A Parallel Matching Algorithm for Convex Bipartite Graphs and Applications to Scheduling*

ELIEZER DEKEL[†] AND SARTAJ SAHNI

Department of Computer Science, University of Minnesota, Minneapolis, Minnesota 55455

An efficient parallel algorithm to obtain maximum matchings in convex bipartite graphs is developed. This algorithm can be used to obtain efficient parallel algorithms for several scheduling problems. Some examples are: job scheduling with release times and deadlines; scheduling to minimize maximum cost; and preemptive scheduling to minimize maximum completion time.

1. INTRODUCTION

A convex bipartite graph G is a triple (A, B, E). $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_m\}$ are disjoint sets of vertices. E is the edge set. E satisfies the following properties:

(1) If (i, j) is an edge of E, then either $i \in A$ and $j \in B$ or $i \in B$ and $j \in A$; i.e., no edge joins two vertices in A or two in B.

(2) If $(a_i, b_j) \in E$ and $(a_i, b_{j+k}) \in E$, then $(a_i, b_{j+q}) \in E$, $1 \le q < k$.

Property (1) above is the bipartite property while property (2) is the convexity property. An example of a convex bipartite graph is shown in Fig. 1.1. $F \subseteq E$ is a *matching* in the convex bipartite graph $G = (A, B, E)$ iff no two edges in F have a common endpoint. $F1 = \{(a_1, b_2), (a_4, b_3), (a_5, b_1)\}$ is a matching in the graph of Fig. 1.1 while $F2 = \{(a_1, b_1), (a_1, b_2), (a_2, b_3)\}$ is not. F is a *maximum cardinality matching* (or simply a maximum matching) in G iff (a) F is a matching and (b) G contains no matching H such that $|H| > |F|$ ($|H|$ = number of edges in H). The matching depicted by solid lines in Fig. 1.1 is a maximum matching in that graph.

In what follows, we shall find it convenient to have an alternate representation of convex bipartite graphs. It is clear that every convex bipartite graph

*This research was supported in part by the Office of Naval Research under contract N00014-80-C-0650.

[†]Current address: Mathematical Sciences Program, University of Texas at Dallas, Richardson, Texas 75080.

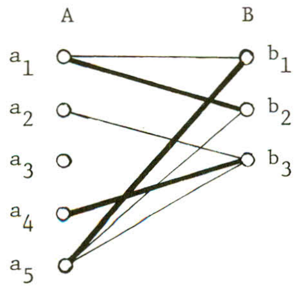


FIG. 1.1. A convex bipartite graph.

$G = (A, B, E)$, $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_m\}$ is uniquely represented by the set of triples:

$$T = \{(i, s_i, h_i) \mid 1 \le i \le n\}$$
$$s_i = \min\{j \mid (a_i, b_j) \in E\}$$
$$h_i = \max\{j \mid (a_i, b_j) \in E\}.$$

In the *triple representation,* we explicitly record the smallest (s_i) and highest (h_i) index vertices to which each a_i is connected. For the example of Fig. 1.1, we have $T = \{(1, 1, 2), (2, 3, 3), (3, 0, 0), (4, 3, 3), (5, 1, 3)\}$.

As an example of the use of matchings in convex bipartite graphs, consider the problem of scheduling a set J of n unit time jobs to minimize the number of tardy jobs. In this problem, we are given a set, of n jobs. Job i has a release time r_i and a due time d_i. It requires one unit of processing. We assume that r_i and d_i are natural numbers. A subset F of J is a *feasible subset* iff every job in F can be scheduled on one machine in such a way that no job is scheduled before its release time or after its due time. A feasible subset F is a *maximum feasible subset* iff there is no feasible subset MAXM of J such that $|Q| > |J|$.

A maximum feasible subset F can be found by transforming the problem into a maximum matching problem on a convex bipartite graph. Without loss of generality, we may assume that $\min\{r_i\} = 0$; $r_i < d_i$, $1 \le i \le n$; and $\max\{d_i\} \le n$. The convex bipartite graph corresponding to J is given by the triple set $T = \{(i, s_i, h_i) \mid s_i = r_i,\ h_i = d_i - 1\}$. Figure 1.2 shows an example of a job set and the corresponding convex bipartite graph G. Vertex i of A represents job i while vertex i in B simply represents the time slot $[i, i + 1]$. There is an edge from job i to time slot $[j, j + 1]$ iff $r_i \le j < d_i$. Hence, every matching in G represents a feasible subset of J. Also, corresponding to every feasible subset of J there is a matching in G. Clearly, a maximum cardinality feasible subset of J can be easily obtained from a maximum matching of G. In addition, a maximum matching also provides the time slots in which the jobs should be scheduled.

$r_i = s_i$	$d_i = h_i + 1$
0	2
0	3
2	4
2	6
4	5
2	6

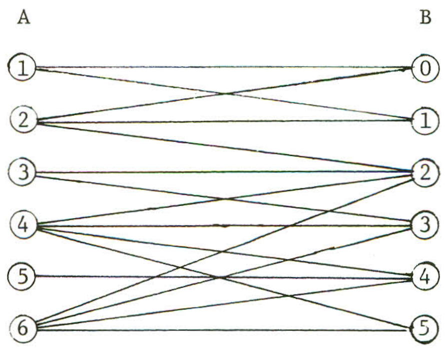


FIGURE 1.2

We shall see several other examples of the application of matchings in convex bipartite graphs in later sections.

Glover [5] has obtained a rather simple algorithm to find a maximum matching in a convex bipartite graph $G = (A, B, E)$. Let $h_i = \max\{j \mid (a_i, b_j) \in E\}$, $1 \leq i \leq |A|$. Glover's algorithm considers the vertices in B one by one starting at b_1. We first determine the set R of remaining vertices in A to which the vertex b_j currently being considered is connected. Let q be such that $a_q \in R$ and $h_q = \min_{a_p \in R}\{h_p\}$. Vertex b_j is matched to a_q and a_q deleted from the graph. The next vertex in B is now considered. Observe that Glover's algorithm is essentially the same as that suggested by Jackson [6].

A straightforward implementation of Glover's algorithm has complexity $O(mn)$. When m is $O(\log \log n)$, a more efficient implementation results from the use of the fast priority queues of van Emde Boas [4, 8]. The resulting implementation has complexity $O(m + n \log \log n)$. The fastest sequential algorithm known for the matching problem is due to Lipski and Preparata [8]. It differs from Glover's algorithm in that it examines the vertices of A one by one rather than those of B. This algorithm has complexity $O(n + mA(m))$, where $A(\cdot)$ is the inverse of the Ackermann's function and is a very slowly growing function.

In this paper, we are primarily concerned with parallel algorithms. In Section 2, we obtain a parallel algorithm for maximum matchings in convex bipartite graphs. In Section 3, we use this parallel algorithm to obtain efficient parallel algorithms for three scheduling problems. All our analyses will assume the availability of as many processing elements (PEs) as needed. This is in keeping with much of the research done on parallel algorithms. In practice, of course, only k processors (for some fixed k) will be available. Our analyses are easily modified for this case. It will be apparent that if our algorithm has time complexity $t(n)$ using $g(n)$ PEs, then with k PEs $(k < g(n))$, its complexity will be $t(n)g(n)/k$.

