ACT-14 DECEMBER, 1978

COORDINATED SCIENCE LABORATORY

APPLIED COMPUTATION THEORY GROUP

EFFICIENT ALGORITHMS FOR FINDING MAXIMUM MATCHINGS IN CONVEX BIPARTITE GRAPHS AND RELATED PROBLEMS

W. LIPSKI, JR.
F. P. PREPARATA

REPORT R-834 UILU-ENG 78-2227

UNIVERSITY OF ILLINOIS – URBANA, ILLINOIS

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* EFFICIENT ALGORITHMS FOR FINDING MAXIMUM MATCHINGS IN CONVEX BIPARTITE GRAPHS AND RELATED PROBLEMS		5. TYPE OF REPORT & PERIOD COVERED Technical Reports
		6. PERFORMING ORG. REPORT NUMBER R-834; UILU-ENG 78-2227
7. AUTHOR(s) W. Lipski, Jr. and F. P. Preparata		8. CONTRACT OR GRANT NUMBER(s) NSF MCS 76-17321; DAAB-07-72-C-0259; DAAG-29-78-C-0016
9. PERFORMING ORGANIZATION NAME AND ADDRESS Coordinated Science Laboratory University of Illinois at Urbana-Champaign Urbana, Illinois 61801		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Joint Services Electronics Program		12. REPORT DATE December, 1978
		13. NUMBER OF PAGES 30
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT *(of this Report)* Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)* Maximum Matching; Convex Bipartite Graph; Maximum Independent Set; Greedy Algorithms; Gale-Optimal Matching; Scheduling Algorithms		

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A bipartite graph G = (A,B,E) is convex on the vertex set A if A can be ordered so that the elements of A connected to any element b in vertex set B form an interval of A; G is doubly convex if it is convex on both A and B. For these types of graphs Glover discovered a simple rule for finding maximum matchings. Letting $|A| = m$ and $|B| = n$, in this paper we describe an implementation of Glover's rule which runs in time $O(m+n\log\log n)$ on a convex graph, and in time $O(m+n)$ on a doubly convex graph. We also show that, given a maximum matching in a convex bipartite graph G, a corresponding maximum set of independent

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

20. ABSTRACT (continued)

vertices can be found in time $0(m+n)$. Finally, we briefly discuss some generalizations of convex bipartite graphs and some extensions of the previously discussed techniques to instances in scheduling theory.

UILU-ENG 78-2227

EFFICIENT ALGORITHMS FOR FINDING MAXIMUM MATCHINGS IN CONVEX BIPARTITE GRAPHS AND RELATED PROBLEMS

by

W. Lipski, Jr. and F. P. Preparata

This work was supported in part by the National Science Foundation under Grant MCS76-17321 and in part by the Joint Services Electronics Program (U.S. Army, U.S. Navy and U.S. Air Force) under Contracts DAAB-07-72-C-0259 and DAAG-20-78-C-0016.

EFFICIENT ALGORITHMS FOR FINDING MAXIMUM MATCHINGS IN CONVEX BIPARTITE GRAPHS AND RELATED PROBLEMS

W. Lipski, Jr.* and F. P. Preparata†
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign

ABSTRACT. A bipartite graph G = (A,B,E) is convex on the vertex set A if A can be ordered so that the elements of A connected to any element b in vertex set B form an interval of A; G is doubly convex if it is convex on both A and B. For these types of graphs Glover discovered a simple rule for finding maximum matchings. Letting $|A| = m$ and $|B| = n$, in this paper we describe an implementation of Glover's rule which runs in time O(m+nloglogn) on a convex graph, and in time O(m+n) on a doubly convex graph. We also show that, given a maximum matching in a convex bipartite graph G, a corresponding maximum set of independent vertices can be found in time O(m+n). Finally, we briefly discuss some generalizations of convex bipartite graphs and some extensions of the previously discussed techniques to instances in scheduling theory.

KEYWORDS AND PHRASES: maximum matching, convex bipartite graph, maximum independent set, greedy algorithms, Gale-optimal matching, scheduling algorithms.

CR CATEGORIES: 5.25, 5.32

*On leave from the Institute of Computer Science, Polish Academy of Sciences, P. O. Box 22, 00-901 Warsaw, PKiN, Poland

†Also with the Departments of Electrical Engineering and of Computer Science

This work was supported in part by the National Science Foundation under Grant MCS76-17321 and in part by the Joint Services Electronics Program under Contracts DAAB-07-72-C-0259 and DAAG-29-78-C-0016.

EFFICIENT ALGORITHMS FOR FINDING MAXIMUM MATCHINGS IN CONVEX BIPARTITE GRAPHS AND RELATED PROBLEMS

W. Lipski, Jr. and F. P. Preparata

1. Introduction

Matching problems constitute a traditionally important topic in combinatorics and operations research [8] and have been the object of extensive investigation. Particularly interesting is the problem of finding a maximum matching in a bipartite graph, which is stated as follows: Let $G = (A,B,E)$ be an undirected bipartite graph, where A and B are sets of vertices, and E is a set of edges of the form (a,b) with $a \in A$ and $b \in B$. A subset $M \subseteq E$ is a *matching* if no two edges in M are incident to the same vertex; M is of *maximum cardinality* (or simply, *maximum*) if it contains the maximum number of edges. As noted by Hopcroft and Karp [7], this problem has many applications, such as the chain decomposition of a partially ordered set, the determination of coset representatives in groups, etc. Hopcroft and Karp have also developed the best known algorithm for this problem.

A special instance of the problem, with some industrial applications, was originally discussed by Glover [6] and referred to as matching in a convex bipartite graph. A bipartite graph G is *convex* on A if an ordering "$\leq$" of the elements of A can be found so that for any $b \in B$ and distinct a_1 and a_2 in A (with $a_1 \leq a_2$)

$$(a_1,b) \in E \text{ and } (a_2,b) \in E \Rightarrow (a,b) \in E \text{ for any } a \in A \text{ such that } a_1 \leq a \leq a_2$$

In other words, G is convex on A when there is an ordering on A such that for any $b \in B$ the set of vertices of A connected to b forms an interval in this ordering. In such a bipartate graph we let BEG[b] and END[b] denote the "smallest" and "largest" elements in the interval of the elements of A connected to b. Naturally, if $b \in B$ is isolated, the set $A(b)$ is empty and BEG[b] = END[b] = Λ, the empty symbol. In what follows we assume that there is no isolated vertex in B.

When this property holds, the maximum matching problem is considerably easier to solve. In fact Glover proved that the following simple procedure yields a maximum cardinality matching (we assume that both A and B be given as sequences of integers from 1 to $|A|$ and $|B|$ respectively; MATCH[i] denotes the element of B matched to $i \in A$):

Algorithm 0

```
begin       for i: = 1 to |A| do
               begin U: = {k:(i,k) ∈ E and k has not been deleted from B}
                     if U ≠ ∅  then (* find j ∈ U to be matched to i *)
                       begin j: = element  in U with minimum value of END
                             MATCH[i]: = j
                             Delete j from B
                       end
                     else MATCH[i]: = Λ    (* i unmatched *)
               end
end
```

In words, element i of A is matched to an available element j of B whose corresponding interval ends the closest to i. The most time consuming task of this algorithm is the formation of the set U and the associated determination of an element $j \in U$ with the smallest value of END[j]: for any given $i \in A$, it involves scanning all the elements of B connected to i. Thus the running time of this task is clearly $O(|E|)$, as pointed out by Lawler [8].

