AN INTRODUCTION TO MATCHING

Jack Edmonds

An Introduction to Matching

Jack Edmonds

Lectures

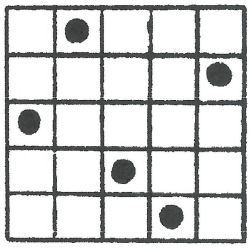
Ann Arbor, Spring 1967

I. The Optimum Assignment Problem

For rectangular array (matrix), N, we define a matching in N to be a subset M of the positions in N such that each column and each row of N contains at most one member of M .

For any square array N , we define a transversal or a perfect matching M to be a subset of the positions in N such that each column and each row of N contains exactly one member of M .

The optimum assignment problem is, given any n x n array N of real numbers, find in N a transversal the sum of whose entries is maximum, i.e., an "optimum" transversal.



A transversal in N "assigns" the rows of N to the columns of N . Where the columns are people and the rows are jobs, and where each numerical entry represents the value of the person of that column at the job of that row, an optimum transversal represents an optimum assignment of the people to the jobs.

A well-known generalization of the assignment problem is the integer Hitchcock-transportation problem: Given a rectangular array N of real numbers, c_{ij}, and given an integer $a_i \geq 0$ for each row i and an integer $b_j \geq 0$ for each column j, assign a non-negative integer x_{ij} to each position (i,j) so that

(1) for every i, $\sum_j x_{ij} = a_i$,

(2) for every j, $\sum_i x_{ij} = b_j$,

and so that $\sum_{ij} c_{ij} x_{ij}$ is minimum (or maximum).

If a_i represents the number of refrigerators available at factory i, and b_j represents the number of refrigerators ordered by dealer j, and c_{ij} represents the cost of shipping a refrigerator from i to j, then $\sum_{i,j} c_{ij} x_{ij}$ represents the total cost of the particular manner $[x_{ij}]$ of distributing the refrigerators.

The assignment problem is where all $a_i = 1$ and all $b_j = 1$.

A minor variation of the assignment problem is: given a rectangular array N of real numbers find in N a matching whose entries have maximum sum. This variation corresponds to replacing the equality signs in (1) and (2) by inequality signs. Ofcourse, a maximum matching will not contain a position whose entry is negative. In particular if all the entries are negative then the maximum matching will be the empty matching. It is an interesting exercise to discover how any maximum transversal problem can be solved by solving a maximum matching problem, and vice versa.

There are other generalizations and variations of the optimum assignment problem -- most notably integer network flow problems. These lectures, after treating the assignment problem itself will deal (briefly, I'm afraid) with some bizarre variations.

In an $n \times n$ array there are $n!$ different transversals. In particular there $100!$ ways to assign 100 people to 100 jobs. $100!$ is very large. If our method for finding an optimum assignment spent one microsecond per possible assignment, it would take hundreds of years to optimally assign 100 men.

It is a remarkable fact there exists a consistently good algorithm. An algorithm good enough that you could actually do as homework any instance of the assignment problem with 100 people, 100 jobs, and any collection of 3 digit numbers as values. Good enough to be used for many thousands of people and jobs. Ofcourse you have to know how, and it is not easy to discover how.

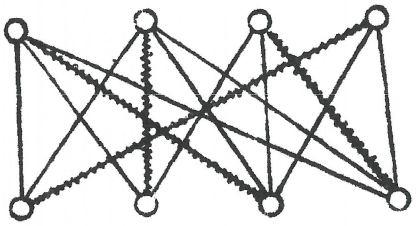
It is an unfortunate fact for most combinatorial problems -- problems very similar to the assignment problem -- that good algorithms are not known. For most such problems, though they are ofcourse finite, the best known methods do considerably worse than one might expect. Such problems include the bulk of integer linear programming problems.

Therefore we do what we can. We experiment with the most promising methods we can find for problems that need answers. And we also try to find classes of problems for which, using special methods, we can predict computational efficiency. These lectures fall

into the latter area.

For ease in giving an arm-waving description of an algorithm, it is convenient to represent the assignment problem by a graph.

A bipartite graph G is one whose set V of nodes partitions into two sets V_1 and V_2 so that no edge of G has both ends in the same set. Thus, each edge meets one node in V_1 and one node in V_2. Denote the set of edges of G as E.



A matching M in a graph is subset of its edges such that no two members meet the same node. A perfect matching in a graph is a subset of its edges such that exactly one member meets each node.

The optimum assignment problem (more precisely, a minor generalization of it) is:

In any given bipartite G, with a real numerical weight c_e for each edge $e \epsilon E$, find if there is one a perfect matching M which maximizes $\sum_{e \epsilon M} c_e$, i.e. "a maximum perfect matching".

After we treat the above problem, we shall treat the same problem where G is not necessarily bipartite. The latter is a very substantial generalization.

Where G is bipartite, with "parts" V_1 and V_2, the nodes of V_1 correspond to rows or jobs, the nodes of V_2 correspond to columns or people. Each edge corresponds to a position in the array.

Any perfect matching in G determines an assignment of people to jobs, as does a perfect matching in the array.

Where G is any graph not necessarily bipartite, the maximum perfect matching problem is the problem of optimally pairing-off a set of objects (the nodes). Admissible pairs and their values are represented by the edges.

We shall see later that various other problems reduce to the matching problem.

A <u>feasible node-weighting</u> of graph G is a vector $[y_v]$ with a component y_v for each node $v \in V$ such that, for every edge $e \in E$,

(3) $y_u + y_w \geq c_e$ where u and w are the ends of e.

Lemma 1. For any perfect matching M of a graph G and for any feasible node-weighting $[y_v]$,

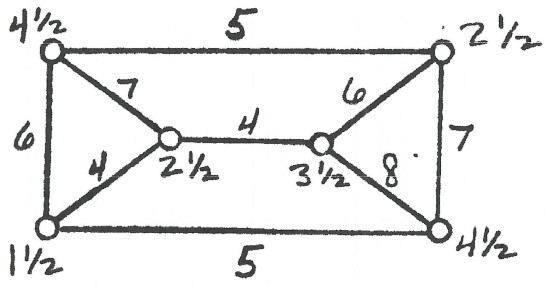
(4) $$\sum_{e \in M} c_e \leq \sum_{v \in V} y_v .$$

Proof: Add up the inequalities (3) which correspond to the edges in M.

It follows from Lemma 1 that if we can find a perfect matching M and a feasible $[y_v]$ such that equality holds in (4), then $\sum_{e \in M} c_e$ must be maximum and $\sum_{v \in V} y_v$ must be minimum.

This is true whether or not G is bipartite. When one can get such a node-weighting along with an M, it provides a very simple guarantee that the M is maximum.

If G is not bipartite, there does not necessarily exist a feasible $[y_v]$ such that Σy_v equals the maximum value of $\sum_{e \in M} c_e$ for G.



For any given edge-weighted bipartite graph G, the optimum assignment algorithm, that we will describe, first chooses any feasible node-weighting $[y_v]$. Then it chooses a matching, not necessarily perfect. Then it successively finds better node-weightings and better matchings until

(1) it finds a node-weighting and a perfect matching for which

$$\sum_{e \in M} c_e = \sum_{v \in V} y_v \text{ , or until}$$

(2) it finds a way to choose node-weightings such that $\sum_{v \in V} y_v \to -\infty$ (in which case, by Lemma 1, there is no perfect matching in G).

