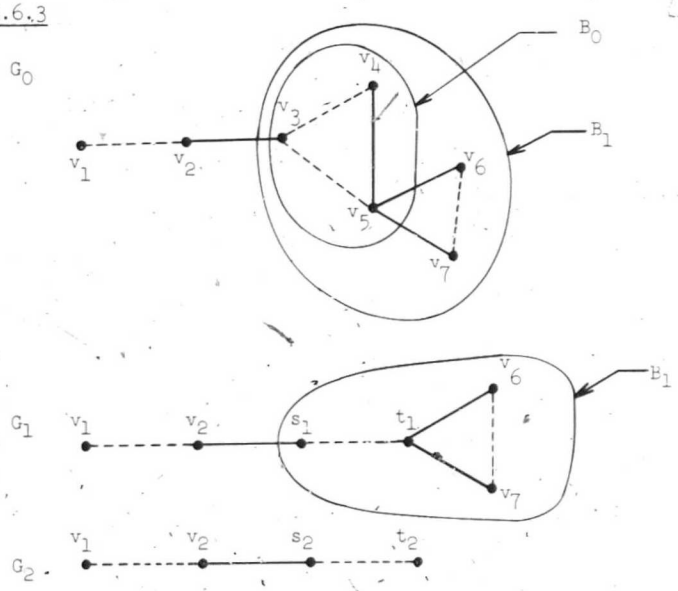


In this example, edge e_{23} could not be used to extend the tree before the blossom was formed. Following the shrinking of B_0, the image of e_{23} can be added to form an augmenting path.

(2) Effect of blossoms in forming other blossoms.

Suppose a blossom B_i forms in G_i which contains a previous blossom B_j. If B_i contains only one vertex of B_j, then B_i is the same type blossom as B_j, and both vertices of B_j (i.e., s_j and t_j) are absorbed into B_i.

Example 3.6.3



This is due to the fact that t_1 and s_1 represent all the vertices within B_0. Their separation is a notational convenience to represent the inner and outer nature of the vertices of B_0.

(3) Action of augmenting on blossoms.

Since blossom vertices are both inner and outer, there are three types of augments which may pass through a blossom:

(1) An augment which uses an odd path through the blossom.

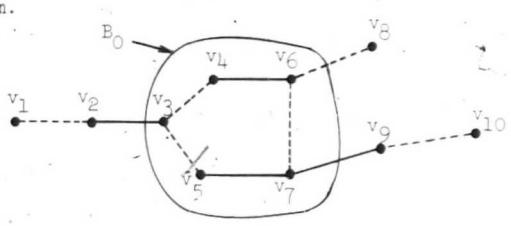
(2) An augment which uses an even path through the blossom.

(3) An augment which originates within the blossom (internal augment).

The following example shows the effect of each type augment upon a light blossom.

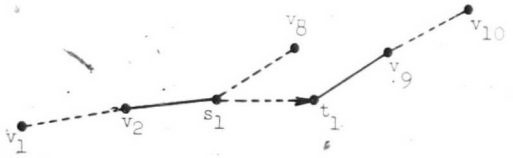
Example 3.6.4

a) G_0

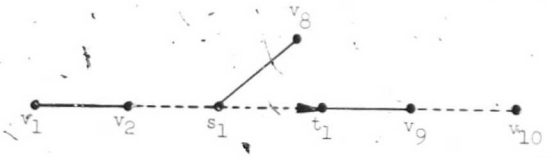


$d = (1, 1, 1, 1, 1, 2, 1, 1, 1)$

b) G_1

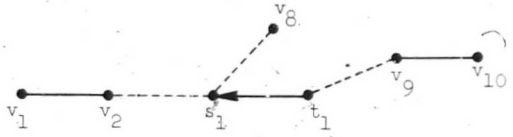


c) G_1 after using the augmenting path $P_1 = \{e_{8,s_1}, e_{s_1,2}, e_{21}\}$



d) G_1 after using the augmenting path

$P_2 = \{e_{10,9}, e_{9,t_1}, e_{t_1,s_1}, e_{s_1,2}, e_{21}\}$.



e) G_1 after using the augmenting path $P_3 = \{e_{s_1,2}, e_{21}\}$.

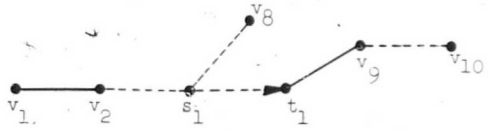


Table 3.6.1 summarizes the effect of each of the above mentioned augments upon the three different blossom types.

Original Blossom Type	Type of Augment: Even Path	Odd Path	Internal Augment
Light	Light	Heavy	Root
Heavy	Heavy	Light	Root
Root	Light	Heavy	Root

Blossom Augmenting

Table 3.6.1

(4) Old blossoms in new trees.

It is more efficient, but not necessary in the cardinality case, to allow blossoms formed in the growth of one tree to remain

shrunk while growing subsequent trees. In the weighted algorithm it is not possible to expand each blossom after every tree and still maintain the proper dual variable relationships on each edge. Thus we must consider how blossoms, which are formed in one tree, can enter subsequent trees.

There are three ways in which a heavy blossom can enter a subsequent tree:

(1) In the same orientation as when first grown.

(2) In reverse orientation (i.e., s_k an inner and t_k an outer vertex).

(3) Only t_k in the tree and used as an outer vertex.

For cases (2) and (3) if the tree becomes Hungarian these blossoms must be expanded to look for further growth.

As in the case of heavy blossoms there are three ways in which a light blossom can enter a tree:

(1) As originally formed (i.e., s_k is outer and t_k following s_k in the tree).

(2) In reverse orientation (i.e., t_k outer, and s_k following t_k in the tree).

(3) Only s_k in the tree as an inner vertex.

Cases (2) and (3) must be expanded if they occur in a Hungarian tree.

There are only two ways in which a root blossom can enter a subsequent tree:

(1) The root blossom can be the root of a new tree, or

(2) the s_k vertex can enter the tree as an outer vertex. The latter case leads to an augmenting path using the odd path through the blossom.

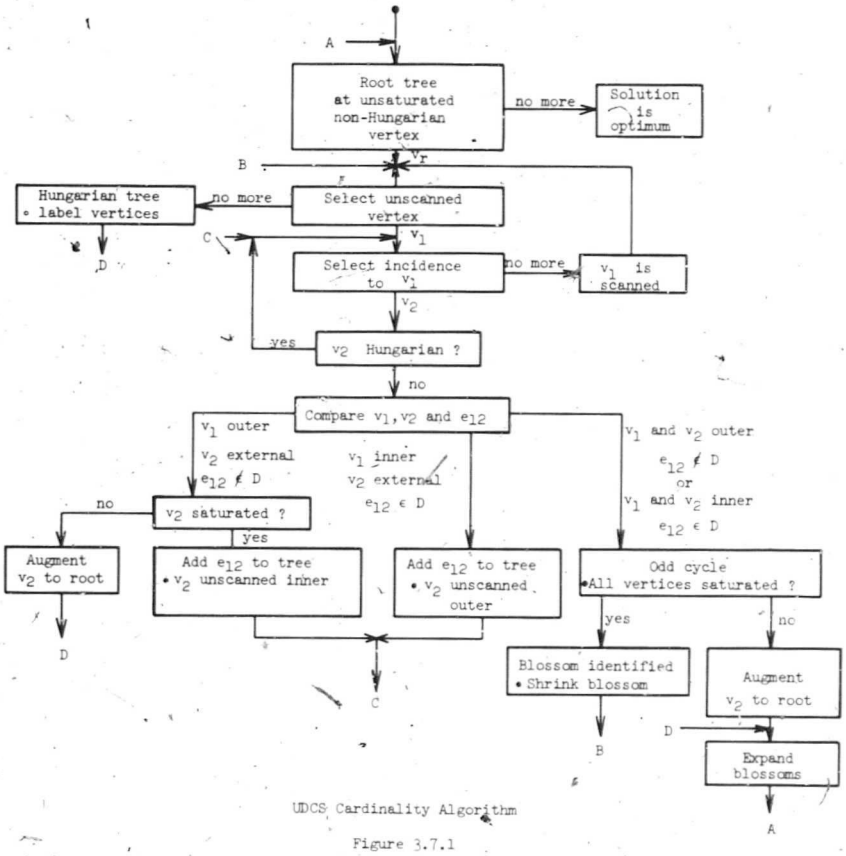
(5) Expanding blossoms

Following each tree of the cardinality UDCS algorithm, all blossoms can be expanded and the proper solution induced. However, for the weighted case blossoms must be left shrunk until their associated dual variables z_k become zero. This can occur when the s_k vertex is used as an inner vertex (matching is an example of this) or when t_k is used as an outer vertex in some tree.

The expansion of UDCS blossoms is similar to the expansion of matching blossoms. The circuit, vertex set, and edge set of the blossom are saved when a blossom is formed. Subsequent augments will affect which edges, in the circuit associated with the blossom, will be in the final solution.

3.7 Cardinality UDCS

Having introduced UDCS blossoms and discussed many of the associated complications, we can now describe the cardinality UDCS algorithm (Figure 3.7.1). An alternating tree is rooted at any unsaturated vertex which has not appeared in a previous Hungarian tree. When vertices first enter the tree they are called unscanned. Subsequently when all incidences from a vertex have been checked the vertex is called scanned.



UDCS Cardinality Algorithm

Figure 3.7.1

The tree is extended until one of three conditions occur:

(1) An augment is found.

(2) An odd cycle forms.

(3) The tree becomes Hungarian.

The situations for which each of these conditions occur is shown in Figure 3.7.1.

If the augment occurs, the cardinality of the solution is increased by interchanging the role of edges in D and not in D along the augmenting path. When an odd cycle forms, if all vertices of the cycle are saturated, then these vertices specify a blossom and it is shrunk. If some vertex v_1 within the odd cycle is unsaturated, then there exists an augmenting path from v_1 to the root. If the tree cannot be extended it is a Hungarian tree, and each vertex of the tree is tagged so that it will not enter into subsequent trees.

After fewer than $|d|$ trees, all vertices are either saturated or belong to a Hungarian tree and the algorithm is completed. A theoretical basis for this algorithm can be obtained by showing that a maximal solution to the entire graph can be found by finding a maximal solution on a Hungarian tree and any maximal solution on the remaining graph. However, in section 3.9, the weighted UDCS algorithm is shown to find the maximum weighted UDCS. Since the cardinality solution can be found by letting $c_{ij} = 1$ for all edges, the results of section 3.9 prove the optimality of the cardinality algorithm as a special case.

3.8 Weighted UDCS

The cardinality and weighted algorithms for a bipartite graph

and the cardinality algorithm for a general graph have already been described. Thus the description of the weighted algorithm on a general graph need only concern those aspects of the algorithm which are unique to this problem.

The algorithm begins by selecting a feasible primal and dual solution and maintains this feasibility while systematically approaching the orthogonality conditions, which are listed below:

(1) $y_i > 0 \Longrightarrow A_i x = d_i$

(2) $z_k > 0 \Longrightarrow T_k x = r_k$

(3) $w_{ij} > 0 \Longrightarrow x_{ij} = 1$

(4) $x_{ij} > 0 \Longrightarrow y_i + y_j + \sum_{e_{ij} \in T_k} z_k + w_{ij} = c_{ij}$.

The algorithm is proved by showing that it obtains a feasible dual and primal solution, that satisfies (1), (2), (3), and (4). Section 3.9 is a proof of the optimality of the algorithm.

The algorithm is summarized below:

(1) The algorithm begins with a feasible primal and dual. $D = \emptyset$ is a feasible primal. The dual variables w and z are initially all zero, and the y variables are selected so that $y_i + y_j \geq c_{ij}$ for all edges.

(2) An alternating tree is rooted at any unsaturated nonzero vertex.

(3) The tree is extended using edges of G^*. G^* is the subgraph of edges for which both endpoints of an edge are not within the same blossom and also $y_i + y_j = c_{ij}$.

In order for $y_i + y_j$ to equal c_{ij}, w_{ij} must be zero. Edges are added to the tree until either:

(a) An augment occurs (step 4),

(b) a blossom forms (step 5), or

(c) no further edges from G^* can be added to the tree (step 6). This is called a Hungarian tree.

(4) If an augmenting path forms in the tree, it is used to increase the cardinality and weight of the solution. The tree is discarded and another tree rooted. This operation reduces the nonorthogonality U_D by at least one.

(5) If a blossom B_k forms, it is shrunk to vertices s_{k+1} and t_{k+1}. s_{k+1} is the outer representation of the vertices of the blossom and t_{k+1} is the inner representation.

(6) If the tree becomes Hungarian, the tree growing process is terminated and a uniform change Δ is made to the dual variables. This change maintains dual feasibility and allows some further move toward optimality. The following changes are made to the dual variables.

(a) If v_i is an outer vertex y_i is lowered by Δ.

(b) If v_i is an inner vertex y_i is raised by Δ.

(c) If the blossom B_k is an outermost blossom and s_k is an outer (inner) vertex, then z_k is increased (decreased) by 2Δ.

(d) If both vertices of an edge in $D \cap G^*$ are outer

(inner) vertices, w_{ij} is increased (decreased) by Δ.

(e) If one vertex of an edge in D is an outer (inner) vertex and the other vertex is an external vertex, w_{ij} is increased (decreased) by Δ.

There are three different conditions which limit the maximum value of Δ:

(1) A new edge enters G*. Δ in this case is limited by the minimum of the four following values:

(a) $\min \{c_{ij} - y_i - y_j\}$ for edges in D that are between an outer and an external vertex.

(b) $\frac{1}{2} \min \{c_{ij} - y_i - y_j\}$ for edges in D that are between two outer vertices, where v_i and v_j are not within the same blossom.

(c) $\min \{w_{ij}\}$ for edges in D that are between an inner and an external vertex.

(d) $\frac{1}{2} \min \{w_{ij}\}$ for edges in D that are between two inner vertices.

If this value of Δ is used, we return to the tree growing phase to extend the tree.

(2) y_i becomes zero. Δ in this case is limited by $\min \{y_i\}$ for all outer vertices. If this value of Δ is used, then one more vertex satisfies orthogonality condition (4). The tree is discarded and another tree rooted.

(3) A blossom must be expanded. Δ in this case is limited by the minimum value of $\frac{1}{2} z_k$ for outermost blossoms for which s_k is an inner vertex or t_k is an outer vertex of the tree. If this value is minimum the appropriate blossom is expanded.

