An Efficient Implementation of Edmonds' Algorithm for Maximum Matching on Graphs

HAROLD N. GABOW

University of Colorado, Boulder, Colorado

ABSTRACT. A matching on a graph is a set of edges, no two of which share a vertex. A maximum matching contains the greatest number of edges possible. This paper presents an efficient implementation of Edmonds' algorithm for finding a maximum matching The computation time is proportional to V^3, where V is the number of vertices; previous implementations of Edmonds' algorithm have computation time proportional to V^4. The implementation is based on a system of labels that encodes the structure of alternating paths.

KEY WORDS AND PHRASES: graph algorithm, matching on a graph, maximum matching, augmenting path, labeling technique

CR CATEGORIES: 5.25, 5.32, 5.41

1. *Introduction*

The problem of finding a maximum matching on a graph has applications in operations research and integer programming. For example, the following is a maximum matching problem:

In a factory, a manager must divide his workers into teams of two. Certain teams are not allowed, because the workers are incompatible. Choose the greatest possible number of teams of compatible workers.

We present an algorithm for finding a maximum matching on a graph. If V is the number of vertices, the run time is proportional to V^3. The space required is $4V$ words in addition to the space needed for the graph.

The approach is a careful implementation of ideas presented by Edmonds [4]. His algorithm has run time proportional to V^4 [4; 6, Erratum]. We improve this by eliminating the process of blossom expansion. Instead, we use a system of labels to store the structure of alternating paths.

This approach is similar to labeling techniques in the matching algorithms of Balinski [1] and Witzgall and Zahn [13]. The former algorithm has run time proportional to V^3, if a stack is used for vertex selection; the latter algorithm can be implemented in time proportional to V^3, using techniques described here. However, both algorithms may label a vertex more than once in a search. This increases the run time and makes it difficult to generalize to other problems, such as finding a maximum weighted matching [13]. The present algorithm overcomes these difficulties [8].

This work was partially supported by the National Science Foundation Graduate Fellowship Program and the National Science Foundation under Grant GJ-1180 at Stanford University, and by the National Science Foundation under Grant GJ-660 at the University of Colorado.

Author's address: Department of Computer Science, University of Colorado, Boulder, CO 80302.

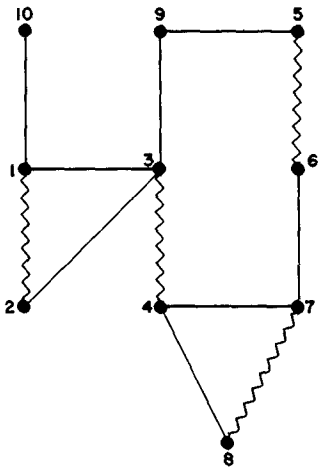


FIG. 1 Matched graph G_1

After summarizing definitions in Section 2, we state the algorithm in Section 3. A proof of correctness is given in Section 4. Section 5 discusses time and space bounds, and applications of the algorithm.

2. *Preliminaries*

This section summarizes some well-known definitions and results.

A graph G consists of a finite set of vertices and a finite set of edges. An *edge* is an (unordered) set of two distinct vertices. The edge containing vertices v and w is denoted vw (or wv). Vertices v and w are *adjacent*. An *adjacency list* for v is a list of the vertices adjacent to v. A *subgraph* of G is a graph whose vertices and edges are in G. A graph is *complete* if any two vertices are adjacent.

A *walk* W is a list of vertices $(v_1, v_2, \cdots, v_n)$, where $n \geq 1$ and v_iv_{i+1} is an edge, for $1 \leq i < n$. A walk is *simple* if no vertex occurs more than once in the list. A *path* is a simple walk. A *cycle* is a walk $(v_1, v_2, \cdots, v_n, v_1)$ such that $n > 2$ and $(v_1, v_2, \cdots, v_n)$ is simple.

Let $W = (v_1, v_2, \cdots, v_n)$ and $X = (v_{n+1}, v_{n+2}, \cdots, v_m)$ be walks. The *reverse* walk of W, denoted *rev* W, is $(v_n, v_{n-1}, \cdots, v_1)$. The *concatenation* of W and X, denoted $W*X$, is $(v_1, \cdots, v_n, v_{n+1}, \cdots, v_m)$. For $W*X$ to be a walk, it is necessary that v_nv_{n+1} is an edge. For $W*X$ to be a path, it is further necessary that W and X are disjoint paths.

A *matching* on a graph is a set of edges, no two of which share a vertex. A *matched graph* (G, M) is a graph G with a matching M. A vertex v is *matched* if it is in some edge of the matching; otherwise v is *unmatched*. M is a *maximum matching* if no matching on G contains more edges than M. Figure 1 shows a matching on a graph G_1 (matched edges are shown as wavy lines). Vertices 9 and 10 are unmatched. The matching is not maximum, since a matching with no unmatched vertices exists.

An *alternating path* in a matched graph is a path $(v_1, \cdots, v_n)$ such that exactly one of every two edges $v_{i-1}v_i$ and v_iv_{i+1} is matched, for $1 < i < n$. An *augmenting path* is an alternating path whose ends v_1 and v_n are distinct unmatched vertices.

If $(v_1, \cdots, v_{2n})$ is an augmenting path in (G, M), a new matching M' is obtained by replacing the matched edges $v_{2i}v_{2i+1}$, $1 \leq i < n$, with the unmatched edges $v_{2i-1}v_{2i}$, $1 \leq i \leq n$. We say the matching M is *augmented* to M', since M' contains one more edge than M. In Figure 1, (10, 1, 2, 3, 4, 8, 7, 6, 5, 9) is an augmenting path. Augmenting gives a maximum matching.

Note that an augmenting path is simple. We cannot augment a matching with a non-

simple alternating walk. This is illustrated in Figure 1 by the walk (10, 1, 2, 3, 4, 8, 7, 4, 3, 9). "Augmenting" does not give a matching, since edges 47 and 48 become "matched."

Augmenting paths are important for this reason:

LEMMA (Berge). *A matched graph* (G, M) *has an augmenting path if and only if* M *is not maximum.*

PROOF. See [2, 4].

As a result, a maximum matching can be found by repeatedly searching for augmenting paths and augmenting the matching. The algorithms in [1, 2, 13] and the one presented here are organized in this way.

3. *Statement of the Algorithm*

This section presents an algorithm, called E, for finding a maximum matching on a graph. First, the basic strategy and the data structures of E are described. Then Algorithm E is stated. An example of how E works is given. Finally, E is compared with Edmonds' algorithm.

The algorithm begins by numbering the vertices and edges of the graph. Below we do not distinguish between a vertex v and its number; we denote both by v. We denote the number of an edge vw as $n(vw)$.

