PARALLEL MATCHING ALGORITHM FOR CONVEX BIPARTITE GRAPHS AND A...
A Parallel Matching Algorithm
for Convex Bipartite Graphs and
Applications to Scheduling
by
Eliezer Dekel and Sartaj Sahni
Technical Report 81-3
April 1981

Cover design courtesy of Ruth and Jay Leavitt.
A Parallel Matching Algorithm for Convex Bipartite Graphs and Applications to Scheduling*
Eliezer Dekel and Sartaj Sahni
University of Minnesota

Abstract

An efficient parallel algorithm to obtain maximum matchings in convex bipartite graphs is obtained. 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.

Key words and phrases: Parallel algorithm, convex bipartite graph, scheduling, complexity.

This research was supported in part by the Office of Naval Research under contract N00014-80-C-3650.
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 $(a_i, b_{j+q}) \in E$, then $(a_i, b_{j+q}) \in E$, $1 \leq q < k$.

Property (1) above is the bipartite property while property (2) is the convexity property. An example convex bipartite graph is shown in Figure 1.1.

![A convex bipartite graph](image)

**Figure 1.1** A convex bipartite graph.

$F \subseteq E$ is a matching in the convex bipartite graph $G=(A, B, E)$ iff no two edges in $F$ have a common endpoint. $F_1=\{(a_1, b_1), (a_4, b_3), (a_5, b_1)\}$ is a matching in the graph of Figure 1.1 while $F_2=\{(a_1, b_1), (a_4, b_3), (a_5, b_1)\}$ 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 $|H| > |F|$ (where $|H|$ is the number of edges in $H$). The matching depicted by solid lines in Figure 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 $G=(A, B, E)$, $A=\{a_1, a_2, \ldots, a_n\}$, $B=\{b_1, \ldots, b_m\}$ is uniquely represented by the set of triples:

$$T = \{(i, s_i, t_i) \mid 1 \leq i \leq n\}$$

$$s_i = \min \{j \mid (a_i, b_j) \in E\}$$
Intuitively, 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 Figure 1.1, we have T = \{(1,1,1), (2,3,3), (3,3,3), (4,3,3), (5,1,1)\}.

As an example of the use of matchings in convex bipartite graphs, consider the problem of scheduling 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. 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 \tilde{F} of J such that |\tilde{F}| > |F|.

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 all the r_i’s and d_i’s are natural numbers; min(r_i) = 0; r_i < d_i, 1 ≤ i ≤ n; and max(d_i) ≤ 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 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 ≤ 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.

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

Glover [4] 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 ≤ i ≤ |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_1 currently being considered is connected. Let q be such that a_q \in R and h_q = min\{h_i \mid a_q \in R\}. Vertex b_1 is matched to a_q and a deleted from the graph. The next vertex in B is now considered. Figure 1.3 specifies Glover’s algorithm more formally. By using appropriate data structures, Glover’s algorithm can be implemented to run in O(mn) time.
Figure 1.2

### procedure MATCH(G, m, n, h)

// G=(A, B, E) is a convex bipartite graph. |A|=n, |B|=m //
// n = max{ |(a_i, b_j) ∈ E| } A maximum matching //
// m is found //

set M; integer m, n, h; graph G
M ← ∅

for i ← 1 to m do
    let R be the set of remaining vertices to which b_i is connected
    if R ≠ ∅ then q ← j such that j∈R and n_j = min_{p∈R} (n_p)
        M ← M ∪ {(a_q, b_i)}
        delete a_q from G
    endif
end for

return ()
end MATCH

Figure 1.3 Glover's maximum matching algorithm.

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.

The parallel computer model used is the shared memory model (SM4). This is an example of a single instruction stream multiple data stream (SIMD) computer. In a SM4 computer, there are p processing elements (PEs). Each PE is capable of performing the standard arithmetic and logical
operations. The PEs are indexed $1, 2, \ldots, p-1$ 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 $T=\{(i, s_i, h_i) | 1 \leq i \leq n\}$ $s_i = c, 1 \leq i \leq n$ 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 Figure 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
Figure 2.1
result of this reordering is shown in Figure 2.2.