The parallel computer model used is the shared memory model (SMM). This is an example of a single instruction stream–multiple data stream (SIMD) computer. In a SMM computer, there are p PEs. Each PE is capable of performing the standard arithmetic and logical operations. The PEs are indexed $0, 1, \ldots, p-1$ and an individual PE may be referenced as in PE(i). Each PE knows its index and has some local memory. In addition, there is a global memory to which every PE has access. The PEs are synchronized and operate under the control of a single instruction stream. An enable/disable mask may be used to select a subset of the PEs that are to perform an instruction. Only the enabled PEs will perform the instruction. Disabled PEs remain idle. All enabled PEs execute the same instruction. The set of enabled PEs may change from instruction to instruction.

If two PEs attempt to simultaneously read the same word of the shared memory, a *read conflict* occurs. If two PEs attempt to simultaneously write into the same word of the shared memory, a *write conflict* occurs. Throughout this paper, we shall assume that read and write conflicts are prohibited.

The reader is referred to [2] for a list of references dealing with graph algorithms, matrix algorithms, sorting, scheduling, etc., on a SMM computer.

2. Parallel Matching in Convex Bipartite Graphs

In Section 1, we showed that every instance of the problem of scheduling jobs to minimize the number of tardy jobs could be transformed into an equivalent instance of the maximum matching in a convex bipartite graph problem. It should be evident that the reverse is also true. Hence, the two problems are isomorphic. A parallel algorithm for a special case of the job scheduling formulation was obtained by us in [1]. In this special case, it was assumed that all jobs have the same release time. This corresponds to the case when the convex bipartite graphs are of the form $T = \{(i, s_i, h_i) \mid 1 \leq i \leq n\}$ and $s_i = c$, $1 \leq i \leq n$ for some c.

We shall now proceed to show how the solution for the special case described above can be used to solve the general case when all the r_i's are not the same. This will be done using the binary tree method described by Dekel and Sahni [2]. Rather than specify the new algorithm formally, we shall describe how it works by means of an example.

A convex bipartite graph is shown in Fig. 2.1. For this graph, $|A| = 14$ and $|B| = 13$. The s_i and h_i values associated with each vertex of A are given in the first two columns of this figure. The first step in our parallel algorithm for maximum matching is to sort the vertices in A in nondecreasing order of s_i. Vertices with the same s_i are sorted into nondecreasing order of h_i. For our example, the result of this reordering is shown in Fig. 2.2.

Following the sort, we identify the distinct s_i values. Let these be

s_i	h_i
4	8
1	4
1	3
1	4
4	10
4	4
1	2
9	11
9	10
12	13
9	13
4	11
12	13
12	12

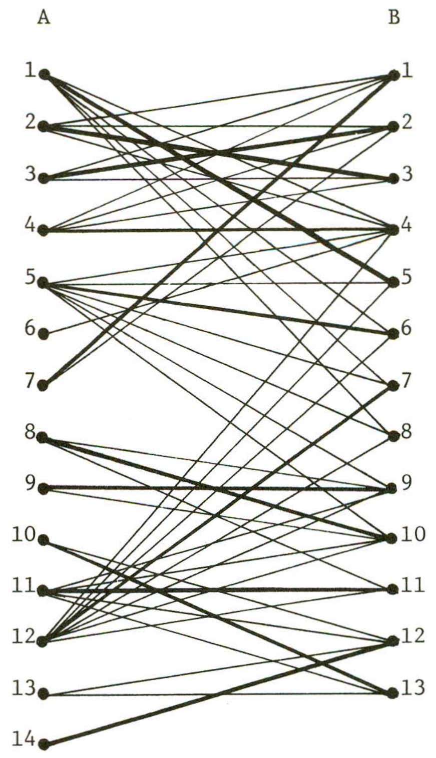


FIGURE 2.1

$R_1, R_2, \ldots, R_k$. Assume that $R_1 < R_2 < \cdots < R_k$. Let $R_{k+1} = \max\{h_i\} + 1$. For our example, $k = 4$ and $R(1 : k+1) = (1, 4, 9, 12, 14)$.

We are now ready to use the binary tree method of [2]. The underlying computation tree we shall use is the unique complete binary tree with k leaf nodes. Figure 2.3 shows the complete binary trees with 4, 5, and 6 leaf nodes. For our example, $k = 4$ and we use the tree of Figure 2.3a. With each node, P, in the computation tree, we associate a contiguous subset $\{u, u+1, u+2, \ldots, v\}$ of the vertices in B. This subset is denoted $[u, v] \cdot P$ or simply $[u, v]$.

i	7	3	2	4	6	1	5	12	9	8	11	14	10	13
s_i	1	1	1	1	4	4	4	4	9	9	9	12	12	12
h_i	2	3	4	4	4	8	10	11	10	11	13	12	13	13

FIGURE 2.2

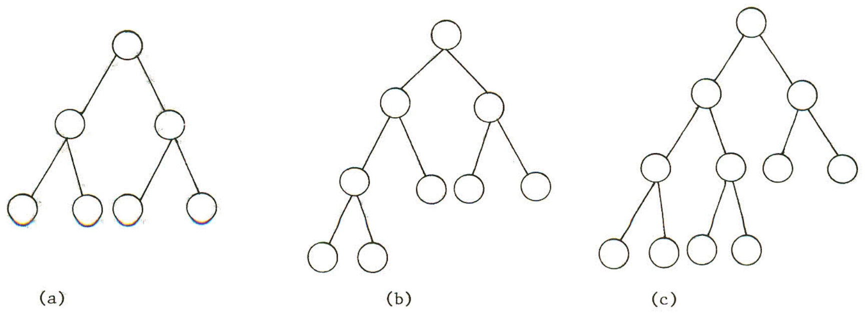


FIG. 2.3. Complete binary trees: (a) 4 leaves; (b) 5 leaves; (c) 6 leaves.

Let the leaf nodes of the computation tree be numbered 1 through k, left to right. If P is the ith leaf node, then $[u, v] \cdot P$ is $[R_i, R_{i+1} - 1]$ (i.e., $u = R_i$ and $v = R_{i+1} - 1$). If P is not a leaf node, then the subset of B associated with P is $[u, v] \cdot \mathrm{LC}(P) \cup [u, v] \cdot \mathrm{RC}(P)$, where LC($P$) and RC($P$) are, respectively, the left and right children of P. The subsets of B associated with each of the vertices in the computation tree for our example are shown in Fig. 2.4. The number in each node of this tree is its index.

Let P be any vertex of the computation tree. Let $[u, v]$ be the subset of B associated with P. The subset of A *available for matching* at node P is denoted $M(P)$ and is defined to be

$$M(P) = \{i \mid u \leq s_i \leq v\}.$$

For example,

$$M(1) = \{1, 2, \ldots, 14\};$$
$$M(2) = \{1, 2, 3, 4, 5, 6, 7, 12\};$$
$$M(4) = \{2, 3, 4, 7\};$$
etc.