In this paper we shall describe a considerably more efficient implementation of Glover's rule and investigate both specializations and generalizations of the original matching problem. Specifically, after considering (Section 2) the maximum matching problem in a convex bipartite graph, we shall analyze the further simplifications which are possible when the graph is doubly convex (Section 3), and the optimal time determination of the maximum set of independent vertices associated with a given maximum matching (Section 4). Finally (Section 5), we succinctly describe two generalizations of the convex matching problem and an extension of the techniques to weighted matching, which directly applies to the solution of a scheduling problem.

2. Maximum matching in convex bipartite graphs: an efficient implementation of Glover's rule.

Let $G = (A,B,E)$ be a bipartite graph convex on A, with $|A| = m$ and $|B| = n$. As before, $A = \{1,2,\dots,m\}$ and $B = \{1,2,\dots,n\}$. For $b \in B$, $A(b) \subseteq A$ denotes the set $\{a:(a,b) \in E\}$; similarly, for $a \in A$, $B(a) \subseteq B$ denotes the set $\{b:(a,b) \in E\}$. Again, we assume that A is ordered so that, for each $b \in B$, $A(b)$ is the interval $[BEG[b], END[b]]$. Notice that if the set A is not initially ordered so that the property of convexity is manifest, the bipartite graph G can be tested for possession of this property - and, if so, rearranged - in time $O(|E| + m + n)$ by means of the Booth-Lueker algorithm [2].

We begin by giving a generalization (and simpler proof) of Glover's rule.

Lemma 1. If $(a,b) \in E$ and $A(b) \subseteq A(c)$, for any $c \in B(a)$, then there is a maximum matching containing (a,b).

Proof. Suppose M is a maximum matching not containing (a,b). If a is unmatched then we may replace the edge of the matching incident to b with (a,b), similarly if b is unmatched. Suppose therefore that $(a,c), (d,b) \in M$ for some $c \in B$, $d \in A$. Since $d \in A(b) \subseteq A(c)$, it follows that $(d,c) \in E$, and we may replace $(a,c),(d,b)$ by $(a,b),(d,c)$ (see Figure 1). $\square$

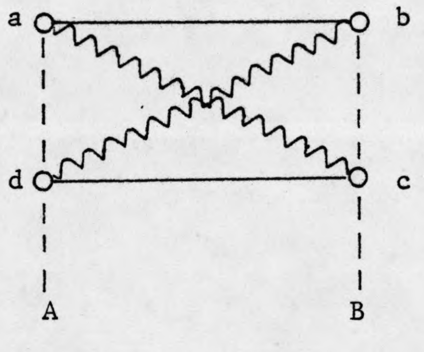

Figure 1. To the proof of Lemma 1. Wiggly edges belong to the matching.

In order to prove that Algorithm 0 correctly finds a maximum matching, let us denote by G_i the graph obtained from G by deleting $1,\ldots,i-1$ from A and MATCH[1],...,MATCH[i-1] from B, together with the edges incident to all these vertices. Let M_i be the set of edges matched by Algorithm 0 to vertices $1,\ldots,i$ in A (we put $M_0 = \emptyset$), and let $A_i(b)$ and $B_i(a)$ be defined for G_i in the same way as A(b) and B(a) were defined for G. We say that M_i can be extended to a maximum matching of G if there is a maximum matching M of G containing M_i; this means that M is the union of M_i and of a maximum matching of G_{i+1}.

Assume inductively that $a \leq m$ and that M_{a-1} can be extended to a maximum matching of G. (This is trivially true for a=1, since M_0 is empty and G_0 coincides with G.) We shall prove that M_a can also be extended to a maximum matching of G. This is obviously true if $B_a(a) = \emptyset$, so assume that $B_a(a) \neq \emptyset$, whence Algorithm 0 chooses MATCH[a] = $b \neq \Lambda$. It is then sufficient to show that there is a maximum matching of G_a containing (a,b). But this is immediate, since for any c in $B_a(a)$ we have $A_a(c) = [a,\text{END}[c]]$; by line 4 of Algorithm 0, we have END[b] $\leq$ END[c] for any $c \neq b$ in $B_a(a)$, whence $A_a(b) \subseteq A_a(c)$, and, by Lemma 1, the claim is established.

As noted earlier, efficiency can be achieved if for a given $a \in A$ the computation of $j \in B_a(a)$ for which END[j] is minimum can be sped-up. We shall now show that, by some additional preprocessing and the use of appropriate data structures, this can be done in time which is sublogarithmic in the size of B.

The basic idea is to try to store the set $B_i(i)$ of unmatched vertices of B connected to a currently inspected vertex $i \in A$ on a priority queue, so that the element $j \in B$ to be matched to i can be found as the least element of the queue. This is indeed possible if the elements of B are relabelled so that END[1] $\leq \dots \leq$ END[n]. Then the least element of the priority queue minimizes the value of END, as required by Glover's rule. In order to complete the description of our implementation, we should specify a method of updating the priority queue, so that its content is changed from $B_i(i)$ to $B_{i+1}(i+1)$ as i is increased by one. It is easy to see that we should delete the least element from the queue (the vertex to be matched to i), then delete all vertices $k \in B$ with END[k] = i and finally insert all vertices $k \in B$ with BEG[k] = i+1. Deleting vertices is easy, since the set of vertices $k \in B$ with END[k] = i appears as an interval in our ordering of B. Inserting vertices can be made easy too, if we precompute an array ORDBEG[1:n] containing the vertices of B sorted according to the parameter BEG, so that BEG[ORDBEG[1]] $\leq \dots \leq$ BEG[ORDBEG[n]]; then the set of vertices $k \in B$ with BEG[k] = i is stored in an interval of consecutive positions of ORDBEG. Notice that both relabelling of vertices in B so that END[1] $\leq \dots \leq$ END[n] and computing the array ORDBEG can be done in time O(m+n) by standard bucket sorting (see e.g. [1]), since in both cases there are n items to be sorted by a key which may assume values from integers 1,...,m.

Next we may take advantage of the fact that the elements in the priority queue are integers in the range [1,n] and employ the priority queue structure developed by van Emde Boas [3,4], which allows each of the standard queue operations to be performed in time O(loglogn) and uses space O(n).

We can now formally describe the matching algorithm, where: QUEUE denotes the just mentioned priority queue _à la_ van Emde Boas (with associated operations MIN, DELETE, INSERT, EXTRACTMIN); MATCH[1:m] ORDBEG[1:n], BEG[1:n], and END[1:n] are arrays of integers, the integer variables nb and ne are counters referring to the arrays ORDBEG and END, respectively (nb-1 and ne-1 count respectively the number of beginnings and ends of intervals [BEG[k],END[k]] found so far.