<table>
<thead>
<tr>
<th>1</th>
<th>7</th>
<th>3</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>1</th>
<th>5</th>
<th>12</th>
<th>9</th>
<th>8</th>
<th>11</th>
<th>14</th>
<th>10</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>h_1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>10</td>
<td>11</td>
<td>10</td>
<td>11</td>
<td>13</td>
<td>12</td>
<td>13</td>
<td>13</td>
</tr>
</tbody>
</table>

Figure 2.2
Following the sort, we identify the distinct $s_i$ values. Let these be $R_1, R_2, \ldots, R_k$. Assume that $R_1 < R_2 < \cdots < R_k$. Let $R_{k+1} = \max(d_i) + 1$. 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 2.3(a). 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 $[u,v].P$ or simply $[u,v]$.

![Figure 2.3: Complete binary trees.](image)

Let the leaf nodes of the computation tree be numbered 1 through $k$, left to right. If $P$ is the $i$th leaf node, then $[u,v].P$ is $[R_i, R_{i+1}]$ (i.e., $u=R_i$ and $v=R_{i+1}$). If $P$ is not a leaf node, then the subset of $B$ associated with $P$ is $[u,v].LC(P) \cup [u,v].RC(P)$ 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 Figure 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:
Figure 2.4

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

For example,

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

etc.

The subset $M(P)$ of $A$ vertices available for matching at $P$ may be partitioned into three subsets $Q(P)$, $I(P)$, and $T(P)$. $Q(P)$ is a maximum cardinality subset of $M(P)$ that may be matched with vertices in $[u, v]$. By algorithm MATCH, this subset is called the matched set. $I(P)$ denotes the infeasible set. It consists of all vertices $i \notin M(P) - Q(P)$ such that $h_i \leq v$. The transferred set $T(P)$ consists of all vertices $i \notin M(P) - Q(P)$ such that $h_i > v$.

Consider node 2 of Figure 2.4. The matching problem defined at this node is given in Figure 2.5. 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, $Q(2) = [1, 2, 3, 4, 5, 7, 12]$; $I(2) = \emptyset$; and $T(2) = \emptyset$. Observe that $|Q(1)|$ is the size of a maximum matching in the original convex bipartite graph. Also, $T(1) = \emptyset$ and $I(1) = A - Q(1)$.

We shall make two passes over the computation tree. The first pass begins at the leaves and moves towards the root. During this pass, the $Q$, $I$, and $T$ sets for each node are computed. The second pass starts at the root and progresses towards the leaves. In this pass, the $Q$ 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 the 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 Figure 2.6. For convenience, this has been translated into the graph language used here.

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

Figure 2.6
An examination of Glover's algorithm (Figure 1.3) reveals that it performs exactly as does procedure FEAS when the restrictions and simplifications applicable to FEAS are incorporated into Figure 1.3.

Hence, for a leaf node of the computation tree, the Q set may be obtained by a direct application of procedure FEAS (or its parallel equivalent). For example, for node 4 of Figure 2.4, we have \( A = M(4) = \{2, 3, 4, 7\} \); \( h_2 = 4 \); \( h_3 = 3 \); \( n_4 = 4 \); \( h_4 = 2 \); \( u = 1 \); and \( v = 4 \). Using FEAS, we obtain \( Q(4) = \{7, 3, 2\} \). Note that \( Q \) consists of exactly those vertices \( i \) with \( \text{MAT}(i) = 1 \). \( I() \) consists of exactly those vertices \( i \) with \( \text{MAT}(i) = \emptyset \) and \( h_i < v \). The remaining vertices form \( T() \). The matched set \( Q \), transferred set \( T \), and infeasible set \( I \) for each of the leaves in our example are shown in Figure 2.6. Null sets are not explicitly shown. So, for node 4, \( I(4) = \emptyset \); \( T(4) = \{4\} \); and \( Q(4) = \{7, 3, 2\} \). The sets are ordered by \( h_i \).