The subset $M(P)$ of A vertices available for matching at P may be partitioned into three subsets MAXM(P), $I(P)$, and $T(P)$. MAXM(P) is a max-

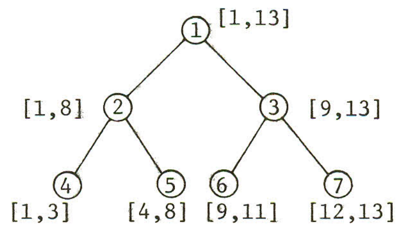
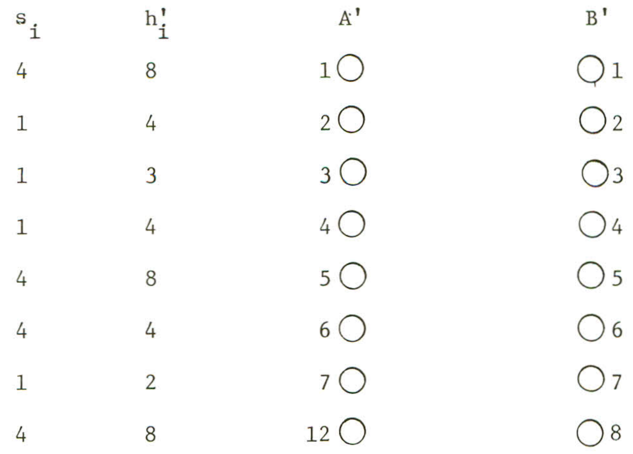


FIGURE 2.4



s_i	h'_i	A'	B'
4	8	1	1
1	4	2	2
1	3	3	3
1	4	4	4
4	8	5	5
4	4	6	6
1	2	7	7
4	8	12	8

FIGURE 2.5

imum cardinality subset of $M(P)$ that may be matched with vertices in $[u, v] \cdot P$; this subset is called the *matched set*. $I(P)$ denotes the *infeasible set*. It consists of all vertices $i \in M(P) - \text{MAXM}(P)$ such that $h_i \leq v$. The *transferred set* $T(P)$ consists of all vertices $i \in M(P) - \text{MAXM}(P)$ such that $h_i > v$.

Consider node 2 of Fig. 2.4. The matching problem defined at this node is given in Fig. 2.5. Note that $h'_i = \min\{v, h_i\}$. A' is the set $M(2)$ and B' is $[u, v] \cdot 2$. If Glover's algorithm is used on this graph, then $\{1, 2, 3, 4, 5, 7, 12\}$ defines a subset of A' that can be matched with vertices in B'. Further, this gives a maximum matching. Hence, $\text{MAXM}(2) = \{1, 2, 3, 4, 5, 7, 12\}$; $I(2) = 6$; and $T(2) = \emptyset$. Observe that $|\text{MAXM}(1)|$ is the size of a maximum matching in the original convex bipartite graph. Also, $T(1) = \emptyset$ and $I(1) = A - \text{MAXM}(1)$.

We shall make two passes over the computation tree. The first pass begins at the leaves and moves toward the root. During this pass, the MAXM, I, and T sets for each node are computed. The second pass starts at the root and progresses toward the leaves. In this pass, the MAXM set for each node is updated so as to correspond to the set of A vertices matched by Glover's algorithm to the B vertices associated with that node.

Pass 1

In this pass, we make extensive use of the parallel algorithm developed in [1] for the case when all the s_i's are the same. For our purposes here, it is sufficient to know the sequential algorithm (FEAS of [1]) that this parallel algorithm is based on. This sequential algorithm is given in Fig. 2.6. For convenience, this has been translated into the graph language used here. The

```
line procedure FEAS($n$, $u$, $v$)
        //Find a maximum matching of vertices in $A$ into the $B$ set $[u, v]$. For every vertex $i \in A$,
        $s_i = u$.//
  1     global MAT(1 : $n$); set $A$; integer $n, u, v, i, j$
  2     sort $A$ into nondecreasing order of $h_i$
  3     MAT(1 : $n$) $\leftarrow$ 0//initialize//
  4     $j \leftarrow u$
  5     for $i \leftarrow 1$ to $n$ do
  6        case
  7        :$j > v$ : return($j$) //all vertices in $B$ matched//
  8        :$j \leq h_i$ : //select $i$// $j \leftarrow j + 1$, MAT($i$) $\leftarrow$ 1
  9        end case
 10     end for
 11     return($j$)
 12     end FEAS
```

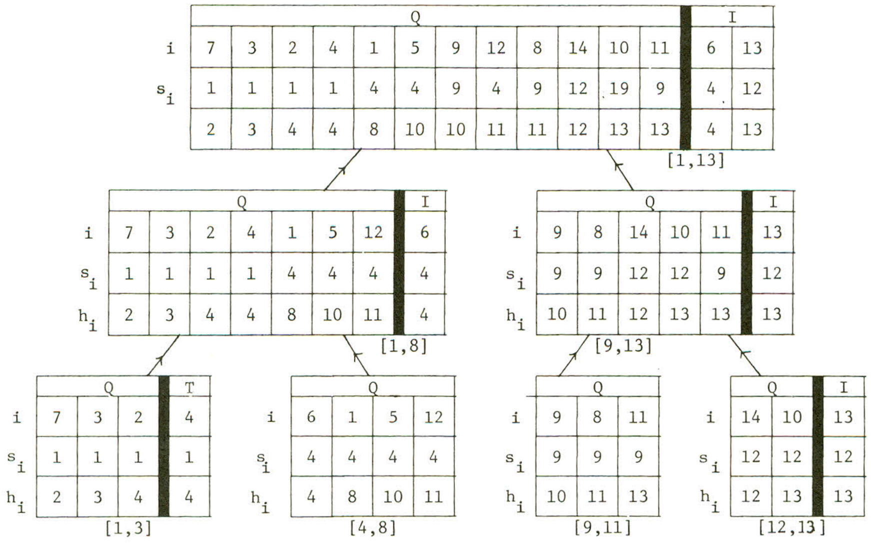
FIGURE 2.6

parallel version of this algorithm has complexity $O(\log n)$ and uses $n/\log n$ PEs [1].

An examination of Glover's algorithm reveals that it performs exactly as does procedure FEAS when the restrictions and simplifications applicable to FEAS are incorporated into it.

Hence, for a leaf node of the computation tree, the MAXM set may be obtained by a direct application of procedure FEAS (or its parallel equivalent). For example, for node 4 of Fig. 2.4, we have $A = M(4) = \{2, 3, 4, 7\}$; $h_2 = 4$; $h_3 = 3$; $h_4 = 4$; $h_7 = 2$; $u = 1$; and $v = 3$. Using FEAS (observe that this algorithm yields the same results regardless of whether the h_i values or the modified values h_i' of Fig. 2.5 are used), we obtain MAXM(4) = $\{7, 3, 2\}$. Note that MAXM consists of exactly those vertices i with MAT(i) = 1. $I(\ )$ consists of exactly those vertices i with MAT(i) = $\emptyset$ and $h_i \leq v$. The remaining vertices form $T(\ )$. The matched set MAXM, transferred set T, and infeasible set I for each of the leaves in our example are shown in Fig. 2.7. Null sets are not explicitly shown. So, for node 4, $I(4) = \emptyset$; $T(4) = \{4\}$; and MAXM(4) = $\{7, 3, 2\}$. The sets are ordered by h_i.

For a nonleaf node P, the MAXM, I, and T sets may be obtained by using the MAXM, I, and T sets of the children of P. Let L and R, respectively, be the left and right children of P. To determine MAXM(P), we first use procedure FEAS with $u = u_R$ and $v = v_R$ ($[u_R, v_R]$ is the subset of B associated with the right child R of P). The A set consists of $T(L) \cup$ MAXM(R). Since both $T(L)$ and MAXM(R) are already sorted by h_i, the sort of line 2 of FEAS can be replaced by a merge. Let S be the subset of $T(L) \cup$ MAXM(R) that has MAT( ) = 1 following the execution of FEAS. The following theorem establishes that MAXM(L) $\cup$ S is a maximum cardinality subset of $M(P)$ that may be matched with vertices in $[u, v] \cdot P$. Hence, MAXM(P) = MAXM(L) $\cup$ S. Following the determination of S, MAXM(L) and S are merged to obtain MAXM(P) in nondecreasing order of h_i.