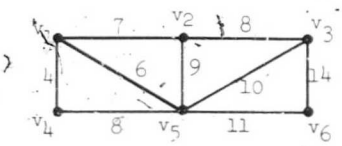
Since there always exist positive outer vertices, Δ is always defined. Having computed Δ, the indicated change is made to the dual variables, and one of the above three steps are executed. Later in the discussion of the ULDCS problem, the Δ variable in the Hungarian process may not be defined. This will indicate that no feasible solution exists.

In summary, after adding at most n edges to a tree, it either augments or the weight of the root goes to zero. In each of these cases the nonorthogonality U_D is decreased by at least one. Therefore, after fewer than $|d|$ trees condition (4) is satisfied. The algorithm grows asymptotically as $|d|\ n^3$.

The next section will show that the algorithm results in a feasible solution which satisfies all the orthogonality conditions and therefore the solution is optimal.

Example 3.8.1 UDCS Example

Consider the following graph with $d = (2, 1, 2, 1, 3, 1)$



Let $y_i = 7$ for all vertices.

Tree 1

(1) Root at v_3 → grow e_{36} → $D_1 = \{e_{36}\}$.

Tree 2

(1) Root at v_3 → Hungarian tree.

(2) Lower y_3 to 3
Raise w_{36} to 4 } → grow e_{35} → $D_2 = \{e_{35}, e_{36}\}$.

Tree 3

(1) Root at v_1 → Hungarian tree.

(2) Lower y_1 to 0 → grow e_{12} → $D_3 = \{e_{12}, e_{35}, e_{36}\}$.

Tree 4

(1) Root at v_4 → Hungarian tree.

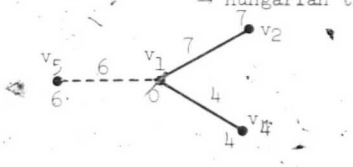
(2) Lower y_4 to 4 → grow e_{14} → $D_4 = \{e_{12}, e_{14}, e_{35}, e_{36}\}$

Tree 5

(1) Root v_5 → Hungarian tree.

(2) Lower y_5 to 6
Raise w_{35} to 1 } → grow e_{15}, e_{12} and e_{14}.

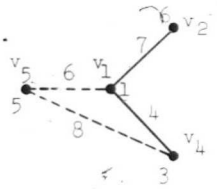
→ Hungarian tree



(3) Lower y_2 to 6, y_4 to 3, and y_5 to 5.
Raise y_1 to 1, and w_{35} to 2. } → grow e_{45}

→ Augment path; $P = \{e_{51}, e_{14}, e_{45}\}$

$D_5 = \{e_{12}, e_{15}, e_{35}, e_{36}; e_{45}\}$



The orthogonality conditions are now satisfied, therefore the solution is optimum.

$y = (1, 6, 3, 3, 5, 7)$ and $w_{35} = 2$, $w_{36} = 4$.

As a check compute W and U.

$W = cx = (7, 6, 10, 14, 8) = 45$

$U = dy + rz + \epsilon w = (2, 1, 2, 1, 3, 1)\ (1, 6, 3, 3, 5, 7)$
$+ 2 + 4 = 45$

3.9 Proof of Optimality

This section will show that the weighted UDCS algorithm obtains an optimum solution. The algorithm is shown to obtain feasible primal and dual solutions which also satisfy the orthogonality conditions, and thus by Theorem B.3, these solutions are optimum.

The primal feasibility constraints are:

(1) $Ax \leq d$

(2) $Tx \leq r$

(3) $Ix \leq \epsilon$

(4) $x \geq 0$.

The dual feasibility constraints are:

(5) $y \geq 0$

(6) $z \geq 0$

(7) $w \geq 0$

(8) $y_i + y_j + \sum_{e_{ij} \in T_k} z_k + w_{ij} \geq c_{ij}$.

The orthogonality conditions are:

(9) $y_i > 0 \Rightarrow A_i x = d_i$

(10) $z_k > 0 \Rightarrow T_k x = r_k$

(11) $w_{ij} > 0 \Rightarrow x_{ij} = 1$

(12) $x_{ij} > 0 \Rightarrow y_i + y_j + \sum_{e_{ij} \in T_k} z_k + w_{ij} = c_{ij}$.

Conditions (1) through (4) follow immediately from the fact that the solution obtained is a UDCS solution. (5), (6), (7) are immediate from the discussion of the algorithm. If y_i is lowered to zero the tree is discarded and another tree is rooted. Each z_k and w_{ij} variable begins at zero and is only allowed to be positive.

Condition (9) was discussed in the previous section. Each tree reduces the total slack of the nonzero vertices, so that eventually (9) is satisfied. z_k is only given nonzero values for blossoms. But it has been shown that each blossom B_k corresponds to the associated $T_k x \leq r_k$ constraint being satisfied with equality. w_{ij} is only allowed to be nonnegative when e_{ij} is in D. Thus the proof reduces to showing (8) and (12).

Proof of (8)

Let $\bar{y}$, $\bar{z}$ and $\bar{w}$ be the final values of the y, z and w

variables. Once a vertex v_i is within a blossom, it is both an inner and outer vertex. The vertex weight of its outer representation v_i is equal to y_i. The vertex weight $y_{i'}$ of its inner representation $v_{i'}$ is equal to $y_i + \sum_{v_i \in S_k} z_k$. When e_{ij} is in T_k for blossom B_k and $v_i \in S_k$, $v_j \notin S_k$, we consider edge e_{ij} as being connected to $v_{i'}$. Therefore before the edge e_{ij} is totally within some blossom we have,

$$y_i + y_j + \sum_{\substack{v_i \in S_k \\ v_j \notin S_k \\ e_{ij} \in T_k}} z_k + \sum_{\substack{v_i \notin S_k \\ v_j \in S_k \\ e_{ij} \in T_k}} z_k + w_{ij} \geq c_{ij}.$$

But once within a blossom y_i and y_j are both reduced by $\frac{1}{2} \bar{z}_k$ for each blossom totally containing e_{ij}, therefore,

$$\bar{y}_i + \bar{y}_j + \sum_{e_{ij} \in T_k} \bar{z}_k + \bar{w}_{ij} \geq c_{ij}.$$

Thus (8) has been proved.

Proof of (12)

In order for e_{ij} to be in the solution (i.e., $x_{ij} = 1$) it must first be in a tree. But in this case,

$$c_{ij} = y_i + y_j + \sum_{\substack{v_i \in S_k \\ v_j \notin S_k \\ e_{ij} \in T_k}} z_k + \sum_{\substack{v_i \notin S_k \\ v_j \in S_k \\ e_{ij} \in T_k}} z_k.$$

Once this equality is obtained it is maintained, as long as $e_{ij} \in D$, so that the proof of (8) goes through for (12) with the inequality

replaced with equality.

Theorem B.5 in Appendix B states that if x_o is a vertex of the polyhedron specified by 1), 2), 3), and 4), then there exists edge weights c such that cx_o is the unique maximum. But the algorithm finds integer solutions for arbitrary edge weights, therefore all vertices have only integer components.

3.10 Summary

In this chapter we have:

(1) characterized solutions to the UDCS problems, and

(2) developed efficient algorithms for finding maximum cardinality or maximum weight UDCS solutions.

The next chapter will use these algorithms in conjunction with a parametric adjustment of edge weights to develop algorithms for the factor problem.

CHAPTER IV

FACTORS

4.1 Introduction

The integer programming formulation of the factors problem is:

1) max cx

subject to:

2) $Ax = d$

3) $x_{ij} \epsilon \{0,1\}$.

The factors problem can obviously be solved as a special case of the ULDCS problem by letting $d^1 = d^2 = d$. However there exists enough extra structure due to the equality constraints that this problem can be solved without using that generality.

First of all, feasibility can be determined by solving,

4) max ϵx

subject to:

5) $Ax \leq d$

6) $x_{ij} \epsilon \{0,1\}$.

If there exists a solution to the equality constraints, it satisfies the inequality constraints and will have as high a cardinality as any other feasible solution to the inequality constraints.

Assuming now that there exists a feasible solution, we wish to determine the maximum weight factor solution. Apply a uniform bias to all edges which is large enough so that if x^1 and x^2 are solution and $|x^1| > |x^2|$ then the weight of x^1 is greater than the weight of x^2. Hence the maximum solution will certainly use the maximum cardinality. But it is easily shown that the solution found in this manner will identify those edges from the graph which constitute the maximum weight

solution to $Ax = d$. Alternately, by using the following transformation on edge weights,

$$c_{ij}' = M - c_{ij},$$

where M is a large integer we can solve the minimization problem:

min cx

subject to:

$Ax = d$,

$x_{ij} \in \{0,1\}$.

A k-factor (regular subgraph) of a graph is a subgraph having the same degree at each vertex. Special k-factors have been studied in the past [3], [22]. For instance, a 1-factor is a perfect matching and a 2-factor is a cycle cover. A feasible solution to the cycle cover problem ($Ax = 2\epsilon$) consists of disjoint cycles.

The question of existence of factors and of finding an optimum factor if one exists will be treated in section 4.2. Previous theoretical work relative to factors will be discussed in section 4.3. In particular a connection will be shown between existence conditions for factors and perfect matching solutions given by Tutte and the concepts of augmenting paths and Hungarian trees. Also the work of Berge [3] and Eastman [8] relative to factors is discussed.

4.2. Existence and Optimality of a d-factor

Unlike the UDCS problem there may not exist a feasible solution to the factor problem. The purpose of this section is to show that the UDCS cardinality algorithm will find a d-factor if one exists, and that the weighted UDCS algorithm combined with a uniform edge weight bias will solve the maximum weight d-factor problem.

Suppose there exists an integer solution x' to the d-factor problem, then the cardinality of x' is $\frac{1}{2}|d|$. But x' is a feasible solution to $Ax \leq d$, and it has as high a cardinality as any other solution to $Ax \leq d$. Finally, any integer solution with that cardinality must be a d-factor, thus the UDCS cardinality algorithm will find a d-factor if one exists.

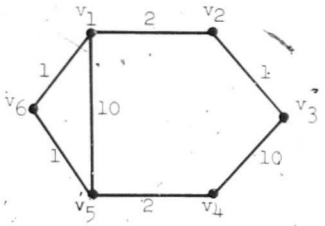
Having determined whether or not there exists a feasible solution to $Ax = d$, we now wish to determine the maximum weight solution. The maximum weight solution to $Ax \leq d$ may not use as many edges as the solution to $Ax = d$, thus the weighted UDCS algorithm will not necessarily yield the maximum weight solution to $Ax = d$.

Example 4.2.1

Let the following graph G be given and find;

(1) the maximum weight solution to $Ax \leq 2\epsilon$, and

(2) the maximum weight solution to $Ax = 2\epsilon$.



The answer to the first problem is,

$$D_1 = \{e_{12}, e_{23}, e_{34}, e_{45}, e_{51}\}$$

which has weight 25.

The answer to the second problem is,

$$D_2 = \{e_{12}, e_{23}, e_{34}, e_{45}, e_{56}, e_{16}\}$$

which has weight 17. In fact, for this example it is easily seen that D_2 is the only feasible solution to $Ax = 2\epsilon$. The optimum solution D_1 achieved a larger weight by using fewer edges.

In order to force the solution of the $Ax \leq d$ problem to also have maximum cardinality, we will use a parametric approach.

Given a graph G with edge weights c and degree constraints d, construct the graph G_N which is G with each c_{ij} replaced by $c_{ij} + N$, where $N = |c|$. We will denote the sum of a set of edges x^1 of G_N as $w_N(x^1)$.

Theorem 4.2.1

If x^1 and x^2 are solution to $Ax \leq d$ and $|x^1| > |x^2|$ then $w_N(x^1) > w_N(x^2)$.

Proof

Since there are $|x^1|$ edges in x^1 and each edge is biased by N, $w_N(x^1) = cx^1 + |x^1|N$. Similarly $w_N(x^2) = cx^2 + |x^2|N$. This results in, $w_N(x^1) - w_N(x^2) = cx^1 - cx^2 + N(|x^1| - |x^2|)$. But $N(|x^1| - |x^2|) > N$ and $cx^1 < N$, so that $w_N(x^2) > w_N(x^1)$. Thus the theorem is proved.

Theorem 4.2.2

If there exists a feasible d-factor solution on G and if x^1 is the maximum weight UDCS relative to $Ax \leq d$ on G_N then x^1 is the maximum weight d-factor on G.

Proof

Let x^2 be any d-factor. x^2 is a solution to $Ax \leq d$, thus $w_N(x^1) \geq w_N(x^2)$, since x^1 is the maximum solution on G_N by hypothesis. x^2 is a maximum cardinality solution to $Ax \leq d$, thus by Theorem 4.2.1, $|x^1| = |x^2|$. Therefore $cx^1 \geq cx^2$, which proves the theorem.

When the cardinality of a solution is constrained to a particular value, in this case $\frac{1}{2}|d|$, a maximization problem can often be

replaced with a minimization problem. Let G_M be G with each c_{ij} replaced by $M-c_{ij}$, where $M = |c|$.

Theorem 4.2.3

If x^1 and x^2 are integer solution to $Ax \leq d$ on G_M, and $|x^1|>|x^2|$ then $w_M(x^2) < w_M(x^1)$.

Theorem 4.2.4

If there exists a feasible d-factor solution on G and if x^1 is the maximum weight UDCS relative to $Ax \leq d$ on G_M then x^1 is the minimum weight d-factor of G.

The proofs of Theorems 4.2.3 and 4.2.4 are very similar to those of Theorems 4.2.1 and 4.2.2. Thus it is seen that both the maximum and minimum weight d-factor solutions can be obtained by use of the weighted UDCS algorithm.

4.3 Previous Theoretical Work Relative to Factors

Berge [3] has studied the factor problem, and in particular its application to the classical four color conjecture. The question of whether four colors are sufficient to color the areas of a map, so that no two adjacent areas have the same color, can be converted into a graphical problem. Namely, can an arbitrary planar 3-regular graph be four colored? A 3-regular graph has degree 3 at each vertex.

The following two theorems show the intimate connection between the factors problem and the 4-color conjecture:

Theorem 4.3.1

A 3-regular graph can be 4-colored if and only if it can be decomposed into three distinct 1-factors (perfect matchings).

Theorem 4.3.2

A 3-regular graph can be 4-colored if and only if it contains

a 2-factor where each component of the 2-factor has an even number of edges.

There exists a technique used by Eastman which very nearly solves the 2-factor problem. Given a graph G with n vertices, a bipartite graph G^1 is formed with $2n$ vertices, by duplicating each vertex of G. The assignment problem [15] is then solved on G^1. The image D in G of this solution is a 2-factor in the sense that $d(i/D) = 2$ for all vertices, however the solution sometimes contains an edge which is used twice. Eastman embedded this approximate solution to the 2-factor problem in a branch and bound structure to search for solutions to the traveling salesman problem.