Thus, the algorithm will prove

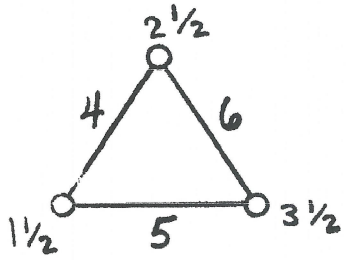
Theorem 2. For any edge-weighted bipartite graph G, which contains a perfect matching, the maximum weight-sum of a perfect matching in G equals the minimum sum of a feasible node-weighting.

P.S. 1 If G contains no perfect matching, then there is no minimum feasible node-weight sum.

By using only integer node-weights, if edge-weights are integers, the algorithm will also prove:

P.S. 2 If the edge-weights are all integers, then a min node-weighting can be chosen to be integers.

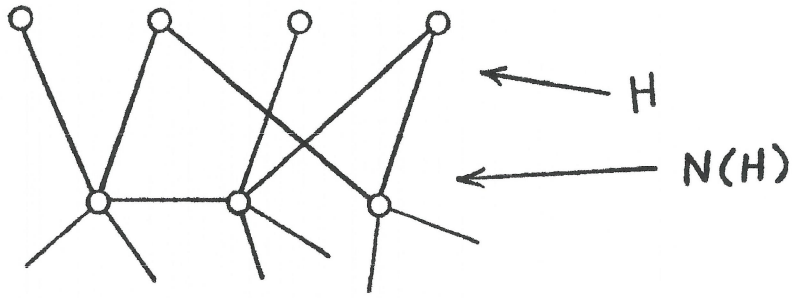
The two postscript as well as the theorem, are false for non-bipartite graphs.



II. A Hungarian Method

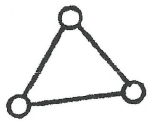
We back up now and state Theorem 1, which will be used as part of the optimum assignment algorithm, to prove Theorem 2 already given.

A subset H of the nodes in a graph G is called <u>hungarian</u>, relative to G, if no two of them are joined by an edge of G and if the set $N(H)$ of neighbors of H has fewer members than H, i.e., $|N(H)| < |H|$. A neighbor of H is a node which is joined by an edge of G to a node in H.



Obviously, a graph which contains a hungarian set H can not contain a perfect matching M, because the $|H|$ members of M which meet the nodes in H would have to meet $|H|$ different members of $N(H)$. This is impossible since $|N(H)| < |H|$.

A non-bipartite graph does not necessarily contain either a hungarian set or a perfect matching.



Theorem 1. A bipartite graph G contains no perfect matching if and only if G has a hungarian set, contained either in part V_1 or part V_2 of G.

Subroutine R1 of the algorithm will prove Thm 1 by finding in any bipartite graph either a perfect matching or a hungarian set. Subroutine R2 of the algorithm will prove Thm 2 by using a hungarian set of a subgraph of G to improve any non-minimum node-weighting of G.

Here is the algorithm, Given a bipartite graph G with a numerical weight c_e on each edge $e \in E$.

Give to G any feasible node-weighting $[y_v]$. (Make the node-weights integers if the edge-weights are.)

For the current feasible node-weighting at any stage of the algorithm, let G' denote the subgraph of G which consists of all nodes of G and those edges e of G such that

(5) $y_u + y_w = c_e$, where u and w are the ends of e.

We call G' the <u>equality subgraph</u> of G relative to $[y_v]$.

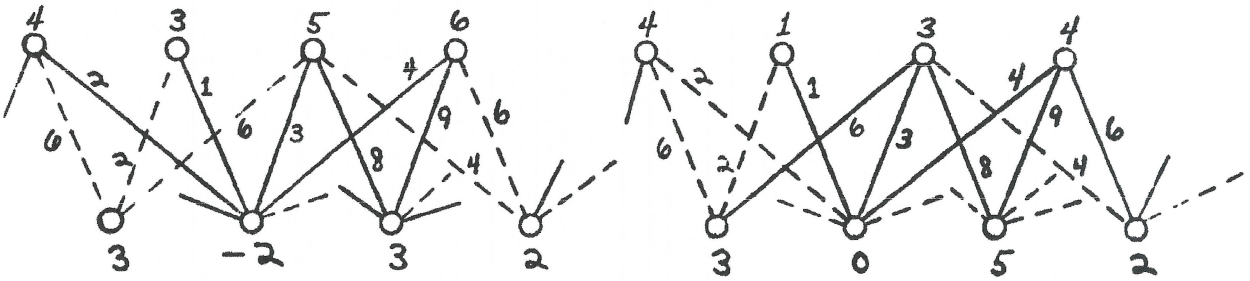
Using routine R1, we find either a perfect matching M in G' or else a hungarian set H' of G' in either part V_1 or V_2. (Since every edge of G has one of its end-nodes in set V_1 and its other end-node in set V_2, the same is true for subgraph G'. Any perfect matching of G' is ofcourse also a perfect matching of G, since G' contains all the nodes of G. However, a hungarian set H' of G' is not in general a hungarian set of G. Since G generally has more edges than G', set H' of nodes generally has more neighbors relative to G than relative to G'.)

Theorem 1 says that G' has either an M or an H'. Routine R1 will be the proof of Theorem 1.

Suppose we find a perfect matching M in G'. Then adding together equations (5), $y_u + y_w = c_e$ for the edges e in M, we get $\sum_{e \in M} c_e = \sum_{v \in V} y_v$, and so M is optimum.

Otherwise, we find a hungarian set H' of G' contained in, say, part V_2. In this case, we apply routine R2:

R2. Suppose that the set $N(H')$, of neighbors of H' relative to G', is not the entire set of neighbors of H' relative to G. Then there are edges e of G which are not in G' and which have one end in H' and the other end not in $N(H')$. Let $\epsilon = \min (y_u + y_w - c_e)$ over all such edges. Clearly, $\epsilon > 0$. Lowering each node-weight in H' by ϵ and raising each node-weight in $N(H')$ by ϵ, we get a new feasible node-weighting. Its sum is smaller because $|N(H')| < |H|$. The equality subgraph G' changes, so we apply R1 again.



Where $N(H')$ is the entire set of neighbors of H', relative to G as well as G', the set H' is hungarian relative to G and so there is no perfect matching in G. In this case, we can take ϵ

to be as large as we please, and still change node-weights as above. This gives Σy_v as small as we please, i.e. $\Sigma y_v \to -\infty$.