[1,13]

	Q												I	
i	7	3	2	4	1	5	9	12	8	14	10	11	6	13
s_i	1	1	1	1	4	4	9	4	9	12	19	9	4	12
	2	3	4	4	8	10	10	11	11	12	13	13	4	13

[1,8]

	Q							I
i	7	3	2	4	1	5	12	6
s_i	1	1	1	1	4	4	4	4
h_i	2	3	4	4	8	10	11	4

[9,13]

	Q					I
i	9	8	14	10	11	13
s_i	9	9	12	12	9	12
h_i	10	11	12	13	13	13

[1,3]

	Q			T
i	7	3	2	4
s_i	1	1	1	1
h_i	2	3	4	4

[4,8]

	Q			
i	6	1	5	12
s_i	4	4	4	4
h_i	4	8	10	11

[9,11]

	Q		
i	9	8	11
s_i	9	9	9
h_i	10	11	13

[12,13]

	Q		I
i	14	10	13
s_i	12	12	12
h_i	12	13	13

FIG. 2.7. Results of first pass.

THEOREM 2.1. *MAXM(L) ∪ S as defined above is a maximum cardinality subset of $M(P)$ that may be matched with vertices in $[u, v] \cdot P$ using algorithm* MATCH.

Proof. The proof is by induction on the height of the subtree of which P is the root (a tree consisting of only a root has height 0). If this height is 1, then MAXM(L) and MAXM(R) are maximum cardinality subsets of $M(L)$ and $M(R)$ that can, respectively, be matched by Glover's algorithm with vertices in $[u, v] \cdot L$ and $[u, v] \cdot R$. If this distance is greater than 1, then MAXM(L) and MAXM(R) satisfy this maximum cardinality matching requirement by induction.

As far as node P itself is concerned, we see that only vertices in $M(L)$ are candidates for matching with vertices in $[u_L, v_L]$ (recall that for vertices in $M(R)$, the s_i value exceeds v_L). Furthermore, when Glover's algorithm is used with the A set being $M(P)$ and the B set being $[u_L, v_R] = [u, v] \cdot P$, vertices in B are considered in the order $u_L, u_L + 1, \ldots, v_L, u_R, \ldots, v_R$. Hence, MAXM($L$) is precisely the subset of $M(P)$ that gets matched with vertices in $[u_L, v_L]$.

The candidates for the remaining vertices in B, i.e., $[u_R, v_R]$, are clearly $T(L) \cup M(R)$. From the way Glover's algorithm works, it is also clear that the vertices of $T(L) \cup M(R)$ that will get matched to $[u_R, v_R]$ are a subset of $T(L) \cup$ MAXM(R). Let this subset be S'. We wish to show that S is a legitimate choice for S'. First, we show that S represents a feasible matching. Then, we shall show that S is in fact selectable by Glover's algorithm.

We know that MAXM(R) can be matched into $[u_R, v_R]$. Let Z be any such matching. Since S is selected by FEAS, we know that every vertex in S can be paired with a distinct vertex in $[u_R, v_R]$ in a such a way that no vertex j in S is paired with a vertex with index greater than h_j. Consider a pairing W that meets this condition. Now suppose that some vertex j in S is paired with a vertex q in $[u_R, v_R]$ with index less than s_j. Clearly, j must be a member of MAXM(R) (as all vertices in $T(L)$ have an s value less than u_R). Suppose that j is matched to j_1 in Z. So, $q < j_1$. If j_1 is free in W, then the pairing of j in W may be changed from q to j_1. If j_1 is not free, then suppose it is matched to j_2. From the restriction on W, it follows that $q < j_1 \le h_{j_2}$. If $j_2 \in T(L)$, then j_2 may be paired with q and j with j_1 (since $q < j_1$, $s_{j_2} < q < h_{j_2}$). If $j_2 \in$ MAXM(R), then suppose that j_2 is matched to j_3 in Z. It is easy to see that $j_3 \ne j_1$. If $q = j_3$ or j_3 is free in W, then we may pair j with j_1 and j_2 with j_3. If q is in the interval $[s_{j_2}, h_{j_2}]$, then we may pair j with j_1 and j_2 with q. If q is not in this interval, then since $q < h_{j_2}$, $q < s_{j_2} \le j_3$. Note that the condition $q < j_{2i+1}$ is preserved. This is needed in case $j_{2i+2} \in T(L)$. Now, let j_4 be the vertex paired with j_3 in W. It should be clear that we can continue in this way and modify W so that j is paired with j_1, j_2 with j_3, j_4 with j_5, etc. In the new pairing, there is one fewer vertex of S that is paired with a vertex with a smaller s value.

Repeating the above construction several times, W can be transformed into a matching such that every vertex $j \in S$ is matched to a vertex q in $[u_R, v_R]$ such that $s_j \le q \le h_j$. Hence, S represents a feasible matching.

Let S' (as defined earlier) be the subset of MAXM(R) $\cup$ $T(L)$ matched by Glover's algorithm to the vertex set $[u_R, v_R]$. We shall now proceed to show that S is a valid choice for S'. Let Z be any matching of MAXM(R) into $[u_R, v_R]$. Let Y be a matching of S' in which all vertices in MAXM(R) $\cap$ S' are matched to the same vertex in $[u_R, v_R]$ as in the matching Z. Let W be a corresponding matching for S. The existence of the matchings Y and W is a consequence of the construction used to show the feasibility of S.

From the definition of S', it follows that $S' \not\subset S$. Also, from the working of FEAS, it follows that $S \not\subset S'$. Let $j \in S$ be a vertex with least h_j such that $j \notin S'$. If no such j exists, then $S = S'$. Assume that j is matched to q in W. If q is free in Y, then S' cannot be of maximum cardinality. So, let $p \in S'$ be matched to q in Y. By definition of Y and W, $p \notin S$. Also, from the order in which FEAS considers vertices, $h_p \ge h_j$ (as otherwise, FEAS would consider p before j and select p for S). Hence, $S' \cup \{j\} - \{p\}$ is also a subset selectable by Glover's algorithm. (Since $h_p \ge h_j$, by ensuring $j < p$, Glover's algorithm will be forced to match j before p.) $S' \cup \{j\} - \{p\}$ agrees with S in one place more than does S'.

By repeating this interchange process, S' may be transformed into S with the result that S is also a maximum cardinality subset of MAXM(R) $\cup$ $T(L)$ that is selectable by Glover's algorithm for matching in $[u_R, v_R]$.