Algorithm E constructs a number of matchings, the last of which is maximum. A matching is stored in the array $MATE$. This array has an entry for each vertex. If v and w are vertices, edge vw is matched if $MATE(v) = w$ and $MATE(w) = v$.

Algorithm E begins with all vertices unmatched. It searches for an augmenting path. If such a path is found, the matching is augmented. The new matching contains one more edge than the previous one. Next E searches for an augmenting path for the new matching. This process is iterated. Eventually, E constructs a matching that has no augmenting path. This matching is maximum, by Berge's Lemma.

Algorithm E searches for an augmenting path in the following way. First an unmatched vertex u is chosen. E scans edges to find alternating paths to u. A vertex v is called *outer* when E finds an alternating path from v to u that starts with a matched edge. Let such a path be $P(v) = (v, v_1, \cdots, u)$, so vv_1 is matched. E sets an entry in the $LABEL$ array for every outer vertex v. Path $P(v)$ can be computed from $LABEL(v)$. If an edge joining an outer vertex v to an unmatched vertex $u' \neq u$ is scanned, E finds an augmenting path, $(u')*P(v) = (u', v, v_1, \cdots, u)$. If no such edge is ever scanned, vertex u is not in an augmenting path.

Figure 2 illustrates a search for an augmenting path to vertex $u = 9$. Figure 2(a) shows paths $P(3)$ and $P(7)$. Figure 2(b) shows the values stored by E. Now we explain these values.

The $LABEL$ entry for an outer vertex is interpreted as either a start label, vertex label, or edge label. In Figure 2, eight vertices are outer. Each is labeled in one of these ways. The remaining vertex, 1, is *nonouter*. This means there is no alternating path from 1 to 9 that starts with a matched edge. Nonouter vertices are drawn hollow in all figures in this paper.

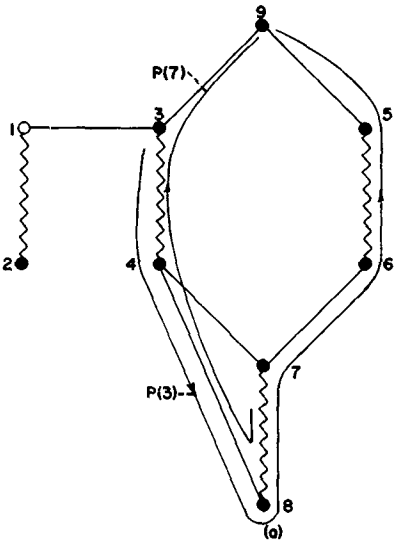
Now we describe the three label types.

Start label. In the search for an augmenting path to the unmatched vertex u, u has a start label. This defines an alternating path, $P(u) = (u)$.

Vertex label. If outer vertex v has a vertex label, $LABEL(v)$ is the number of another outer vertex. Path $P(v)$ is defined as $(v, MATE(v))*P(LABEL(v))$. Using this definition, we compute $P(8)$:

$$P(8) = (8, MATE(8))*P(LABEL(8)) = (8, 7)*(4, 3, 9) = (8, 7, 4, 3, 9).$$

Edge label. If outer vertex v has an edge label, $LABEL(v)$ contains the number of an



v	$MATE(v)$	Label type	$LABEL(v)$	$FIRST(v)$
1	2	nonouter	—	—
2	1	vertex	3	1
3	4	edge	$n(67)$	0
4	3	vertex	9	0
5	6	edge	$n(67)$	0
6	5	vertex	9	0
7	8	edge	$n(48)$	0
8	7	vertex	4	0
9	—	start	—	0

(b)

FIG 2. Search values

edge joining two outer vertices, $LABEL(v) = n(xy)$. Path $P(v)$ is defined in terms of paths $P(x)$ and $P(y)$. As an example, consider vertex 7, which has label $n(48)$. Vertices 4 and 8 are outer, so there are alternating paths $P(4)$ and $P(8)$. Vertex 7 is in $P(8)$. Let $P(8, 7)$ denote the portion of $P(8)$ from 8 to 7, i.e. $P(8, 7) = (8, 7)$. Then $P(7)$ is defined as the path $(rev\ P\ (8, 7))*P(4)$, i.e.

$$P(7) = (rev\ (8, 7))*(4, 3, 9) = (7, 8, 4, 3, 9).$$

Path $P(3)$ is defined similarly. The label of vertex 3 is $n(67)$. Since vertex 3 is in $P(7)$, path $P(3) = (rev\ P(7, 3))*P(6)$.

E also uses an array $FIRST$. If v is an outer vertex, $FIRST(v)$ is the first nonouter vertex in $P(v)$. In Figure 2, the first nonouter vertex in $P(2)$ is $FIRST(2) = 1$. Path $P(7)$ does not contain any nonouter vertices, so $FIRST(7)$ is set to 0, a dummy vertex. If edge 67 is removed, vertices 2, 3, and 5 become nonouter, and $FIRST(7)$ becomes 3.

The array $FIRST$ speeds up the computation. Using $FIRST$, E finds the first nonouter vertex in $P(v)$ with a table look-up. Without $FIRST$, this operation involves computing vertices in $P(v)$ until a nonouter vertex is found. Thus $FIRST$ enables E to do in constant time what otherwise requires time proportional to V, the number of vertices. This speedup is crucial in achieving the $O(V^3)$ time bound.

Now we state Algorithm E in detail.

The vertices of the graph are numbered from 1 to V. E also uses a dummy vertex 0 for boundary conditions.

The edges of the graph are stored in some standard manner, such as an adjacency matrix [9] or adjacency lists. For convenience we choose adjacency lists, using an approach described by Tarjan [11]. Let W be the number of edges in the graph. An array END has entries numbered from $V + 1$ to $V + 2W$. For each edge, there are two consecutive entries, containing the numbers of the vertices in the edge. Thus edge vw is stored as the ith edge when $END(V + 2i - 1) = v$ and $END(V + 2i) = w$. The edge number $n(vw)$ is $V + 2i$. Thus, v and w can be easily computed from $n(vw)$. There is an adjacency list for each vertex v. This list contains the numbers $n(vw)$ of all edges containing v.

The array END and the adjacency lists are referenced implicitly in the algorithm. See, for example, step $E2$, below.

The $MATE$ array has an entry for each vertex. $MATE$ specifies a matching. If v,

$w \neq 0$ are vertices, $MATE(v) = 0$ if v is unmatched; edge vw is matched if $MATE(v) = w$ and $MATE(w) = v$.