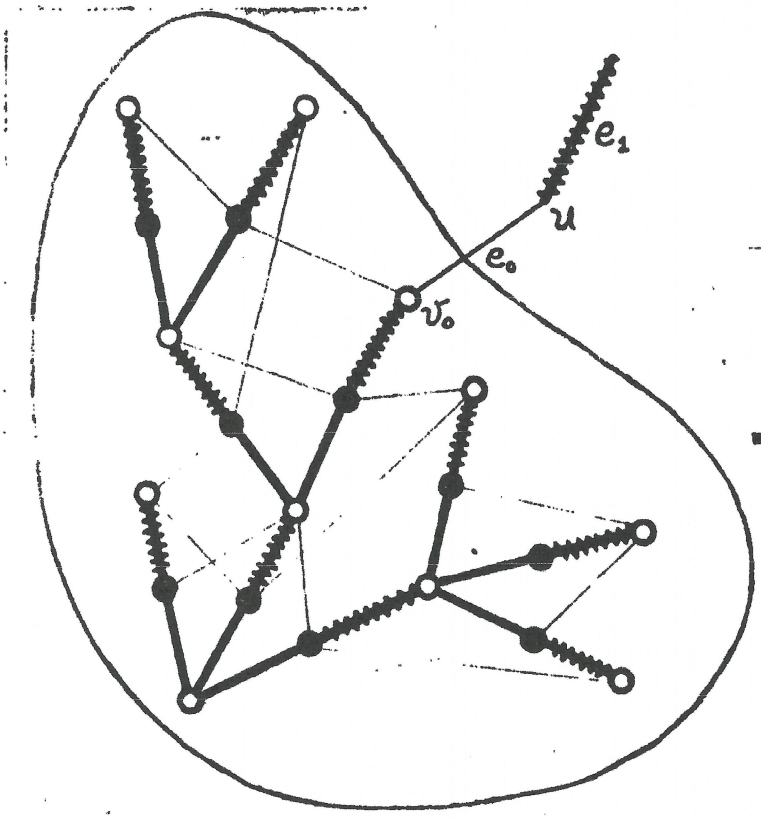
Assuming Thm 1 holds, i.e., assuming there is a valid routine R1, the routine R2 just described proves Thm 2 and PS1. For a node-weighting having min sum, there must be a perfect matching in the corresponding G', because otherwise we can apply Thm 1 and R2 to get a smaller node-weight sum.

PS2 asserted that if edge-weights are integers then a min node-weighting can be chosen to be integers. This follows by applying the above process to any integer-valued feasible node-weighting, because whenever edge-weights and node-weights are integers, ϵ is an integer (or arbitrary $+\infty$).

Let us now describe routine R1, thereby proving Thm 1 and completing the description of an optimum-assignment algorithm. R1 must find either a perfect matching in G' or a hungarian set of G'.

R1: Let M be any matching in G', not necessarily a perfect matching. To begin it might be the empty matching. If M is not perfect, let r be any node which M does not meet. We shall find either a hungarian set of G' which contains r, or else a matching M' in G' which is better than M in the sense that M' meets r as well as all the nodes which M' meets.

In G' , "grow a tree" T of the kind pictured inside the bean
~~oval~~. Node r itself is a tree of type T . Start off with simply it.



——— edge of G' , not in T , and not in M .

edge of M .

edge of T , not in M .

● called an inner node of T .

○ called an outer node of T .

All the outer nodes of T must be in the same part of G' . Thus, no two of them are joined by a edge of G' . The number of inner nodes of T is exactly one less than the number of outer nodes.

For any T either (a), (b), or (c) holds. (a) Every edge in G' which meets an outer node of T meets an inner node of T at its other end. In this case, the set of outer nodes is a hungarian set H of G' . The set of inner nodes is N(H) , relative to G' .

Otherwise some edge e_o of G' meets an outer node, say v_o, of T and a node, say u, not in T. (b) u meets no edge in M. In this case obtain M' from M by interchanging the matching roles of edges in the path P, consisting of u, e_o, and the path from v_o to r in T. Then forget T and, if there is still some node, say r', in G' which does not meet any edge of M', grow in G' another tree rooted at r'. (c) Otherwise, u meets an edge $e_1 \epsilon M$. In this case, enlarge T by adjoining e_o and e_1 to it, so that u becomes an inner node and the other end of e_1 becomes an outer node.

Suppose the number of nodes in G, and thus in G', is $2n$. After T is enlarged at most n times, either (a) or (b) must occur. After at most n differently-rooted trees are grown, either (a) or a perfect matching must occur in G'.

When (a) occurs, we apply R2 to the hungarian set H of G', the set of outer nodes in T. Thus, unless H is also a hungarian set relative to G, the node weights change and the equality subgraph G' changes. The outer nodes of T are then not a hungarian set of the new G'. However, the same matching M is contained in the new G' and the same tree T is contained in the new G'. Case (b) or (c) of R1 applies directly to this M and T in the new G', and so we can continue growing the same tree in the new G'. Thus, R1 is iterated a total of at most n^2 times and the "step" of enlarging a tree is iterated a total of at most n^2 times before one obtains either a perfect matching in some equality subgraph G' or else a hungarian set of G.

III. Bipartite Matching and Linear Programs

By polyhedron (strictly speaking convex polyhedron) we mean the set of all vectors (points) which satisfy some given finite collection of linear equations and linear inequalities.

Let G be any finite graph, not necessarily bipartite. Let E denote the set of edges of graph G. Let V denote the set of nodes of G.

Let there be a real variable x_e for each edge $e \in E$. Let P_G be the polyhedron of vectors $[x_e]$ such that

(1) for every $e \in E$, $x_e \geq 0$, and

(2) for every $v \in V$, $\sum x_e = 1$, where the sum is taken over all edges e which meet node v.

Another way of expressing (1) and (2) is

(1') $x_j \geq 0$, $j \in E$, and

(2') $\sum_j a_{ij} x_j = b_i$, $j \in E$, $i \in V$,

where all $b_i = 1$, where $a_{ij} = 1$ if edge j meets node i, and where $a_{ij} = 0$ if edge j does not meet node i.

Matrix $[a_{ij}]$ is called the incidence matrix of graph G. It has a "row" i for each node $i \in V$ and a "column" j for each edge $j \in E$. Clearly, a matrix is the incidence matrix of some graph if and only if each of its columns contains exactly two 1's and the rest 0's.

We may define a vertex x^o of a polyhedron P to be a point (i.e., a vector $[x_j^o] = x^o$) in P such that, for some linear function $U = \sum_j c_j x_j$ of the points in P, x^o is the only point in P which maximizes the function.

It is a standard theorem that if a given linear function of points in polyhedron P has a maximum (i.e., P is not empty and the function is not unbounded above), then the function is maximized by a vertex of P. Ofcourse, some linear functions on P are maximized by other points as well.

As you know, the linear programming problem is: For the polyhedron P, determined by a given system of linear "constraints", and for a given linear function U of points in P, find a vertex of P which maximizes (or minimizes) U in P.

Any vector $[x_e]$ of zeroes and ones is called the incidence vector of (or simply the vector of) the subset of e's such that $x_e = 1$.

Thus, every subset of edges in G is represented by a unique 0,1 vector, and conversely.

Clearly, where c_e is the weight on edge e in G, the weight-sum of any perfect matching M in G is the value of $\sum c_e x_e$ ($e \epsilon E$) for the vector of M.

Clearly, the 0,1-valued vectors contained in P_G (in fact, the integer-valued vectors contained in P_G) are precisely the vectors of perfect matchings in G.

We shall show that the assignment problem is an instance of linear programming by showing that

Theorem 3. If G is bipartite, then the vertices of P_G are precisely the vectors of perfect matchings in G.

Clearly, even if G is not bipartite, the vector of any perfect matching M in G is a vertex of P_G, since we can display a function $\sum c_e x_e$ which obviously is maximized in P_G only by the vector of M. In particular, where $c_e = 1$ if and only if $e \in M$.

The hard part is to show that every vertex of P_G is the vector of a matching. We shall do so using the duality thm of linear programming and thm 2 about node-weightings for an edge-weighted bipartite G.