Algorithm 1 (Finding maximum matching in convex bipartite graph)

Input: BEG[1:n], END[1:n], ORDBEG[1:n]

$\text{END}[1] \leq \ldots \leq \text{END}[n]$, $\text{BEG}[\text{ORDBEG}[1]] \leq \ldots \leq \text{BEG}[\text{ORDBEG}[n]]$

Output: MATCH[1:m]

(Algorithm on next page)

```
 1  begin  QUEUE: = ∅ ,      nb: = ne: = 1
 2         for  i: = 1 to m do
 3            begin (*find vertex to be matched to i*)
 4               while (nb ≤ n) and (BEG[ORDBEG[nb]] = i) do
 5                  begin   INSERT(ORDBEG[nb])
 6                          nb: = nb + 1
 7                  end
 8               if QUEUE = ∅  then MATCH[i]: = Λ (*i unmatched*)
 9               else begin  MATCH[i]: = MIN
10                           EXTRACTMIN
11                    end
12               while (ne ≤ n) and (END[ne] = i) do
13                    begin DELETE(ne)
14                          ne: = ne+1
15                    end
16            end
17  end
```

From the viewpoint of performance, notice that each term of MATCH[1:m] is processed exactly once (lines 8 or 9), for a total work $O(m)$, while each term of B is inserted into the queue once (line 5) and extracted once (lines 10 or 13). So we conclude that the running time of Algorithm 1 is $O(m + n\log\log n)$.

3. Maximum matching in doubly convex bipartite graphs

As noted by Glover, the maximum matching problem becomes even simpler when the bipartite graph G is doubly convex, i.e., orderings of both A and B exist such that every A(b) is an interval of A and every B(a) is an interval of B.

As before, we assume that G be given as a bipartite graph convex on A, that is, as a set $\{< \mathrm{BEG}[b], \mathrm{END}[b] > : b \in B\}$ representing intervals of A. A preliminary task is to test whether the set B can be reordered so that for each $a \in A$ the set B(a) be an interval of B.

Pictorially, we may display G by means of a set of segments (Figure 2a): specifically, in the plane (x,y), we let the segment $y = b$, $\mathrm{BEG}[b] \leq x \leq \mathrm{END}[b]$ represent the interval A(b) (in the sequel this will be briefly referred to as segment b). If we next join the extremes of adjacent segments, i.e., introduce in this diagram edges $(\mathrm{BEG}[i], \mathrm{BEG}[i+1])$ and $(\mathrm{END}[i], \mathrm{END}[i+1])$, for $i = 1,2,\ldots,n-1$, the set of segments is enveloped by two polygonal lines called the left and right boundaries, which together with the first and last segments of the given set form a simple polygon. In this representation, G is convex on B if the intercept of a vertical line with this polygon consists of a single segment: thus G is convex on B if and only if the segments can be rearranged so that both boundaries are bitomic, as shown in Figure 2d (that is, in the resulting relabelling of elements of B, for some $1 \leq r_1 \leq n$, $\mathrm{BEG}[1] \geq \ldots \geq \mathrm{BEG}[r_1]$ and $\mathrm{BEG}[r_1] \leq \ldots \leq \mathrm{BEG}[n]$; similarly for some $1 \leq r_2 \leq n$, $\mathrm{END}[1] \leq \ldots \leq \mathrm{END}[r_2]$ and $\mathrm{END}[r_2] \geq \ldots \geq \mathrm{END}[n]$). We shall now describe a linear time - hence optimal - algorithm which tests G for double convexity and, if this property holds, produces the desired ordering of B.

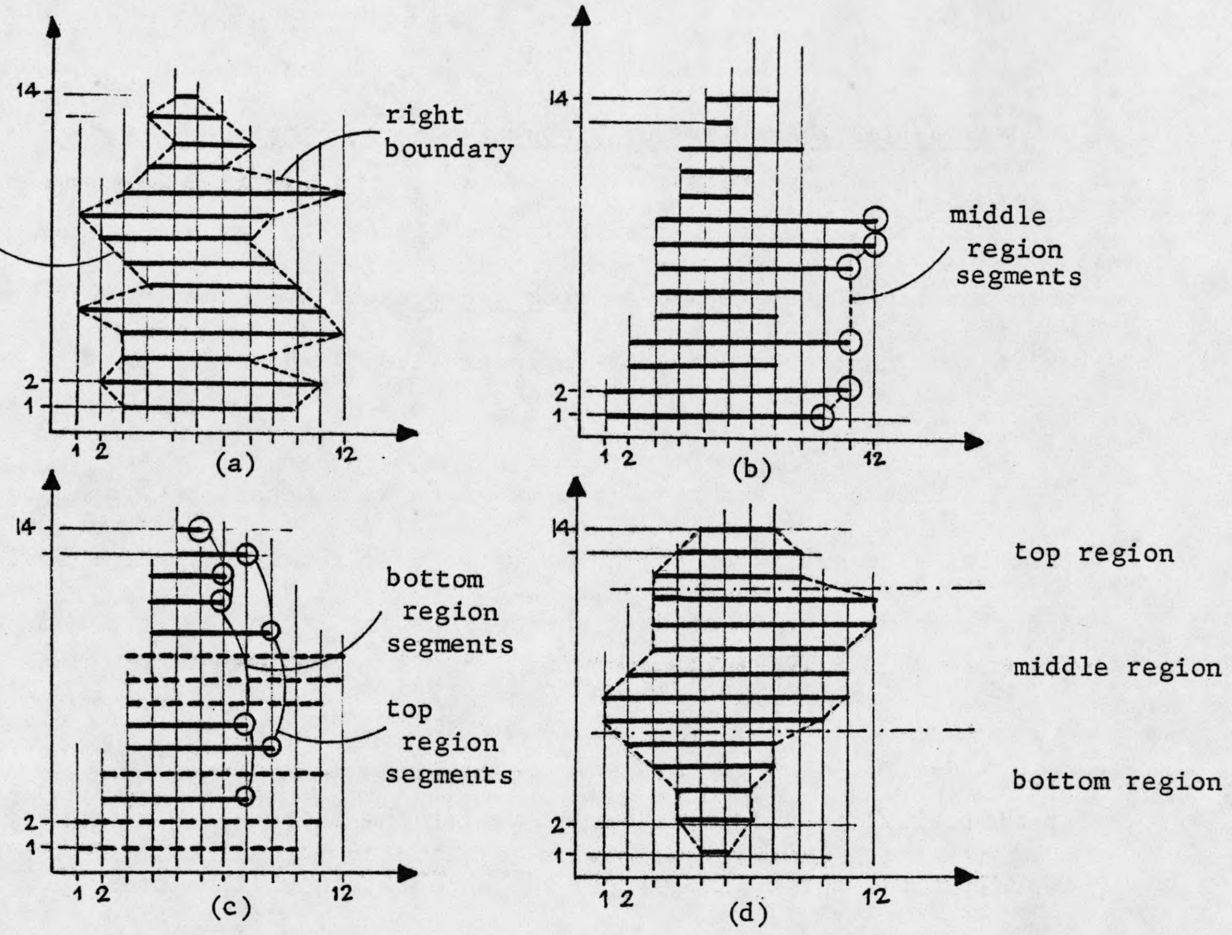

Figure 2. Different polygons corresponding to the same set of segments. (a) arbitrary order; (b),(c) ordered by nonincreasing BEG; (d) ordered to exhibit double convexity.