Tutte has studied factors extensively [22,23,24]. He gives necessary and sufficient conditions for the existence of perfect matchings and d-factors. These conditions are theoretical only, and no efficient algorithm for detection of a factor is given. Also, his results deal only with the existence of a solution and say nothing about the maximum weight case.

We will introduce enough of Tutte's notation in order to state the conditions and then show a connection between these conditions and the augmenting path viewpoint which is the main unifying theme of both Edmonds' work and of this thesis.

S and T are distinguished sets of vertices.

$G(S)$ is the graph obtained from G by removing vertices of S and all edges incident to vertices of S.

$H(S)$ is the number of components of $G(S)$ which contain an odd number of vertices.

$d(i/G)$ is the number of edges of the graph G which are incident to v_i.

$e(S,T)$ is the number of edges with one end in S and one end in T.

$q(S,T)$ is the number of components (C_i) of $G(S \cup T)$ for which, $e(C_i, T) + \sum_{v_k \in C_i} d_k \equiv 1 \pmod 2$. A component which satisfies this condition will be called an odd component.

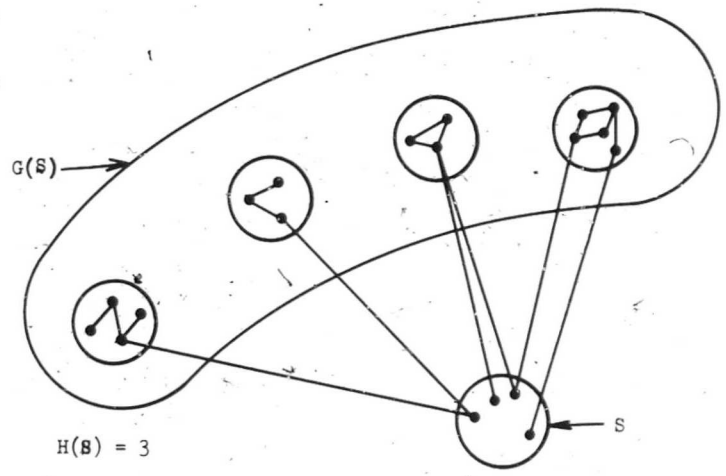
Theorem 4.3.3 (Tutte)

G has no 1-factor (perfect match) if and only if there exists an $S \subset V$ such that $H(S) > |S|$.

Consider Figure 4.3.1 which shows a graph G decomposed relative to a set of vertices S. Suppose that $H(S)$ is greater than $|S|$. If we imagine that each C_i (component of $G(S)$) is maximally matched, then every component which has an odd number of vertices will still have an exposed vertex. Match $|S|$ of the exposed vertices of these components to vertices of S. At this point there exist at least two odd components of $G(S)$ which can not be connected to S, and each vertex of S is matched to some different odd component of $G(S)$.

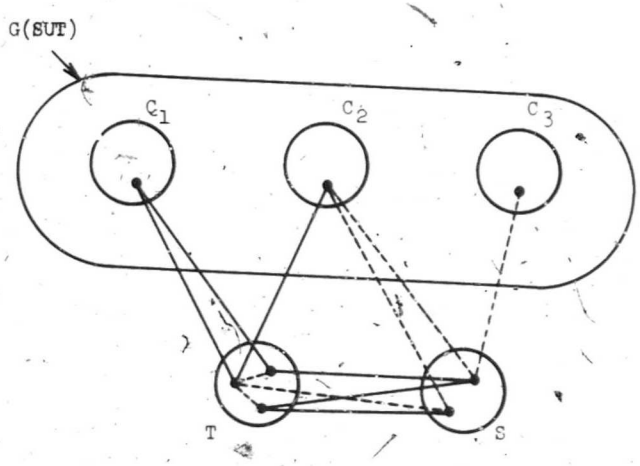
Let us now attempt to construct an augmenting path from one exposed vertex, $v_1 \in C_1$ to another, $v_2 \in C_2$. The only way to move from C_1 to S is by an unmatched edge. Now having arrived at some vertex v_3 in S, the next edge of the alternating path must be a matched edge which by construction terminates in some component C_i of $G(S)$, which has no exposed vertices. Now the return from C_i to S must be on an unmatched edge, since there is only one matched edge between S and C_i, and augmenting paths are edge disjoint.

In general, whenever the path is in S, the next edge must be matched which takes the path to a component which has no exposed vertices. Thus no augmenting path exists and the Berge theorem (Theorem 2.2.1) says that the matching is maximum. But a matching which has



Partition of G Relative to Theorem 4.3.3

Figure 4.3.1



Partition of G Relative to Theorem 4.3.4

Figure 4.3.2

exposed vertices is not a 1-factor, thus the connection between Tutte's condition and augmenting paths is shown. We can go further than this by relating the Hungarian tree concept to the set S defined by Tutte.

Suppose that in solving the cardinality matching problem using Edmonds' algorithm a Hungarian tree J is formed. Let S be the set of inner vertices of the Hungarian tree. The outer vertices of a Hungarian tree are connected only to other vertices of the tree. But from the structure of Hungarian trees we know that there are more outer vertices than inner. Thus this choice of S satisfies the conditions of Theorem 4.3.3.

The construction of the Hungarian tree actually demonstrates a set S as required by Tutte. Further the Hungarian tree implies that a 1-factor doesn't exist. Tutte also states the necessary and sufficient conditions for the existence of a general d-factor.

Theorem 4.3.4

G has no d-factor if and only if there exists two disjoint subsets of vertices S and T, such that

$$(1) \quad \sum_{v_i \in S} d_i < q(S,T) + \sum_{v_i \in T} (d_i - d(i/G(S)))$$

Figure 4.3.2 depicts the partition of a graph G relative to the two sets S and T. Again, as in matching, we will construct a solution on G which is not a d-factor and then argue that no UDCS augmenting paths exist. This, along with Theorem 3.2.1, shows that no d-factor exists.

The constructed solution D consists of:

(1) All edges in $T \times T$.

(2) All edges in $T \times C_i$, for each component C_i of $G(S \cup T)$.

(3) Any maximum solution to C_i which is compatible with the edges defined in (2).

Each vertex v_i in T now has slack,

$$u_i = d_i - d(i/G(S)).$$

Thus in order for vertices of T to satisfy $Ax = d$, they must find

$$\sum_{v_i \in T} (d_i - d(i/G(S)))$$

incidences with vertices in S.

Also by this construction we know that at least $q(S,T)$ components of $G(S \cup T)$ require an additional incidence from S. But condition (1) implies that the degree constraints on S preclude the addition of the required edges.

Continuing the construction by adding as many $S \times T$ edges as possible and further adding at most one edge from each odd component to S. This construction must leave some odd component C_i with an unsaturated vertex, and also with no edges in D between C_i and D.

Suppose an alternating path is started from an unsaturated vertex of C_i using an edge with one end in S. The only way to continue the path from S is by an edge in D, whose other end is in T, or in one of the saturated C_i's. The path having arrived at T or such a C_i can only return to S. Thus no cardinality augmenting path exists and the suggested connection between augmenting paths and Tutte's conditions has been shown.

The T and S sets for UDCS can also be related to the Hungarian tree concept if we restrict ourselves to the case of bipartite graphs. If a Hungarian tree forms during the UDCS problem it has already been shown that some vertex is necessarily unsaturated, thus a d-factor doesn't exist. Let D be a maximum solution to the $Ax \leq d$ problem and let J be a Hungarian tree grown relative to some unsaturated vertex v_i.

Recall the definition of a UDCS Hungarian tree from section 3.3. All edges incident to an outer vertex and not in D are also incident to inner vertices, and all edges incident to an inner vertex and in D are also incident to outer vertices. Also all inner vertices of the tree are saturated by edges incident to T. Thus let S be the set of all inners, and T the set of all outers. Notice that the S and T sets satisfy the constructed solution on the S and T sets defined by Tutte. All edges between a vertex of T, and a vertex of a C_1, are in D, and all edges between vertices of S are not in D.

CHAPTER V

UPPER AND LOWER DEGREE CONSTRAINED SUBGRAPHS (ULDCS)

5.1 Introduction

The integer program for the ULDCS problem is:

$$\max cx$$

subject to:

$$d^1 \leq Ax \leq d^2$$

$$x_{ij} \in \{0,1\}.$$

The description of this problem is similar to that of the UDCS problem (Chapter 3). The proof of the sufficiency of added constraints to insure integrality will not be given, but the discussion of the algorithm which actually achieves the integer solution will be presented. The ULDCS algorithm uses the UDCS algorithm as a subroutine to achieve the maximum solution.

This problem is shown in section 5.2 to contain many of the problems discussed so far. For example, matching, UDCS, and factors are special cases of the ULDCS problem.

In section 5.3 the relationship between the cardinality UDCS problem and the cardinality ULDCS problem will be characterized and an algorithm developed. Section 5.4 develops and describes the algorithm for the weighted ULDCS problem.

5.2 Relationships between Problems

The purpose of this section is to demonstrate the connection between many of the problems studied in this thesis as well as several other classical optimization problems. Many different types of relationships are shown, some of which show containment, others equivalence.

These relationships will be described in the order specified by the circled numerals in Figure 5.2.1.

1) By letting $d^1 = 0$ it can be seen that the ULDCS algorithm solves the UDCS problem.

2) This relationship can be obtained by showing that the UDCS and LDCS problems are equivalent, along with the fact that the UDCS problem is contained in the ULDCS problem.

3) By letting $d^1 = d^2 = d$ the ULDCS algorithm solves the maximum d-factor problem. Also the minimum d-factor problem can be solved by a transformation of edge weights. That is, let $c_{ij}^1 = N - c_{ij}$, where N is a sufficiently large integer.

4) The UDCS and LDCS problems are equivalent.

UDCS problem

max cx

subject to:

$Ax \leq d^2$

$x_{ij} \epsilon \{0,1\}$

LDCS problem

min cx

subject to:

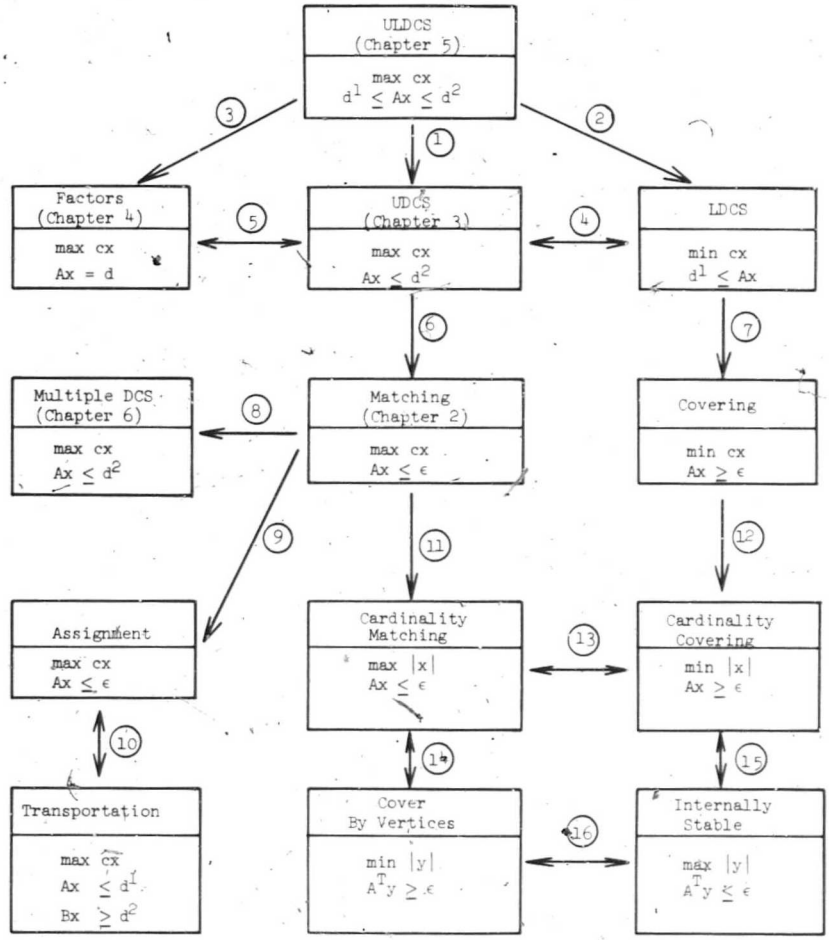
$d^1 \leq Ax$

$x_{ij} \epsilon \{0,1\}$

Subtracting $A\epsilon$ from both sides of $Ax \leq d^2$ and rearranging, results in $A(\epsilon - x) \geq A\epsilon - d^2$,

where $\epsilon - x$ is the set complement of x. Maximizing cx is the same as minimizing $c(\epsilon - x)$, since $c\epsilon$ is a constant. Thus we can write the UDCS problem as:

min $(\epsilon - x)c$



Relationships Between Various Problems

Figure 5.2.1

subject to:

$A(\epsilon - x) \leq (A\epsilon - d^2)_t$

$x_{ij} \epsilon \{0,1\}$

We are now in a position to show that the LDCS problem solves the UDCS problem. Given a UDCS problem specified by a degree constraint d^2, let $d^1 = A\epsilon - d^2$ and solve the LDCS problem. The solution obtained will be the set complement of the desired UDCS solution. Similarly the UDCS algorithm can be shown to solve the LDCS problem. Thus the problems are equivalent.

5) Chapter 4 was devoted to showing this relationship. The edges of the graph were biased by a sufficiently high positive integer to insure a maximum cardinality solution. If the solution was not a factor then it was shown that the problem had no feasible solution. Similarly the minimum factor problem was solved by use of the UDCS algorithm.

6) Letting $d^2 = \epsilon$ shows that matching is a special case of the UDCS problem.

7) Letting $d^1 = \epsilon$ shows that covering is a special case of the LDCS problem.

8) The MDCS and matching problems are shown to be equivalent in Chapter 6. Basically the original graph G is converted to a graph with many more vertices and edges, and the matching algorithm is used to find a solution on this graph. The solution found on this expanded graph is shown to induce the maximum multiple DCS on G.

9) The classical assignment problem of operations research is just the matching problem on a bipartite graph. Thus the matching problem contains the assignment problem.

10) The classical transportation problem is equivalent to the assignment

problem. It is obvious that the assignment problem is a special case of the transportation problem, however the converse is not so obvious. A technique similar to that used in Chapter 6 is utilized by Dantzig [6] to show this relationship. Each vertex of the transportation problem, with supply or demand d_i, is expanded into d_i vertices with supply or demand equal to 1.