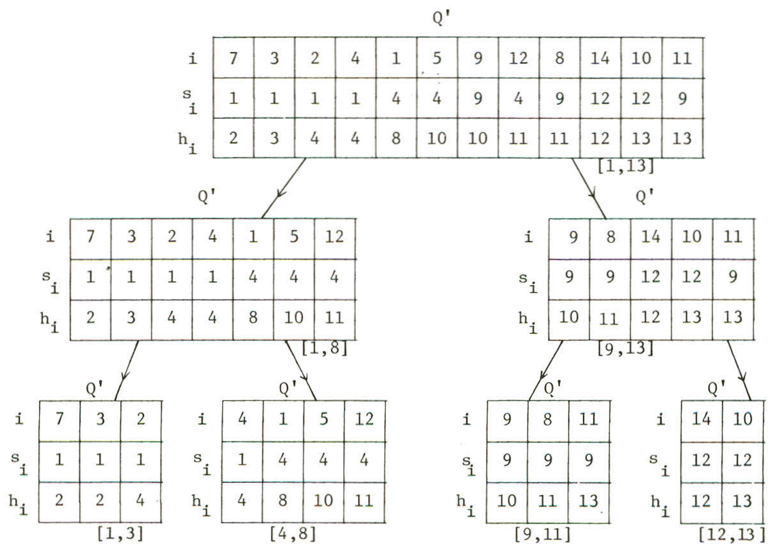
Hence, MAXM(L) $\cup$ S is a maximum cardinality subset of $M(P)$ selectable by Glover's algorithm for matching in $[u, v] \cdot P$. ■

Once MAXM(P) is known, $T(P)$ and $I(P)$ are easily computed. Actually, as $I(P)$ is never used, we may omit its computation. Figure 2.7 shows the MAXM, I, and T sets (except when empty) for all nodes in our example.

Pass 2

In the second pass, for each vertex P of the computation tree; we compute a set MAXM$'(P)$ which represents the set of A vertices matched by Glover's algorithm with the set $[u, v] \cdot P$. With respect to the matching shown by solid lines in Fig. 2.1, we see that if P is the root, then MAXM$'(P) = \{1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 14\}$; if P is node 3 of the computation tree, then MAXM$'(P) = \{8, 9, 10, 11, 14\}$.

If P is the root node, then MAXM$'(P)$ = MAXM(P), by definition of MAXM(P). Let P be any nonleaf node for which MAXM$'(P)$ has been computed. Let L and R be the left and right children, respectively, of P. Let $[u, v] \cdot L = [u_L, v_L]$ and $[u, v] \cdot R = [u_R, v_R]$. Let $V = \{j \mid j \in \text{MAXM}'(P)$ and $s_j < v_R\}$. Let W be the ordered set obtained by merging together V and MAXM(L) (note that both V and MAXM(L) can be maintained so that they are in nondecreasing order of h_i and that W is also in nondecreasing order of h_i). MAXM$'(L)$ consists of the first min $\{|W|, v_L - u_L + 1\}$ vertices in W. The correctness of this statement may be established by induction on the level of P. MAXM$'(R)$ is readily seen to be MAXM$'(P)$ − MAXM$'(L)$. Figure 2.8 shows the MAXM$'(\,)$ sets for all the vertices in the computation tree of our example.



[1,13]

i	7	3	2	4	1	5	9	12	8	14	10	11
s_i	1	1	1	1	4	4	9	4	9	12	12	9
h_i	2	3	4	4	8	10	10	11	11	12	13	13

[1,8]

i	7	3	2	4	1	5	12
s_i	1	1	1	1	4	4	4
h_i	2	3	4	4	8	10	11

[9,13]

i	9	8	14	10	11
s_i	9	9	12	12	9
h_i	10	11	12	13	13

[1,3]

i	7	3	2
s_i	1	1	1
h_i	2	2	4

[4,8]

i	4	1	5	12
s_i	1	4	4	4
h_i	4	8	10	11

[9,11]

i	9	8	11
s_i	9	9	9
h_i	10	11	13

[12,13]

i	14	10
s_i	12	12
h_i	12	13

matching={(7,1),(3,2),(2,3),(4,4),(1,5),(5,6),(12,7),(9,9),(8,10)
(11,11),(14,12),(10,13)}

FIG. 2.8. Second pass.

From the MAXM′( ) sets of the leaves, the matching is easily obtained. If P is a leaf, and $[u, v] \cdot P = [a, b]$, then the first vertex in MAXM′(P) is matched with a, the second with $a + 1$, etc. (note that MAXM′(P) is in nondecreasing order of h_i). The matching for our example is also given in Fig. 2.8.

Complexity Analysis

The initial ordering of A by s_i and within s_i by h_i can be done in $O(\log^2 n)$ time using $n/2$ PEs [11, 12]. During the first pass, the computation of MAXM( ) requires the use of FEAS and a merge. The use of FEAS (without the sort) takes $O(\log n)$ time and requires $O(|M(P)|/\log|M(P)|)$ PEs [1]. The merge at node P takes $O(\log n)$ time with $|M(P)|/2$ PEs. Since MAXM can be computed in parallel for all nodes on the same level of the computation tree, $O(\log n)$ time is needed per level. The total time for the first pass is $O(\log^2 n)$ and $n/2$ PEs are needed. Pass 2 requires only some merging per node. The total cost of this pass is also $O(\log^2 n)$ and $n/2$ PEs suffice.

Hence, the overall complexity of our parallel algorithm for maximum matching in convex bipartite graphs is $O(\log^2 n)$. The PE requirement is $O(n)$.

Another complexity measure often computed for parallel algorithms is the effectiveness of processor utilization (EPU) (see [1, 2, 14]). For any problem P and parallel algorithm A, this is defined as follows:

$$\mathrm{EPU}(P, A) = \frac{\text{complexity of fastest ``known'' sequential algorithm for } P}{\text{complexity of } A * \text{number of PEs used by } A}.$$

For our algorithm, we have an EPU that is $\Omega((n + mA(m))/(\log^2 n * n))$ (recall that $m = |B|$).

3. Applications to Scheduling

We have already seen one application of matchings in convex bipartite graphs to scheduling. This application was to a problem that is actually isomorphic to the matching problem: viz., find the maximum number of unit time jobs from $J = \{(r_i, d_i) \mid 1 \leq i \leq n,\ r_i \text{ and } d_i \text{ are natural numbers}\}$ that can be scheduled on a single machine such that no job is scheduled before its release time r_i or after its due time d_i. In this section, we shall look at three other scheduling problems that can be solved in an efficient manner using the parallel matching algorithm developed in previous section.

3.1 *Scheduling to Minimize the Maximum Cost*

In this problem, we are given n unit time jobs. Associated with each job is a release time r_i (which is a natural number) and a nondecreasing cost function $c_i()$, $1 \leq i \leq n$. Let $S = (s_1, s_2, \ldots, s_n)$ be a one-machine schedule for the n jobs. s_i denotes the start time of job i. Let q_i be as defined below:

$$q_i = \begin{cases} c_i(s_i) & s_i \geq r_i \\ \infty & s_i < r_i. \end{cases}$$

The cost of S is defined to be max $\{q_i\}$. We wish to find a schedule with minimum cost.

As pointed out by Rinnooy Kan [13], this problem may be solved by first determining the finish time of a minimum finish time schedule for the n jobs. If the n jobs are already in nondecreasing order of release times r_i, then a minimum finish time schedule has start times s_i given by

$$\begin{aligned} s_1 &= r_1 \\ s_i &= \max\{r_i, s_{i-1} + 1\}. \end{aligned} \tag{3.1}$$

This equation is a special case of Eq. (6.1) of [1] and so we may use algorithm PADMIS of [1] to obtain the s_i's. The finish time, C^*, of a minimum finish time schedule for the n jobs is $s_n + 1$.

