# Efficient labelling algorithms for the Maximum Non Crossing Matching Problem

by

Federico Malucelli[1], Thomas Ottmann[2], Daniele Pretolani[1]

**Abstract:** *Consider a bipartite graph; let's suppose we draw the origin nodes and the destination nodes arranged in two columns, and the edges as straight line segments. A non crossing matching is a subset of edges such that no two of them intersect. Several algorithms for the problem of finding the non crossing matching of maximum cardinality are proposed. Moreover an extension to weighted graphs is considered.*

**Keywords:** *Non Crossing Matching, VLSI layout, permutation, longest increasing subsequence.*

## 1. Introduction

**Consider a bipartite graph** $G=(O,D,E)$, with $O$ and $D$ origin and destination node sets respectively ($|O|=|D|=n$), and $E$ a set of edges $(i,j)$, $i\in O$ and $j\in D$ ($|E|=m$). For each origin node $i$ let $FS(i)$ be the list of edges incident in $i$. Suppose we draw the origin nodes and the destination nodes arranged in two columns, and the edges as straight line segments between origins and destinations. A non crossing matching is a subset of edges $M\subseteq E$ such that no two edges of $M$ intersect (including intersections at nodes). The Maximum Non Crossing Matching (MNCM) is the problem of finding the non crossing matching of maximum cardinality. Let $p$ denote the cardinality of the MNCM. Problems arising in several fields can be modelled as MNCM: for example the 3-Side Switch Box in VLSI design has been presented in [2], where an $O(n^2)$ time algorithm is proposed. The MNCM problem can be reduced to the one of finding the longest increasing subsequence in a permutation of size $m$. An algorithm for this problem has been proposed by Fredman [1] and slightly improved by Widmayer and Wong [7]; in our case this algorithm has complexity $O(m+m\log p)$. In this paper some labelling algorithms for the MNCM, which work directly on the bipartite graph, will be proposed. The overall complexity is improved to $O(m\log\log n)$ or to $O(m+\min\{np, m\log p\})$. Finally the Maximum Weight Non Crossing Matching (MWNCM) will be introduced.

## 2. The labelling algorithm

We identify both origin nodes and destination nodes with numbers in the set $\{1,2,\ldots,n\}$; the nodes are numbered in increasing order from the top to the bottom, hence two edges $(i,j)$ and $(h,k)$ cross iff ($i\le h$ and $j\ge k$) or ($i\ge h$ and $j\le k$).

The algorithm is organized in two phases: a first phase during which labels are assigned to all the edges of $E$; a second phase during which the edges of the MNCM are selected. The label $L(i,j)$ assigned to edge $(i,j)$ corresponds to the cardinality of the partial MNCM that includes $(i,j)$ and lies entirely above it (i.e. it includes only edges $(h,k)$ such that $h<i$ and $k<j$). The value of the maximum assigned label gives the cardinality of the MNCM for $G$, as can be easily proved [2]. Once labels have been defined, the selection phase is trivial.

---

[1]Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy; e.mail carra@dipisa.di.unipi.it

[2]Institut für Informatik, Universität Freiburg, Rheinstr. 10-12, D-7800 Freiburg i. Br., Germany.

The labelling phase is the crucial part of the algorithm. This phase can be carried out scanning the origin nodes from the top to the bottom, setting the label for each edge incident to the current origin node. We assign a label $LN(j)$ to each destination node $j$; all node labels are initially set to 0. During each iteration of the labelling algorithm given below some node labels will be increased. After completion of the labelling phase $LN(j)$ is the maximum label assigned to an edge incident to $j$. The labelling algorithm can be described as follows:

{Step 0} for each $j \in D : LN(j) := 0$;
for $i := 1$ to $n$ do
{Step 1} for each edge $e = (i,j) \in FS(i) : L(e) := 1 + \max \{ LN(k) : k < j \}$;
{Step 2} for each edge $e = (i,j) \in FS(i)$: $LN(j) := \max \{ LN(j) , L(e) \}$;

The selection of the edges in an MNCM can be carried out as follows:

let $k = \max\{L(e), e \in E\}$;
select an edge $e_k$ with label $k$;
while $k > 1$ do
select an edge $e_{k-1}$ with label $k-1$ and not intersecting the edge $e_k$;
$k := k-1$;

If the edges are arranged in a suitable data structure (i.e. a bucket list) then, since each edge is considered only once, the selection phase takes $O(m)$ time. It is easy to see that this phase gives a non crossing matching of maximum cardinality.