11) The cardinality matching problem is solved by the weighted matching problem where $c = \epsilon$

12) The cardinality covering problem is solved by the weighted covering problem where $c = \epsilon$

13) Norman and Rabin [18] have pointed out this relationship between the cardinality covering and matching problems. Given a maximum cardinality matching, which leaves a certain set of vertices exposed, add one edge to each exposed vertex to achieve the minimum covering. Conversely, given a minimum cardinality covering solution, retain one edge from each component of the solution to achieve the maximum cardinality matching.

14) and 15) These relationships follow by formal application of the duality principle of linear programming if noninteger solutions are allowed. When the requirement of integrality is imposed these relationships no longer hold. For instance, the integrality of the matching problem does not guarantee the integrality of the associated dual linear program. At first this seems surprising, but there is a simple intuitive reason for this. The integrality constraints tend to lower the value of the maximum matching since the new problem has more constraints. But by the theory of duality the dual program cost associated with the integerized matching problem equals this lower value. Thus the dual value was

also lowered by these added constraints. But from the viewpoint of the dual problem, the dual cost should increase since we are adding constraints to a minimization problem. Thus these two problems are not the same. The minimum cover of edges by vertices problem has been studied by Lawler [16] and Lorentzen [17]. An efficient algorithm has not been developed for this problem.

16) The minimum cover of edges by vertices can be shown equivalent to the maximum internally stable set of vertices problem by a set complementary argument similar to that used in 4).

5.3 Cardinality ULDCS

In this section we will demonstrate an interesting relationship between the UDCS and ULDCS cardinality problems. This relationship will result in an algorithm for the ULDCS problem which uses the UDCS algorithm as a subroutine.

We will first require several definitions. Partition the vertices of G, relative to a ULDCS solution x and given constraints $d^1 \leq Ax \leq d^2$, into the following types:

(1) v_i is nonfeasible (NF) if $d_i^1 > A_i x$.

(2) v_i is lower saturated (LS) if $d_i^1 = A_i x < d_i^2$.

(3) v_i is upper and lower saturated (ULS) if $d_i^1 = A_i x = d_i^2$.

(4) v_i is feasible (F) if $d_i^1 < A_i x < d_i^2$.

(5) v_i is upper saturated (US) if $d_i^1 < A_i x = d_i^2$.

$p(x)$ is the total infeasibility of x relative to a lower bound constraint d^1.

$$p(x) = \sum_{v_i \in NF} (d_i^1 - A_i x)$$

type path leads to a contradiction.

Assume that there exists a type 1 path P. Form $x^3 = x^2 \oplus P$. x^3 can be shown to be a solution to $Ax \leq d^2$. But $|x^3 \oplus x^1| < |x^2 \oplus x^1|$, which contradicts the assumed minimality of $|x^2 \oplus x^1|$.

Assume that there exists a type 2 path P. Form $x^3 = x^2 \oplus P$. x^3 can also be shown to be a feasible solution to $Ax \leq d^2$. But $|x^3| > |x^2|$ which contradicts the assumed maximality of x^2.

Assume that there exists a type 3 path P. Form $x^3 = x^1 \oplus P$. x^3 can be shown to be a feasible solution to $d^1 \leq Ax \leq d^2$. But $|x^3| > |x^1|$, which contradicts the assumption that x^1 is maximum.

Since none of the three type paths can exist, it can be concluded that $|x^1 \oplus x^2| = 0$, or $x^1 = x^2$, which proves the theorem.

Corollary 5.3.1

If there exists a solution to $d^1 \leq Ax \leq d^2$, let x^1 be one such solution which has maximum cardinality. Let N be the cardinality of the maximum solution to $Ax \leq d^2$. Then $|x^1| = N$.

Proof

From Theorem 5.3.1 x^1 is a maximum cardinality solution to $Ax \leq d^2$, thus $|x^1| = N$.

This is a surprising result, since it seems reasonable that adding the lower bound constraints d^1 could cause the maximum cardinality solution of the $d^1 \leq Ax \leq d^2$ problem to be lower than that of the $Ax \leq d^2$ problem.

This result suggests that the ULDCS cardinality problem can be solved by first solving the UDCS cardinality problem, and then searching for even-length alternating paths which reduce the infeasibility of NF vertices.

The following theorems show that such an approach will locate a maximum ULDCS solution if one exists.

Theorem 5.3.2

Let x^2 be a maximum cardinality solution to $d^1 \leq Ax \leq d^2$, and x^1 a maximum solution to $Ax \leq d^2$. If x^1 is not a solution to $d^1 \leq Ax \leq d^2$, there exists a path $P \subset x^1 \oplus x^2$, such that $p(x^3) < p(x^1)$, where $x^3 = x^1 \oplus P$.

Proof

An even length alternating path P will be constructed from a nonfeasible (NF) vertex v_1 to either a feasible (F) or upper saturated (US) vertex v_2. Consider the effect of using P to form $x^3 = x^1 \oplus P$:

(1) $d(i/x^3) = d(i/x^1)$ for all intermediate vertices of the path P.

(2) $d(2/x^3) = d(2/x^1) - 1$. Since v_2 is either F or US, this has no effect on infeasibility.

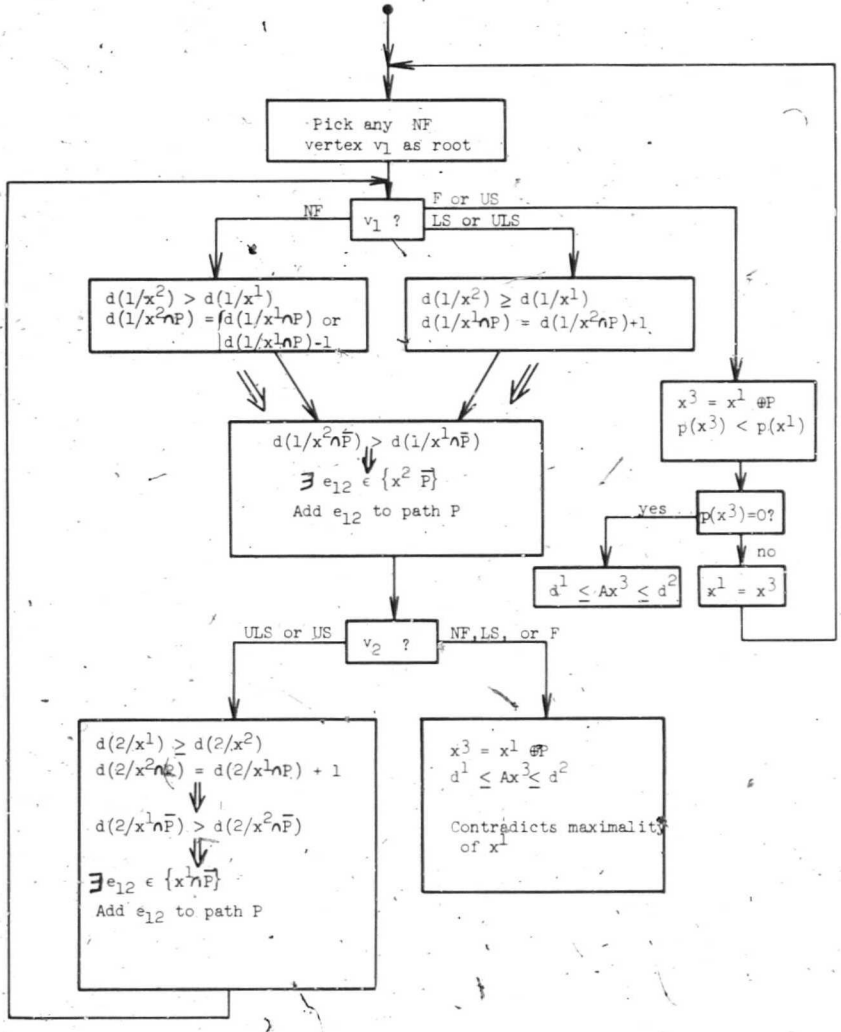
(3) $d(1/x^3) = d(1/x^1) + 1$. Since v_1 is a NF vertex, this results in $p(x^3) = p(x^1) - 1$.

The net effect of using the path is to reduce infeasibility by one.

Figure 5.3.1 is a diagram of the construction which generates the path P. The detailed steps of the proof are given in the figure. Basically at each stage in the construction, either the path can be extended or the proper path exists. Since the number of edges are finite the path will eventually be found.

Theorem 5.3.3

If x^2 is a maximum cardinality solution to $Ax \leq d^2$, and



Constructive Proof of Theorem 5.3.2

Figure 5.3.1

x^1 is a maximum cardinality solution to $d^1 \leq Ax \leq d^2$, then x^1 can be constructed from x^2 using a sequence of paths $\{P_i\}$, where each $P_i \subset x^1 \oplus x^2$.

Proof

Refer again to Figure 5.3.1. Theorem 5.3.2 guarantees the existence of a path P in $x^1 \oplus x^2$ which reduces $p(x^1)$. Now apply Theorem 5.3.2 recursively letting the next x^1 equal the previously generated x^3. Since $p(x^1)$ is a nonnegative integer and it is reduced by 1 for each application of Theorem 5.3.2, we can conclude that Theorem 5.3.3 is true.

Theorem 5.3.4

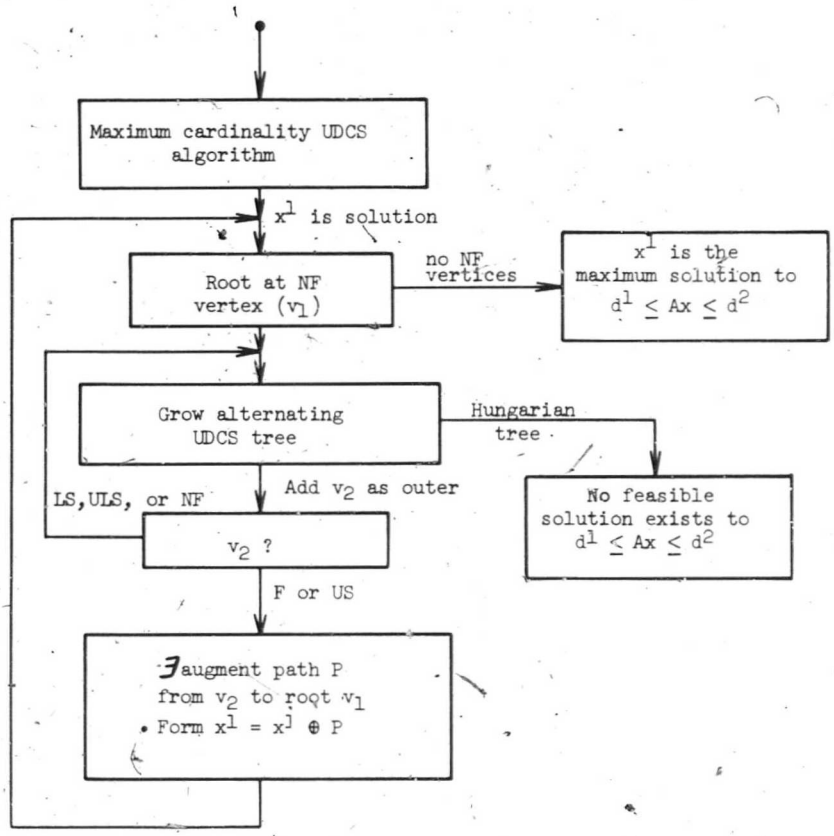
Let x^1 be a maximum cardinality solution to $Ax \leq d^2$ and assume that there exists a feasible solution to $d^1 \leq Ax \leq d^2$. If v_i is NF, then there exists a F or US outer vertex relative to an alternating tree rooted at v_1, and thus a path which reduces $p(x^1)$ by 1.

Proof

By Theorem 5.3.2 we know that there exists such a path to a F or US vertex v_2. The question is, will the UDCS cardinality algorithm find it? Since the path exists, v_2 will be contained in the tree. But then v_2 must be either an inner or outer vertex. If it is an inner vertex there exists a augmenting path which contradicts the assumed maximality of x^1. Thus v_2 is an outer vertex of the tree.

We now have an algorithm (Figure 5.3.2) which finds a maximum cardinality solution to $d^1 \leq Ax \leq d^2$:

(1) Use the UDCS cardinality algorithm to find any maximum



Maximum Cardinality ULDCS Algorithm

Figure 5.3.2

cardinality solution x^1 to $Ax \leq d^2$.

(2) Root a tree at any NF vertex. Use the UDCS tree growing routine to grow until some outer vertex is F or US. The path from this outer vertex to the root is then used to reduce the infeasibility of the solution.

(3) Iterate (2) until the solution is lower feasible or until a Hungarian tree is grown indicating that a solution does not exist.

5.4 Weighted ULDCS

The integer program describing this problem was stated in section 5.1. The linear program which has only integer solution is:

(1) max cx

subject to:

(2) $d^1 \leq Ax \leq d^2$

(3) $Tx \leq r$

(4) $Ix \leq \epsilon$.

(5) $x \geq 0$

The associated dual linear program is:

(6) min $(d^2y - d^1p + rz + \epsilon w)$

subject to:

(7) $A^Ty - A^Tp + T^Tz + Iw \geq c$

(8) $y \geq 0$, $p \geq 0$, $z \geq 0$, $w \geq 0$.

A typical dual equation is given by

$$y_i + y_j - p_i - p_j + \sum_{e_{ij} \epsilon T_k} z_k + w_{ij} \geq c_{ij}.$$

The algorithm is based upon this primal-dual formulation, and the orthogonality conditions which are given below:

(9) $x_{ij} > 0 \Rightarrow y_i + y_j - p_i - p_j + \sum_{e_{ij} \in T_k} z_k + w_{ij} = c_{ij}$

(10) $y_i > 0 \Rightarrow v_i$ is US or ULS

(11) $p_i > 0 \Rightarrow v_i$ is LS or ULS

(12) $z_k > 0 \Rightarrow T_k x = r_k$

(13) $w_{ij} > 0 \Rightarrow x_{ij} = 1$

T is the same matrix as used in Chapter 3 for the UDCS problem. The p variables are the dual variables associated with the lower bound constraints $d^1 \leq Ax$.