For simplicity, let us assume that $r_1 = 0$. Once C^* has been computed, a schedule that has minimum cost may be obtained by constructing the convex bipartite graph, G, with $A = \{1, 2, \ldots, n\}$ and $B = \{0, 1, \ldots, C^* - 1\}$. There is an edge from i in A to j in B iff $j \geq r_i$. (A represents the jobs and $j \in B$ represents the time slot $[j, j + 1]$.) A cost $c_i(j)$ is associated with edge (i, j).

From our choice of C^*, it follows that the graph just constructed has a maximum matching M of size n. Every such matching defines a feasible schedule. A matching for which the maximum edge cost is minimum defines the optimal matching for our scheduling problem.

As suggested by Rinnooy Kan [13], this matching can be obtained by performing a binary search on the $O(n^2)$ edge costs in G. At each iteration of this binary search, all edges with cost greater than c are deleted from G and a maximum matching in the resulting convex bipartite graph G' found. If this matching is of size less than n, then there is no matching in G with maximum edge cost no more than c; otherwise there is. The time complexity of Rinnooy Kan's scheme is readily seen to be $O(nA(n) \log n)$.

Complexity Analysis

The initial sort by release times can be done in $O(\log^2 n)$ time using $O(n)$ PEs [12]. C^* can be found in $O(\log n)$ time using n PEs and the algorithm of [1]. Each iteration of the binary search requires the construction of G' and the solution of a matching problem. The first of these can be done in $O(\log^2 n)$ time using $O(n^2/\log^2 n)$ PEs. The second task takes $O(\log^2 n)$ time and $O(n)$

PEs. The total time needed to determine the matching with minimum cost is therefore $O(\log^3 n)$ and the number of PEs needed is $O(n^2/\log^2 n)$. Hence, the overall complexity of the parallel scheduling algorithm is $O(\log^3 n)$. The number of PEs used is $O(n^2/\log^2 n)$. The EPU of our parallel algorithm is $\Omega(nA(n) \log n/(\log^3 n * n^2/\log^2 n)) = \Omega(A(n)/n)$.

3.2. *Job Sequencing with Release Times and Deadlines*

Once again, each job has a processing requirement that is one unit. However, this time there is a release time r_i, deadline d_i, and weight w_i associated with job i. The r_i's and d_i's are natural numbers. No job may be scheduled before its release time. We are interested in obtaining a maximum weight subset of the n jobs such that all jobs in this subset can be scheduled on a single machine in such a manner that none starts before its release time or finishes after its deadline.

This problem can be solved efficiently by a sequential greedy algorithm. Jobs are considered in nonincreasing order of the weights. If the job, i, currently being considered can be scheduled together with all the others so far scheduled such that no job violates its release time or deadline, then job i is in the optimal schedule. Otherwise, it is not. The sequential complexity of this algorithm is $O(n^2)$ [7].

For the parallel algorithm, we resort to a convex bipartite graph formulation. The A set represents the n jobs; so, $|A| = n$. The B set represents time slots. Let $a = \min \{r_i\}$ and $b = \max \{d_i\}$. $B = \{a, a + 1, \ldots, b - 1\}$. In the convex bipartite graph corresponding to the scheduling problem, there is an edge from i in A to $j \in B$ iff $r_i \leq j < d_i$. The weight of each edge incident on vertex $i \in A$ is w_i. We wish to find a matching M for which the sum of edge weights is maximum.

Our parallel algorithm for this is based on the greedy sequential algorithm for job sequencing with release times and deadlines. Define the binary relation R on the weights w_i as follows:

$$w_i R w_j \quad \text{iff} \quad w_i < w_j \quad \text{or} \quad (w_i = w_j \text{ and } i > j).$$

Let G_i be the convex bipartite graph obtained from G by deleting all edges with weight w_j such that $w_j \, R \, w_i$. Clearly, the following determines whether or not i is in the maximum weight matching:

(a) Determine the size of the maximum cardinality matching in G_i. Let this be s_i.

(b) Delete all edges incident on vertex $i \in A$ of G_i. Determine the size of the maximum cardinality matching in the new graph. Let this be s'_i.

(c) i is in the maximum weight matching iff $s_i = s'_i + 1$.

So, each vertex can determine, in parallel, whether or not it is in the maximum weight matching. To avoid read conflicts, we need to make n copies of the graph G. This can be done in $O(\log n)$ time using $O(n^2)$ PEs.

To see this, observe that as G is a convex bipartite graph, it can be represented in $O(n)$ space. So, n PEs can make one copy of G in $O(n)$ time. Next, $2n$ PEs are used to make a copy of each of these copies. Now we have four copies of G. At step $\log n$, $n^2/2$ PEs make one copy of each of the existing $n/2$ copies in $O(1)$ time. G_i can be constructed in $O(1)$ time using $O(n)$ PEs per vertex i. Then, steps (a), (b), and (c) above can be performed in $O(\log^2 n)$ time using a total to $O(n^2)$ PEs. (Note that only pass 1 of the parallel convex matching algorithm need be executed.) The total time needed to determine the subset of A in the maximum weight matching is therefore $O(\log^2 n)$.

Once we have obtained the above subset of A, the actual matching can be determined by deleting from G all vertices that are not in this subset. A maximum cardinality matching on the resulting graph yields the desired matching with maximum weight. This corresponds to an optimal schedule for the scheduling problem. It takes an additional $O(\log^2 n)$ time to determine this matching.

The complexity of the parallel algorithm is $O(\log^2 n)$ and its EPU is $\Omega(n^2/(\log^2 n * n^2)) = \Omega(1/\log^2 n)$.

EXAMPLE 3.1. Figure 3.1 gives an example of a job set. Let $H_i = \{j \mid iRj\}$. Figure 3.2 gives the $H_i s$, $s_i s$, and s_i''s, and s's i's for our example.

3.3. *Preemptive Scheduling with Precedence Constraints*

Let J be a set of n unit time jobs. Let P be a precedence relation on J. For $i \in J$ and $j \in J$, iPj iff i must be completed before j can commence. We may assume that P is a partial order. Hence, P may be represented as a directed acyclic graph (dag) as in Fig. 3.3 (strictly speaking, P is the transitive closure of this graph). The directed edge $\langle i, j \rangle$ means that i must be completed before j can start (i.e., iPj). We also assume that jobs have been indexed so that iPj implies $i < j$. This is true of the indexing (of nodes) in Fig. 3.3.