The $LABEL$ array has an entry for each vertex. In a given search, a vertex v is outer if $LABEL(v) \geq 0$. If v has a vertex label, $LABEL(v)$ is a vertex number between 1 and V. If v has an edge label, $LABEL(v)$ is an edge number between $V + 1$ and $V + 2W$. These classifications are used implicitly in the algorithm, in tests like "If the vertex is outer, then. . . ." See, for example, step $E4$.

The $FIRST$ array has an entry for each vertex. In a given search, if v is an outer vertex then $FIRST(v)$ is the first nonouter vertex in $P(v)$.

The algorithm is presented below in a high-level language similar to Knuth's [10]. E is the main routine. It uses subroutines L and R.

E constructs a maximum matching on a graph It starts a search for an augmenting path to each unmatched vertex u. It scans edges of the graph, deciding to assign new labels or to augment the matching

*E*0. [Initialize.] Read the graph into adjacency lists, numbering the vertices 1 to V and the edges $V + 1$ to $V + 2W$. Create a dummy vertex 0 For $0 \leq i \leq V$, set $LABEL(i) \leftarrow -1$, $MATE(i) \leftarrow 0$ (all vertices are nonouter and unmatched) Set $u \leftarrow 0$

*E*1. [Find unmatched vertex] Set $u \leftarrow u + 1$. If $u > V$, halt; $MATE$ contains a maximum matching Otherwise, if vertex u is matched, repeat step $E1$ Otherwise (u is unmatched, so assign a start label and begin a new search) set $LABEL(u) \leftarrow FIRST(u) \leftarrow 0$.

*E*2 [Choose an edge] Choose an edge xy, where x is an outer vertex. (An edge vw may be chosen twice in a search—once with $x = v$, and once with $x = w$.) If no such edge exists, go to $E7$. (Edges xy can be chosen in an arbitrary order. A possible choice method is "breadth-first": an outer vertex $x = x_1$ is chosen, and edges x_1y are chosen in succeeding executions of $E2$, when all such edges have been chosen, the vertex x_2 that was labeled immediately after x_1 is chosen, and the process is repeated for $x = x_2$. This breadth-first method requires that Algorithm E maintain a list of outer vertices, $x_1, x_2 \cdots$.)

*E*3. [Augment the matching.] If y is unmatched and $y \neq u$, set $MATE(y) \leftarrow x$, call $R(x, y)$, then go to $E7$ (R completes the augment along path $(y)*P(x)$)

*E*4. [Assign edge labels.] If y is outer, call L, then go to $E2$ (L assigns edge label $n(xy)$ to nonouter vertices in $P(x)$ and $P(y)$)

*E*5. [Assign a vertex label.] Set $v \leftarrow MATE(y)$. If v is nonouter, set $LABEL(v) \leftarrow x$, $FIRST(v) \leftarrow y$, and go to $E2$ (See Figure 3)

*E*6. [Get next edge.] Go to $E2$ (y is nonouter and $MATE(y)$ is outer, so edge xy adds nothing).

*E*7. [Stop the search] Set $LABEL(0) \leftarrow -1$. For all outer vertices i, set $LABEL(i) \leftarrow LABEL(MATE(i)) \leftarrow -1$ Then go to $E1$ (now all vertices are nonouter for the next search).

L assigns the edge label $n(xy)$ to nonouter vertices Edge xy joins outer vertices x, y. L sets *join* to the first nonouter vertex in both $P(x)$ and $P(y)$. Then it labels all nonouter vertices preceding *join* in $P(x)$ or $P(y)$ See Figure 4.

*L*0. [Initialize.] Set $r \leftarrow FIRST(x)$, $s \leftarrow FIRST(y)$. If $r = s$, return (no vertices can be labeled).

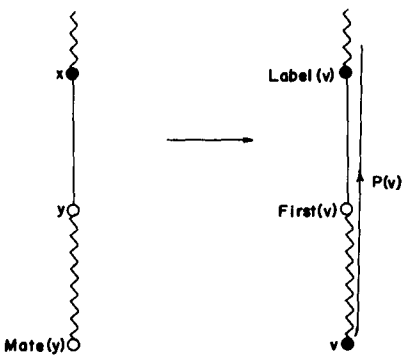


FIG. 3 Assigning a vertex label

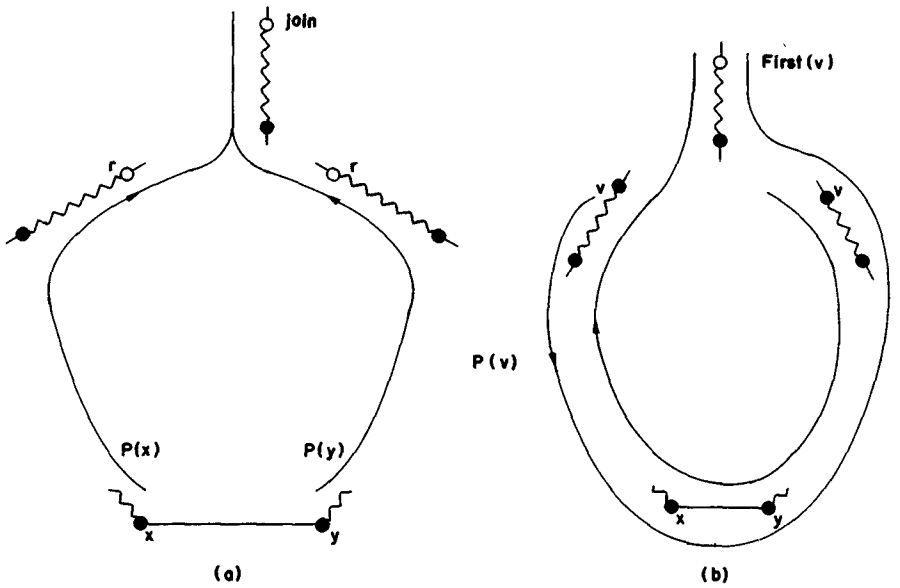


FIG. 4. Assigning edge labels

Otherwise flag r and s. (Steps $L1$–$L2$ find *join* by advancing alternately along paths $P(x)$ and $P(y)$. Flags are assigned to nonouter vertices r in these paths. This is done by setting $LABEL(r)$ to a negative edge number, $LABEL(r) \leftarrow -n(xy)$. This way, each invocation of L uses a distinct flag value.)

$L1$. [Switch paths] If $s \neq 0$, interchange r and s, $r \leftrightarrow s$ (r is a flagged nonouter vertex, alternately in $P(x)$ and $P(y)$).

$L2$. [Next nonouter vertex.] Set $r \leftarrow FIRST(LABEL(MATE(r)))$ (r is set to the next nonouter vertex in $P(x)$ or $P(y)$). If r is not flagged, flag r and go to $L1$ Otherwise set *join* $\leftarrow r$ and go to $L3$.