For a nonleaf node \( P \), the \( Q \), \( I \), and \( T \) sets may be obtained by using the \( Q \), \( I \), and \( T \) sets of the children of \( P \). Let \( L \) and \( R \), respectively, be the left and right children of \( P \). To determine \( Q(P) \), we 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 \( T(L) \cup Q(R) \). Since both \( T(L) \) and \( Q(R) \) 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 \( T(L) \cup Q(R) \) that has \( \text{MAT}() = 1 \) following the execution of FEAS. The following theorem establishes that \( Q(L) \cup S \) is a maximum cardinality subset of \( M(P) \) that may be matched with vertices in \( [u,v].P \). Hence, \( Q(P) = Q(L) \cup S \).

Theorem 2.1: \( S \) as defined above is a maximum cardinality subset of \( M(P) \) that may be matched with vertices in \( [u,v].P \) using algorithm MATCH.

Proof: The proof is by induction on the distance of \( P \) from the farthest leaf in the subtree of which it is a root. If this distance is 1, then \( Q(L) \) and \( Q(R) \) are maximum cardinality subsets of \( M(L) \) and \( M(R) \) that can, respectively, be matched by Glover's algorithm with vertices in \( [u,v].L \) and \( [u,v].R \). If this distance is greater than 1, then \( Q(L) \) and \( Q(R) \) 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 \( v_R \)). Furthermore, when Glover's algorithm is used with the \( A \) set being \( M(P) \) and the \( B \) set being \( [u,v].P \), vertices in \( B \) are considered in the order \( u_L, u_L+1, \ldots, v_L, u_R, \ldots, v_R \). Hence, \( Q(L) \) 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 \( T(L) \cup Q(R) \). 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 \( Q(R) \) 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 matched to a vertex in \([u_R,v_R] \) in a such a way that no vertex \( j \) in \( S \) is matched to a vertex with index greater than \( h_j \). Consider a matching \( W \) that

---

**Figure 2.7:** Results of first pass.

<table>
<thead>
<tr>
<th></th>
<th>Q</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>7</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>S</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>h</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Q</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>9</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>h</td>
<td>10</td>
<td>11</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Q</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>1</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>S</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>h</td>
<td>12</td>
<td>13</td>
<td>13</td>
</tr>
</tbody>
</table>

---

- \( [1,3] \) - \( [4,6] \) - \( [9,11] \) - \( [12,18] \)
meets this condition. Now suppose that some vertex $j$ in $S$ is matched to a vertex $q$ in $[u_R, v_R]$ with index less than $s_j$. Clearly, $j$ must be a member of $Q(R)$ (as all vertices in $T(Z)$ have an $s$ value less than $u_R$). Suppose that $j$ is matched to $j_1$ in $Z$. If $j_1$ is free in $W$, then the matching of $j$ in $W$ may be changed from $q$ to $j_1$. If $j_1$ is not free, then suppose it is matched to $j_2$. If $j_2 \in T(L)$, then $j_2$ may be matched to $q$ and $j$ to $j_1$ (note that $q < j_1$ and so $s_q < s_j$).

If $j_2 \in Q(R)$, then suppose that $j_2$ is matched to $j_3$ in $Z$. It is easy to see that $j_2 \neq j_3$. If $q \neq j_3$ or $j_3$ is free in $W$, then we may match $j$ to $j_1$ and $j_2$ to $q$. Otherwise, let $j_4$ be the vertex matched to $j_3$ in $W$. It should be clear that we can continue in this way and modify $W$ so that $j$ is matched to $j_1$, $j_2$ to $j_3$, $j_4$ to $j_1$, etc. In the new matching, there is one fewer vertex of $S$ that is matched to a vertex with 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 $q$ such that $u_R < s_j < q < h_j < v_R$. Hence, $S$ represents a feasible matching.

Let $S'$ (as defined earlier) be the subset of $Q(R) \cup T(L)$ 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 $Q(R)$ into $[u_R, v_R]$. Let $Y$ be a matching of $S'$ in which all vertices in $Q(R) \cap S'$ 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 $S' \neq S$. Also, from the working of FEAS, it follows that $S \neq S'$. Let $j \in S$ be a vertex with least $h$, such that $j \not\in S'$. 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 $W$, $p \not\in S$. Also, from the order in which FEAS considers vertices, $h > n$. Hence, $S' \cup \{j\} \setminus \{p\}$ is also a subset selectable by Glover’s algorithm. $S' \cup \{j\} \setminus \{p\}$ 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 $Q(R) \cup T(L)$ that is selectable by Glover’s algorithm for matching in $[u_R, v_R]$.