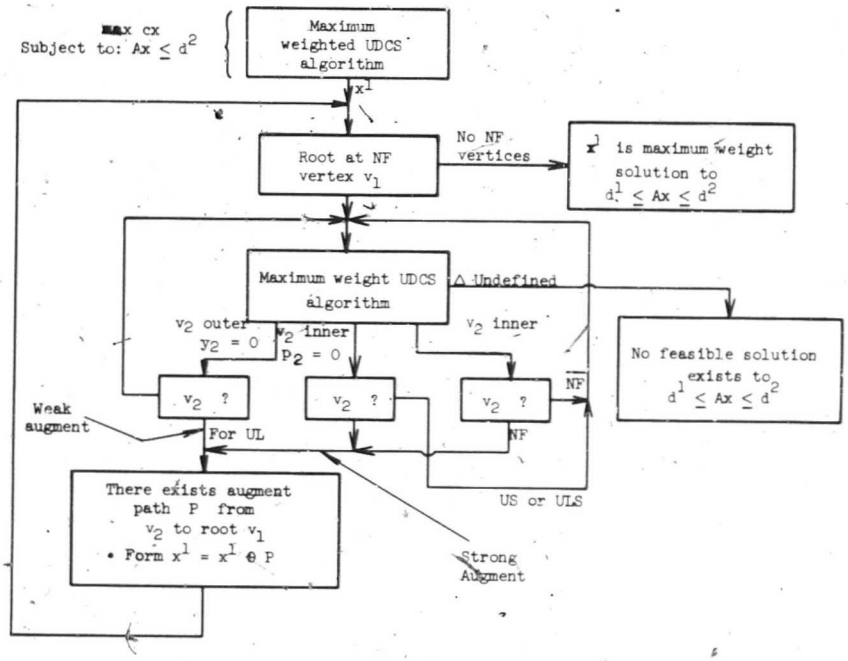
The ULDCS algorithm uses the UDCS algorithm as a subroutine. The algorithm has two major phases. During phase 1 the UDCS algorithm finds the maximum UDCS solution. During phase 2 a succession of paths are found which systematically reduce the infeasibility of the phase 1 solution relative to the lower bound. Figure 5.4.1 is a flow diagram of the algorithm.

At the end of phase 1, all orthogonality conditions are satisfied, but the solution is not primal feasible. During phase 2 the only condition which is allowed to become nonorthogonal is (11). The p variables are allowed to increase on NF vertices in order to search for paths that reduce infeasibility. If all the nonfeasible vertices can be made feasible, condition (11) will again be satisfied. Thus the convergence of the algorithm depends upon depletion of these nonfeasible vertices.

The major steps of the algorithm are summarized below:

(1) Solve the UDCS algorithm yielding x^1.

(2) Root at any NF vertex v_r and grow a tree with the UDCS algorithm. If a path is found which reduces the $p(x^1)$, it is implemented and the tree discarded. Otherwise the tree eventually becomes Hungarian.



Maximum Weight ULDCS Algorithm

Figure 5.4.1

(3) Basically the same events occur in the ULDCS Hungarian process, as in the UDCS case. A Δ is calculated which is just large enough to cause some new event to occur. If Δ is undefined then there does not exist a feasible solution to the problem.

(4) If there exists a solution, the algorithm will find a succession of paths, each of which will reduce the infeasibility by at least one. The algorithm is arranged so that as soon as the nonfeasible vertices are eliminated, the orthogonality conditions are satisfied.

The Hungarian process for the ULDCS algorithm will now be described. There are both a p and y variable associated with each vertex v_i. At most one of these two variables is nonzero at any time. One of the following events occurs whenever the Hungarian process is used:

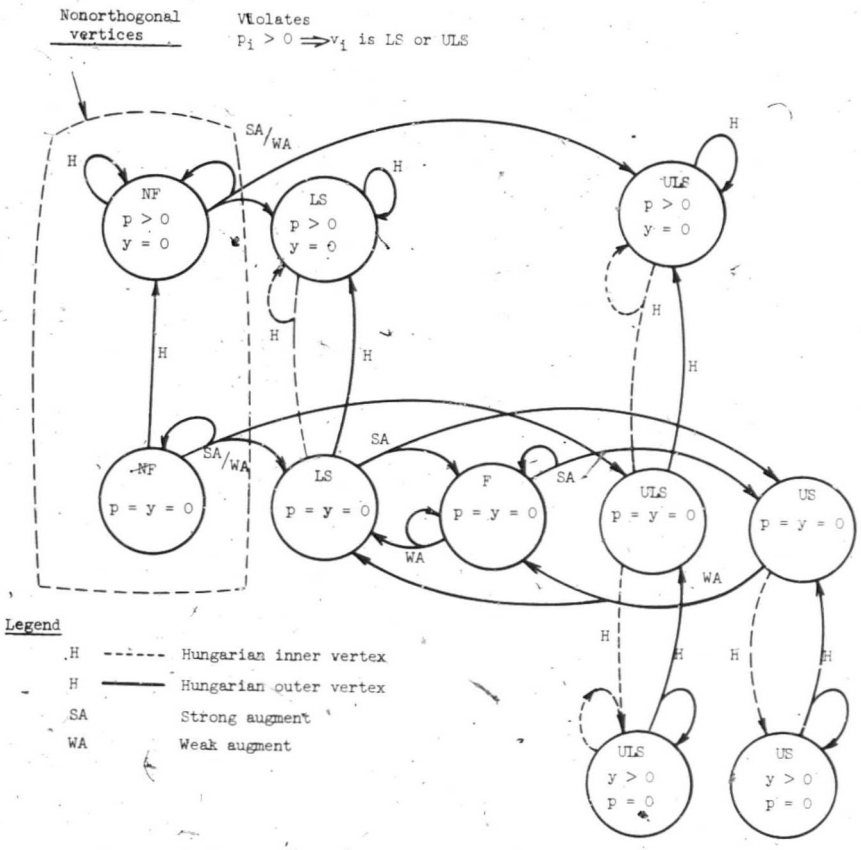
1) Bring a new edge into G^*.

$$\Delta_1 = \min \{\min_{E_1} (c_{ij} - y_i - y_j + p_i + p_j), \min_{E_2} (w_{ij})\}$$

$$E_1 = \{e_{ij} | v_i \text{ is outer, } v_j \text{ is external, } e_{ij} \notin D\}$$

$$E_2 = \{e_{ij} | v_i \text{ is inner, } v_j \text{ is external, } e_{ij} \in D\}$$

If a new edge is brought into G^*, the UDCS algorithm attempts to extend the tree. When the tree can no longer be extended, and no augmenting paths exist, the tree is again Hungarian.



Vertex State Transition Diagram (ULDCS Algorithm)

Figure 5.4.2

2) Form a blossom.

$\Delta_2 = \min \left\{ \min_{E_1} \frac{1}{2}(c_{ij}' - y_i - y_j + p_i + p_j), \min_{E_2} \frac{1}{2}(w_{ij}) \right\}$

$E_1 = \{e_{ij} | v_i \text{ and } v_j \text{ outer vertices}, e_{ij} \notin D\}$

$E_2 = \{e_{ij} | v_i \text{ and } v_j \text{ inner vertices}, e_{ij} \in D\}$

If a blossom forms it is shrunk and the algorithm continues to grow the tree. A vertex within a blossom can be used as an inner or outer vertex. If as a result of the Hungarian weight adjustment, y_i becomes zero for a blossom vertex that is not ULS then an augmenting path exists. For if the vertex is F or US there exists a weak augmenting path, but if it is LS, F, or NF there exist a strong augmenting path. These various cases are shown in Figure 5.4.2.

3) Detect a weak augment.

$\Delta_3 = \min (y_i)$, for all vertices that are outer and F or US. Notice that $p_i' = 0$ for vertices that are F or US therefore the minimization is over vertices with positive values of y_i. If the Δ found by the Hungarian process is due to a zero weight outer vertex v_i, then the classification of that vertex is important. If it is a F or US vertex then an edge can be removed from v_i without effecting the orthogonality at v_i. But using the weak augmenting path from v_i to the root reduces the non-feasibility of the root, and thus aids convergence.

4) Detect a strong augment.

$\Delta_4 = \min (p_i)$, for all vertices that are inner and NF.

If the minimum Δ is due to an inner vertex v_i, then there may exist a strong augmenting path from v_i. If v_i is NF, LS, or F the augment can be used. This decreases the infeasibility of the root.

5) Expand a blossom.

$\Delta_5 = \min \frac{1}{2}(z_k)$, $\forall$ k such that s_k is inner or t_k is outer. If the minimum Δ is associated with a blossom B_k, the blossom must be expanded and all the associated changes to the graph made just as for the matching and UDCS algorithms.

6) No feasible solution exists.

When none of the Δ_i above are defined it can be shown that no feasible solution exists.

For cases 1-5 above $\Delta = \min (\Delta_1, \Delta_2, \Delta_3, \Delta_4, \Delta_5)$. The dual variables are adjusted based on this value of Δ, and the appropriate step is taken. For outer vertices y_i is lowered or p_i is raised by Δ while for inner vertices y_i is raised or p_i is lowered by Δ.

Figure 5.4.2 demonstrates the effect of the Hungarian process and augmenting paths upon the vertex states. In particular note that the nonorthogonal states can never be entered from orthogonal states.

This chapter has described the relationship between the various problems studied in this thesis. Also efficient cardinality and weighted ULDCS algorithms were presented along with an interesting connection between the UDCS and ULDCS problems.

CHAPTER VI

MULTIPLE DEGREE CONSTRAINED SUBGRAPHS (MDCS)

6.1 Introduction

The multiple degree constrained subgraph (MDCS) problem [10] can be stated as an integer program as follows:

(1) max cx

subject to:

(2) $Ax \leq d$

(3) x_{ij} integer.

In the ULDCS problem an edge could be used only once, but for the MDCS problem an edge can be used any integer number of times to satisfy the degree constraints.

Section 6.2 will show the equivalence of the MDCS and matching problem and also show that the UDCS algorithm can be used to solve the MDCS problem. Previous theoretical results by Tutte [22] will be discussed in section 6.3.

6.2 Equivalence of MDCS and Matching

This section will show the equivalence of the MDCS and matching problems. Matching is a special case of MDCS as can be seen by letting the degree constraints be all ones. It remains to be shown that the matching problem implies the MDCS solution.

Let G be a given graph with vertex set V and edge set E. Let x be a vector of nonnegative integers, where x_{ij} is the number of times e_{ij} is to be used in the solution.

We show this equivalence by constructing a new graph H from

G, where each vertex v_i in G becomes d_i separate vertices in H. The matching on H is used to identify the MDCS solution on G.

Construction 1

Each vertex v_i in G has a degree constraint d_i. Let the vertex set of H be formed by expanding v_i into d_i separate vertices $(v_{i1}, v_{i2}\ldots)$, each with degree constraint 1. If e_{ij} is in G then there exists an edge between each image of v_i and v_j in H. Thus e_{ij} becomes $d_i d_j$ separate edges in H. This set of edges is called E_{ij}.

Construction 2

If x is a solution to the MDCS problem then construct y, the image of x in H as follows:

(1) Choose any $x_{ij} > 0$. Assign any x_{ij} of the edges of E_{ij} to y in such a way that no two edges touch the same vertex.

(2) Continue for each $x_{ij} > 0$, thus generating a matching y on H.

Since $\sum_{\forall j} x_{ij} \leq d_i$, it is clear that enough vertices exist to complete the above construction. Further, if at any stage $x_{ij} > 0$, the edges of E_{ij} connect every v_{ik} to every v_{jl}, thus the construction can always be made.

Conversely if y is any matching of H, let $x_{ij} = |E_{ij} \cap y|$. $\sum_j x_{ij} = \sum_j |E_{ij} \cap y| \leq d_i$, since y is a matching.

For the above constructions, if $e_{kl} \in E_{ij}$, then $c_{kl} = c_{ij}$. Thus the cost of solutions on G and H connected by the above construction are equal.

Theorem 6.2.1

x is a maximum solution to the MDCS problem on G

if and only if

y is a maximum solution to the matching problem on H, where x and y are related by the above constructions.

Proof

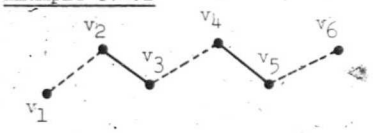
Suppose y is not a maximum matching of H, then there exists a z such that $w(z) > w(y)$. Let z^1 be the image in G of z by construction 2. x is the image of y from the statement of the theorem.

Now, $w(y) = w(x)$ and $w(z) = w(z^1)$, thus $w(x) < w(z^1)$. So that if z^1 is a feasible solution the "only if" part of the theorem is true. By construction 2, $d(i/z^1) = \sum_{\forall j} |E_{ij} \cap z| \leq d_i$ and thus z^1 is feasible.

If x is not maximum, then there exists a MDCS solution z such that $w(z) > w(x)$. Let y and z^1 be the images in H of x and z. $w(x) = w(y)$ and $w(z) = w(z^1)$, thus $w(y) < w(z^1)$. But by construction 2, any MDCS solution becomes a feasible matching of H, and the theorem is proved.

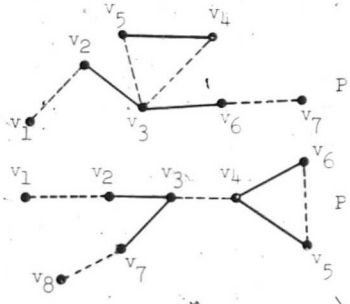
The augmenting "path" concept for MDCS problems is more complicated than any of the previous problems. For matching an augmenting path was a simple path. In the UDCS problem an augmenting path was edge disjoint but vertices could occur more than once. For the MDCS problem augmenting "paths" are not even edge disjoint.

Example 6.2.1



Matching

$P = \{e_{12}, e_{23}, e_{34}, e_{45}, e_{56}\}$



UDCS

$P = \{e_{12}, e_{23}, e_{34}, e_{45}, e_{53}, e_{36}, e_{67}\}$

MDCS

$P = \{e_{12}, e_{23}, e_{34}, e_{45}, e_{56}, e_{64}, e_{43}, e_{37}, e_{78}\}$

One way to characterize MDCS augmenting paths is by the use of Theorem 6.2.1. If a MDCS solution G is not maximum then the associated matching on H is not maximum. But then there exists a matching type augmenting path P in H. The image in G of a matching augmenting path P in H is then a typical MDCS augmenting path.

By the previous results we can now solve the MDCS problem:

(1) Construct H from G by construction 1.

(2) Solve the maximum matching problem on H to obtain the solution y.

(3) Construct x, the image of y in G, by construction 2. By Theorem 6.2.1, x is a maximum MDCS solution.

This raises two questions:

(1) What is the asymptotic growth of this algorithm?

(2) Is there a more direct algorithm?

The growth of the matching algorithm (n^4) is based upon the number of vertices (n) of the graph. The graph H has $|d|$ vertices, thus its growth is bounded by $|d|^4$.