In the rest of this section we shall always assume that the convex bipartite graph G under consideration is connected. In fact, it is very easy to find connected components of a convex bipartite graph. It is sufficient to scan vertices $i \in A$ in increasing order and to count the number of beginnings and the number of endings of intervals found up to vertex i. Each time these two counts coincide, a new connected component is found. With the elements of B labelled so that END[1] $\leq \ldots \leq$ END[n], and with the array ORDBEG as in Algorithm 1, the determination of connected components can be done in $O(m+n)$ time.

Referring to Figure 2d, it is easy to see that the polygon displaying the double convexity of an arbitrary G consists - up to the reversal of the ordering of B - of three regions (not all simultaneously empty): a <u>middle</u> region, where both left and right boundaries are nondecreasing (i.e., both BEG[j] and END[j] are nondecreasing with increasing j, assuming that the labelling of elements of B coincides with the bottom to top ordering of segments in the given geometric representation); a <u>top</u> region where the left and right boundaries are nondecreasing and nonincreasing, respectively; a <u>bottom</u> region where the left and right boundaries are nonincreasing and nondecreasing, respectively. Moreover, all segments of the top region are nested, starting with the topmost segment of the middle region, similarly, all segments of the bottom region are nested, starting with the bottommost segment of the middle region.

It is easy to see that our description need not define the three regions uniquely, if there are different elements in B with the same value of BEG or END; to guarantee the uniqueness we require that all segments in the bottom region have $\text{BEG}[j] > \min_{1 \leq k \leq n} \text{BEG}[k]$, and all segments in the top region have $\text{END}[j] < \max_{1 \leq k \leq n} \text{END}[k]$.

Suppose that we initially index the elements of B so that the pairs $\langle \text{BEG}[j], \text{END}[j] \rangle$, $j = 1, \ldots, n$ are in lexiographic ascending order; this can be done by bucket sorting these elements on the parameter BEG, and then (stably) bucket sorting the resulting sequence on the parameter END, all in time $O(m+n)$. Once this ordering of segments $\{A(b) : b \in B\}$ is available (see Figure 2b), we shall first extract from it the subsequence of segments to be assigned to the middle region. To complete the test, we must verify whether the remaining segments can be successfully assigned to either top or the bottom regions. Since for segments in these regions, the orderings BEG and END are contragradient, we must preliminarily alter the order of the segments not assigned to the middle region, so that for any two such consecutive segments j and j+1, $(BEG[j] = BEG[j+1]) \Rightarrow (END[j] \geq END[j+1])$: this can be obviously done in linear time by a straightforward use of a stack (Figure 2c). Next, we must test whether the resulting sequence can be partitioned into two subsequences, for each of which the parameter END is nonincreasing: if this is feasible, then the two subsequences of segments will respectively form the top and bottom regions. More exactly, we should do the partitioning in such a way, that the resulting subsequences of segments be nested as previously explained. We guarantee this by assigning the extremal segments of the middle region to the sequence to be partitioned.

The whole task is performed by the following algorithm, which computes for each segment j a parameter Y[j] denoting its order in the final arrangement. This algorithm also makes use of a special subroutine, which - if at all possible - partitions in linear time a sequence of integers into two nonincreasing subsequences; for example, (4,6,3,5,4) is partitioned into (4,3) and (6,5,4). This simple subroutine is described formally in an appendix. Its additional feature, which is important for the correctness of our algorithm, is that the first term of the sequence is assigned to the first subsequence.

Algorithm 2 (Testing for double convexity of a connected convex bipartite graph)

Input: BEG[1:n],END[1:n]

The pairs <BEG[j],END[j]>, j = 1,...,n are in lexicographic increasing ordering

Output: Y[1:n]

Vertices $j \in B$ relabelled so that for $1 \leq j < n$

BEG[j] < BEG[j+1], or BEG[j] = BEG[j+1] and END[j] ≥ END[j+1]

```
 1  begin (* find last segment jm of middle region *)
 2       jm: = 1
 3       for j: = 2 to n do
 4          if END[j] ≥ END[jm] then jm: = j
        (* extract segments not in internal part of middle region *)
 5       e: = END[1], ℓ: = 0
 6       for j: = 1 to n do
 7          if (END[j] ≥ e) and (j≠1) and (j≠jm) then e: = END[j]
 8          else begin ℓ: = ℓ + 1
 9                     S[ℓ]: = j
10               end
11          relabel the elements of B so that for 1 ≤ j < n
              (BEG[j] = BEG[j+1]) ⇒ (END[j] ≥ END[j+1])
12          reorder S[1:ℓ] so that for 1 ≤ p < ℓ
              (BEG[S[p]] = BEG[S[p+1]]) ⇒ (END[S[p]] ≥ END[S[p+1]])
13          partition S[1:ℓ] into two subsequences SUB1[1:ℓ1] and
              SUB2[1:ℓ2], such that END[SUB1[1]] ≥ ... ≥ END[SUB1[ℓ1]]
              and END[SUB2[1]] ≥ ... ≥ END[SUB2[ℓ2]]
14          k1: = k2: = k3: = 1
15          for j: = 1 to n do (* determine Y[j] *)
16             if SUB1[k1] = j then (* j belongs to bottom region *)
17               begin Y[j]: = ℓ1 - k1 + 1
18                     k1: = k1 + 1
19               end
20             else if SUB2[k2] = j then (* j belongs to top region *)
21                    begin Y[j]: = n - ℓ2+k2
22                          k2: = k2+1
23                    end
24                  else (* j belongs to middle region *)
25                    begin Y[j]: = ℓ2+k3
26                          k3: = k3+1
27                    end
28  end
```

It is straightforward to conclude that Algorithm 2 runs in time $O(n)$.

We can now describe the maximum matching algorithm, which makes use of a DEQUE (doubly-ended-queue) as an auxiliary data structure; as is well-known, DEQUE has two distinguished elements, top and bottom, and the following repertoire of instructions: INSERTTOP, DELETETOP, INSERTBOTTOM, and DELETEBOTTOM.