The ℓ.p. dual of maximizing $U = {}_j\Sigma c_j x_j$. subject to $x_j \geq 0$ and ${}_j\Sigma a_{ij} x_j = b_i$,is minimizing $W = {}_i\Sigma b_i y_i$ subject to ${}_i\Sigma a_{ij} y_i \geq c_j$. The duality thm says that max U = min W if these extrema exist.

In particular, the dual of maximizing $U = \sum c_e x_e$ $(e \in E)$ in P_G is minimizing $W = \sum y_v$ $(v \in V)$ subject to $y_u + y_w \geq c_e$ for every e, where u and w are the end-nodes of e. That is, minimizing the sum of feasible node-weights, as in Thm 2.

Thus, it follows from Thm 2 and the ℓ.p. duality thm that if G is bipartite, then, for any U, the max of U for vectors in P_G equals the max of U for vectors of perfect matchings.

Therefore, since all vectors of perfect matchings are in P_G, U is maximized in P_G by the vector of a perfect matching.

For any vertex x° of P_G, suppose that U is a linear function that is maximized in P_G only at x°. Then x° must be the vector of a perfect matching. So Thm 3 is proved.

Conversely, Thm 3 and the ℓ.p. duality thm immediately imply Thm 2. So, in view of ℓ.p. duality, Thm 2 and Thm 3 are equivalent.

Where bipartite graph G is a square array whose rows i and columns j are the nodes, and whose positions (i,j) are the edges. Thm 3 is well-known as G. Birkhoff's theorem on "doubly stochastic matrices" (1946)

An n by n doubly stochastic matrix is defined to be an n by n matrix $[x_{ij}]$ such that:

all $x_{ij} \geq 0$, and

(3) for every fixed i, $_j\Sigma x_{ij} = 1$, and

(4) for every fixed j, $_i\Sigma x_{ij} = 1$. (see page)

Matrices are vectors, indexed differently. The collection of n by n doubly stochastic matrices is a polyhedron. The Birkhoff thm says that the vertices of this polyhedron are the n by n permutation matrices. A permutation matrix is a matrix such that there is a 1 in each row, a 1 in each column, and all other entries are zeroes.

Thm 2, essentially, is due to Egervary (1931). The algorithm here for the assignment problem, essentially, is due to Kuhn and Munkres (1955-57).

Where the 1 in (3) is replaced by any prescribed integers $a_i \geq 0$ and the 1 in (4) is replaced by any prescribed integers $b_j \geq 0$, we get the linear constraints of the integer transportation problem, relations (1) and (2) of section I. These constraints, together with $x_{ij} \geq 0$ for every i and j, define a polyhedron, say P_T. Theorem 3 readily generalizes to the fact that all the vectices of P_T are integer-valued vectors. Thus we have the very well-known fact that the integer transportation problem is an instance of linear programming.

Indeed, historically the first, and still the most prominent, algorithms for the transportation problem are direct applications of the simplex method.

Theorem 3, and consequently Theorem 2, and consequently even Theorem 1, are readily (and often) proved using general ℓ.p. techniques together with the special properties of the incidence matrix $[a_{ij}]$ of a bipartite graph.

Similarly, extremal (integer) flows in a network can be treated by applying general ℓ.p. techniques to matrices $[a_{ij}]$ which are the "incidence matrices of directed graphs". And, on the other hand, the combinatorial algorithm that we described for the assignment problem is very closely related to the combinatorial methods of Ford and Fulkerson for network flow problems.

Two advantages of these combinatorial methods are (1) for practical purposes, they are more efficient than simplex methods, and (2) for theoretical purposes, they provide theoretical bounds on

efficiency of computation that are so for not available for simplex methods. The disadvantage of these combinatorial methods is that they have not been satisfactorialy extended to general ℓ.p. problems.

IV. Matching in a general graph

So far these lectures have been small variations on old stuff. Now I would like to show one of the ways in which these combinatorial methods can be generalized to certain problems which can not be treated directly as linear programs -- at least not in the usual sense. Actually they are linear programs determined implicitly by astronomical collections of linear constraints. Surprisingly, these problems are just as tractable as the assignment problem -- in spite of the traditional views as to why the assignment problem is so tractable.

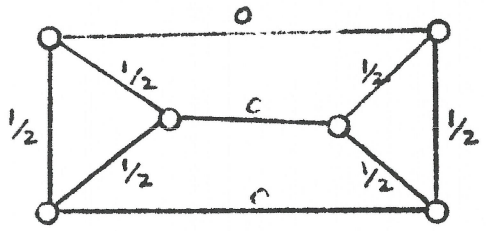
We treat the problem of finding a maximum-weight perfect matching in an edge-weighted graph G which is not necessarily bipartite. In particular we can take G to be a complete graph -- a graph in which every pair of nodes is joined by an edge.

Recall that we have already defined a polyhedron P_G, for any graph G, by the linear constraints (1) and (2) in section III. We have already observed that the only integer-valued vectors contained in P_G are the vectors of perfect matchings in G. That is, an integer-valued vector $[x_e]$ is contained in P_G if and only if its components are 0's and 1's, and the 1-components correspond to the edges of a perfect matching.

We have already observed that where c_e is the weight on e in G, the weight-sum of any perfect matching M is the value of $\Sigma\, c_e x_e (e \in E)$ for the vector of M.

Hence, the maximum perfect-matching problem, for any edge-weighted graph G is an instance of "integer linear programming". Integer linear programming is the problem of maximizing a given linear function by an integer-valued vector subject to some given linear constraints.

If all the vertices of polyhedron P_G were vectors of matchings, as in the case where G is bipartite, then the maximum matching problem would be simply an instance of linear programming. However, when G is not bipartite, P_G has vertices which are fractional-valued.



The picture shows one of the simplest graphs G such that P_G contains a fractional vertex as well as several vertices which are vectors of perfect matchings. The numbers on the edges are the components of the fractional vertex in this P_G. Notice that where the edge-weights are the numbers on the edges in the picture on page 6, the maximum-weight of a perfect matching is 18. At the fractional vertex of P_G, $\sum_{e \in E} c_e x_e$ has the value 19, which by no accident is also the min sum of a feasible node-weighting.

A main idea in tackling the general perfect-matching problem is to chop off the fractional vertices of P_G so as to obtain a polyhedron P'_G such that all the vectors of perfect matchings are still contained in P'_G and such that P'_G doesn't have any fractional vertices, i.e., all its vertices are vectors of perfect matchings. In other words, obtain the convex hull, P'_G, of the vectors of perfect matchings.

Theorem 4. For any graph G, the convex hull of the vectors of perfect matchings in G is the polyhedron P'_G given by the following linear constraints:

(1) for every edge $e \in E$: $x_e \geq 0$;

(2) for every node $v \in V$: $\sum x_e = 1$, where the sum is taken over all edges e which meet node v.

(3) for every subset s of nodes which has cardinality $|s| = 2q_s + 1$ for some positive integer q_s, $\sum x_e \leq q_s$, where the sum is taken over all edges e which have both ends in s.

Any vector of a matching, say M, in G satisfies (3) for any set s, since no more than q_s edges of M can have both ends in s, and since such edges are the only ones in M which appear in (3). Therefore, every vector of a perfect matching of G is in the polyhedron P'_G.

