An Improved Maximum Matching Algorithm in a Permutation Graph[‡‡]

Frank Bauernöppel[‖*††] (bauernoe@informatik.hu-berlin.de)
Evangelos Kranakis[*†] (kranakis@scs.carleton.ca)
Danny Krizanc[*†] (krizanc@scs.carleton.ca)
Anil Maheshwari [‡†*†††] (manil@tifrvax.tifr.res.in)
Jörg-Rüdiger Sack[*†**] (sack@scs.carleton.ca)
Jorge Urrutia[§†] (jorge@csi.uottawa.ca)

May 18, 1995

Note : This draft improves the results presented in Section 3 of [1].

1 Introduction

We present an $O(n \log^2 n)$ time algorithm for computing a maximum matching in a permutation graph on n-vertices. Our results are based on the algorithm of [12] for a two processor scheduling problem of [2]. The algorithm of [12] runs in $O(n + m)$ time, where n and m are the vertices and dependencies, respectively of the given graph. Through vector dominance and using computational geometry techniques we provide a novel and improved algorithm for maximum matching in permutation graphs. We establish that the problem has an $O(n \log^2 n)$ solution. Furthermore, if the dependency graph of a scheduling problem is known to be a permutation graph, then we now have an improved two-processor scheduling algorithm (if the number of edges is $\Omega(n \log^2 n)$).

2 Maximum Matching in a Permutation Graph

As pointed out, e.g., in [10], there is a strong relation between maximum matchings in co–comparability graphs and the following scheduling problem: Let $G = (V, E)$ be a directed acyclic graph; let G have n vertices and m edges. Vertex $v \in V$ is a *successor* of a vertex $u \in V$ if there is a directed path from u to v in G. A *two-processor scheduling* for G is an assignment of time units $1, 2, 3, \ldots$ to the vertices $v \in V$ such that

[*] Carleton University, School of Computer Science, Ottawa, ON, Canada
[†] Research supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada) grant.
[‡] Tata Institute of Fundamental Research, Bombay 400 005, India
[§] University of Ottawa, Department of Computer Science, Ottawa, ON, Canada
[‖] Institut für Informatik, Humboldt-Universität zu Berlin, 10099 Berlin, Germany
[**] Research supported in part under an R&D agreement between Carleton University and ALMERCO Inc.
[††] Work by the author was carried during a stay at Carleton University.
[‡‡] The results presented in this addendum will be presented at ICALP'95.

1. each vertex $v \in V$ is assigned exactly one time unit,
2. at most two elements are assigned the same time unit, and
3. if v is a successor of u in G, then u is assigned a lower time unit than v.

The edges of G represent dependencies among the set of n vertices (tasks) to be executed. The largest time unit assigned to a vertex is called the length of the schedule.

It has been shown that the following holds [4]: Given a directed acyclic graph G with n vertices then there is a two-processor scheduling for G of length l iff there is a matching of size $n - l$ in the undirected complement graph G'. Two vertices u and v are connected by an undirected edge in G' iff neither uv nor vu is an edge in graph G. Moreover, a matching in G' can be obtained from a schedule by simply matching all pairs of vertices that are assigned the same time unit. Note that G' is a co–comparability graph.

Efficient algorithms for finding a tightest two-processor schedule are known [5]. We follow the approach of [2] that leads to a linear time algorithm for the scheduling problem [12] when the graph G is transitively closed. This approach uses a vertex numbering assigning numbers $1, 2, \ldots$ to the n vertices of G in an increasing order. By $L(u)$ we denote the label of vertex u and by $N(u)$ we denote the list $(L(v_1), L(v_2), \ldots, L(v_k))$ of the labels of the successors v_i of u in G sorted in decreasing order.

Suppose the numbers $1, 2, \ldots, k-1$ have already been assigned. A vertex u is labeled with value $L(u) = k$, if

1. all successors of u in G are already labeled, and
2. for each other vertex u' fulfilling 1, the sorted list $N(u')$ is lexicographically not smaller than $N(u)$. (Ties are broken arbitrarily.)