Hence, $Q(L) \cup S$ is a maximum cardinality subset of $M(P)$ selectable by Glover’s algorithm for matching in $[u, v]$.
Once $Q(P)$ is known, $T(P)$ and $I(P)$ are easily computed. Actually, as $I(P)$ is never used, we may omit its computation. Figure 2.6 shows the $Q$, $I$, 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 $Q'(P)$ which represents the set of all vertices matched by Glover’s algorithm, with the set $[u,v]_P$. With respect to the matching shown by solid lines in Figure 2.1, we see that if $P$ is the root, then $Q'(P) = [1, 2, 3, 4, 5, 7, 8, 9, 13, 11, 12, 14]$; if $P$ is node 3 of the computation tree, then $Q'(P) = [3, 9, 10, 11, 14]$.

If $P$ is the root node, then $Q'(P) = Q(P)$, by definition of $Q(P)$. Let $P$ be any nonleaf node for which $Q'(P)$ has been computed. Let $L$ and $R$ be the left and right children, respectively, of $P$. Let $[u,v]_L = [u_L,v_L]$ and $[u,v]_R = [u_R,v_R]$. Let $V = \{j\} \cup Q'(P)$ and $h_i < u_i$. Let $w$ be the ordered set obtained by merging together $V$ and $Q(L)$ (note that both $V$ and $Q(L)$ can be maintained so that they are in nondecreasing order of $h_i$ and that $w$ is also in nondecreasing order of $h_i$). $Q'(L)$ consists of the first $\min \{|w|, v-u+1\}$ vertices in $w$. The correctness of this statement may be established by induction on the distance of $P$ from the root. $Q'(R)$ is readily seen to be $Q'(P) - Q'(L)$. Figure 2.7, shows the $Q'()$ sets for all the vertices in the computation tree of our example.

From the $Q'(P)$ sets of the leaves, the matching is easily obtained. If $P$ is a leaf, and $[u,v]_P = [a,b]$, then the first vertex in $Q'(P)$ is matched with $a$, the second with $a+1$, etc. (note that $Q'(P)$ is in nondecreasing order of $h_i$). The matching for our example is also given in Figure 2.7.

Complexity Analysis

The initial ordering of $A$ by $h_i$ and within $h_i$ by $S_i$ can be done in $O(\log^2 n)$ time using $\sqrt{n}/2$ PEs ([6] and [7]). During the first pass, the computation of $Q(P)$ requires the use of FEAS and a merge. The use of FEAS (without the sort) takes $O(\log n)$ time and requires $O(M(P)/\log^2 M(P))$ PEs. The merge at node $P$ takes $O(\log n)$ time with $M(P)/2$ PEs. Since, $Q$ 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.
Figure 2.3 Second pass

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], and [9]). For any problem $P$ and parallel algorithm $A$, this is defined as follows:

$$EPU(P,A) = \frac{\text{complexity of fastest sequential algorithm for } P}{\text{complexity of } A} \times \frac{1}{\text{number of PEs used by } A}$$

For our algorithm, we have an EPU that is $O(nm/(\log^2 n \cdot n)) = O(m/\log^2 n)$ (recall that $m = |B|$).
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 \( J=\{ (r_i, d_i) \mid 1 \leq i \leq n \} \) 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 sections.

### 4.1 Scheduling to Minimize the Maximum Cost

In this problem, we are given \( n \) unit time jobs. Associated with each job is a release time \( r_i \) 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:

\[
q_i = \begin{cases} 
  c_i(s_i) & s_i \leq r_i \\
  \infty & s_i > r_i 
\end{cases}
\]

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 [8], 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:

\[
(4.1) \quad s_i = r_i \\
  s_i = \min( r_i, s_{i-1} + 1 )
\]

This equation is a special case of Equation (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 \( B=\{1, 2, \ldots, C^*-1\} \). 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 [8], 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.