It is not so obvious that every vertex of P'_G is a vector of a perfect matching. We prove this by means of an algorithm -- in a manner analogous to our proof of Thm 3 by means of the assignment algorithm.

The idea of treating a combinatorial problem by chopping away at a polyhedron to eliminate undesirable vertices is by no means new. A long time ago Kuhn and Dantzig and others took this approach to the traveling salesman problem. I believe Motzkin tried it for the "3-dimensional assignment problem". Gomery discovered finite algorithms for integer linear programming which operate by chopping locally until the answer is a vertex of the resulting polyhedron. His methods could be adapted to give finite algorithms for describing the convex hull of the integer vectors in any given bounded polyhedron.

For the matching problem, and certain cousins, the chopping-idea has been completely successful in the following senses. (1) We get a succinct and useful description of precisely the relevant polyhedra. (2) We get an algorithm that is really good.

Later, I'll describe a representative, "optimum branching", of one other essentially different class of problems for which polyhedron-chopping has been completely successful.

Let $U = \sum c_e x_e (e \in E)$ be any linear function of vectors $[x_e]$, determined by an arbitrary specification of edge-weights for G. To maximize U by a vertex of P'_G is a linear program. Our purpose is to solve this $\ell.p.$ by the vector of a perfect matching. Since U is arbitrary and since for every vertex of P'_G there is a U which is maximized only by that vertex, this will prove Thm 4.

We now describe the dual of our ℓ.p. Like ℓ.p. duals everywhere, it has a variable for each constraint of the primal, other than non-negativity, and it has a constraint for every variable of the primal, as well as the non-negative constraint on each dual variable that corresponds to an inequality-constraint of the primal.

In particular, it has a variable y_v for each node $v \in V$, the "node-weight" for v. And it has variable y_s, an "odd-set-weight", for each subset $s \subset V$ such that $|s| = 2q_s + 1$ where q_s is a positive integer. Recall that $|s|$ means the number of nodes in s.

The variables y_v are allowed to go negative since they correspond to equations of the primal. The constraints of the dual are

(4) for every set s, $y_s \geq 0$;

(5) for every edge e, $f_e(y) = y_u + y_w + \sum y_s \geq c_e$, where u and w are the ends of e, and where the summation is over all sets s which contain both ends of e.

Thus, a dual weighting $y = [y_v, y_s]$ is called <u>feasible</u> if (4) and (5) hold.

The linear function to be minimized is

(6) $W = \Sigma\, y_v + \Sigma\, q_s y_s$.

The duality theorem tells us that a vector $x^o = [x_e^o]$ maximizes $U = \Sigma\, c_e x_e$ subject to (1), (2), and (3) if and only if there is some vector $y^o = [y_v^o, y_s^o]$ satisfying (4) and (5), for which $U(x^o) = W(y^o)$. We shall find such an x^o which is the vector of a perfect matching.

More directly useful to our purpose is the so-called "complementary slackness theorem" on dual ℓ.p.'s. For our particular dual ℓ.p.'s it says that a vector $x^o = [x_e^o]$ maximizes U subject to (1), (2), and (3), and a vector $y^o = [y_v^o, y_s^o]$ minimizes W subject to (4) and (5), if and only if

(7) for every variable y_s, either $y_s^o = 0$ or else equality holds in the corresponding constraint (3);

(8) for every variable x_e, either $x_e^o = 0$ or else equality holds in the corresponding constraint (5).

Where $x^o = [x_e^o]$ is the vector of a perfect matching M of G,

(7) says that for every set s of nodes, either $y_s^o = 0$ or else q_s edges of M have both ends in s;

(8) says that for every edge e in M, equality holds in the corresponding constraint (5).

The matching algorithm finds a perfect matching M and a vector $y^o = [y_v^o, y_s^o]$ such that (4),(5),(7), and (8) hold. This guarantees that M is optimum and it also proves Thm 4.

Vectors $[y_v, y_s]$ have an awful lot of components. Fortunately, however, the ones we deal with have no more non-zero components than the number of edges in G.

V. A Matching Algorithm

Given any edge-weighted graph G. The algorithm starts out just like for the bipartite case. We choose a feasible node-weighting $[y_v]$, i.e., values of y_v such that (5) holds with all $y_s = 0$. Let G' be the equality subgraph of G, relative to this $[y_v, y_s]$. That is, G' consists of all nodes of G and those edges e for which equality holds in the corresponding constraint (5), $f_e = c_e$.

If we can find a perfect matching in G' then it will be optimum because then it and the dual weights will satisfy (4), (5), (7), and (8). Generally we will not be able to find a perfect matching in this G' or in any other such G' determined by a node-weighting. However, lets try to, just as in the bipartite case.

Choose any matching M, not necessarily perfect, in G'. If there is a node r which M doesn't meet, start growing in G' a tree T rooted at r, just as we do for the bipartite case.

Recall that when G is bipartite, either (a), (b), or (c) must hold for T in G'. (See Section II).

Bipartite or not, when we spot an occurrance of (c), we enlarge T in G'. Bipartite or not, when we spot an occurrance of (b), we get a better matching in G', we discard the current T in G', and if the matching is still not perfect we start another tree T.

In case (a), the outer nodes of T comprise a hungarian set H relative to G', and the inner nodes of T comprise the set $N(H)$ of neighbors of H relative to G'. Bipartite or not, when (a) occurs we change the dual weighting $[y_v, y_s]$ so that other edges of G enter G'. We then continue to treat the same T relative to the new G'. In general, the way the dual-weighting changes is more complicated than for a bipartite G where we don't have any positive weights y_s. Indeed, so far our weights y_s are all zero. We shall have to describe how some of them become positive. We'll do so after we describe the concept of pseudo-node.

When G is not bipartite, a fourth case (d) can occur:

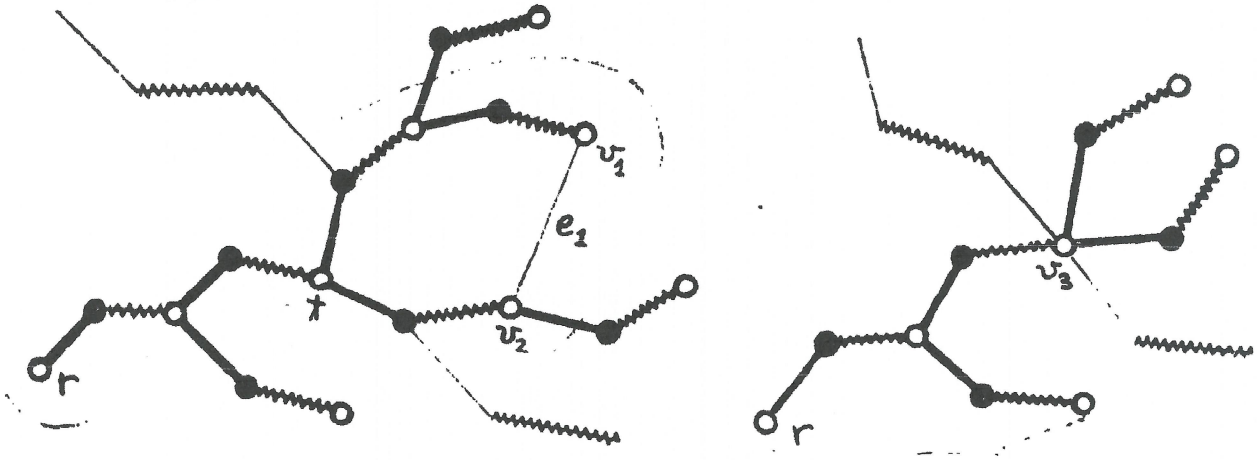
(d) Two outer nodes, say v_1 and v_2, of T are joined by an edge, say e_1, of G'.