Algorithm 3 (Finding maximum matching in doubly convex bipartite graph)

Input: BEG[1:n], END[1:n], Y[1:n]

BEG[j] < BEG[j+1], or BEG[j] = BEG[j+1] and END[j] $\geq$ END[j+1]

for $1 \leq j < n$

Output: MATCH[1:m]

```
begin  DEQUE: = $\emptyset$,  j: = 1
       for i: = 1 to m do
           begin (* find element in B to be matched to $i \in A$ *)
              while (BEG[j] = i) and (j $\leq$ n) do
                    begin (* insert j into deque *)
                          if (DEQUE = $\emptyset$) or (Y[j] > Y[top]) then INSERTTOP(j)
                          else INSERTBOTTOM(j)
                          j: = j+1
                    end
              if (DEQUE = $\emptyset$) then MATCH[i]: = $\Lambda$ (* i unmatched *)
              else if END[top] < END[bottom] then
                       begin MATCH[i]: = top
                             DELETETOP
                       end
                    else begin MATCH[i]: = bottom
                               DELETEBOTTOM
                         end
              while (DEQUE $\neq \emptyset$) and (END[top] = i) do DELETETOP
              while (DEQUE $\neq \emptyset$) and (END[bottom] = i) do DELETEBOTTOM
           end
end
```

Notice that each element of B is inserted into and deleted from the DEQUE exactly once, and that each of the standard deque operations can be executed in constant time; it follows that the entire matching can be computed in time $O(m+n)$.

4. Finding a maximum independent set of vertices in a convex bipartite graph

Closely related to the maximum matching problem in bipartite graphs is the determination of a maximum independent set (of vertices), that is, of a maximum cardinality set of vertices of the bipartite graph G such that no two of them are connected. It is well-known (see, e.g. [10]) that a maximum independent set can be derived from a maximum matching M by standard alternating path techniques as follows (see Figure 4): (i) direct every edge $e \in M$ from A to B, and any $e \in E-M$ from B to A; (ii) letting B_0 denote the set of unmatched vertices in B, find the sets $A_1 \subseteq A$ and B_1 $(B_0 \subseteq B_1 \subseteq B)$ of vertices reachable from B_0; (iii) construct the maximum independent set as $I \triangleq B_1 \cup (A-A_1)$. Therefore the entire problem reduces

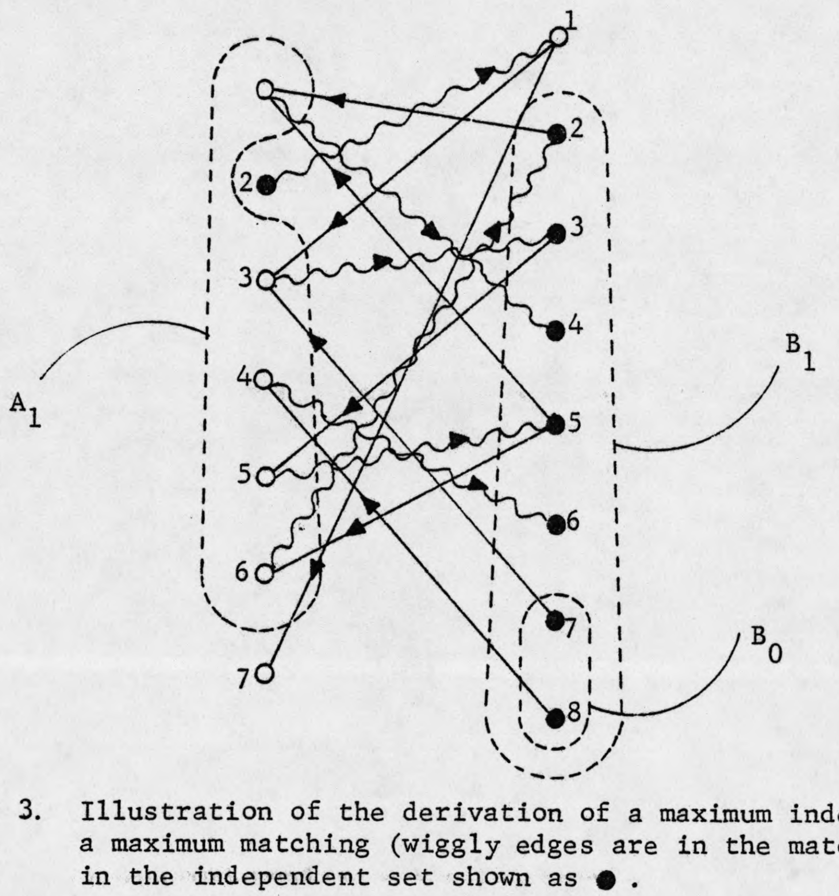

Figure 3. Illustration of the derivation of a maximum independent set from a maximum matching (wiggly edges are in the matching M): vertices in the independent set shown as ● .

to finding all the vertices of G which are reachable from B_0. A most interesting fact we shall now show is that, when G is convex, this reachable set can be obtained in time $O(n+m)$ so that the determination of a maximum independent set runs in total time $O(m+n\log\log n)$, or $O(m+n)$ if G is doubly convex, the computation of the maximum matching being the dominant task (notice that, once A_1 and B_1 are known, I is obtainable in time $O(n+m)$).

As usual, the graph G is described by the two arrays BEG[1:n] and END[1:n]; MATCH[1:n] gives for each $i \in A$ either Λ or the element of B matched to it. We assume that the elements of B be ordered so that $BEG[i] \leq BEG[i+1]$, $1 \leq i < n$. Due to the property of convexity, for each $b \in B_0$ the set $A(b)$ of vertices reachable by a single edge from it form an interval of A; from any matched vertex a in this interval we reach a single vertex $MATCH[a] \in B$, which in turn reaches another interval $A(MATCH[a])$ of A. Notice thar $A(b)$ and $A(MATCH[a])$ necessarily overlap, so by the convexity of G their union is a single interval. Therefore, initially we place in a queue all the elements of B_0 in increasing order, and starting with the smallest one j_1, we determine a single <u>extended interval</u> $A^*(j_1) \supseteq A(j_1)$ of A, which is the set of all elements of A which are reachable from j_1 ($A^*(j_1)$ could be informally viewed as the "closure" of $A(j_1)$). This extended interval is constructed by scanning $A(j_1)$ in <u>decreasing</u> order starting from $END[j_1]$ and currently updating the extremes of the reached interval; once the scanning reaches the lower extreme without further downward extension of the interval, then if the interval has been extended upward beyond $END[j_1]$, scanning is resumed in ascending order starting from

END[j_1] until the same terminating condition occurs, and this process is repeated until no further extension - either downward or upward - is possible. At this point the construction of interval A*(j_1) has been completed. We then extract the next element j_2 from the queue and begin the construction of A*(j_2). Notice that if A*(j_1) and A(j_2) are disjoint (Figure 4a), BEG[j_2] must be larger than the upper extreme of A*(j_1). Since by hypothesis, BEG[j_1] $\leq$ BEG[j_2], it follows that only downward extensions