**3. Implementation and complexity.**

In our algorithm the techniques described in [1], [7] are modified to work on general bipartite graphs. Instead of maintaining the node label $LN(j)$ for each destination node $j$ during each iteration of the loop in the labelling algorithm it is sufficient to mantain an array $P$ which gives, for each node label $k$ assigned so far, the topmost destination $j$ such that $LN(j)=k$. Let $K$ denote the maximum currently assigned node label. Then we can replace {Step 0} in the labelling algorithm by setting $K := 0$ and $P(0) := 0$. Furthermore, in {Step 1} the assignment of edge label $L(e)$ to edge $e$ can be done referring to the array $P$ instead of referring to the node labels, since it is easy to see that

$\max \{ LN(k): k < j \} = \max \{ k : P(k) < j , 0 \le k \le K \}$.

Finally, we can directly update $P$ and $K$ and, therefore, replace {Step 2} of the labelling algorithm as follows:

{Step 2} for each edge $e = (i,j) \in FS(i)$:
$k := L(e)$;
if $k \le K$ then $P(k) := \min\{ P(k), j\}$
else $K := K + 1$; $P(K) := j$;

The complexity of our algorithm depends on the implementation of the max operation in {Step 1}. Note that updating $K$ and the array $P$ in {Step 2} takes $O(m)$ overall time.

Further on we assume that the edges in each $FS(i)$ are sorted with respect to the index of destination node. The $FS(i)$ lists can be sorted in $O(m)$ time as follows: first, collect all edges with the same destination node $j$ in a list $L_j$, $1 \le j \le n$; then scan through the non empty lists in the sequence $L_1,\ldots,L_n$ and append each edge $(i,j)$ to $FS(i)$.

A simple implementation of {Step 1} requires the scanning of both $FS(i)$ and $P$, in the same order as in a merge operation between two lists; in this case the labelling of $FS(i)$ takes $O(|FS(i)|+K)$ time, and the overall complexity is $O(m+np)$.
For sparse graphs (i.e. when $|FS(i)|<<K$), we can perform a binary search for each edge $(i,j)$ instead of scanning the whole sequence $P$, thus obtaining an $O(|FS(i)|\log(K))$ time bound for {Step 1}, and an overall $O(m\log p)$ complexity.

**Remark 3.1** For each origin node $i$ we can choose between the scanning of $P$ and the binary search, depending on the current values of $K$ and $|FS(i)|$; the resulting complexity is $O(m+\min\{np, m\log p\})$.

**Remark 3.2** Consider the class of convex bipartite graphs, that is graphs in which for each origin $i$ $FS(i)$ contains exactly the edges $(i,j)$ with $j$ in an interval [3]; on this class the problem can be solved in $O(m+(n-p)\log p)$ time implementing {Step 1} as follows:

- find the label $k$ for the edge $(i,j)\in FS(i)$ with maximum index $j$; note that we can check in constant time if $j>P(K)$, and in this case we have $k = K+1$; hence, the binary search is not necessary for the $p$ edges $(i,j)$ that allow to increase $K$;
- scan $P$, in decreasing order, starting from $P(k)$; it is easy to see that at most $|FS(i)|$ elements of $P$ must be scanned.

**Remark 3.3** We can label each edge in less than logarithmic time using a *bounded dictionary* [5] or the *priority queue* defined in [6]. These data structures allow to perform set-manipulation operations and queries on a subset $S$ of the integers in the interval $[1..N]$ in $O(\log\log N)$ time and $O(N)$ space: in our case the set $S$ contains the values $P(1),\ldots,P(K)$, and thus $N=n$. For each edge $(i,j)$ the max operation and the updating of $P$ and $S$ can be carried out in $O(\log\log n)$ time; the overall complexity of the algorithm becomes $O(m\log\log n)$, with an $O(m)$ space bound.