Muntz and Coffman [10] have developed a level algorithm to obtain minimum finish time preemptive schedules for J (as above) when the number of machines available is 2. Their algorithm also works for the case when the processing times are mutually commensurable (rather than simply 1 unit). A set of times $\{t_1, t_2, \ldots, t_n\}$ is *mutually commensurable* iff each is a multiple

i	1	2	3	4	5	6	7	8	9	10	11
r_i	1	4	5	1	3	0	4	2	6	2	4
d_i	3	6	6	3	6	1	6	6	7	3	7
w_i	50	55	65	40	70	20	60	80	60	30	85

FIGURE 3.1

$H_1 = \{2,3,5,7,8,9,11\}$

$s_i = 6$ $s_i' = 5$ Job 1 is in

$H_2 = \{3,5,7,8,9,11\}$

$s_i = 5$ $s_i' = 5$ Job 2 is out

$H_3 = \{5,8,11\}$

$s_i = 4$ $s_i' = 3$ Job 3 is in

$H_4 = \{1,2,3,5,7,8,9,11\}$

$s_i = 6$ $s_i' = 6$ Job 4 is out

$H_5 = \{8,11\}$

$s_i = 3$ $s_i' = 2$ Job 5 is in

$H_6 = \{1,2,3,4,5,7,8,9,10,11\}$

$s_i = 7$ $s_i' = 6$ Job 6 is in

$H_7 = \{3,5,8,11\}$

$s_i = 5$ $s_i' = 4$ Job 7 is in

$H_8 = \{11\}$

$s_i = 2$ $s_i' = 1$ Job 8 is in

$H_9 = \{3,5,7,8,11\}$

$s_i = 5$ $s_i' = 5$ Job 9 is out

$H_{10} = \{1,2,3,4,5,7,8,9,11\}$

$s_i = 6$ $s_i' = 6$ Job 10 is out

$H_{11} = \{\emptyset\}$

$s_i = 1$ $s_i' = 0$ Job 11 is in

FIGURE 3.2

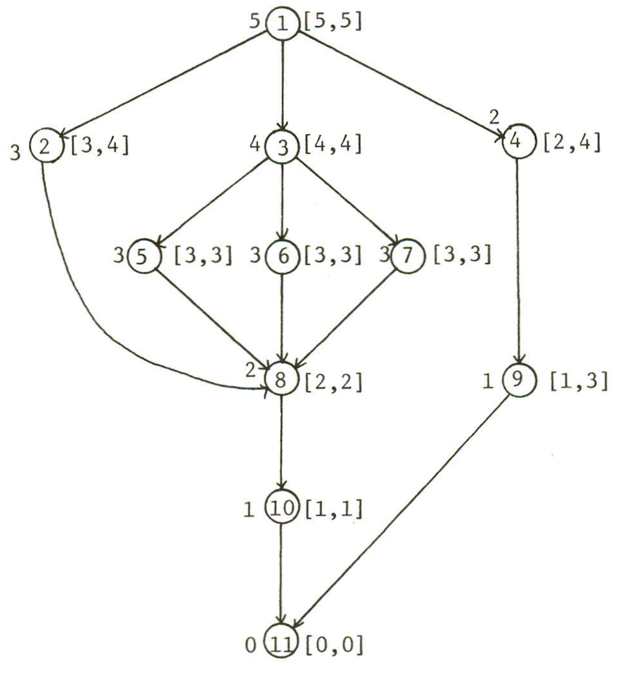


FIGURE 3.3

```
procedure SCH
  $L \leftarrow$ maximum level
  $S_i \leftarrow$ all jobs in $J$ with level $= i,\ 0 \le i \le L$
  for $k \leftarrow L$ to 1 do
    if $|S_k| = 1$ then find the largest $j$ such that $j < k$ and $S_j$ contains a job all of whose
                        predecessors are in $S_L, S_{L-1}, \ldots, S_{k+1}$.
                        If no such $j$ exists then exit endif.
                        Let $q$ be this value of $j$ and let $r$ be the job in $S_q$ with the above property
                        $S_q \leftarrow S_q - \{r\};\ S_k \leftarrow S_k \cup \{r\}$
    endif
  endfor
  preemptively schedule the jobs in each $S_i$ using McNaughton's rule [9].
  Jobs in $S_i$ precede those in $S_{i-1}$.
end SCH
```

FIG. 3.4. The Muntz–Coffman Algorithm.

of some number w. In the case of mutually commensurable times, each job is broken into a chain of jobs each requiring w units of processing. In this section, we shall deal directly only with the case of unit time tasks.

Let G be the dag representation of the precedence relation P. A node in G is *terminal* iff its out-degree is 0. The *level* of a node in G is the length of the longest path from v to any terminal node. All terminal nodes have level 0. In Fig. 3.3, the number outside each node is its level.

Muntz and Coffman's [10] algorithm to obtain minimum finish time two-machine schedules is given in Fig. 3.4. One observes that the objective of the **for** loop of this algorithm is to minimize the number of sets S_i with $|S_i| = 1$. This can be done in parallel as follows. Let L be the maximum level in the precedence graph G. Let L_i be the level of vertex i in G. Let D_i be the length of the longest path ending at i. Let $H_i = L - D_i$. The $[L_i, H_i]$ pairs of each vertex in our example are given in Fig. 3.3. Note that H_i is the largest j such that job i could possibly be in S_j following the execution of the **for** loop in the Muntz–Coffman algorithm.

Set	S_5	S_4	S_3				S_2		S_1		S_0
i	1	3	5	6	7	2	8	4	10	9	11
L_i	5	4	3	3	3	3	2	2	1	1	0
H_i	5	4	3	3	3	4	2	4	1	3	0
non critical						X		X		X	
candidate set	X	X					X		X		X

FIGURE 3.5

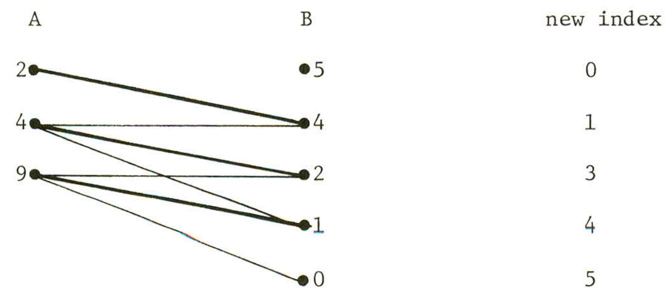


FIG. 3.6. Convex bipartite graph.

A job i is said to be *critical* iff $L_i = H_i$. Only noncritical jobs are candidates for displacement to other sets. Also, only those sets S_i that initially have exactly one critical job are candiates to receive a new job during the **for** loop. Note that every S_i must have at least one critical job. In Fig. 3.5, the noncritical jobs have been marked with an X. Sets that are candidates to receive a new job have also been marked with an X. Double vertical lines mark set boundaries.

To determine the movement of noncritical jobs to candidate sets, we construct a convex bipartite graph in which the A set consists of the noncritical jobs while the B set consists of the candidate sets. There is an edge from $i \in A$ to $j \in B$ iff $L_i \leq j \leq H_i$. For our example, the resulting graph is shown in Fig. 3.6. We intend to use a maximum matching M in the graph constructed above as follows: if (i, j) is an edge of the matching M, then job i is moved to set S_j; vertices of A that are not matched to any vertex in B remain in their original set.

The maximum matching M is to be obtained from the convex bipartite graph constructed above as follows:

(1) Reindex the vertices in B. The new index of vertex $i \in B$ is $L - i$. So, for $i \in A$, $s_i = L - H_i$ and $h_i = L - L_i$.

(2) Use Glover's algorithm considering vertices in B in increasing order of index.

Let $S'_L, S'_{L-1}, \ldots, S'_0$ be the sets obtained by updating $S_L, S_{L-1}, \ldots, S_0$ using the maximum matching above. These updated sets satisfy the following properties:

(a) There is no job pair (i, j) such that iPj and $i \in S'_a$, $j \in S'_b$ for $a \leq b$.

(b) There is no partition of the job set J that satisfies property (a) and has a smaller number of sets of size 1 than in $S'_L, \ldots, S'_0$.

For our example, the maximum matching M is $\{(2, 4), (4, 2), (9, 1)\}$ (solid lines in Fig. 3.7). So, $S'_i = S_i$, $i = 0, 1, 2, 5$. $S'_3 = \{5, 6, 7\}$; and $S'_4 = \{3.2\}$. Properties (a) and (b) hold for our example.