$L3$. [Label vertices in $P(x)$, $P(y)$.] (All nonouter vertices between x and *join*, or y and *join*, will be assigned edge labels. See Figure 4(a).) Set $v \leftarrow FIRST(x)$ and do $L4$. Then set $v \leftarrow FIRST(y)$ and do $L4$. Then go to $L5$.

$L4$ [Label v] If $v \neq join$, set $LABEL(v) \leftarrow n(xy)$, $FIRST(v) \leftarrow join$, $v \leftarrow FIRST(LABEL(MATE(v)))$ and repeat step $L4$ (See Figure 4(b).) Otherwise continue as specified in $L3$.

$L5$ [Update $FIRST$.] For each outer vertex i, if $FIRST(i)$ is outer, set $FIRST(i) \leftarrow join$. (*Join* is now the first nonouter vertex in $P(i)$)

$L6$. [Done] Return

$R(v, w)$ rematches edges in the augmenting path. Vertex v is outer. Part of path $(w)*P(v)$ is in the augmenting path. It gets rematched by $R(v, w)$ (Although R sets $MATE(v) \leftarrow w$, it does not set $MATE(w) \leftarrow v$. This is done in step $E3$ or another call to R.) R is a recursive routine.

$R1$. [Match v to w] Set $t \leftarrow MATE(v)$, $MATE(v) \leftarrow w$. If $MATE(t) \neq v$, return (the path is completely rematched)

$R2$. [Rematch a path.] If v has a vertex label, set $MATE(t) \leftarrow LABEL(v)$, call $R(LABEL(v), t)$ recursively, and then return.

$R3$. [Rematch two paths.] (Vertex v has an edge label) Set x, y to vertices so $LABEL(v) = n(xy)$, call $R(x, y)$ recursively, call $R(y, x)$ recursively, and then return.

We illustrate how E constructs a maximum matching on graph G_1 of Figure 1. Initially, all vertices are unmatched. E searches for an augmenting path to vertex 1. The first edge chosen, 12, forms such a path. An augment is done by placing 12 in the matching. E sets $MATE(1) \leftarrow 2$, $MATE(2) \leftarrow 1$.

In a similar manner, edges 34, 56, and 78 are matched. This gives the matching in Figure 1.

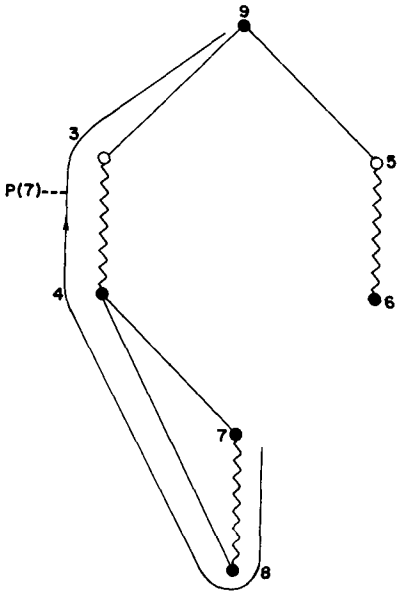


FIG. 5. Search from vertex 9

In the last search, vertex 9 gets a start label. Edge 93 is scanned, and vertex 4 gets a vertex label; similarly, vertices 6 and 8 get vertex labels. When E scans edge 48, vertex 7 gets an edge label. The result is Figure 5. (Only scanned edges are shown. The $LABEL$ values of outer vertices are shown in Figure 2.)

Now we describe how vertices 3 and 5 are labeled, as shown in Figure 2. E scans edge 67, and subroutine L is called to assign the label $n(67)$.

L computes *join* in steps $L0$–$L2$, as follows:

1. In step $L0$, the first nonouter vertex in $P(6)$ is computed as $FIRST(6) = 5$. The first nonouter vertex in $P(7)$ is computed as $FIRST(7) = 3$. Vertices 5 and 3 are flagged by setting $LABEL(5) \leftarrow LABEL(3) \leftarrow -n(67)$.

2. In step $L2$, the next nonouter vertex in $P(7)$ is computed as $FIRST(9) = 0$. Vertex 0 is flagged.

3. In step $L2$, the next nonouter vertex in $P(6)$ is computed as $FIRST(9) = 0$. Since 0 is already flagged, *join* is set to 0.

In steps $L3$–$L4$, L assigns the label $n(67)$ to vertices 5 and 3.

L resets $FIRST(i)$ for $i = 4, 6, 7, 8$, in step $L5$. No nonouter vertices remain in $P(i)$, so $FIRST(i)$ is set to 0.

Finally, L returns.

Now E continues scanning edges. Vertex 2 gets a vertex label; the result is Figure 2. When edge 32 is scanned, vertex 1 gets an edge label, $n(32)$. Finally edge 1 10 is scanned, and the augmenting path $(10)*P(1)$ is found.

The augment is done in step $E3$ and subroutine R. Step $E3$ matches vertex 10, and calls $R(1, 10)$ to rematch the remainder of $(10)*P(1)$. Figure 6(a) shows the result of $R(1, 10)$: Edge 1 10 is matched, and two recursive calls are pending on R, $R(3, 2)$ and $R(2, 3)$. (In Figure 6(a), edge 12 is drawn half-wavy, denoting $MATE(2) = 1$ but $MATE(1) \neq 2$.) Path $P(1)$ is defined as $(rev\ P(2, 1))*P(3)$. The call $R(3, 2)$ processes path $P(3)$. Figure 6(b) shows the matching when $R(3, 2)$ is complete. ($R(3, 2)$ makes recursive calls $R(6, 7)$ and $R(7, 6)$.) Then the call $R(2, 3)$ processes path $rev\ P(2, 1)$. It sets $MATE(2) = 3$, completing the augment.

Now $MATE$ contains a maximum matching. The algorithm halts in step $E1$.