## 4. The weighted problem

Let $w_{ij}$ be a real number associated to each edge $(i,j)\in E$. The Maximum Weight Non Crossing Matching (MWNCM) is defined as the non crossing matching $M$ with the maximum sum of $w_{ij}$ over $(i,j)\in M$. Note that the MWNCM is not necessarily the MNCM, moreover if there are some negative $w_{ij}$ then $M$ may even be non maximal.

The basic structure of the algorithm remains unchanged. The meaning of the labels becomes the following: $L(i,j)$ is the weight of the partial MWNCM which includes edge $(i,j)$ and lies entirely above $(i,j)$. Consequently the labelling operation of {Step 1} becomes:

$$L(i,j) := w_{ij} + \max\{LN(k): k<j\}.$$

In order to compute $\max\{LN(k): k<j\}$ efficiently we use a simplified version of the priority search tree (PST) [4]. A PST is a structure for storing sets of points in a two-dimensional space; here, we regard the pairs $(j, LN(j))$ as points in a 2-space. This structure allows to perform insertion and deletion of pairs, and *range query* operations such as finding the pair $(x,y)$ with maximum $y$ value and $x$ belonging to a given interval. The time complexity of each of these operations is $O(\log N)$, where $N$ is the number of pairs contained in the structure. It is easy to see that the max operation in {Step 1} of the labelling phase can be reduced to a range query on the subset $[1,\ldots,j-1]$ of the destination nodes. Update operations in {Step 2} can be carried out performing a deletion and a subsequent insertion into the PST. Since there will be at most $n$ pairs in the PST, the overall complexity of the algorithm is $O(m\log n)$. The space requirement for the PST is $O(n)$, hence the space complexity of the algorithm remains

unchanged.

## 5. Conclusions

In this paper we proposed several algorithms for the Maximum Non Crossing Matching problem. The main characteristic of all these algorithms is that the problem is directly approached, as in [2], without reducing it to the permutation case. Comparing our algorithm to the one presented in [2], it should be observed that for sufficiently sparse graphs (i.e. $m$=O($np$)) we obtain a O($np$) worst case complexity, which is slightly better than O($n^2$). Moreover, the space requirement is reduced from O($n^2$) to O($m$). On the other hand remark that, applying the Widmayer and Wong algorithm to the permutation corresponding to a dense graph (i.e. $m$=O($n^2$)), the resulting complexity is O($n^2 \log p$), which is worse than O($n^2$).
Another stimulating open problem is the one of finding the Maximal NCM of minimum weight, which seems to be more difficult than the MWNCM: it is possible to devise a trivial O($m^2$) algorithm; the possibility to improve this complexity deserves further investigations.

## Acknowledgements

We wish to thank the referees for their comments, and B. Nilsson for helpful discussions.

## References

[1] M.L. Fredman, "On Computing the Length of Longest Increasing Subsequences", **Discrete Mathematics**, 11, 1, (1975), p. 29-35.

[2] Y. Kajitami and T. Takahashi, "The non cross matching and applications to the 3-side switch box routing in VLSI layout design", Proc. International Symposium on Circuits and Systems 1986, p 776-779.

[3] E. Lawler, "Combinatorial Optimization: Networks and Matroids", Holt, Rinehart and Winston, (1976).

[4] E. M. McCreight, "Priority search trees", **Siam Journal on Computing**, v. 14 (1985), n. 2, p 257, 276.

[5] K. Melhorn and S. Näher, "Bounded Ordered Dictionaries in O(log log $N$) Time and O($n$) space", **Information Processing Letters** 35 (1990), p. 183-189.

[6] P. van Emde Boas, "Preserving Order in a Forest in Less than Logarithmic Time and Linear Space, **Information Processing Letters** v. 6, n. 3 (1977), p. 80-82.

[7] P. Widmayer and C.K. Wong, "An Optimal Algorithm for the Maximum Alignment of Terminals", **Information Processing Letters** 10 (1985), p. 75-82.