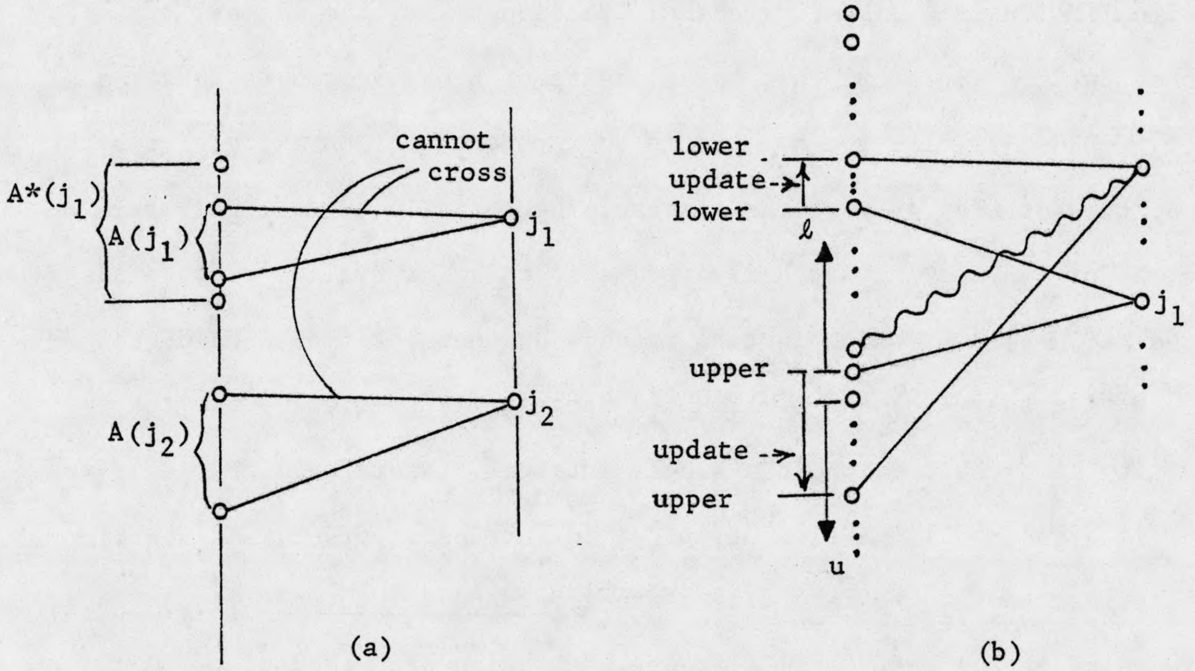

Figure 4. (a) Illustration of the case where A*(j_1) and A(j_2) are disjoint.
(b) Explanation of the meaning of variables "lower", "upper", ℓ and u.

of $A(j_2)$ may meet previously scanned elements of A. To avoid any time-consuming unnecessary repeated scanning, we must ensure than any previously scanned interval be skipped in subsequent processing, so that each element of A be scanned at most once. This objective is achieved by means of a stack: as soon as the construction of $A^*(j)$, for some $j \in B_0$, is completed, its lower and upper extremes are inserted into the stack, whose content - at a generic instant - is a sequence $-1, i_1, e_1, i_2, e_2, \ldots, i_k, e_k$, such that, for $1 \leq p < k$, $e_p + 1 < i_{p+1}$, $[i_p, e_p]$ is an interval of A, and $\bigcup_{p=1}^{k} [i_p, e_p]$ is the set of all scanned elements of A. The reachability algorithm uses as auxiliary data structures a QUEUE, containing the elements of B_0 ordered according to nondecreasing value of BEG, and a STACK, for storing the sequence of scanned intervals, as already noted. The intuitive significance of the program variables lower, upper, ℓ, and u is as follows (see Figure 4b): lower and upper denote respectively the current boundaries of the extended interval being constructed; ℓ and u are pointers used in scanning, running downward and upward respectively.

<u>Algorithm 4</u> (Finding the set of vertices in A reachable by alternating paths from the set of unmatched vertices in B in a convex bipartite graph)

<u>Input</u>: BEG[1:n], END[1:n], MATCH[1:m]

QUEUE containing the unmatched vertices $b \in B$ in increasing order

$BEG[1] \leq \ldots \leq BEG[n]$

<u>Output</u>: The set $\bigcup_{p=1}^{k} [i_p, e_p] \subseteq A$ of vertices reachable from unmatched vertices $b \in B$, represented by a sequence $-1, i_1, e_1, i_2, e_2, \ldots, i_k, e_k$ stored on STACK

```
 1 begin
 2   STACK ⇐ -1
 3   while QUEUE ≠ ∅ do (*  find vertices reachable from first(QUEUE)*)
 4     begin j ⇐ QUEUE
 5       if END[j] > top(STACK) then (* new vertices to be scanned *)
 6         begin  ℓ: = END[j]+1, lower: = BEG[j], u: = upper: = END[j]
 7            repeat (* extend interval of vertices reached from j *)
 8              while ℓ > lower do (* scan downward *)
 9                begin  ℓ: = ℓ-1
10                  if MATCH[ℓ] ≠ Λ then (* ℓ is matched *)
11                    begin lower: = min (lower, BEG[MATCH[ℓ]])
12                          upper: = max (upper, END[MATCH[ℓ]])
13                    end
14                  if ℓ <  top(STACK)+1 then (* skip interval *)
15                    begin  ℓ ⇐ STACK
16                           ℓ ⇐ STACK
17                           lower: = min(lower,ℓ)
18                    end
19                end
20              while u < upper do (* scan upward *)
21                begin u: = u+1
22                  if MATCH[u] ≠ Λ then  (* u is matched *)
23                    begin lower: = min(lower, BEG[MATCH[u]])
24                          upper: = max(upper, END[MATCH[u]])
25                    end
26                end
27            until (ℓ=lower) and (u=upper) (* extended interval completed *)
28            STACK ⇐ lower
29            STACK ⇐ upper
30         end
31     end
32 end
```

To analyze the performance of Algorithm 4, we note that each element of A is scanned at most once (either by loop 8 or by loop 20); the extremes of extended intervals are pushed into (lines 28 and 29) and popped from STACK (lines 15 and 16) at most once, thereby allowing the conclusion that the algorithm runs in time O(m+n).

5. Generalizations and related problems

In this section we shall briefly describe two interesting generalizations of the notion of a convex bipartite graph to which Glover's rule, and hence the efficient algorithms previously described, are applicable, and an extension of the techniques to a weighted matching problem, which models a significant scheduling application.

5.1. Simple chessboards: a generalization of doubly convex bipartite graphs

Algorithm 3 can be applied to a class of convex bipartite graphs more general than that of doubly convex graphs. In order to describe this class we shall need some definitions. By a chessboard we shall mean any finite collection of unit squares with integer coordinates on a plane. Any such unit square will be denoted by coordinates $\langle x,y\rangle$ of its left lower corner. A chessboard is simple if for any of its squares $\langle x,y_1\rangle$, $\langle x,y_2\rangle$, where $y_1 \leq y_2$, it contains all squares $\langle x,y\rangle$, $y_1 \leq y \leq y_2$ (see Figure 5). Rows and columns of a chessboard are defined in the natural way as maximal horizontal and vertical sequences of adjacent squares, respectively. We may allow a simple chessboard to be cut vertically in some places to make some squares nonadjacent (such as $\langle 6,8\rangle$ and $\langle 7,8\rangle$ in Figure 5), provided the line along which we cut touches the boundary of the chessboard. Let A and B be the set of columns and rows of a simple chessboard, respectively, and let us consider the bipartite graph $G = (A,B,E)$, where $(a,b) \in E$ iff column a and row b intersect (i.e., have a square in common). This graph is convex on A (but not necessarily doubly convex), the required ordering of A being given by the natural left-to-right ordering of columns. It is easily seen that any matching in G corresponds to a set of nonattacking rooks on this chessboard (see Figure 5). If the j^{th} row of a simple chessboard consists of squares $<x,Y[j]>$, $BEG[j] \le x \le END[j]$, then the maximum cardinality set of nonattaching rooks on this chessboard is found by Algorithm 3 in time linear in the number of rows and columns. The reason why Algorithm 3 works correctly is that similarly to the doubly convex case, the sequence of ends of rows "seen" from any column of a simple chessboard is bitonic, whence the sequence of the values of END for vertices $j \in B$ (rows of the chessboard) stored on the DEQUE is also bitonic, and we may find a vertex with the minimal value of END either at the top or at the bottom of the DEQUE. We leave details to the reader.