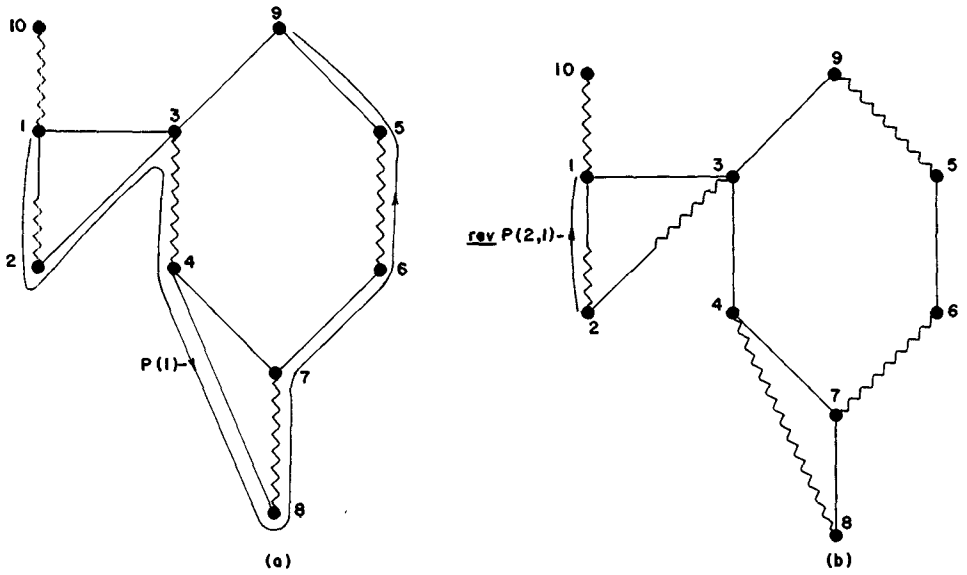


FIG. 6. Augment path (10)*P(1)

For comparison we briefly describe how Edmonds' algorithm [4] finds the same matching on G_1. We discuss the search for an augmenting path to vertex 9.

The search begins by growing a tree consisting of the edges in Figure 5, except for edge 48. When this edge is scanned, it completes a cycle, (4, 7, 8, 4). Edmonds defines a *blossom* as an odd number of vertices joined by a maximally matched cycle. Vertices 4, 7, and 8 form a blossom. These vertices and the edges between them are shrunk into a single vertex, b. Vertex b is adjacent to any vetrex adjacent to 4, 7, or 8, b is matched with vertex 3. The result is a reduced graph G_1'.

Now the problem is to find an augmenting path in G_1'. Suppose the path (10, 1, 2, 3, b, 6, 5, 9), corresponding to (10)*P(1), is found. The matching in G_1' is augmented. So edge b6 becomes matched. Then blossom b is expanded into the original cycle (4, 7, 8, 4). Vertex 7 is matched to 6. The remaining vertices are matched along edges of the cycle. The result is a maximum matching.

The intermediate steps that find the augmenting path in G_1' are similar. Two more blossoms are shrunk. (These correspond to edge labels $n(67)$ and $n(32)$.) In the augment, these two blossoms are expanded and rematched.

The implementation of this elegant algorithm requires some care. A time bound of $O(V^4)$ results from (possibly) V^2 blossom expansion operations, each requiring time $O(V^2)$. Algorithm E avoids shrinking and expansion by recording the pertinent structure of blossoms in *LABEL* and *FIRST*. This results in a factor of V speedup.

4. *Proof of Correctness*

This section proves that Algorithm E constructs a maximum matching. It shows that E constructs valid augmenting paths; each matching is augmented correctly; and the last matching is maximum.

It is convenient to introduce the dummy vertex, 0, to handle boundary conditions. In any search, we assume vertex 0 is nonouter, and is "matched" with the unmatched vertex u. We also extend the paths $P(v)$ to vertex 0, as follows:

Definition 1. An *outer path* is an alternating path $(v, v_1, \cdots, u, 0)$ that starts with a matched edge vv_1 and ends with the dummy vertex, 0.

The definition guarantees an outer path contains at least one nonouter vertex.

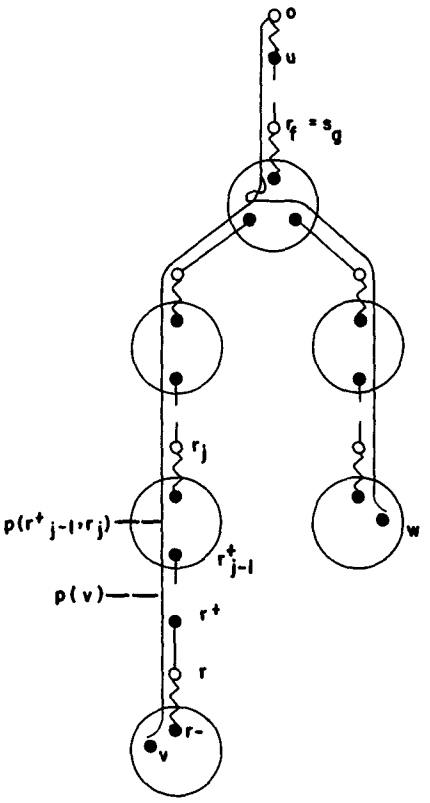


FIG. 7. Search graph

The first task is to prove that in step $E3$, $(y)*P(x)$ is an augmenting path. It suffices to show that $P(x)$ is an outer path. We do this below, in Corollary 1. The main issue is proving $P(x)$ is simple.

We start by defining a search graph, which gives the properties of a search conducted by E. Functions p, f, and l in the definition correspond to P, *FIRST*, and *LABEL* in Algorithm E.

Definition 2. A *search graph* (G, O, u, p, f, l) consists of: a matched graph G; a set of vertices, O, called *outer vertices*; an unmatched outer vertex u; a function p, mapping an outer vertex v to $p(v)$, an outer path starting at v; a function f, mapping an outer vertex v to $f(v)$, the first nonouter vertex in $p(v)$; and a function l, mapping certain outer vertices v to $l(v)$, another outer vertex. In addition, the following properties are satisfied for an outer vertex v. Let $r = f(v)$, the first nonouter vertex in $p(v)$; let r^- be the vertex matched with r; and let $r^+ = l(r^-)$.

2.1. Path $p(v) = p(v, r)*p(r^+)$.

2.2. An outer vertex x in $p(v, r)$ has $f(x) = r$.

Figure 7 shows an outer path $p(v)$ in a search graph. All outer vertices x with the same vertex $f(x)$ are grouped in a circle. Vertex $r = f(v)$ is the first nonouter vertex in path $p(v)$. Consider a vertex x in v's circle. Properties 2.1 and 2.2 imply that $p(x)$ consists of a path inside the circle to r^-; followed by edges r^-r, rr^+; followed by $p(r^+)$. (Note that the circles in Figure 7 correspond to blossoms in Edmonds' algorithm.)

Figure 2 shows a search graph constructed by E. The outer vertex 3 illustrates property 2.2, since all vertices x in $P(3, 9)$ have $FIRST(x) = 0$.

The decomposition property 2.1 at first seems too weak. It seems natural that $p(v) = (v, v_1, \cdots, v_{2i-1}, v_{2i}, \cdots)$ decomposes as $p(v, v_{2i-1})*p(v_{2i})$, for any i. However this is false. In Figure 2, $p(3) \neq (3, 4)*p(8)$.