This proof of equivalence between matching and MDCS problems is an interesting result, but it does not result in a good bound on the growth of the algorithm. A more direct method of solution is by use of the UDCS algorithm. Allow each edge e_{ij} of G to be expanded into $\min(d_i, d_j)$ edges. Now x_{ij} for each of these edges can be restricted

to be one or less without losing any solution to the original problem. The growth of the MDCS problem is thus the same as that of the UDCS problem, that is $|d|\ (n^3)$.

A graph with no odd cycles is called a bipartite graph. This class of graphs is important in that their incidence matrices are unimodular, and thus the difficulty of noninteger solutions to matching and DCS problems does not exist.

A graph with no even cycles also has interesting properties. Each component must have either zero or one odd circuit, since if two odd cycles exist then an even circuit must also exist. We will now show that the edges that are in a maximum MDCS solution (i.e., $x_{ij} > 0$) form a subgraph that contains no even cycles.

Many of the theoretical results associated with matching and DCS problems are complicated by the nonuniqueness of solutions. This can be caused by equal edge weights or by sets of edges having equal total weight.

There are two methods which can be used to handle nonuniqueness. One method, which was used in Chapter 3, is to select one of many equal solutions based on the minimality of the symmetric difference with a reference solution. A more direct method of breaking ties will be discussed here.

From an algorithmic viewpoint it is usually sufficient to find any maximum solution, although the existence of ties may allow further optimization relative to some other constraint. When it is necessary to explicitly exclude ties, the following procedure may be used:

(1) Arbitrarily order the edges in E, $E = \{e_1, e_2, \ldots, e_n\}$.

(2) Add 2^{-i} to the i^{th} edge.

We are assuming that the original weights are integers. Obviously no two edges now have the same weight. Sets of edges which did not have the same integer sum before are not equal now, since the total amount added to all edges is less than one. Now consider two sets which had the same integer sum. Since the sets are not identical examine their symmetric difference. The set which has the first edge relative to the ordering, now has the largest weight. This is true since,

$$2^i > \sum_{k=i+1}^{n} 2^k$$

Thus no two sets have the same weight.

Lemma 6.2.1

Let x be the unique solution to a MDCS problem. G^1 is the subgraph of G containing the edges for which $x_{ij} > 0$. Then there does not exist an even circuit in G^1.

Proof

Let $C = (e_1 e_2 \ldots e_n)$ be any even circuit in G^1.

Let $E_{odd} = \{e_i | i \text{ is odd}\}$ and $E_{even} = \{e_i | i \text{ is even}\}$.

$w(E_{odd}) = \sum_{e_i \in E_{odd}} c_i$ and $w(E_{even}) = \sum_{e_i \in E_{even}} c_i$.

Now if we assume that all ties have been broken it follows that $w(E_{odd}) \neq w(E_{even})$. Assume $w(E_{odd}) > w(E_{even})$.

Let $\Delta = \min_{e_i \in C} \{x_i\}$

Now an alternate solution can be constructed as follows:

$$x_i^1 = x_i + \Delta \qquad \text{for } x_i \in E_{odd}$$

$$x_i^1 = x_i - \Delta \qquad \text{for } x_i \in E_{even}$$

First notice that x^1 is a feasible solution to the original degree constraints, since at each vertex of the circuit Δ is added and subtrated the same number of times. But, $w(x^1) = w(x) + \Delta(w(E_{odd})) - \Delta(w(E_{even}))$. Hence $w(x^1) > w(x)$, which contradicts the assumed maximality of x. Thus no even cycles can exist in the unique solution of a MDCS problem.

6.3 Tutte's Conditions

Tutte has studied a special case of this problem. Namely, when does a solution exist which satisfies the degree constraints with equality. He calls a graph G d-soluble if there exists a solution x such that $Ax = d$. The notation used here was introduced in section 4.3.

Theorem 6.3.1 (Tutte)

G is not d-soluble if and only if there is a subset S of V such that,

$$\sum_{v_i \in S} d_i < q(S) + \sum_{v_i \in T(S)} d_i .$$

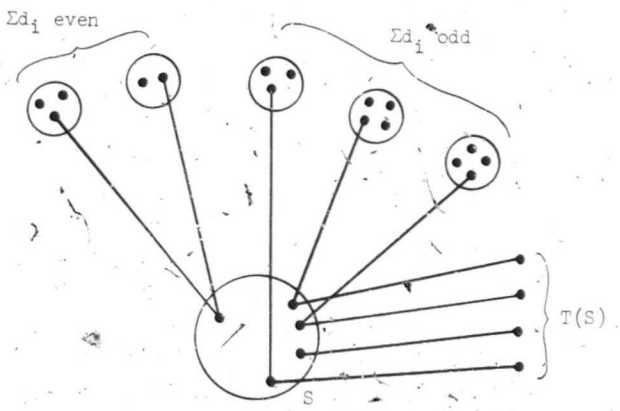
Only the sufficiency of this condition will be discussed here. The necessity is considerably more difficult.

$q(S)$ is the number of components of $G(S)$ which have more than one vertex and for which the total degree constraints are odd. $T(S)$ is the set of components of $G(S)$ which have only one vertex.

Consider the situation if such an S exists (Figure 6.3.1).

Assume that there exists a maximum solution to each component for which the degree constraints have an odd sum. Each of these $q(S)$ components must use an edge from S to satisfy the constraints with equality. Similarly the vertices in $T(S)$ require $\sum_{v_i \in T(S)} d_i$ edges from S in order to satisfy the constraints with equality. But the condition of this theorem states that the total degree allowable to vertices of S is less than the sum of those two numbers. Thus there is no way to complete this solution to satisfy the constraints with equality. A more complete discussion of this would require a further characterization of MDCS augmenting paths.

The above characterization is theoretical and does not suggest an efficient algorithm to determine feasibility and identify the solution if one exists. We can solve the d-soluble problem by use of the MDCS cardinality algorithm just as the UDCS algorithm was used to solve the factor problem (Chapter 4). Also the maximum (or minimum) weighted solution can be found by a parametric approach as in the case of factors.



Partition of G relative to Theorem 6.3.1

Figure 6.3.1

CHAPTER VII

APPLICATIONS AND FURTHER RESEARCH

7.1 Parallel Computation

Karp [13], and Reiter [20], have developed a model for parallel computation of recursive algorithms. A parallel computation is viewed as a directed graph in which a vertex J_i represents a computational job to be performed on data provided on the incoming edges. The output of the computation is placed on the outgoing edges as data to some other computation.

The minimum time for which the jobs can be recursively solved is called T. Associated with this time T is an activity schedule for each computation; job J_i is active from t_{i1} to t_{i2}.

The question which naturally arises from this formulation is, what is the minimum number of computers required to execute this given schedule?

We first form the compatibility graph C. The vertices are the jobs J_i. An edge e_{ij} exists between jobs J_i and J_j, if and only if the activity times of J_i and J_j are disjoint.

The above problem can now be stated as, find the minimum cardinality cover of the vertices of C by complete subgraphs of C. Since if a subgraph is complete, relative to a particular subset of jobs, then all those jobs can be executed on a single computer.

In this form it can now be seen that this problem is an instance of the general covering problem [1], for which no efficient algorithm exists.

However, the matching problem can provide an upper bound on the number of computers required. Suppose there are N jobs. Let M be a maximum cardinality matching on C. Assign one computer to each pair of jobs identified by the edges in the matching, and one computer to each job which is not incident to an edge in M. The total number of computers required by this assignment is $N - |M|$. But since $|M|$ is maximum, $N - |M|$ is minimum. In general this assignment provides an upper bound on the number of computers required.

In the following special cases the matching solution will be optimal.

Case 1.

Due to other considerations no more than two jobs can be assigned to a computer.

Case 2.

If the graph C possesses any of the following special characteristics:

(1) C has no cycles.

(2) C is a bipartite graph.

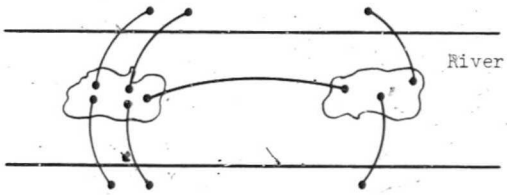
(3) C has no 3-cycles.

A more abstract formulation of this problem can be given, which may be suggestive of application in other areas. Namely, find the minimum rank equivalence relation which is contained within a given reflexive, symmetric relation.

7.2 Routing Problems

It was Euler who first solved the classical Königsberg bridge

problem [4], where seven bridges are given with the topology shown in Figure 7.2.1.



The Königsberg Bridge Problem

Figure 7.2.1

The problem, studied in 1736 by Euler, was this: is it possible to describe a continuous route which traverses each bridge exactly once? The following statements give the answer for a more general problem, and specifically states that the Konigsberg bridge problem has no solution.

An Euler graph is a graph in which all vertices are of even degree.

An Euler circuit is a cycle which uses each edge of a graph exactly once.

It can be shown that a connected graph G contains an Euler circuit if and only if G is an Euler graph.

A problem closely related to the existence of an Euler circuit is the Chinese Postman's Problem [4]. Here we wish to find a minimum weighted cycle of a graph which uses each edge of the graph at least once. It is easily seen that, in general, some edges of G must be

duplicated in order to satisfy the Euler conditions. Thus the problem reduces to finding the minimum weighted set of edges which when added to G causes each vertex to have even degree. An algorithm for this problem, and its theoretical justification, is given in Appendix C. Basically the matching algorithm (Chapter 2) and the shortest path algorithm [14] are combined to solve this problem.

Another related problem is to find the minimum p chains which together use each edge at least once. A chain is a continuous route with distinct end points. This problem requires the determination of the best k-cardinality matching of a graph together with the shortest path algorithm.

The application of these problems to the selection of optimum routings is immediate. Let each edge be a particular task to be accomplished and the weight of the edge be the cost of tranversing this edge. Then the optimum route for a single serviceman to take in order to accomplish each task at least once and return to the starting point is given by the solution to the Chinese Postman's Problem. If k servicemen are available and they do not have to return to the starting point, the minimal k chain solution is the optimum routing strategy.

Another application might be in the testing of the readiness of a communication network. A test message could be sent from a particular terminal and routed over each link of the system. Return of the message to the originating terminal without error, would give an indication of the present status of the network.

7.3 Interconnection and Scheduling

A general class of interconnection problems [4] can be formulated as degree constrained subgraph problems. For example, suppose a number of central computer complexes are to be interconnected so that they will be capable of sharing their various capabilities when time dependent demands occur. Assume that each computer C_i based on projected demands, and its own memory capacity, computing speed, and input/output capabilities, is to be directly linked to at least d_i other computers. Suppose that the cost of providing the connection between computer C_i and computer C_j is a real positive number c_{ij}.

The interconnection problem is to design the system of interconnections having minimum total cost and which satisfy the demands d_i of the various stations. Let A be the computer versus connecting link incidence matrix, and assume that any of the connections are feasible. x is a $\{0,1\}$ vector where $x_{ij} = 1$ implies computer C_i and computer C_j are to be connected. This problem can be stated as an integer program as follows:

$$\min cx$$

subject to:

$$Ax \leq d$$

$$x_{ij} \in \{0,1\}$$

This is the lower degree constrained subgraph problem (LDCS) which has been shown to be equivalent to the UDCS problem in Chapter 5.

In the same context consider that the interconnection of the computers has already been specified and we wish to operate the system.

Here we identify a cost c_{ij} when the connecting link between computers C_i and C_j is in use. Now as the demands on the various computers fluctuate the degree constraints d_i will vary. The introduction of an upper bound on the degree can be used to prevent a computer from becoming saturated. Thus the general ULDCS problem could be used to dynamically schedule this computer system.

The same comments can be made relative to the design or scheduling of any system. For instance, the vertices may be power stations, communications centers, or states of a finite automaton.

7.4 Communication Network Capacity

One method which is used to rate a multiple line communications network is the maximum call miles that can be simultaneously connected. The communications network can be viewed as a weighted graph where each vertex t_i is a terminal. Each terminal has an upper bound d_i on the number of calls which can simultaneously be handled. If two terminals, t_i and t_j, can directly communicate, then there is an edge e_{ij} of the graph between the associated vertices. The edge weight m_{ij} is the number of miles between terminals t_i and t_j.

A closely related problem, which is the linear programming dual of this problem, is the minimum line length to provide the maximum call mileage.

The problem of maximum call miles can be stated as an integer program as follows:

max mx

subject to:

$Ax \leq d$

x_{ij} integer

Where **A** is the terminal-branch incidence matrix and m is the branch mileage vector between connected terminals. The maximum call traffic through each terminal is specified by the d vector. In this form it can be seen that this is a MDCS problem (Chapter 6).

This problem has been studied at Bell Laboratories by J. W. Suurballe [21]. He has developed algorithms for this problem and an extension where there is a separate upper limit on the number of calls between each pair of terminals.

The integer programming formulation of that problem is:

max cx

subject to:

$Ax \leq d$

$Ix \leq f$

x_{ij} integer,

where f_{ij} is the maximum allowable number of calls between terminals t_i and t_j.

His algorithms begin with a feasible primal, but an infeasible dual. The initial selection of primal and dual solutions is simplified, since the mileage between terminals satisfy the triangle inequality.

If G^1 is the subgraph defined by the set of edge e_{ij} for which $x_{ij} > 0$, then the optimum solution to a MDCS problem has the following characteristics (see Chapter 6):

(1) No even cycles exist in G^1

(2) Each component of G^1 has either 0 or 1 odd circuit. Suurballe uses this characterization to find a good initial primal solution.

The MDCS algorithm described in Chapter 6, where each edge e_{ij} is expanded into min (d_i, d_j) edges, can be used to solve the above problem. The MDCS algorithm can also be extended to the edge capacitated case by expanding each edge into min (d_i, d_j, f_{ij}) edges. The algorithms of Chapter 6 differ from the Suurballe algorithms in that the primal and dual are always feasible. This reduces the number of cases which must be considered in the algorithm.

7.5 Conclusions and Further Research

The purpose of this research was to characterize the solutions for the DCS class of combinatorial problems, to develop efficient algorithms for finding those solutions, and to find engineering applications. These goals have all been realized to some extent, but much further research is now suggested. There are three classifications for this further effort:

(1) Further investigation into DCS problems.

(2) Identification of other applications.

(3) Use of these techniques to solve other combinatorial problems.

(1) DCS investigations

In this catagory would fall such efforts as further programming, and specification of initial conditions for each of the algorithms. Also

a parametric study of the optimum k-edge solutions for each of the algorithms would be of interest.

(2) Other applications

Some applications have been discussed but many more are sure to follow with further investigation. In order to determine the applicability of these or related techniques to a particular application, an intimate knowledge of the application is required. A few of the areas which could be investigated in more detail are:

(1) Design of integrated circuit layouts with prescribed regularity of connections.