The correctness of (a) follows from the indexing scheme imposed on the jobs (viz., iPj implies $i < j$) and the unique way in which Glover's algorithm

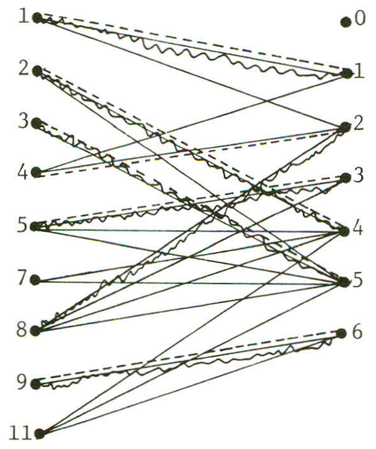


FIGURE 3.7

works. Suppose that statement (a) is incorrect. Then, there exists a job pair (i, j) such that iPj and $i \in S'_a$, $j \in S'_b$ and $a \leq b$. Since no such job pair exists in $S_L, \ldots, S_0$, it must be the case that job j is noncritical and is matched to b (old indexing) in M (as only noncritical jobs can change sets and no job moves to a set of smaller index). From the definition of L and H and the knowledge that iPj, it follows that $H_i \geq L_i$, $H_j \geq L_j$, $L_i > L_j$, and $H_i > H_j$. If job i is critical, then it is the case that $L_i > H_j$. So, job j cannot possibly be matched to a set b with $A \leq b$. Hence, we may assume that i is also noncritical. Again, if i is not matched in M, then $a > b$ as for a to be less than or equal to b, it must be the case that $L_i < H_j$. But since $L_i > L_j$, $h_i < h_j$ and i must get matched to b before j can (see Glover's alogrithm). Hence, i and j must both be matched in M. i is matched to a and j to b. Hence, $a \neq b$. If $a < b$ then vertex b is considered before a (as its new index is $L-b < L-a$). Since iPb, it must be the case that $H_i \geq b \geq L_i$. Also, $L_i > L_j$ implies that $h_i < h_j$. Hence, Glover's algorithm will match i to b and not j to b.

The truth of property (b) is established using the following reasoning. If $i \in B$ is such that no vertex of A is matched to i in M, then $|S'_i| = 1$. To see this, observe that S_i has exactly one critical job. Every noncritical job in S_i is a candidate for matching with $i \in B$. Thus, if no job is matched with i, then all these noncritical jobs are matched elsewhere and $|S'_i| = 1$. It is readily seen that if $i \in B$ is matched in M, then $|S'_i| > 1$. Hence, the number of sets of size 1 is minimized when a maximum matching is used.

Complexity Analysis

The steps in our parallel preemptive scheduling algorithm are:

(1) Determine L_i and H_i, $1 \leq i \leq n$

(2) Mark the noncritical jobs

(3) Mark the candidate sets
(4) Construct the convex bipartite graph
(5) Find the maximum matching M
(6) Reassign matched jobs to their new sets
(7) Schedule jobs in the same set using McNaughton's rule.

Step 1 $\leq$ can be performed in $O(\log^2 n)$ time using the critical path algorithm developed by Dekel, Nassimi, and Sahni [3]. This algorithm uses $O(n^3/\log n)$ PEs. The noncritical jobs can be identified in $O(1)$ time using n PEs. The sets $S_0, \ldots, S_L$ may be obtained in $O(\log n)$ time by sorting on L_i. This requires $O(n^2)$ PEs. The candidate sets can be marked in $O(\log n)$ time and the resulting convex bipartite graph may be constructed in an additional $O(1)$ time using n^2 PEs. The matching requires $O(\log^2 n)$ time and $O(n)$ PEs while the reassigning takes $O(1)$ time. A parallel implementation of McNaughton's rule appears in [1]. This has complexity $O(\log n)$ and uses $O(n/\log n)$ PEs.

Hence, the overall complexity of our seven-step parallel algorithm for preemptive scheduling is $O(\log^2 n)$. The number of PEs used is $O(n^3/\log n)$. Since the fastest sequential algorithm known for this problem (i.e., the Muntz–Coffman algorithm) has complexity $O(n^2)$, the EPU of our parallel algorithm is $O(n^2/(\log^2 n * n^3/\log n)) = O(1/(n \log n))$.

4. Conclusions

We have developed fast parallel algorithms for several versions of the matching problem for convex bipartite graphs. In Section 2, we explicitly considered the maximum cardinality matching problem. In Sections 3.1 and 3.2, we considered the problems of obtaining a maximum matching that minimized the maximum weight edge used as well as one that maximized the total weight of the used edges. Both these problems were discussed in connection with the scheduling problems in which they naturally arose. Finally, the maximum cardinality matching algorithm was again used to solve the preemptive scheduling problem of Section 3.3 efficiently.

This paper has further enhanced the utility of the binary tree method of Dekel and Sahni [2] for the design of parallel algorithms. It should also be pointed out that while all of our complexity analyses have assumed the availability of as many PEs as needed, our algorithms can be used when fewer PEs are available. The complexity of each algorithm will increase by no more than the shortfall in PEs. So if only half the number of PEs is available, then the time needed will at most double (except for a possible constant increase in overhead).

References

1. Dekel, E., and Sahni, S. Parallel scheduling algorithms. *Oper. Res.* **31,** 1 (1983), 24–49.
2. Dekel, E., and Sahni, S. Binary trees and parallel scheduling algorithms. *IEEE Trans. Comput.* **C-32,** 3 (March 1983), 307–315.

3. Dekel, E., Nassimi, D., and Sahni, S. Parallel matrix and graph algorithms. *SICOMP* **10,** 4 (Nov. 1981), 657–675.

4. Emde Boas, P. van. Preserving order in a forest in less than logarithmic time and linear space. *Inform. Process. Lett.* **6** (1977), 80–82.

5. Glover, F. Maximum matching in a convex bipartite graph. *Naval Res. Logist. Quart.* **14** (1967), 313–316.

6. Jackson, J. R. Scheduling a production line to minimize maximum tardiness. Research Rep. 43, Management Science Research Project, University of California, Los Angeles, 1955.

7. Lageweg, B. J., and Lawler, E. L. Private communication. Cited in Lenstra, J. K. *Sequencing by Enumerative Methods*. Mathematisch Centrum, Amsterdam, 1976, p. 22.

8. Lipski, W., Jr., and Preparata, F. P. Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems. *Acta Inform.* **15** (1981), 329–346.

9. McNaughton, R. Sequencing with deadlines and loss functions. *Management Sci.* **6** (1959), 1–12.

10. Muntz, R. R., and Coffman, E. G., Jr. Optimal preemtive scheduling on two-processor systems. *IEEE Trans. Comput.* **C-18** (1969) 1014–1020.

11. Nassimi, D., and Sahni S. Bitonic sort on a mesh connected parallel computer. *IEEE Trans Comput.* **C-28,** 1 (Jan. 1979), 2–7.

12. Preparata, F. P. New parallel-sorting schemes. *IEEE Trans. Comput.* **C-27,** 7 (July 1978), 669–673.

13. Rinnooy Kan, A. H. G. *Machine scheduling problems, classification complexity, and computation*. Nijhoff, The Hague, 1976.

14. Savage, C., Parallel algorithms for graph theoretic problems. Ph.D. thesis, University of Illinois, Urbana, Aug. 1978.