Now we derive the structure of path $p(v)$ shown in Figure 7. Let the nonouter vertices in $p(v)$ be r_j, for $0 \leq j \leq J$. Thus $r_0 = r = f(v)$ and $r_J = 0$. Property 2.1 shows that for any i in $0 \leq i \leq J$,

$$p(v) = p(v, r_0)*p(r_0^+, r_1)* \cdots *p(r_{j-1}^+, r_j)* \cdots *p(r_i^+). \tag{1}$$

A vertex $x \neq r_j$ in $p(r_{j-1}^+, r_j)$ is outer, and $f(x) = r_j$.

Next we derive the relationship between two outer paths $p(v)$ and $p(w)$, as shown in Figure 7. Let the nonouter vertices in $p(w)$ be s_k, for $0 \leq k \leq K$. Let z be the first outer vertex in $p(v)$ that is also in $p(w)$. Decomposition (1) shows $f(z)$ is a nonouter vertex in both paths, $f(z) = r_f = s_g$ for some indices f, g. So it is easy to see that $p(v)$ and $p(w)$ are identical after $r_f = s_g$, and that $p(v, r_{f-1})$ and $p(w, s_{g-1})$ are disjoint. Thus, as shown in Figure 7, $p(v)$ and $p(w)$ both partition into three subpaths. The first subpaths, $p(v, r_{f-1})$ and $p(w, s_{g-1})$, are disjoint; the last subpaths, $p(r_f^+)$ and $p(s_g^+)$, are identical; the middle subpaths, $p(r_{f-1}^+, r_f)$ and $p(s_{g-1}^+, s_g)$, intersect arbitrarily.

Now we show that E maintains a search graph. First we formally define the function P:

Definition 3. The *outer path function* P for Algorithm E is defined (recursively) as follows:

1. The unmatched vertex u has outer path $P(u) = (u, 0)$.

2. If v has a vertex label, $LABEL(v)$ is the number of an outer vertex, and $P(v) = (v, MATE(v))*(LABEL(v))$.

3. If v has an edge label, $LABEL(v)$ is the number of edge xy, where x and y are outer vertices. Either $v \in P(x)$ or $v \in P(y)$. In the former case,

$$P(v) = (rev\ P(x, v))*P(y); \qquad \text{otherwise, } P(v) = (rev\ P(y, v))*P(x).$$

LEMMA 1. *Each time step E2 is executed in Algorithm E, a search graph is formed by* $(G, O, u, P, FIRST, LABEL)$.

PROOF. The argument is by induction. Assume a search graph is formed, with outer vertices O. Below we show that if E assigns an edge label $n(xy)$ to new vertices O', then a search graph is formed, with outer vertices $O \cup O'$. The case where E assigns a vertex label is left as an easy exercise.

Edge labels are assigned in step $E4$ and subroutine L. From Figure 7, we see steps $L0$–$L4$ work as follows:

In steps $L0$–$L2$, consecutive nonouter vertices in $P(x)$ and $P(y)$ are flagged. (In a search graph, the nonouter vertex after r is $f(l(r^-))$.)

In step $L2$, *join* is set to the first nonouter vertex common to $P(x)$ and $P(y)$. (In Figure 7, *join* $= r_f = s_g$.)

In step $L4$, a label is assigned to each nonouter vertex v preceding *join* in $P(x)$ or $P(y)$.

Now we check the search graph properties for an outer vertex v. We assume first $v \in O$, and then $v \in O'$.

If $v \in O$, let r be the vertex $FIRST(v)$ before L is executed. We assume r is labeled in step $L4$, since otherwise there is nothing new to check. Either $r \in P(x)$ or $r \in P(y)$; assume the former. Figure 7, applied to paths $P(v)$ and $P(x)$, shows that these paths are identical after r. So after L is executed, the first nonouter vertex in $P(v)$ is *join*. The value $FIRST(v)$ is set to *join* in step $L5$. Thus array $FIRST$ is maintained correctly.

Properties 2.1 and 2.2 follow easily from (1). Thus vertices in O satisfy all search graph properties.

Now we check these properties for a vertex $v \in O'$. Before step $L4$, v is a nonouter vertex in $P(x)$ or $P(y)$. Assume the former, so $P(v) = (rev\ P(x, v))*P(y)$. This defines an outer path (see Figure 7). In particular, $P(v)$ is simple, since $P(x, v)$ and $P(y)$ are disjoint. Thus $P(v)$ is defined correctly.

The remaining search graph properties for v follow from those for vertices $x, y \in O$.

The lemma now follows by induction. □

COROLLARY 1. *E labels vertices v so $P(v)$ is an outer path starting at v.*

Thus we see that E constructs valid augmenting paths. Next we show that E augments a matching correctly. We begin with two useful definitions.

In an augment, step $E3$ and subroutine R change values of $MATE$. A nonzero vertex v is *originally matched* if $MATE(v)$ is unchanged from its value before step $E3$ begins. For example, in Figure 6(a), all vertices except 1, 10, and 0 are originally matched.

Define a partial order on vertices, $\circledS$, as follows: $v \circledS w$ means that v is an outer vertex, and v is labeled before w is labeled. In Figure 2, $9 \circledS 7 \circledS 1$. We consider vertices labeled in the same call to L as being labeled simultaneously. So neither $3 \circledS 5$ nor $5 \circledS 3$.

LEMMA 2. *Let $R(v, w)$ be called, with v an outer vertex and $w \notin P(v)$. Suppose the first vertex of $P(v) = (v, v_1, v_2, \cdots)$ that is not originally matched is v_{2m+1}, and $v \circledS v_{2m+1}$.*

Then R changes $MATE(v_i)$, $0 \leq i \leq 2m$, to give a maximum matching of the path $(w)*P(v, v_{2m})$ (i.e. *$MATE(v) = w$, and for $1 \leq i \leq m$, $MATE(v_{2i-1}) = v_{2i}$, $MATE(v_{2i}) = v_{2i-1}$*).

PROOF. The proof is by induction on m. The argument falls into three cases: $m = 0$; $m > 0$ and v has a vertex label; $m > 0$ and v has an edge label. For details, see [8]. □

COROLLARY 2. *E augments a matching correctly.*

PROOF. We show that after step $E3$ and R are executed, $MATE$ is changed according to an augment along the path $(y)*P(x)$.

The value $MATE(y)$ is set correctly in step $R3$. When $R(x, y)$ is called, all vertices in $P(x)$ are originally matched, except vertex 0. The hypotheses of the lemma are satisfied with $v_{2m+1} = 0$. So when $R(x, y)$ returns, $MATE$ is changed to give a maximum matching of $(y)*P(x)$. □

The final task is to show the last matching is maximum. Suppose vertex u is unmatched in the last matching. In some execution of step $E1$, a search is started from vertex u. This search terminates without augmenting. Let S_u denote the search; let M_u denote the matching when search S_u is made. We investigate how subsequent searches interact with S_u. The following concept is central [4].