Once the vertex labeling is complete, the matching is found in a greedy manner by a list schedule according to decreasing vertex labels: Starting with the highest label n, a vertex is matched with the highest label possible.

The $|V| + |E|$ time algorithm of [12] is optimal when the restriction graph G is given explicitly. This yields a $O(n^2)$ time maximum matching algorithm for the corresponding co–comparability graph G'.

Since permutation graphs can be represented in $O(n)$ space, we are interested in a faster algorithm for this class of graphs. Using the geometric interpretation of permutation graphs (i.e. the class of permutation graphs is the same as that of the vector dominance graphs in the plane, see [1]), we show that simple geometric arguments and data structures suffice to design a matching algorithm whose run-time is sublinear in the number of edges of the graph.

To achieve this, partition the vertices into *levels*. The level of a vertex $v \in V$ is the length of the longest path from v to a vertex of outdegree zero. It is easy to see that the following holds: If vertex $u \in V$ is at a higher level than vertex $v \in V$, then $L(u) > L(v)$. This can be shown by an inductive argument. Let l_u, l_v be the level of u and v respectively. Then, v has at least one successor at level $l_v - 1 \geq l_u$ whereas u has no successor at this level.

This partitioning into levels corresponds to vertex domination in the geometric representation of a permutation graph. The partition into levels can be done in $O(n \log n)$ time. All that remains is to determine the order in which the vertices within a level are labeled. Instead of determining sorted lists $N(v)$ we use a geometric argument. Denote by $DomReg(p)$

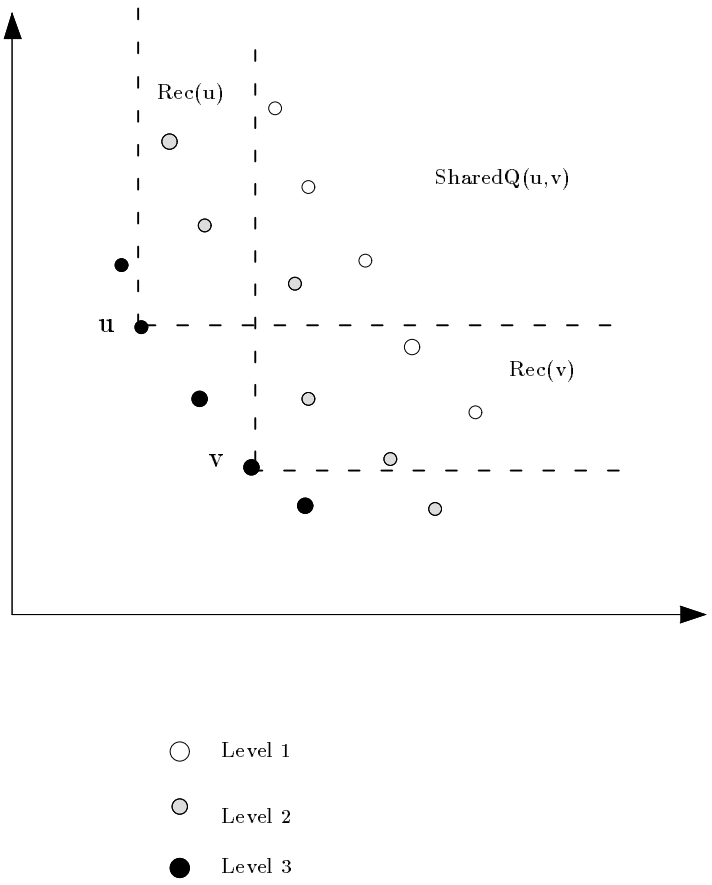

Figure 1: The regions $Rec(u)$, $Rec(v)$ and $SharedQ(u,v)$.

the upper right quadrant of an axis-aligned coordinate system whose origin is at point p. In this section, a point p dominates a point q if q lies in $DomReg(p)$. Let R be any region of the plane, then $Max(R)$ denotes the maximum label of all labeled points which lie in R, it is set to zero if R contains no labeled point.