(2) Design and scheduling of communications networks.

(3) Interconnection philosophy of parallel processing computer structures.

(3) Other theoretical extensions

The linear programming dual to noninteger matching, namely the minimum cover of edges by vertices, has not been efficiently solved. it is probable that investigations of this problem, using the characterization given by Edmonds [11] and Berge [3], would lead to an efficient algorithm.

Most of the problems which have been discussed are special cases of the general covering problem [1] where the covering subsets are restricted to having exactly two elements. A logical step in approaching the general problem would be investigation of subsets of three elements.

The addition of other constraints to the solution could be

investigated. For instance, we might ask that the solution subgraph be a forest, a spanning tree, have k components, or possess some other restricted topology.

APPENDIX A

CARDINALITY MATCHING PROGRAM

The matching problem of Chapter 2 was programmed for the IBM 7090 using the MAD language (January 6, 1967 version). An outline of the program functions, data structures, a listing of the program, and computational experience are contained in this appendix.

This program was used to study a series of graphs having from 5 to 35 vertices. Primarily, complete graphs were used, since they cause the algorithm to do the maximum amount of work. The results of this study are shown in Figure A.1.

For the graphs studied the maximum computation time grew approximately as $n^{2.5}$, with increasing vertices. This is conjectured to be due to the fact that the predicted n^4 growth is only experienced by a small portion of the algorithm. It would require a much larger value of n to be able to determine the asymptotic value of growth.

The matching program consists of 6 parts whose main functions are described below:

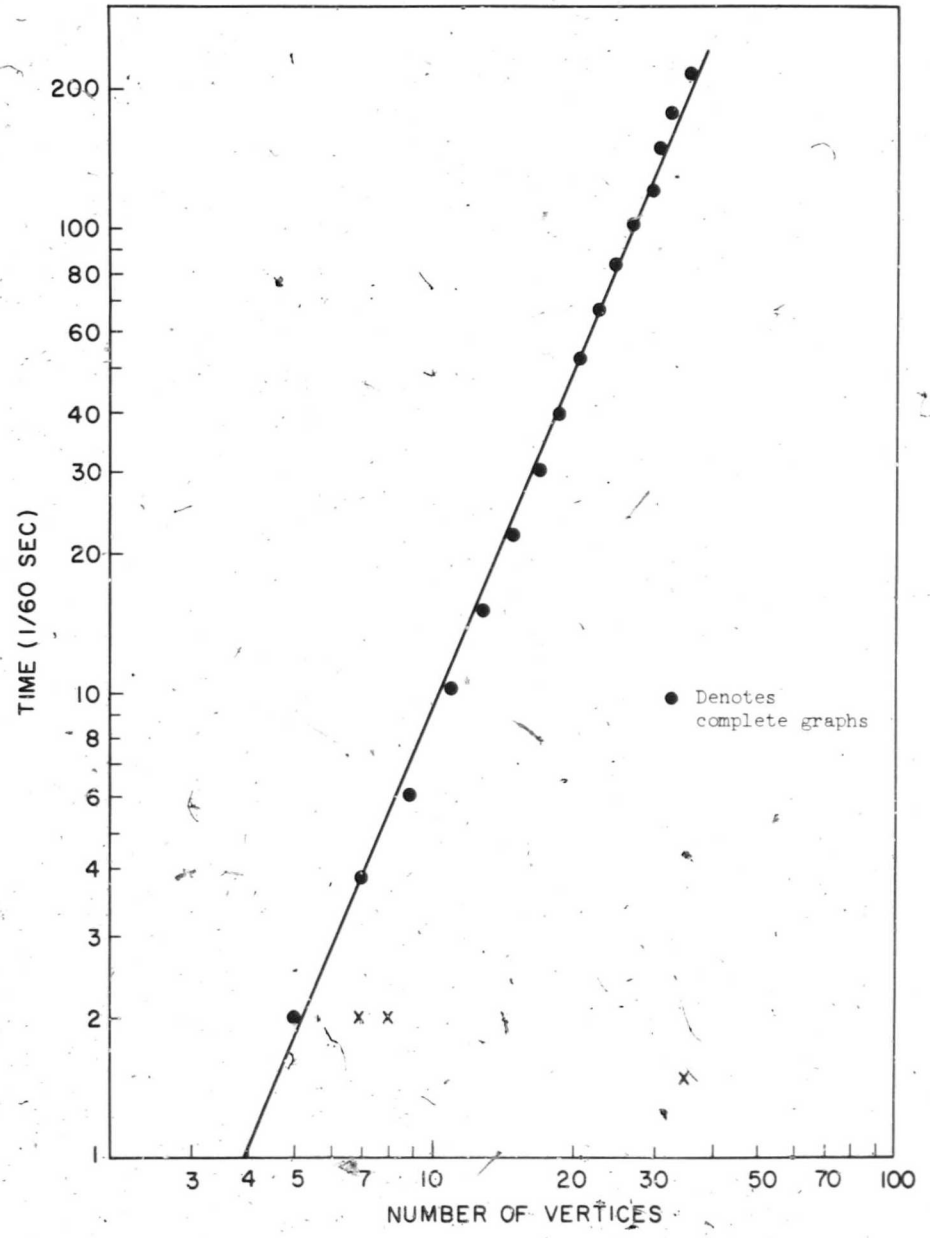
(1) Main

(a) Finds a root for the next tree.

(b) Finds an edge between two simple vertices such that the tree can be extended.

(c) Directs the flow of the program.

(d) Determines completion of the algorithm and prints the results.



Matching Computational Experience

Figure A.1

(2) Grow

(a). Adds two simple vertices and two edges to the tree.

(b) Records backtrace information.

(3) Blossom

(a) Backtraces two paths from outer vertices to the root to determine the odd circuit HP_k of a blossom B_k.

(b) Reclassifies blossom vertices as outer vertices.

(c) Updates B, T, and M vectors.

(d) Computes vertex set BV(k) of blossom B_k.

(4) Augment

(a) Backtraces path to root.

(b) Adjusts matching along path.

(5) Expand

Expands all blossoms into simple vertices.

(6) Hungarian

Tags vertices of Hungarian tree so they will not enter into subsequent trees.

The important data structures are listed below:

N - An integer from 1 - 35. This input determines the size of the graph

A_i - {0,1} vector of N bits. The vertices connected to vertex v_i. These inputs determine which edges are in the graph.

C(I) - The classification of simple vertices.

E is exposed.

H is Hungarian.

IN is inner.

Out is outer.

SOUT is scanned outer.

NU is neuter. This is a vertex which is matched but is not in a tree.

T(I) - Backtrace information for the tree. I runs from 1 to 2N. Blossoms are numbers from N+1 to 2N. One word per vertex or blossom.

B(I) - Outermost blossom containing simple vertex I. If I is not in a blossom then B(I) = I. If a blossom K is an outermost blossom then B(K) = K. If a blossom is not being used then B(K) = 0.

M(I) - Matching information. One word per vertex or blossom.

HP(K,L) - Blossom odd circuit storage. N words reserved from each blossom. Each word is a vertex or blossom of the circuit.

BV(K) - The simple vertices of blossom K.

LP(K) - The length of the circuit of blossom K.

```
$ COMPILE MAD, EXECUTE, PRINT OBJECT.

MAD (06 JAN 1967 VERSION) PROGRAM LISTING ... ... ...


                DIMENSION A(36),C(36),T(72),M(72),MP(36*36),BV(72),LP(36)
                DIMENSION P(36), Q(36), B(72)
                NORMAL MODE IS INTEGER
      L0        READ AND PRINT DATA
                E=1
                H=2
                IN=3
                OUT=4
                SOUT=5
                NU=6
                ST=DAYTIM.(0)
                LM=1
                 THROUGH L1, FOR I=1,1,I.G.2*N
                T(I)=0
                M(I)=0
                WHENEVER I.G.N, TRANSFER TO L1A
                B(I)=I
                C(I)=E
                BV(I)=LM
                LM=2*LM
      L1        CONTINUE
                WHENEVER CG.NE.1, TRANSFER TO LC1
                THROUGH LC2, FOR I=1,1,I.G.N
                A(I)=.N.BV(I)
      LC2       CONTINUE
      LC1       CG=0
                THROUGH L2, FOR EX=1,1,EX.G.N
                WHENEVER C(EX).E.E,TRANSFER TO L3
      L2        CONTINUE
                SP=DAYTIM.(0)
                DT=SP-ST
                PRINT COMMENT $ TIME$
                PRINT RESULTS DT
                 THROUGH L52, FOR I=1,1,I.G.N
      L52       PRINT RESULTS M(I)
                TRANSFER TO L0
      L1A       B(I)=0
                BV(I)=0
                TRANSFER TO L1
      L3        ROOT=EX
                C(EX)=OUT
      L4         THROUGH L5, FOR IV=1,1,IV.G.N
                WHENEVER C(IV).E.OUT,TRANSFER TO L6
      L5        CONTINUE
                TRANSFER TO HUNG
      L6        THROUGH L8, FOR JV=1,1,JV.G.N
                WHENEVER A(IV).A.BV(JV).E.0,TRANSFER TO L8
                 WHENEVER C(JV).E.E
                 TRANSFER TO AUG
                 OR WHENEVER C(JV).E.NU
                 TRANSFER TO GROW
                 OR WHENEVER C(JV).E.OUT
                 TRANSFER TO L9
```

```
                OTHERWISE
                TRANSFER TO L8
                END OF CONDITIONAL
L8              CONTINUE
                C(IV)=SOUT
                TRANSFER TO L4
L9              WHENEVER B(IV).E.B(JV), TRANSFER TO L8
                TRANSFER TO BLOSS
GROW            AA=B(IV)
                BB=B(JV)
                CC=M(BB)
                T(BB)=AA
                T(CC)=BB
                B(BB)=IN
                C(CC)=OUT
                TRANSFER TO L8
HUNG            THROUGH L13, FOR I=1,1,I.G.N
                WHENEVER C(I).E.SOUT.OR.C(I).E.IN, TRANSFER TO L14
L13             CONTINUE
                TRANSFER TO EXPAND
L14             C(I)=H
                TRANSFER TO L13
AUG             C(JV)=OUT
                BB=B(IV)
                AA=B(JV)
L15             M(AA)=BB
                M(BB)=AA
                WHENEVER BB.E.ROOT, TRANSFER TO EXPAND
                AA=T(BB)
                BB=T(AA)
                TRANSFER TO L15
BLOSS           THROUGH L16, FOR KK=N+1,1,KK.G.2*N
                WHENEVER B(KK).E.0,TRANSFER TO L17
L16             CONTINUE
                PRINT COMMENT $ L16 ERRORS$
                TRANSFER TO L0
L17             K=KK-N
                P(1)=B(IV)
                THROUGH L18, FOR IP=1,1,P(IP).E.ROOT
L18             P(IP+1)=T(P(IP))
                Q(1)=B(JV)
                THROUGH L19, FOR IQ=1,1,Q(IQ).E.ROOT
L19             Q(IQ+1)=T(Q(IQ))
L22             WHENEVER IQ.E.1.OR.IP.E.1,TRANSFER TO L20
                WHENEVER P(IP-1).NE.Q(IQ-1),TRANSFER TO L20
L21             IP=IP-1
                IQ=IQ-1
                TRANSFER TO L22
L20             L=1
L20A            HP(K,L)=P(IP)
                WHENEVER IP.E.1, TRANSFER TO L23
                L=L+1
                IP=IP-1
                TRANSFER TO L20A
L23             I=0
L24             WHENEVER I.E.IQ-1,TRANSFER TO L25
                L=L+1
                I=I+1
                HP(K,L)=Q(I)
                TRANSFER TO L24
```

```
L25         LP(K)=L
            BV(KK)=0
L31         WHENEVER HP(K,1).E.ROOT, TRANSFER TO L32
            AA=HP(K,1)
            BB=T(AA)
            M(KK)=BB
            M(BB)=KK
            T(KK)=BB
L33         B(KK)=KK
            THROUGH L26,FOR I=1,1,I.G.L
            AA=HP(K,I)
            B(AA)=KK
            M(AA)=0
L26         T(AA)=0
            THROUGH L27, FOR J=1,1,J.G.2*N
             WHENEVER B(J).E.0, TRANSFER TO L27
            WHENEVER B(B(J)).E.KK, TRANSFER TO L28
            WHENEVER T(J).E.0, TRANSFER TO L27
            WHENEVER B(T(J)).NE.KK, TRANSFER TO L27
            T(J)=KK
L27         CONTINUE
            TRANSFER TO L8
L32         ROOT=KK
            TRANSFER TO L33
L28         B(J)=KK
            BV(KK)=BV(KK).V.BV(J)
            WHENEVER J.G.N, TRANSFER TO L27
            C(J)=OUT
            TRANSFER TO L27
EXPAND      THROUGH L34, FOR KK=N+1,1,KK.G.2*N
             WHENEVER B(KK).E.KK, TRANSFER TO L38
L34         CONTINUE
L35         THROUGH L36, FOR I=1,1,I.G.N
            WHENEVER C(I).E.E.OR.C(I).E.H,TRANSFER TO L36
            C(I)=NU
L36         CONTINUE
            THROUGH L37, FOR I=1,1,I.G.2*N
L37         T(I)=C
             TRANSFER TO L2
L38         K=KK-N
            WHENEVER M(KK).E.0, TRANSFER TO L39
            THROUGH L40, FOR J=1,1,J.G.N
L42         WHENEVER B(J).E.KK, TRANSFER TO L41
L40          CONTINUE
            PRINT COMMENT $ L40 ERROR$
            TRANSFER TO L0
L41         WHENEVER A(J).A.BV(M(KK)).E.0, TRANSFER TO L40
            THROUGH L43, FOR L=1,1,L.G.LP(K)
L44         BB=HP(K,L)
            WHENEVER BV(BB).A.BV(J).E.0, TRANSFER TO L43
            M(BB)=M(KK)
            M(M(KK))=BB
            M(KK)=0
            TRANSFER TO L46
L43         CONTINUE
            PRINT COMMENT $ L43 ERROR$
            TRANSFER TO L0
L46         B(KK)=0
             B(BB)=BB
            I=L
```

```
            TRANSFER TO L47
L39         BB=HP(K,I)
            L=1
            TRANSFER TO L46
L47         WHENEVER I.E.LP(K), TRANSFER TO L55
L49         I=I+1
            WHENEVER I.E.L, TRANSFER TO L59
            AA=HP(K,I)
            WHENEVER I.E.LP(K), TRANSFER TO L50
L51         I=I+1
            BB=HP(K,I)
            M(AA)=BB
            M(BB)=AA
            B(AA)=AA
            B(BB)=BB
            TRANSFER TO L47
L55         I=0
            TRANSFER TO L49
L50         I=0
            TRANSFER TO L51
L59         THROUGH L56, FOR I=1,1,I.G.2*N
            WHENEVER B(I).E.KK, TRANSFER TO L57
L56         CONTINUE
            TRANSFER TO EXPAND
L57         THROUGH L58, FOR L=1,1,L.G.LP(K)
            WHENEVER BV(I).A.BV(HP(K,L4).E.0, TRANSFER TO L58
            B(I)=HP(K,L)
            TRANSFER TO L56
L58         CONTINUE
            PRINT COMMENT $ L57 ERRORS
            TRANSFER TO L0
            END OF PROGRAM
```

APPENDIX B

LINEAR PROGRAMMING AND CONVEX POLYHEDRA

Let A be a real valued matrix and b, c, x, and y real valued vectors. Let X be the set of all vectors which satisfy:

(1) $Ax \leq b$

(2) $x \geq 0$

Let Y be the set of all vectors such that:

(3) $A^T y \geq c$

(4) $y \geq 0$

A vector $x \in X$ or $y \in Y$ is called feasible with respect to the given constraints.