Definition 4. The *Hungarian subgraph H for vertex u* is a subgraph of G. It consists of all edges containing an outer vertex of S_u, and all vertices in these edges.

In G_1, if edge 23 is deleted, Figure 2 shows the Hungarian subgraph for vertex 9. Note these obvious properties of a Hungarian subgraph H: In search S_u, each edge of H is chosen at least once in step $E2$. If vertex $v \in H - u$, the matched edge containing v is in H.

The basic result is that no augmenting path constructed after S_u intersects H.

LEMMA 3. *Suppose a matching M agrees with M_u on H, $M \cap H = M_u \cap H$. Then no augmenting path for M contains a vertex in H.*

PROOF. The hypothesis implies we can refer to "a matched edge in H" unambiguously. We do so below.

Let Q be an augmenting path for M containing a vertex in H. We derive a contradiction, proving the lemma.

Path Q is not contained entirely in H. So we can set $Q = (v_1, v_2, \cdots, v_{2n})$, where for some i in $1 \leq i < n$, vertex $v_{2i+1} \in H$ and $v_{2i+2} \notin H$. The matched edge $v_{2i}v_{2i+1}$ is in H. So a vertex in this edge is outer in S_u. Since v_{2i+1} is nonouter, v_{2i} is outer.

Choose an index j, $0 \leq j < i$, so path $Q' = (v_{2j}, v_{2j+1}, \cdots, v_{2i+1})$ is in H, and vertex v_{2k} is outer for $j < k \leq i$ but nonouter for $j = k$. This can be done, since if v_{2k} is outer in H, then the matched edge $v_{2k-2}v_{2k-1}$ is in H. (Note if $j = 0$, we have $v_0 = 0$ and $v_1 = u$.)

We derive a contradiction by calculating $FIRST(v_{2j+1})$ at the end of S_u. If x and y are adjacent outer vertices, then at the end of S_u, $FIRST(x) = FIRST(y)$. (This results from executing subroutine L on edge xy.) Applying this observation to vertices in path Q' shows $FIRST(v_{2j+1}) = v_{2k+1}$, the first nonouter vertex in Q'. (This vertex exists, since v_{2i+1} is nonouter.)

However, path $P(v_{2j+1}) = (v_{2j+1}, v_{2j}, \cdots)$, so $FIRST(v_{2j+1}) = v_{2j}$. Since $v_{2j} \neq v_{2k+1}$, we have a contradiction. □

Now we show E works correctly.

TABLE I. WORST-CASE TIME BOUNDS

Steps	Time	Executions per search	Total time
E0	$V + W$	—	V^2
E1	c	V	V
E2	c	$2W$	V^2
E3–R	V	1	V^2
E4–L0	c	W	V^2
L1–L6	V	$V/2$	V^2
E5	c	$V/2$	V^2
E6	c	W	V^2
E7	V	1	V^2

COROLLARY 3. *E halts with a maximum matching specified by MATE.*

PROOF. First note that E always halts. We have seen that subroutines L and R halt. So any search eventually stops, in step $E2$ or $E3$. E starts a finite number of searches in step $E1$. So Algorithm E halts.

Corollary 2 shows that *MATE* specifies a valid matching when E halts. To prove the last matching is maximum, it suffices to show no augmenting path exists, by Berge's Lemma. Let u be an unmatched vertex. Lemma 3 shows no augment made by E after S_u involves edges in H. So Lemma 3 can be applied to the last matching, to show there is no augmenting path to u. □

We conclude this section by describing a useful modification to Algorithm E, due to Edmonds [4]. The idea is to ignore a Hungarian subgraph H in searches after S_u. (By Lemma 3, searching in H is fruitless.) We change step $E2$ as follows:

E2′. [Choose an edge.] Choose an edge . . . If no such edge exists, go to *E1*.

Step $E2'$ causes step $E7$, which unlabels vertices, to be skipped after S_u. It is easy to check that in the modified algorithm, a vertex $y \in H$ is never labeled in a search after S_u.

This modification speeds up the algorithm if a maximum matching contains unmatched vertices. It does not change the worst-case time bound.

5. *Efficiency and Applications*

Algorithm E requires at most $O(V^3)$ time units when executed on a random access computer. This is seen from Table I, which gives simple bounds on the time for each step. For example, steps *E4–L0* can be executed in a constant amount of time (c); in a given search, they are executed at most W times (where W is the number of edges); since there are at most V searches, and $W \leq V(V-1)/2$, the total time for these two steps is $O(V^3)$. (Note that in step $E2$ we assume edges are chosen in a breadth-first or similar method, where a list of outer vertices is maintained. The list allows an unexamined edge to be chosen in time c.[1])

Families of graphs that require time $O(V^3)$ can be constructed. We describe such a family, assuming Algorithm E uses the breadth-first method in step $E2$. Similar families can be constructed for other methods [8].

Figure 8(a) shows a graph G_{6m}, with a maximum matching. This graph is formed from vertices $1, 2, \cdots, 6m$ by joining vertices $1, 2, \cdots, 4m$ in a complete graph, K_{4m}, and joining vertex $2i-1$ with vertex $4m+i$, for $1 \leq i \leq 2m$. Adjacency lists contain vertices in numerical order. Figure 8(b) shows the intermediate matching with $2m$ edges constructed by Algorithm E. Figure 8(c) illustrates a typical search to match vertex $4m+i$. (Here i is odd.) All vertices except $1, 3, 5, \cdots, 2i-1$ are made outer. An augmenting

[1] The run time of Algorithm E is at most $O(VW\alpha(W, V))$, if an efficient set merging algorithm is used to maintain *FIRST* in step *L5*. Here α is a very slowly growing function; $\alpha(W, V) \leq 3$ for all graphs that can be stored in an existing computer memory [12]. For sparse graphs ($W \ll V^2$), this variant of E is preferable.