When G is bipartite, any two outer nodes of T must be in the same part, V_1 or V_2, of G, and so (d) can not occur.

One can immediately verify that for any G' (regardless of whether G is bipartite), either (a), (b), (c), or (d) holds for T in G'.

When (d) occurrs, matters get especially tricky. .

Let P_1 be the path in T from v_1 back to r . Let P_2 be the path in T from v_2 back to r . Paths P_1 and P_2 together with the edge e_1 joining v_1 and v_2 form what we call a flower.



It consists of a stem: the path between r and t ; and a blossom B : the polygon. Ofcourse if P_1 and P_2 happen not to run together until they get back to r , then t = r and hence the stem is just the node r . It is also possible for $t = v_1$, or even $t = r = v_1$. The number of edges and the number of nodes in blossom B is odd and greater than 1, i.e., $2q + 1$ where q is some positive integer.

We now obtain a new graph G , a new subgraph G' , and a new tree T , from the ones we've got, by shrinking to a single pseudo-node, v_3 , the blossom B and all edges that have both ends in B .

The edges of the matching M that are not shrunken away do form a new matching M in the new G'. The new T, formed by the edges of the old T which are not shrunken away, is a tree in the new G' having the correct structure relative to the new M. The pseudo-node v_3 is an outer-node of the new T. If $t = r$, then v_3 is the root of the new T.

Whenever we shrink a blossom we remember it so that later we can expand the pseudo-node to recover it. We never bother to remember the edges of the current matching in a blossom that we shrink, because they are not likely to be compatible with the matching that is current when we expand the pseudo-node back to the blossom.

The algorithm continues as before, considering occurrances of (a), (b), (c), or (d), relative to the new G, G', M, and T.

Each time we spot an occurrance of (d), we shrink the blossom, thereby obtaining still another G, G', M, and T. A blossom containing pseudo-nodes might be shrunken into another pseudo-node, so we can have pseudo-nodes "inside" the pseudo-nodes of G'. The set, say s, of all real nodes inside a pseudo-node always has odd cardinality, because the sum of an odd number of odd numbers is odd. We remember every blossom we shrink so that any pseudo-node can be expanded anytime that it is not inside another pseudo-node.

Whenever we spot an occurrance of (c), we enlarge the T in the current G' .

Whenever we spot an occurrance of (b), we get a new matching in the current G' such that fewer nodes in G' are left unmet by the new matching. When this happens we discard T . If there is another node r in G' still not met by the matching, we start growing in the same G' another tree rooted at that r . Pseudo-nodes of G' formed from blossoms of earlier trees may become inner nodes or outer nodes of this tree.

Whenever there are no cases of (b), (c), or (d) to spot, we have case (a), i.e., the outer nodes of T are a hungarian set H relative to G' and the inner nodes of T are the neighbor set $N(H)$ of H relative to G'. We must in this case consider changing the dual weighting $y = [y_v, y_s]$.

In general, for the current feasible dual-weighting, $y = [y_v\ y_s]$, a y_s is positive only if s is the set of real nodes inside some pseudo-node, either a pseudo-node of the current G' or a pseudo-node at any level inside a pseudo-node of the current G' . For every edge e which is either an edge of G' or an edge of a currently shrunken blossom, we have equality, $f_e(y) = c_e$, for the constraint (5) corresponding to e ; conversely, if $f_e(y) = c_e$ holds for an edge e of the current G , i.e., for an edge e that is not inside a current pseudo-node, then e is an edge of G' . These are the senses in which a pseudo G' is the <u>equality</u> subgraph of a pseudo G .

Assuming these conditions hold, we now describe how, when (a) holds, to get a new feasible dual-weighting such that these conditions continue to hold and such that either we have a new G' relative to which (b), (c), or (d) holds for T, or else we dispose of a pseudo inner node of T, or else we have $W \to -\infty$, in which case there is no perfect matching in G. We choose ϵ to be as large as possible subject to the following constraints with right-hand sides given by the current dual-weighting.

(9) For every edge e of G, not in G', such that one end of e is an outer node of T and the other end of e is not in T, $\epsilon \leq f_e(y) - c_e$.

(10) For every edge e of G, not in G', such that both ends of e are outer nodes of T, $2\epsilon \leq f_e(y) - c_e$.

(11) For every s which is the set of all real nodes inside a pseudo inner node of T, $2\epsilon \leq y_s$.

(The pseudo inner nodes that we refer to here are nodes of the current G, not pseudo nodes inside of pseudo nodes.)

We assume for the moment that at least one such constraint exists, so that ϵ has a maximum.

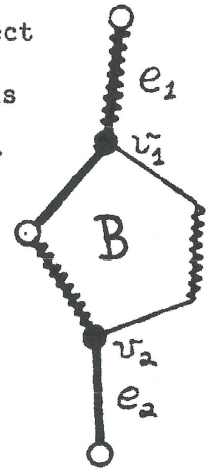
Now we change the dual weighting $[y_v, y_s]$ as follows. For every real node v which is either an outer node of T or else inside a pseudo outer node of T, lower y_v by ϵ. For every s

which is the set of all real nodes inside a pseudo outer node of T, raise y_s by 2ϵ. For every real node v which is either an inner node of T or else inside an inner node of T, raise y_v by ϵ. For every s which is the set of all real nodes inside a pseudo inner vertex of T, lower y_s by 2ϵ.

Suppose that the size of ϵ was determined by equality in some instance of either (9) or (10). Let $\dot{e}$ denote the corresponding edge. After the ϵ-adjustment of the dual-weighting, the new G' is a certain different subgraph of the same G (perhaps having pseudo-nodes). Subgraph G' is determined by G and the new dual-weighting. The same T and M are in this new G'. The edge e enters G'. If only one end of e is an outer node of T (and the other end not in T), then we have immediately an occurrance of either (b) or (c). If both ends of e are outer nodes of T, then we immediately have an occurrance of (d), a blossom to shrink as previously described.

Suppose the size of ϵ was determined by equality in some instance of (11). Let v denote the corresponding pseudo inner node. Let e_1 denote the edge of the current matching which meets v. Edge e_1 is in T. Let e_2 denote the other edge of T which meets v. After the ϵ-adjustment of the dual-weighting, we expand v to the blossom, say B, whose shrinking introduced v. This expansion gives rise to a new G and G'. It is easy to verify that B is part of the new G'. That is, $f_e(y) = c_e$ holds for the edges e of B. Let v_1 denote the node of B which edge e_1 meets ;

let v_2 denote the node of B which e_2 meets. The only node of B which is met by the current matching is v_1 ; we are able to add to the matching certain edges of B so as to get a new matching M, in the new G' , which meets all the nodes of B . The edges of the current T remain in the new G' . Unless $v_1 = v_2$, they do not form a tree in the new G' . However, they, together with (a certain one of the two paths in B joining v_2 to v_1 ~~B , minus the non-matching edge in B which meets v_2~~ , do form our new tree T . This new T does have the correct structure relative to the new G' and new M . Relative to this new T , M , G' , G , and $[y_v, y_s]$, we now return to looking for further occurrances of (a), (b), (c), or (d). Incidentally, in the situation we just treated, i.e., where the size of ϵ was determined by equality in an instance of (11), it might be that $\epsilon = 0$. This isn't relevant to the treatment.