A primal linear program has the form:

(5) $\max \; cx \; (=W)$

subject to:

(6) $Ax \leq b$

(7) $x \geq 0$

A dual linear program is defined relative to a given primal program. The dual to (5), (6), and (7) is:

(8) min by $(=U)$

subject to:

(9) $A^T y \geq c$

(10) $y \geq 0$

A pair of solutions (x, y) to a given primal-dual system of linear programs are called orthogonal if:

$(b-Ax)y = 0$, and

$(c-A^Ty)x = 0$.

The following theorems from linear programming theory are used in this thesis.

Theorem B.1

If x and y are feasible solutions to the given primal and dual linear programs, then $W = cx \leq by = U$.

Theorem B.2

If x and y are feasible solutions to the given primal and dual linear programs and $U = W$, then x and y are optimum solutions.

Theorem B.3

If x and y are feasible solutions to the given primal and dual linear programs and are also orthogonal, then x and y are optimum.

A convex polyhedron is the set of points (vectors) which satisfy a system of linear inequalities. The sets X and Y are convex polyhedra. A vector x is a vertex of the convex polyhedron X if and only if there does not exist two other vectors x_1 and x_2 in X such that

$$x = \tfrac{1}{2}(x_1 + x_2).$$

Let there be given a convex polyhedron X defined by $Ax \leq b$, $x \geq 0$, and let c be a cost vector.

Theorem B.4

The maximum value of cx occurs at some vertex of the convex polyhedron X, possibly at other points as well.

Theorem B.5

If x_0 is a vertex of the convex polyhedron defined by $Ax \leq b$ and $x \geq 0$, then there exists a c such that the unique maximum of cx occurs at x_0.

APPENDIX C

CHINESE POSTMAN'S PROBLEM AND THE DETECTION OF A NEGATIVE CYCLE

In this appendix we will present algorithms for the solution of the Chinese Postman's Problem, and for the detection of a negative cycle in a graph.

The Chinese Postman's Problem is to find a minimum continuous route through a graph which uses each edge of the graph at least once. In section 7.2 the problem was shown to be equivalent to finding the minimum weighted set of edges which, when added to G, cause each vertex to have even degree.

A forest is a subgraph which contains no cycles.

An S-odd subgraph is a subgraph of G whose set of odd-degree vertices is a given set S. S must be of even cardinality.

An S-forest is a forest in G whose set of odd-degree vertices is a given set S. Again, S must be of even cardinality. Obviously an S-forest is an S-odd subgraph, but the converse is not necessarily true.

Given a weighted connected graph G with a set S of vertices of odd degree, a minimum weight Euler graph G_D can be constructed by duplicating edges M such that M is a minimum S-odd subgraph.

An S-odd subgraph Q of minimum weight is an S-forest, since if a cycle C exists, $Q^1 = Q - C$ is an S odd subgraph of lower weight. Thus the Chinese Postman's Problem is equivalent to finding a

APPENDIX C

CHINESE POSTMAN'S PROBLEM AND THE DETECTION OF A NEGATIVE CYCLE

In this appendix we will present algorithms for the solution of the Chinese Postman's Problem, and for the detection of a negative cycle in a graph.

The Chinese Postman's Problem is to find a minimum continuous route through a graph which uses each edge of the graph at least once. In section 7.2 the problem was shown to be equivalent to finding the minimum weighted set of edges which, when added to G, cause each vertex to have even degree.

A <u>forest</u> is a subgraph which contains no cycles.

An <u>S-odd subgraph</u> is a subgraph of G whose set of odd-degree vertices is a given set S. S must be of even cardinality.

An <u>S-forest</u> is a forest in G whose set of odd-degree vertices is a given set S. Again, S must be of even cardinality. Obviously an S-forest is an S-odd subgraph, but the converse is not necessarily true.

Given a weighted connected graph G with a set S of vertices of odd degree, a minimum weight Euler graph G_D can be constructed by duplicating edges M such that M is a minimum S-odd subgraph.

An S-odd subgraph Q of minimum weight is an S-forest, since if a cycle C exists, $Q^1 = Q - C$ is an S odd subgraph of lower weight. Thus the Chinese Postman's Problem is equivalent to finding a

APPENDIX C

CHINESE POSTMAN'S PROBLEM AND THE DETECTION OF A NEGATIVE CYCLE

In this appendix we will present algorithms for the solution of the Chinese Postman's Problem, and for the detection of a negative cycle in a graph.

The Chinese Postman's Problem is to find a minimum continuous route through a graph which uses each edge of the graph at least once. In section 7.2 the problem was shown to be equivalent to finding the minimum weighted set of edges which, when added to G, cause each vertex to have even degree.

A <u>forest</u> is a subgraph which contains no cycles.

An <u>S-odd subgraph</u> is a subgraph of G whose set of odd-degree vertices is a given set S. S must be of even cardinality.

An <u>S-forest</u> is a forest in G whose set of odd-degree vertices is a given set S. Again, S must be of even cardinality. Obviously an S-forest is an S-odd subgraph, but the converse is not necessarily true.

Given a weighted connected graph G with a set S of vertices of odd degree, a minimum weight Euler graph G_D can be constructed by duplicating edges M such that M is a minimum S-odd subgraph.

An S-odd subgraph Q of minimum weight is an S-forest, since if a cycle C exists, $Q^1 = Q - C$ is an S odd subgraph of lower weight. Thus the Chinese Postman's Problem is equivalent to finding a

minimum S-forest in the original graph G. The following algorithm is proposed to solve this problem by use of the matching algorithm.

Algorithm

1. Form G_s from G. The vertex set of G_s is S. Each pair of vertices v_i and v_j of G_s is connected by an edge e_{ij} whose weight c_{ij} is given by the shortest path in G between v_i and v_j.

2. Find the minimum weight matching M_s of G_s which has cardinality $\frac{1}{2}|S|$. First change the edge weights of G_s by the following formula,

$$c'_{ij} = \max_{e_{ij} \in G_s} (c_{ij}) - c_{ij}.$$

then solve the maximum weight matching problem. Since the graph is complete (i.e., there is an edge between each pair of vertices) the solution will contain $\frac{1}{2}|S|$ edges.

3. Let M' be the preimage of M_s in G. That is, for each edge e_{ij} in M_s, use the path p_{ij} between v_i and v_j which was used to compute c_{ij} in G_s. Then M is the minimum S-odd subgraph of G_s, and the problem is solved.

The following theorem provides the justification for the above algorithm.

Theorem C.1

M_s is a minimum weight maximum cardinality matching on G_s

if and only if M is a minimum S-forest of G.

Proof

Construct a matching M_s of G_s from the forest M in the following manner. Take any two vertices, v_1 and v_2, of S which are in the same component of the forest M. Remove p_{12}, the unique path in the forest M between v_1 and v_2, from M. Continue to identify pairs of vertices in this way until all vertices of S are exhausted. Consider the matching M_s on G_s which is defined by the above pairing of vertices. Since $w(p_{ij}) \geq c_{ij}$, due to the use of the shortest path algorithm, we have,

(1) $w(M) \geq w(M_s) \geq w(N_s)$,

where N_s, is the minimum weight maximum cardinality matching on G_s.

Conversely construct an S-odd subgraph N from N_s in the following manner. For each e_{ij} in N_s, let p_{ij} be the shortest path in G between v_i and v_j. Let $N = \bigcup_{e_{ij} \in N_s} p_{ij}$.

N is an S-odd subgraph, therefore,

(2) $w(N_s) = w(N) \geq w(M)$,

Combining (1) and (2) we obtain,

(3) $w(N_s) = w(M) = w(M_s)$,

and the theorem is proved.

A related problem is the detection of the existence of a negative cycle in a given graph. Let F be the subgraph of negative edges. If a cycle exists in F, then there obviously exists a negative cycle. Therefore we assume that F is a forest. We define S as the set of

vertices which have odd degree relative to F. Thus F is automatically an S-forest.

The algorithm consists of four steps:

(1) Form G_A from G, where the weight of edge e_{ij} in G_A is equal to the absolute value of c_{ij}.

(2) Form G_S from G_A. The vertices of G_S are the vertices of S. There is an edge between each pair of vertices v_1 and v_2 of G_S and its weight is equal to the weight of the shortest path between v_1 and v_2 in G_A.

(3) Find M_S, the minimum weight maximum cardinality matching on G_S.

(4) Form M from M_S. For each edge $e_{ij} \epsilon M_S$, p_{ij} is the shortest path in G_A between v_i and v_j.

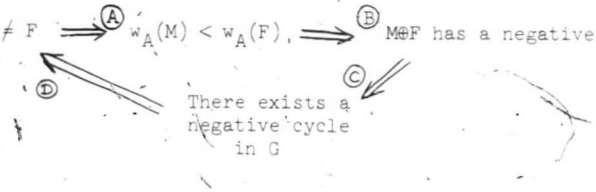
$$M = \bigcup_{e_{ij} \epsilon M_S} p_{ij}$$

Theorem C.2

There exists a negative cycle in G if and only if $M \neq F$.

Proof

The theorem will be proved in the following sequence.



M is a minimum S-odd subgraph on G_A. Therefore $w_A(M) < w_A(F)$ and thus (A) is proved.

Since M and F have odd degree only at vertices in S, each component of $M \oplus F$ satisfies the Euler conditions and is therefore a cycle. But $w_A(M) < w_A(F)$ implies that $w(M \oplus F)$ is negative. Thus at least one component of $M \oplus F$ is a negative cycle and (B) is proved. But that same component is also a cycle in G, therefore (C) is proved.

Now suppose that there exists a negative cycle C. Let $Q = F \oplus C$, where Q is an S-odd subgraph.

$$w_A(F) = w_A(F-C) + w_A(F \cap C)$$

$$w_A(Q) = w_A(F-C) + w_A(C-F)$$

Since w(C) is negative, $w_A(F \cap C) > w_A(C-F)$. Thus $w_A(F) > w_A(Q)$, which implies that F is not the minimum S-odd subgraph. Therefore $M \neq F$, and (D) is shown.

REFERENCES

(1) M. L. Balinski, "Integer Programming: Methods, Uses, Computations," Management Science, Vol. 12, and 13, November 1965, pp. 253-313.

(2) C. Berge, "Two Theorems in Graph Theory," Proc. Natl. Acad. Sci. US, 43, pp. 842-844 (1957).

(3) C. Berge, "The Theory of Graphs and its Applications," John Wiley & Sons, New York, 1962, pp. 213-218.

(4) R. G. Busacker and T. L. Saaty, "Finite Graphs and Networks," McGraw-Hill, Inc., New York, 1965.

(5) G. B. Dantzig, "Application of the Simplex Method to a Transportation Problem," Activity Analysis of Production and Allocation, John Wiley & Sons, Inc., New York, 1951, pp. 359-373.

(6) G. B. Dantzig, "Linear Programming and Extensions," Princeton University Press, 1963, pp. 319-321.

(7) M. Davis, "Computability and Unsolvability," McGraw-Hill, 1958.

(8) W. L. Eastman, "Linear Programming With Pattern Constraints," Harvard University Ph.D., July 1958.

(9) J. Edmonds, "Paths, Trees, and Flowers," Canadian J. Math., pp. 449-467, May 1965.

(10) J. Edmonds, "Maximum Matching and a Polyhedron with 0,1-Vertices," J. Res. NBS, 69B (Math and Math Phys.), No. 1 and No 2, pp. 129-130, Jan.-June 1965.

(11) J. Edmonds, "Covers and Packings in a Family of Sets," Bull. Amer. Math Soc., 68, pp. 494-499 (1962).

(12) D. R. Fulkerson, "An Out-of-Kilter Method for Minimum-Cost Flow Problems," J. Siam, Vol. 9, No. 1, March 1961, pp. 18-27.

(13) R. M. Karp and R. E. Miller, "Properties of a Model for Parallel Computations: Determinacy, Termination, and Queueing," IBM Research Paper RC 1285.

(14) J. B. Kruskal Jr., "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem," Proc. Amer. Math. Soc. 1, 48-50, 1956.

(15) H. W. Kuhn, "The Hungarian Method for the Assignment Problem," Naval Research Logistics Quarterly, Vol. 2, pp. 83-97 (1955).

(16) E. L. Lawler, "Covering Problems: Duality Relations and a New Metnod of Solution," J. Siam, Vol. 14, No. 5, Sept. 1966.

(17) L. C. Lorentzen, "Notes on Covering of Arcs by Nodes in an Undirected Graph," Oper. Reas. Center, Univ. of Calif., July 1966.

(18) R. Z. Norman and M. O. Rabin, "An Algorithm for a Minimum Cover of a Graph," Proc. Amer. Math Soc., 10, pp. 315-319.

(19) J. Peterson, "Die Theorie der Regularen Graphen," Acta. Math, Vol. 51 (1891), pp. 193-220.

(20) R. Reiter, "Initiation Timing in a Model for Parallel Computation," Technical report SIL-66-3, Systems Engineering Laboratory of the Department of Electrical Engineering, The University of Michigan, March 1966.

(21) J. W. Suurballe, Bell Laboratories, Private Communication.

(22) W. T. Tutte, "A Short Proof of the Factor Theorem for Finite Graphs," Canadian J. Math, 6, pp. 347-352 (1954).

(23) W. T. Tutte, "The Factorization of Linear Graphs, "J. London Math Soc., 22, pp. 107-111 (1947).

(24) W. T. Tutte, "The Factors of Graphs" Can. J. Math, pp. 314-318 (1952).