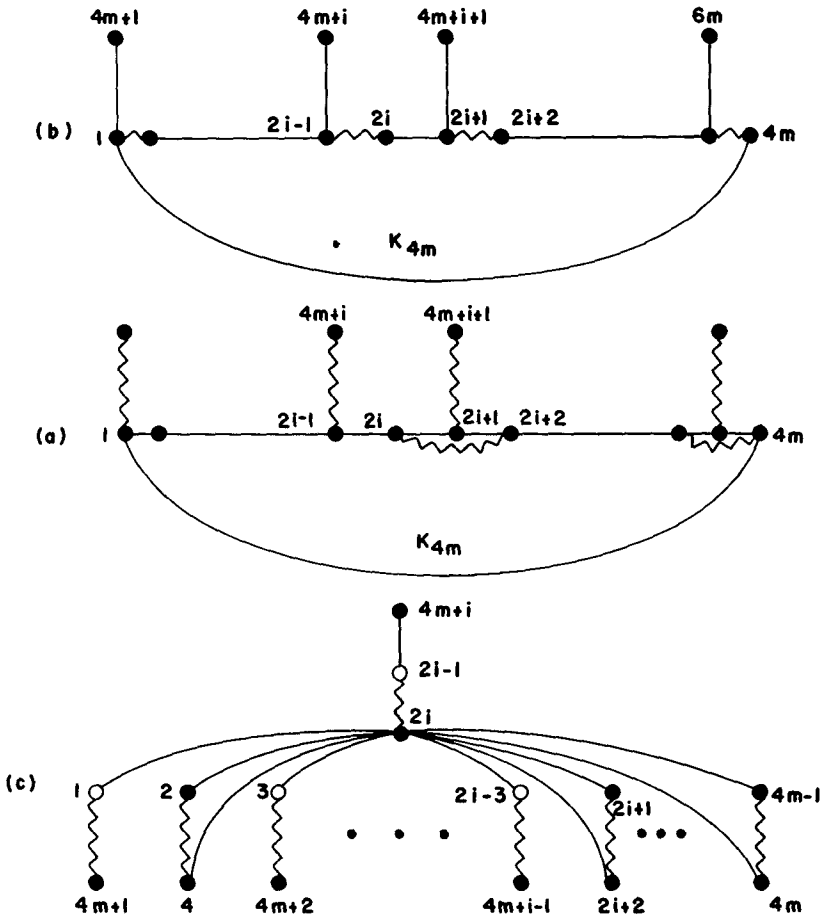


FIG. 8. Worst-case graph G_{6m}

path to vertex $4m + i + 1$ is found when outer vertex $2i + 1$ is chosen in step $E2$. Over $4mi$ edges are chosen in this search, and over $4m^3$ edges are scanned in the last m searches. Thus the time is $O(V^3)$.

Several experiments were conducted with an implementation of E in Algol W on the IBM 360/165. For the worst-case graphs described above, run times proportional to $V^{2.8}$ were observed, over the interval $11 \leq m \leq 24$ ($66 \leq V \leq 144$, $968 \leq W \leq 4608$), with times from .18 to 1.6 sec. For a similar experiment on Edmonds' algorithm, times proportional to $V^{3.5}$ were observed (versus V^4 predicted) with time 1.7 sec for $m = 11$. Experiments on E on "random" graphs gave times one order of magnitude faster than worst-case graphs with 3200 edges [8].

The space used by the Algol W implementation of E is $5V + 4W$ words. This includes $V + 4W$ words for the graph; $2V$ words for *MATE* and *LABEL*; V words for *FIRST*, also used by the stack of recursive calls to R; and V words for a list of outer vertices for step $E2$.

Algorithm E can be used to speed up the scheduler devised by Fujii et. al. [6]. They solved the following problem: Compute an optimal schedule for N tasks to be executed by two processors, assuming the tasks have equal length and arbitrary precedence constraints. Their approach is to construct a *compatibility graph*, showing which tasks can be executed simultaneously; then find a maximum matching on the compatibility graph; finally, sequence the matched task pairs and the unmatched tasks according to precedence con-

straints. This algorithm was thought to require time $O(N^4)$[6, Erratum]. But the first and last steps can be executed in time $O(N^3)$, and Algorithm E finds the matching in time $O(N^3)$. So the scheduler is an $O(N^3)$ algorithm. (Recent work by Coffman and Graham [3] solves this scheduling problem in time $O(N^2)$. Their elegant method does not employ matchings directly.)

Algorithm E can be generalized to find maximum matchings on weighted graphs. In a *weighted graph*, each edge has a weight that is a real number. The problem is to find a matching with maximum weight. Matching on ordinary graphs is the special case of this problem where all edges have equal weight. Edmonds [5] first developed an efficient $(O(V^4))$ algorithm for this problem. The generalization of Algorithm E takes time $O(V^3)$[8].

ACKNOWLEDGMENTS. The author wishes to thank Professor Harold Stone of Stanford University for assisting in the preparation of this manuscript, and Professor Eugene Lawler of the University of California at Berkeley for communicating his stimulating ideas on matching.

REFERENCES

(Note. Reference [7] is not cited in the text.)

1. BALINSKI, M.L. Labelling to obtain a maximum matching In *Combinatorial Mathematics and Its Applications*, R.C. Bose and T.A Dowling, Eds , U. of North Carolina Press, Chapel Hill, N.C., 1967, pp. 585-602
2. BERGE, C. Two theorems in graph theory. *Proc. Nat. Acad. Sci. 43* (1957), 842-844.
3. COFFMAN, E J. JR., AND GRAHAM, R L Optimal scheduling for two-processor systems. *Acta Informatica 1* (1972), 200-213.
4. EDMONDS, J. Paths, trees and flowers. *Canad J. Math. 17* (1965), 449-467
5. EDMONDS, J Maximum matching and a polyhedron with 0,1-vertices. *J. Res. Nat. Bur Standards 69B* (1965), 125-130
6. FUJII, M., KASAMI, T., AND NINOMIYA, K. Optimal sequencing of two equivalent processors. *SIAM J. Applied Math. 17* (1969), 784-789; Erratum, *20* (1971), 141.
7. GABOW, H. An efficient implementation of Edmonds' maximum matching algorithm Tech. Rep. 328, Comput. Sci. Dep , Stanford U., Stanford, Calif , 1972
8. GABOW, H. Implementations of algorithms for maximum matching on nonbipartite graphs. Ph.D. diss., Comput. Sci. Dep., Stanford U., Stanford, Calif., 1973.
9. HARARY, F. *Graph Theory*. Addison-Wesley, Reading, Mass., 1969.
10. KNUTH, D *The Art of Computer Programming, Vol. 1*. Addison-Wesley, Reading Mass , 1968.
11. TARJAN, R.E. An efficient planarity algorithm. Tech. Rep. 244, Comput. Sci. Dep., Stanford U., Stanford, Calif., 1971.
12. TARJAN, R.E. Efficiency of a good but not linear set union algorithm. *J. ACM 22*, 2 (April 1975), 215-225.
13. WITZGALL, D., AND ZAHN, C T. JR. Modification of Edmonds' algorithm for maximum matching of graphs. *J. Res. Nat. Bur. Standards 69B* (1965), 91-98.

RECEIVED APRIL 1973; REVISED JULY 1975