We have finished describing all situations of the algorithm, except the two terminal situations.

One of the terminal situations is when we obtain in some G' , perhaps with pseudo nodes, a matching M which is perfect. We then expand pseudo nodes v , one after another, to the polygons B that they represent. Immediately before a pseudo node v is expanded, the matching M , that we have, is perfect in the graph G' with with node v , that we have. Let e denote the unique edge of M which, in that G' , meets node v. Expanding v to polygon B gives us a larger G' containing B instead of v . In this G' ,

e is the only edge of M that meets a node of B. By adjoining to M certain edges of B, we obtain a perfect matching M for this larger G'. Unless there are no more pseudo-nodes, we then treat some pseudo-node of this G', perhaps one in B, in the same way.

Eventually, we get a perfect matching M in the original graph G. It will be an optimum perfect matching, because it and the dual-weighting, that we have, together satisfy conditions (4), (5), (7), and (8).

The other terminal situation is an occurrance of (a) for which there are no constraints (9), (10), or (11). In this case, ϵ can be chosen as large as we please. By using $\epsilon \to \infty$ to change the dual weighting $[y_v, y_s]$, as already described, we get a feasible $[y_v, y_s]$ such that $W \to -\infty$. Hence, there can be no perfect matching in G.

This completes the description of the algorithm. Just as when G is bipartite, we can observe a bound, relative to the size of G, on the number of operations in applying the algorithm to G, which shows the algorithm to be not only finite, but very good. At the same time, Theorem 4 is proved.

VI. Theorems of Tutte, Peterson, and Konig

The algorithm also provides proof of the following theorem of W.T. Tutte (1947), analogous to Theorem 1. We define a Tutte family in a graph G to be a family of disjoint connected subgraphs G_i of G such that each G_i contains an odd number of nodes (perhaps one node) and such that, upon shrinking every G_i to a node v_i, the set of nodes v_i is a hungarian set of the resulting graph. (Tutte does not describe the family in this way.)

Theorem 5. A graph contains a perfect matching if and only if it does not contain a Tutte family.

The "only if" part is fairly easy. If G contains a Tutte family of subgraphs G_i and also a perfect matching M, then, because G_i has an odd number of nodes, at least one edge say e_i, of M has one end in G_i and the other end not in G_i. After shrinking, the edges e_i meet the nodes v_i, and have distinct nodes at their other ends. This is impossible, however, since the set of nodes v_i is hungarian.

To prove the "if" part recall the terminal situation of the algorithm where, for an occurrance of (a), there are no constraints (9), (10), and (11). The absence of a constraint (11), means that no inner node of T is pseudo. Since the outer nodes of T are a hungarian set H of G', the absence of constraints (9) and (10) means that H is also a hungarian set of G. Its neighbor set

$N(H)$ consists of the inner nodes of T, all real. The nodes in H, call them nodes v_i, are obtained by shrinking disjoint connected subgraphs G_i of the original, real-noded, G. Each G_i contains an odd number of nodes. Therefore, the original, real-noded, G contains a Tutte family.

The only alternative is the other terminal situation of the algorithm, and it yields a perfect matching. Thus, Tutte's theorem is proved.

Ofcourse, a much more pleasant proof can be obtained by stripping-down the algorithm to one for simply finding in any G either a perfect matching or a Tutte family. Indeed, finding an algorithm for the latter was a main hurdle in finding the algorithm that maximizes weight-sum. Tutte's original proof of Theorem 5 is fascinatingly unalgorithmic, and it prompted a number of programmatic efforts on the subject.

The subject of matchings started over 75 years ago with the 4-color map conjecture. The conjecture, still unproved, says that for any way of dividing up the plane into a "map", by a connected graph (a "planar" one) embedded in the plane so that every edge lies on the boundary of two different regions, the regions can be colored with only four colors so that any two regions having an edge in common are colored differently. The property of every edge lying on the boundary of two different regions of the map is equivalent to the planar graph containing no isthmus. An <u>isthmus</u>

of a connected graph, planar or not, is an edge whose deletion leaves the graph unconnected.

By "perturbing"at each node the conjecture easily reduces to the case where the graph has degree 3 at each node. The degree of a graph G at a node is the number of edge-ends in G that meet the node.

An interesting theorem, which we won't prove, is that: A planar map having degree 3 at every node can be colored (properly) with 4 colors if and only if the graph of the map contains three mutually disjoint perfect matchings. Thus, the 4-color conjecture is equivalent to the statement that any 3-degree, connected, planar graph with no isthmus contains 3 mutually disjoint perfect matchings.

In 1891, Peterson made the following contribution toward proving the 4-color conjecture.

Theorem 6. A 3-degree, connected graph G with no isthmus, whether planar or not, contains a perfect matching.

Let's prove this using Tutte's theorem. Suppose a graph G as described in Theorem 6 contains no perfect matching. Then it contains a Tutte family of subgraphs G_i. Since G_i contains an odd number of nodes, since each node has odd degree, and since the collection of edges meeting nodes in G_i has collectively an even number of edge-ends, the number of edges of G having one end in G_i, and one end not in G_i, is odd.

Not every G_i can have as many as 3 such edges, since the shrunken G_i's are a hungarian set whose smaller neighbor set must meet all such edges, and no node in the neighbor set can meet more than 3 such edges. Therefore, there is at least one subgraph G_i such that exactly one edge, say e, has one end in G_i and one end not in G_i. Edge e is then an isthmus of G, contradictin the hypothesis. So Theorem 6 is proved.

By deleting the edges of a perfect matching M from the G of Theorem 6, we are left with simply a collection of mutually disjoint polygons. Clearly, the set of edges in this collection of polygons can be partitioned into two perfect matchings if and only if each of the polygons contains an even number of edges. Considerable effort has been spent on trying to prove that when G is planar, there exists an M such that $G-M$ consists of even polygons.

If the G of Theorem 6 is bipartite, then, for any M, $G-M$ consists of even polygons, because it is easy to show that every polygon in a bipartite graph is even. Thus, the 4-color conjecture is proved for any planar map whose graph G is 3-degree and bipartite.

Indeed, though little is known about partitiioning the edges of a general k-degree graph into k perfect matchings, we do have the following theorem about k-degree bipartite graphs. The elementary theory of bipartite graphs, including this theorem and Theorem 1, is due to Konig (circa 1925).

Theorem 7. The edges of any k-degree bipartite graph can be partitioned into k perfect matchings.

For any perfect matching M in a k-degree bipartite graph G, clearly G-M is a (k-1) degree bipartite graph. Hence, it suffices to show that any k-degree bipartite G contains an M. If G doesn't contain an M, then, by Theorem 1, it contains a hungarian set H. Together the $|H|$ nodes in H meet $k \cdot |H|$ different edges. At the other ends of all these edges is $N(H)$. But this is impossible, since the degree of each node in $N(H)$ is only k, and $N(H) < |H|$. So Theorem 7 is proved.