**Complexity Analysis**

The initial sort by release times can be done in $O(\log^2 n)$ time using $O(n)$ PEs [7]. $C^*$ 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 $O(\log n)$ time and $O(n)$ PEs. The total time needed to determine the matching with minimum cost is therefore $O(\log n)$ and the number of PEs needed is $O(n/\log n)$. Hence, the overall complexity of the parallel scheduling algorithm is $O(\log n)$. The number of PEs used is $O(n/\log n)$. The complexity of the fastest sequential algorithm known for this problem (i.e., the algorithm described above) is $O(n^2 \log n)$. Hence, the EPU of our parallel algorithm is $O(n \log n/(\log n^2 n^2/\log^2 n))=O(1).

### 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 $i$. 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 nondecreasing 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. This feasibility test can be carried out in $O(n)$
time. So, the sequential algorithm has complexity $O(n^2)$.

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 $r_i < j < d_i$. The weight of each edge incident on vertex $i \in A$ is $w_i$. We wish to find a matching $A$ 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:

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

Let $G_i$ be the convex bipartite graph obtained from $G$ by deleting all edges with weight $w_i$ such that $w_i \; R \; w_j$. 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. $G_i$ can be constructed in $O(1)$ time using $O(n^2)$ PEs per vertex $i$. Then, steps a), b), and c) above can be performed in $O(\log^2 n)$ time using a total of $O(n^2)$ 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 $O(\log^3 n)$. The number of PEs can easily be reduced to $O(n^2/\log n)$.

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.

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

Example 3.1: Figure 3.1 gives an example job set. Let
Figure 3.2 gives the $H_i$'s for our example. The convex graph $G_i$ is shown in Figure 3.3. The maximum matching in $G_i$ is shown by broken lines. The maximum matching in the graph obtained from $G_i$ by deleting edges incident on vertex 4 is shown by wavy lines. These two matchings are of the same cardinality. Hence, vertex 4 is not in the maximum weight matching.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_i$</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$d_i$</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>$w_i$</td>
<td>50</td>
<td>55</td>
<td>65</td>
<td>40</td>
<td>70</td>
<td>20</td>
<td>60</td>
<td>80</td>
<td>60</td>
<td>30</td>
<td>85</td>
</tr>
</tbody>
</table>

Figure 3.1

### 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 Figure 3.4 (strictly speaking, $P$ is the transitive closure of this graph). The directed edge $(i,j)$ 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 Figure 3.4.

Muntz and Coffman [5] 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 $\{t_1, t_2, ..., t_n\}$ is mutually commensurable iff each is a multiple 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.
H_1 = \{2,3,5,7,8,9,11\} \\
s_1 = 6 \quad s_1' = 5 \quad \text{Job 1 is in}

H_2 = \{3,5,7,8,9,11\} \\
s_1 = 5 \quad s_1' = 5 \quad \text{Job 2 is out}

H_3 = \{5,8,11\} \\
s_1 = 4 \quad s_1' = 3 \quad \text{Job 3 is in}

H_4 = \{1,2,3,5,7,8,9,11\} \\
s_1 = 0 \quad s_1' = 6 \quad \text{Job 4 is out}

H_5 = \{8,11\} \\
s_1 = 3 \quad s_1' = 2 \quad \text{Job 5 is in}

H_6 = \{1,2,3,4,5,7,8,9,10,11\} \\
s_1 = 7 \quad s_1' = 6 \quad \text{Job 6 is in}

H_7 = \{3,5,8,11\} \\
s_1 = 5 \quad s_1' = 4 \quad \text{Job 7 is in}

H_8 = \{11\} \\
s_1 = 2 \quad s_1' = 1 \quad \text{Job 8 is in}

H_9 = \{3,5,7,8,11\} \\
s_1 = 5 \quad s_1' = 5 \quad \text{Job 9 is out}

H_{10} = \{1,2,3,4,5,7,8,9,11\} \\
s_1 = 6 \quad s_1' = 6 \quad \text{Job 10 is out}

H_{11} = \{\emptyset\} \\
s_1 = 1 \quad s_1' = 0 \quad \text{Job 11 is in}

\textbf{Figure 3.2}

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 Figure 3.4, the number outside each node is its level.

Muntz and Coffman's [5] algorithm to obtain minimum finish time 2-machine schedules is given in Figure 3.5. 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 from any vertex with no predecessors to i. Let H_i = L - D. The [L_i, D_i] pairs of each vertex in our example is given in Figure 3.4. 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.