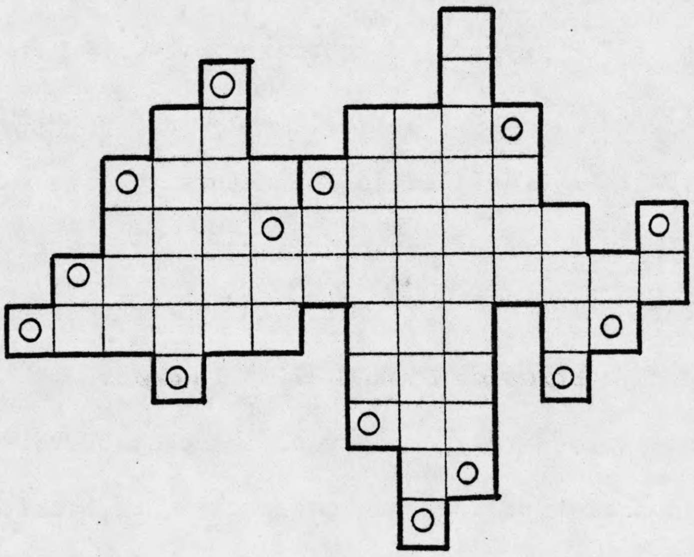

Figure 5. A simple chessboard with a maximum set of nonattacking rooks found by Algorithm 3.

5.2. Bipartite graphs convex on a tree-ordered set

Glover's rule works correctly in a more general situation, where the sets $A(b)$, $b \in B$ are (sets of vertices of) paths in a directed tree (for concreteness we shall assume that the tree is directed toward the root; families of sets of this type are of some importance in file organization, see [9]). The convex case is easily seen to correspond to a tree degenerating into a single path. Assume that a directed tree with vertex set A is represented by an array S[1:m] which gives the successor S[a] of any vertex $a \in A$ ($S[a] = \Lambda$ if a is the root). Similarly as in the convex case, let A(b) be represented by the pair <BEG[b],END[b]>, meaning that A(b) is the set of vertices of the path in the tree, beginning at BEG[b] and ending at END[b]. From the array S we can easily produce, in O(m) time, a topological ordering of A, i.e., a linear ordering of the elements of A, in which the distance to the root, or the rank of a vertex, is nonincreasing. We may also assume that the predecessors of any vertex appear consecutively in this ordering, and that if a_1 appears earlier than a_2 then all predecessors of a_1 appear earlier than all predecessors of a_2. This is always the case if the ordering is found by a breadth-first search of the tree. The algorithm for finding a maximum matching in our bipartite graph processes the vertices of A according to the just described ordering and runs as follows. Instead of a single priority queue, we maintain a collection of priority queues; at any instant in the execution of the algorithm there are as many distinct queues as there are vertices of A with the same value of rank currently being processed. Each time we encounter a vertex $i \in A$ which is a leaf of the tree we initialize a new priority queue and insert into it all vertices $j \in B$ with BEG[j] = i; each time we have processed all predecessors of a vertex a, we merge the queues corresponding to them into one queue corresponding to a. All other details are the same as in Algorithm 1. The reason why our procedure works correctly is as follows. The priority queue Q corresponding to a vertex a contains all so far unmatched vertices $b \in B$ such that $a \in A(b)$. The paths starting at a and ending at vertices END[b], b in Q, are nested one in another, exactly as in the convex case, whence the same agrument based on Lemma 1 can be applied to prove that matching a to the vertex b in Q with the minimal value of END guarantees that the matching obtained will be of maximal cardinality.

If we apply the mergeable heap structure described by van Emde Boas [3], which allows the priority queues to be efficiently merged, then we can achieve $O(m + A(n)n\log\log n)$ time complexity, where A(n) is the functional inverse, very slowly growing, of a function of Ackerman type (see Tarjan [11]).

Our algorithm can be used to find a maximum set of nonattacking rooks on a chessboard satisfying the following condition: any two squares $<x,y_1>$ $<x,y_2>$ can be joined by a sequence $<x,y_1> = <x^{(1)},y^{(1)}>,<x^{(2)},y^{(2)}>,\ldots,<x^{(k)},y^{(k)}> = <x,y_2>$ of adjacent (i.e., having an edge in common) squares with $x^{(i)} \geq x$, $1 \leq i \leq k$. In words, the chessboard does not branch as we go from left to right (see Figure 6). The tree-like ordering of the set A of columns of such a chessboard is defined so that a column containing square $<x+1,y>$ is the successor of column containing square $<x,y>$.

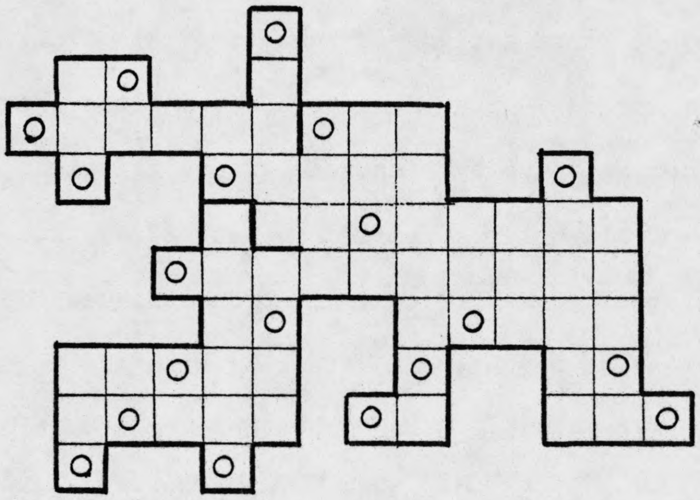

Figure 6. A chessboard and a maximum set of nonattacking rooks found by a modification of Algorithm 1.

5.3. <u>Gale-optimal matchings and one-processor scheduling of independent tasks</u>

It is clear that Algorithm 4 can be modified so that it finds an alternating path in a convex bipartite graph - if there is one - in linear time. Using such a modified algorithm as a subroutine in the standard method of finding a maximum matching, based on repeatedly augmenting a matching along an alternating path (see, e.g. [8]), we can obtain an algorithm of complexity $O(n(m+n))$. Of course, it is less efficient than the $O(m+n\log\log n)$ Algorithm 1. However, there is a situation when the standard alternating path algorithm is of interest.