Now let u and v be two points on a common level and assume w.l.o.g. that u lies above v, i.e. u's $y-$coordinate is larger than v's. The intersection of $DomReg(u)$ and $DomReg(v)$ is a quadrant called $SharedQ(u,v)$. Then the region $DomReg(u)$ can be partitioned into $SharedQ(u,v)$ and the remaining half-open rectangle, called $Rec(u)$; similarly, for v (see Figure 1). Now, observe that $L(u) > L(v)$ if and only if $Max(Rec(u))$ is greater than $Max(Rec(v))$.

This reduces the problem of performing a comparison operation of the form "$L(u) > L(v)$" (as needed for sorting each layer) to a comparison between two integers (labels) obtained via Maximum-Labeled-Element-Queries in half-open rectangles. There are different approaches to solving such queries: one is to state the problem as a (dynamic) 3-d range searching problem where the third coordinate is the label, the other, taken here, is to use the range priority search trees (called as *range trees*, see e.g., [11]). A range tree stores the points

sorted by x-coordinate in its leaves. Located at each internal node is a y-sorted list of all points in its subtree.

To perform a Maximum-Labeled-Element-Query in a half open rectangle R, we must find the maximum labeled point in R, where R is bounded by $[x_a, x_b]$ and $[y_a, +\infty]$. To answer these queries we build two data structures. The first data structure is a range tree that reports the maximum layer number l among all layer numbers corresponding to each point in R. Given l and R, the second data structure reports the maximum labeled point in the layer l, among all points of l lying inside R.

The (priority) range tree is computed as follows. Sort the points with respect to increasing x-coordinate. Build a balanced binary search tree T over them. At each internal node u sort the points, which are in the subtree rooted at u, with respect to their y-coordinates and determine the maximum label among all points in the subtree. For each point p_i at node u of T, compute the maximum layer number among all points at u which have higher y-coordinate with respect to p_i. Also assume that there are cross-ranking pointers associated between a node and its parent and between a node and its sibling. It can be seen that the preprocessing takes $O(n \log n)$ time and $O(n \log n)$ space using the algorithm of [3].

The queries in the first tree are answered as follows. We locate the $O(\log n)$ roots of subtrees spanning the x-range $[x_a, x_b]$ and for each of these we use the sorted y-lists to compute the maximum layer number in its x-range. The maximum layer number is the maximum of at most $O(\log n)$ values computed in the above step. Note that we do not have to perform a binary search in the sorted y-lists, since we can locate y_a using the cross ranking information. Thus the queries in the first tree can be answered in $O(\log n)$ time.

The second data structure is computed for every layer of the point set. Observe that any layer l is $x - y$ monotone. Hence l can be represented by an array A_l, where the points in A_l follow the respective order. Given the half open rectangle R, the points of l in R can be located by performing a binary search on A_l by choosing the appropriate x and y coordinates as the keys. Note that the points of l in R forms an interval in A_l. So the problem of computing the maximum labeled point in R of l reduces to that of computing the maximum element of an interval in A_l. We know that an array containing integers in the range $1, ..., n$ can be preprocessed in $O(n)$ time and the maximum interval queries can be reported in $O(1)$ time [6]. Thus the second data structure can be computed in $O(n)$ time using $O(n)$ storage and the queries can be answered in $O(\log n)$ time. Now we describe our algorithm.

1. Compute the vector dominance representation of the permutation graph.
2. Partition the point set into layers, $1, 2, \ldots, k$ and assign each point its layer number.
3. Build the first data structure - the range tree.
4. Assign arbitrarily the labels to the points in the first layer from the range $1, ..., n_1$, where n_1 is the number of points on the first layer. All other points are initialized to 0 as their labels. Build the second data structure for the points in the first layer.
5. For layers $i = 2, 3, \ldots, k$ do
   sort the points on layer i (using the above described comparison operator) and assign consecutive labels to the points.
6. Perform a greedy matching on the labeled graph.

Note that the labels for the layer i are computed using (only) the labels of the layers 1 to i-1; the locations of all points remain unchanged. The total time per point is therefore $O(\log n)$, since the queries in both data structures can answered in $O(\log n)$ time and the second data structure is built in linear time, once we assign labels to each point on that layer. If an optimal sorting algorithm is used, the total number of queries can be bounded by $\sum_{i=1}^{k} n_i \log n_i$, where n_i denotes the cardinality of layer i. Thus all labels can be assigned in $O(n \log^2 n)$ time. Now we show how the greedy matching can be performed.