A job i is said to be critical iff \( L_i = H_i \). Only non-critical jobs are candidates for displacement to other sets. Also, only those sets S_i that initially have exactly one critical job are candidates to receive a new job during the for loop. Note that every S_i must have at least one
critical job. In Figure 3.6, 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+A to j+B iff $L_1 < j < L_i$. For our example, the resulting graph is shown in Figure 3.7.

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:
procedure SCH

L ← maximum level
S_i ← all jobs in J with level=i, 0<i<L
for k ← 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 processors are in S_i, S_{i+1}, ..., S_L.
    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 ← S_q - {r}; S_k ← S_k U {r}
endfor
preemptively schedule the jobs in each S_i using
McNaughton's rule.
Jobs in S_i precede those in S_{i-1}.
end SCH

Figure 3.5 The Muntz-Coffman Algorithm
(1) reindex the vertices in B. The new index of vertex \( i \in B \) is \( i' = L - i \). 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 \( S'_0, S'_{1}, \ldots, S'_{n} \) be the sets obtained by updating \( S_0, S_{1}, \ldots, S_{n} \) using the maximum matching above. These updated sets satisfy the following properties:
(a) There is no job pair \((i,j)\) such that \(i P j\) and \(i \not< S' a\), \(j \not< S' b\) for \(a \not< b\).

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

For our example, the maximum matching \(M\) is \(((2,4),(4,2),(9,1))\) (solid lines in Figure 3.7). So, \(S'_1 = S_1, i=0,1,2,5\). \(S'_2 = \{5,6,7\}\) 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. \(i P j\) implies \(i < j\)) and the unique way in which Glover’s algorithm works. Suppose that statement (a) is incorrect. Then, there exists a job pair \((i,j)\) such that \(i P j\) and \(i \not< S' a\), \(j \not< S' b\) and \(a \not< b\). Since no such job pair exists in \(S' L, \ldots, S' J\), it must be the case that job \(j\) is noncritical and it 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 \(i P j\), it follows that \(H_i > L_j, h_i > L_j, L_i > L_j, H_j > H_i\). 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 \(a \not< b\). Hence, we may assume that \(i\) is also noncritical. Again, if \(i\) is not matched in \(M\), then \(a \not< b\) as for \(a\) to be less than or equal to \(b\), it must be the case that \(L_i \not< H_j\). But since \(L_i > L_j, h_i \not< h_i\) and \(i\) must get matched to \(b\) before \(j\) can (see Glover’s algorithm). Hence, \(i\) and \(j\) must both be matched in \(M\). \(i\) is matched to \(a\) and \(j\) to \(b\). Hence, \(a \not< b\). If \(a \not< b\) then vertex \(b\) is considered before \(a\) (as its new index is \(L-b<L-a\)). Since \(i P b\), it must be the case that \(H_i > b \not< L_i\). Also, \(L_i > L_j\) implies that \(h_i < h_i\). 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 \(i \not< B\) is such that no vertex of \(A\) is matched to \(i\) in \(M\), then \(\mid S'_i \mid = 1\). To see this, observe that \(S'_i\) has exactly one critical job. Every noncritical job in \(S'_i\) is a candidate for matching with \(i \not< B\). Thus, if no job is matched with \(i\), then all these noncritical jobs are matched elsewhere and \(\mid S'_i \mid = 1\). It is readily seen that if \(i \not< B\) is matched in \(M\), then \(\mid S'_i \mid > 1\). 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 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 $S_1, \ldots, S_r$ can be obtained in $O(\log n)$ time by sorting on $L_i$. This requires $O(n')$ 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.

dence, 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 $O(n^2/(\log^2 n \cdot n^3/\log n)) = O(1/(n \log n))$.

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).
5. References


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

Eliezer Dekel and Sartaj Sahni

Department of the Navy
Office of Naval Research
Arlington, Virginia 22217

April 1981

25

UNCLASSIFIED

UNCLASSIFIED

Parallel algorithm, convex bipartite graph, scheduling, complexity.

An efficient parallel algorithm to obtain maximum matching in convex bipartite graphs is obtained. 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 completion time.