The mystery of the 4-color conjecture seems not due to mystery about planar graphs and maps. The subject of "planarity" is very well understood. The mystery is due to the lack of a satisfactory theory about the combinatorics of coloring, i.e., partitioning. If you could find a good algorithm for deciding, for any given 3-degree graph G, whether the edges of G can be partitioned into 3 perfect matchings, then you could probably settle the 4-color conjecture.

VII. Degree-Constrained Subgraphs

Given any graph G with a real numerical weight c_e for each edge $e \in E$ and an integer b_v for each node $v \in V$, find in G, if there is one, a subgraph M which has degrees b_v at nodes v and whose edges have maximum weight-sum. This is called the "optimum b-matching problem" or the "optimum degree-constrained subgraph problem" (for "undirected graphs").

Where $b = [b_v]$, $v \in V$, a <u>b-matching</u> M in G is a subset $M \subset E$ of edges such that b_v edge-ends of edges in M meet node v. (Tutte and many other authors in graph theory would say "b-factor" rather than "b-matching".) Obviously there is a 1-1 correspondence between the b-matchings in a graph G and the b-degree subgraphs of G that contain all the nodes of G.

We allow G, and hence M, to contain loops and multiple parallel edges. A <u>loop</u> is an edge such that both of its ends meet the same node. Several edges are said to be <u>parallel</u> to each other if they all meet the same one or two nodes. Loops and multiple parallel edges are superfluous in the 1-matching problem.

Tutte (1954) generalized his Theorem 5 of the last section to a characterization of graphs G which, for given $b = [b_v]$, do not contain a b-matching. Using similar devices we shall show how to get a good algorithm for optimum b-matching

In fact, we shall generalize further, thereby including directly the integer network flow of problem of Ford and Fulkerson. The latter may be regarded as the following: Given any directed graph (network) G with a real numerical weight (cost) c_e for each edge (arc) $e \in E$ and with an integer b_v for each node $v \in V$, find in G, if there is one, a subset $M \subset E$ of edges such that, for every $v \in V$, the number of edge-ends of M directed toward v minus the number of edge-ends of M directed away from v equals b_v, and such that $\sum_{e \in M} c_e$ is maximum (or minimum). A negative b_v is called a source, and a positive b_v is called a sink.

To get the appropriate common generalization of undirected graph and directed graph, we introduce the concept of "bidirected graph". A graph G is called bidirected if each edge-end of G has either a +1 or a -1 on it. Equivalently, each edge-end is directed either toward or away from the node it meets, independently of how the other end of the same edge is directed. Equivalently, each end of an edge is either a front-end or a rear end, independently of what the other end of the same edge is.

The degree of a node v in a bidirected graph G is the number of front-ends in G that meet v minus the number of rear ends in G that meet v. With this new definition of degree, the optimum degree-constrained subgraph problem is the same as stated above.

A bidirected G is directed if every edge of G has a front end and a rear end. A bidirected G is undirected if every edge of G has two front ends. Another interesting case is where every

edge has either two front ends or two rear ends.

Another generalization is obtained by introducing single ended objects called slacks, positive slacks and negative slacks, i.e., front slacks and rear slacks. A slack in a graph meets only one node. A graph G with slacks is regarded as undirected when all ends in G including those of slacks are front ends.

Slacks conveniently represent upper and lower bound degree-constraints. Suppose we wish to find a maximum weight subgraph M of G such that the degree of M at node v is at least b_v^1 and at most b_v^2. In other words, suppose we wish to find in G a maximum weight b-matching where $b = [b_v]$ and where each b_v is in the interval $b_v^1 \leq b_v \leq b_v^2$. Obtain from G a new graph G_o by introducing at each node v of G, $b_v^2 - b_v^o$ negative slacks and $b_v^o - b_v^1$ positive slacks where b_v^o is some integer between b_v^1 and b_v^2. Give these slacks weight zero. Finding a maximum weight b_v^o - matching in G_o is equivalent to finding a maximum weight b-matching in G.

You may ask why I don't introduce "edges" with 3 ends. I would if I knew a good algorithm for handling them.

Another valuable generalization, suggested by the transportation and integer flow problems, is to maximize $U = \sum c_e x_e (e \epsilon E)$ by an integer-valued vector $x = [x_e]$ that satisfies

(1) for every element $e \epsilon E$, either an edge e or a slack e, $0 \leq x_e \leq \alpha_e$;

(2) for every node $v \in V$,

$$\sum_{e} a_{ev} x_e = b_v, \quad (e \in E),$$ where

$a_{ev} = 0$ if e does not meet v,

$a_{ev} = 1$ if e has one front end at v,

$a_{ev} = -1$ if e has one rear end at v,

$a_{ev} = 2$ if e is a loop with two front ends at v, and

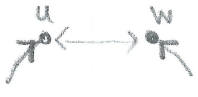
$a_{ev} = -2$ if e is a loop with two rear ends at v.

Matrix $[a_{ev}]$ is called the incidence matrix of bidirected graph G. Set V is the set of nodes in G. Set E is the set of edges, including loops, and also the slacks in G. The integer b_v is the degree-constraint at v. The integer α_e is called the capacity on e.

When every α_e is 1, this problem is simply the b-matching problem relative to graph G. When the α_e's are any positive integers, the problem is the b-matching problem relative to the graph obtained from G by replicating α_e times the element e. We also permit $\alpha_e = \infty$. Of course, if the problem has a solution it must be possible to replace $\alpha_e = \infty$ by a large α_e. However, as a matter of fact, infinite capacities are much easier to handle than finite capacities.

We now describe how any optimum b-matching problem can be reduced to an optimum 1-matching problem.

We describe first how any b-matching problem on a bidirected graph G can be reduced to a b*-matching problem on an undirected graph G*. For each node v in G, let there be two nodes, say u and w, in G*. Let all the front ends at v be front ends at u in G*. Let all the rear ends at v be front ends at w in G*. Let there be in G* a new edge e with a front end at u, a front end at w, and any



appropriately large capacity. Let the degree constraints b^*_u and b^*_w be appropriately large and such that $b^*_u - b^*_w = b_v$. Let every edge or slack in G have the same weight in G* as in G. Let the new edges in G* have weight zero.

One can verify that if G is directed then G* is bipartite.

We next describe how any b-matching problem on an undirected graph G can be reduced to a b*-matching problem on an undirected graph G* such that the edge capacities are all ∞ and such that there are no slacks.

For each edge e of G having finite capacity α_e, and meeting say nodes u and w in G, replace e by a path P_e of three edges joining u to w; give the two new nodes, interior to P_e, degree constraints equal to α_e. Give one of the non-middle edges of P_e the weight that e had; give zero weights to the other two edges of P_e. Let r denote a special new node. For each slack e of G, having capacity α_e, and meeting say node v in G, replace e by a path P_e of two edges joining v to r; give the new node, interior to path P_e, a degree-constraint equal to α_e or b_v, whichever is smaller. Give the edge of P_e that meets v the weight that slack e had; give zero weight to the other edge of P_e. Let there be a zero-weighted loop with both (front) ends at r. Give r any appropriately large degree-constraint whose parity is such that the sum of the degree-constraints on all nodes is even. The result of this construction is the desired b*-matching problem.

Next we describe how to reduce any b-matching problem on an undirected graph G, such that there are no slacks and such every edge e has capacity $\alpha_e = \infty$, to a perfect matching problem (like treated in section V) on