Suppose that there is a weight $w(b) \geq 0$ associated with every $b \in B$, and that we are looking for a matching which maximizes the sum of weight of matched vertices in B. Since assignable subsets of B - i.e., subsets that can be covered by a matching - form a matroid, it follows that the matching we are looking for can be found by a matroid greedy algorithm (see Lawler [8] for the explanation of all notions related to matroids). More exactly, our matching can be obtained as follows: (i) order the vertices in B according to nonincreasing weight, (ii) starting with the empty matching, scan B in this order; for any $b \in B$, augment the current matching along an alternating path starting at b and ending at an unmatched vertex in A, if such a path exists, or leave b unmatched otherwise. Notice that after the augmentation process in step (ii), vertices which were matched remain matched (probably to different vertices), and vertices which were left unmatched before, remain unmatched. It can be proved (Gale [5], see also [8]), that the matching M so obtained is Gale-optimal, i.e. optimal in the following strong sense: Let $\{b_1,\ldots,b_k\} \subseteq B$, $w(b_1) \geq \ldots \geq w(b_k)$ be the set of vertices covered by M. Then for any other matching M', the set $\{c_1,\ldots,c_\ell\} \subseteq B$, $w(c_1) \geq \ldots \geq w(c_\ell)$ of vertices covered by M' satisfies the condition $\ell \leq k$, $w(b_1) \geq w(c_1),\ldots,w(b_\ell) \geq w(c_\ell)$. (Notice that both the greedy algorithm and the notion of Gale-optimality depend only on the ordering of B according to the weights, and not on the actual values of the weights.)

It is obvious that a Gale-optimal matching of a convex bipartite graph can be obtained in $O(n(m+n))$ time by the greedy algorithm, using a modification of Algorithm 4, as explained at the beginning of this subsection.

There is an interesting relationship between Gale-optimal matchings in convex bipartite graphs and the problem of scheduling a set B of n independent (no precedence constrains) tasks on one processor, where each task takes one unit of processing time, there is a <u>starting time</u> BEG[j] and <u>deadline</u> END[j] for every task j, and a <u>penalty</u> p(j) which must be paid if this task is not executed in the time interval [BEG[j],END[j]] (we assume that time is integer-valued). It is easy to see that any schedule minimizing the total penalty corresponds to a Gale-optimal matching in a convex bipartite graph defined by arrays BEG,END, and with $w(j) = M-p(j)$ ($M > \max_{1 \leq j \leq n} p(j)$): the vertex i matched to task $j \in B$ determines the unit interval of time when j is to be executed (see Lawler [8], Chapter 7). We conclude that an optimal schedule for this problem can be obtained in $O(n(m+n))$ time (m is the maximal deadline). Of course, if all penalties are equal, i.e., when we simply maximize the number of tasks executed, then the optimal schedule can be obtained in $O(m+n\log\log n)$ time by Algorithm 1.

As a closing remark, we note that the maximum matching problem on a general bipartite graph G corresponds to the situation where for any $b \in B$ the set A(b) is a collection of t(b) intervals of A. It is an almost straightforward extension of our discussions in Sections 2 and 4, to show that the standard approach based on augmenting paths [8] can be implemented - both for the maximum matching and for the Gale-optimal matching - in time $O(n(m+t\log\log n))$ where $t = \sum_{b \in B} t(b)$ is the total number of intervals in the given G.

Appendix

Algorithm A (Partitioning a sequence of n integers into two nonincreasing subsequences)

Input ; $S[1{:}\ell]$ - the original sequence

Output: $\mathrm{SUB1}[1{:}\ell 1]$, $\mathrm{SUB2}[1{:}\ell 2]$ - two nonincreasing subsequences into which $S[1{:}\ell]$ is partitioned $S[1] = \mathrm{SUB1}[1]$

```
1  begin  $\ell 1 := \ell 2 := 0$,   $\mathrm{SUB1}[0] := \mathrm{SUB2}[0] := \infty$
2     for $i := 1$ to $\ell$ do
3        if  $S[i] \le \mathrm{SUB1}[\ell 1]$ then (* add S[i] to first subsequence *)
4           begin  $\ell 1 := \ell 1 + 1$
5                  $\mathrm{SUB1}[\ell 1] := S[i]$
6           end
7        else if $S[i] \le \mathrm{SUB2}[\ell 2]$ then (* add S[i] to second subsequence *)
8                begin  $\ell 2 := \ell 2 + 1$
9                       $\mathrm{SUB2}[\ell 2] := S[i]$
10               end
11            else stop (* no partitioning possible *)
12 end
```

To prove the correctness of the algorithm, first notice that we always have $\mathrm{SUB1}[\ell 1] \le \mathrm{SUB2}[\ell 2]$, the inequality being strict except for $\ell 1 = \ell 2 = 0$. If now, for some i, we reach the condition $\mathrm{SUB1}[\ell 1] < \mathrm{SUB2}[\ell 2] < S[i]$ (line 11) it is clear that the original $S[1{:}\ell]$ contains an *increasing* subsequence of length 3, which makes impossible its partitioning into two *nonincreasing* subsequences.

One may note that the algorithm easily generalizes to an algorithm for partitioning an arbitrary sequence of length ℓ into the minimal possible number of nonincreasing subsequences, in time $O(\ell \log d)$, where d is this minimial number of subsequences, or - equivalently - the maximal length of an increasing subsequence in the given sequence.

References

1. Aho, A. V., Hopcroft, J. E., and Ullman, J. D. The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, MA, 1974.

2. Booth, K. S., and Lueker, G. S. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. J. Comp. System Sci. 13 (1976), 335-379.

3. Emde Boas, P. van Preserving order in a forest in less than logarithmic time. Proc. 16th Annual Symp. on Foundations of Comp. Sci., Univ. of California, Berkeley, Oct. 1975, pp. 75-84.

4. Emde Boas, P. van Preserving order in a forest in less than logarithmic time and linear space. Information Proc. Lett. 6 (1977), 80-82.

5. Gale, D. Optimal assignments in an ordered set: an application of matroid theory. J. Combinatorial Theory 4 (1968), 176-180.

6. Glover, F. Maximum matching in convex bipartite graph. Naval Res. Logist. Quart. 14 (1967), 313-316.

7. Hopcroft, J. E. and Karp, R. M. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM J. Comput. 2(1973), 225-231.

8. Lawler, E. L. Combinatorial Optimization: Networks and Matroids. Holt, Rinehart and Winston, New York, NY, 1976.

9. Lipski, W. Information storage and retrieval - mathematical foundations II (Combinatorial problems). Theoret. Comput. Sci. 3 (1976), 183-211.

10. Lipski, W., Lodi, E., Luccio, F., Mugnai, C., and Pagli, L. On two dimensional data organization II. Tech. Rep. S-77-43, Inst. of Comp. Sci., Univ. of Pisa, Italy, December 1977. To appear in Fundamenta Informaticae.

11. Tarjan, R. E. Efficiency of a good but not linear set union algorithm. J. ACM 22 (1975), 215-224.