The matching can be done by a sequence of $O(n)$ Maximum-Labeled-Element-Queries using the quadrant $DomReg(p)$ for finding point q to be matched with p. It examines points in decreasing order of their labels and tries to match them. Once p and q are matched, both of them are deleted from the point set. To compute the matching, a dynamic range-range priority search tree is used. Sort the points with respect to x-coordinate and arrange them in a binary search tree (i.e. the primary tree). At each node of the tree, sort all points in its subtree with respect to y-coordinate and build a binary search tree over them (i.e. secondary tree). This data structure can be built in $O(n \log n)$ time and $O(n \log n)$ space using the algorithm of [3]. Given a point p, the maximum-labeled query is performed as follows. Locate the $O(\log n)$ roots of subtrees in the primary tree spanning the x-range of $DomReg(p)$ and for each of these we use the secondary tree to compute $O(\log n)$ roots of subtrees spanning the y-range of $DomReg(p)$. So in all we have $O(\log^2 n)$ values, and the maximum labeled point q is the maximum of these values. Thus the maximum point q can be reported in $O(\log^2 n)$ time. The next step is to remove the point q from the data structure. Locate the leaf of the primary tree containing the point q. Now walk up this tree till its root and at each intermediate node, update the secondary tree starting from the leaf containing q till its root. Since there are only $O(\log n)$ secondary trees to be updated, the total time for deletion of q from the data structure is $O(\log^2 n)$ time. Hence the greedy matching can be computed in $O(n \log^2 n)$ time. Above results are summarized in the following theorem.

Theorem 2.1 *A maximum matching in permutational graphs can be computed in $O(n \log^2 n)$ time where n is the number of vertices of G.*

Since complement graphs of permutation graphs are permutation graphs we get the following result.

Theorem 2.2 *The two-processor task scheduling for dependency graphs known to be permutation graphs can be solved in $O(n \log^2 n)$ time where n is the number of processes (not dependencies).*

References

[1] F. Bauernoppel, E. Kranakis, D. Krizanc, A. Maheshwari, J.-R. Sack and J. Urrutia, *Planar Stage Graphs: Characterizations and Applications*, Technical Report TR-95-06, School of Computer Science, Carleton University, Ottawa, Canada, 1995 (submitted for publication).

[2] Coffman, E. G. Jr.; Graham, R. L.; "Optimal scheduling for two-processor systems" Acta Informatica 1 (1972), 200-213.

[3] R. Cole, "Parallel Merge Sort", SIAM J. Comp. 17:4 (1988), 770-785.

[4] Fujii, M.; Kasami, T.; Ninomiya, K. "Optimal sequencing of two equivalent processors." SIAM J. Appl. Math. 17:4 (1969), 784-789.

[5] Gabow, H. N. "An almost-linear algorithm for two-processor scheduling" J. ACM 29:3 (1982), 766-780.

[6] J. JáJá, An introduction to parallel algorithms, Addison-Weseley Publishing Company, 1992.

[7] J. H. van Lint and R. M. Wilson, A Course in Combinatorics, Cambridge University Press, 1992.

[8] L. Lovász, Combinatorial Problems and Exercises, North Holland Publishing Company, 1979.

[9] S. Micali and V. V. Vazirani, "An $O(\sqrt{V}E)$ algorithm for finding maximum matching in general graphs", in: Proc. 21st Ann. IEEE Symp. Foundations of Computer Science, (1980),17-27.

[10] Moitra, A.; Johnson, R., C. "A Parallel algorithm for maximum matching on interval graphs." 18th Intl. Conf. on Parallel Processing III (1989), 114-120.

[11] F. P. Preparata and M. I. Shamos, Computational Geometry: An Introduction, Springer-Verlag, New York, 1985.

[12] Sehti, R. "Scheduling graphs on two processors" SIAM J. Comp. 5:1 (1976), 73-82.