

AN OPTIMAL ALGORITHM FOR THE MAXIMUM ALIGNMENT OF TERMINALS

P. WIDMAYER * and C.K. WONG

IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598, U.S.A.

Communicated by H.R. Wiehle

Received 25 January 1984

Revised 20 June 1984

We consider the problem of finding a maximum subset of a given set of wires connecting two rows of terminals with fixed positions, such that no wires in the subset cross. We derive an algorithm that runs in $O(p + (n - p) \lg(p + 1))$ time, where n is the number of wires given and p is the maximum number of noncrossing wires; in many practically relevant cases, e.g., when p is very high, it needs only linear time. We show how an extension of the algorithm solves the more general problem, where the positions of some terminals have some flexibility, within the same time bound.

Keywords: Maximum independent set, permutation graph, longest common subsequence, longest increasing subsequence, VLSI wiring, maximum alignment

1. The alignment problem for fixed terminals

In recent studies on VLSI wiring [3], cascode-switch macro wiring [9], and the pin permutation problem [7], the problem of finding a maximum subset of noncrossing wires in a situation as in Fig. 1 is encountered. For example, noncrossing short wires can be realized in the polysilicon level, thereby substantially saving wiring space on the metal level and reducing net capacitance.

In the top row, terminals are numbered from 1 to n from left to right; in the bottom row, these numbers are arbitrarily permuted. Each pair of terminals having the same numbers is to be connected by a straight wire. In general, some of the wires cross (see Fig. 1). The problem is to find a subset of maximum size of the set of wires such that no two wires in the subset cross each other; we call this a maximum alignment of the terminals, where the terminals are in fixed positions. Later on, we will also consider the case where some of the terminals are allowed to move in

specified ranges. This problem arises in a more general version of the pin permutation problem [8].

2. Previous results

The first paper known to us to study this problem in a different context is [1]; there, an algorithm with $O(n^2)$ worst-case running time for the (equivalent) maximum size independent set problem for permutation graphs is proposed. Fredman [2] presents an algorithm for computing the length of a longest increasing subsequence with worst-case running time of $O(n \lg n)$; he proves that $\Omega(n \lg n)$ is also a lower bound in the worst case.

Since then, numerous papers on the more general problem of computing a longest common subsequence of two given strings have appeared. Hirschberg [4], Hunt and Szymanski [5] and Nakatsu et al. [6] have designed algorithms that are especially fast in special cases related to our problem of finding the longest ascending subsequence in a string of the integers 1 to n . Hirschberg [4] has presented an $O(n \cdot p)$ algorithm, where

* On leave from: Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Universität Karlsruhe, Postfach 6380, 7500 Karlsruhe, Fed. Rep. Germany.

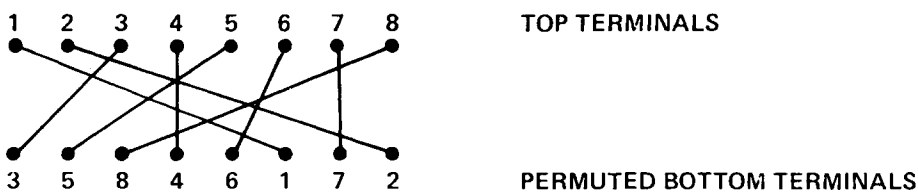


Fig. 1. An example of the fixed terminal alignment problem.

p is the size of the solution, i.e., in our case, the number of noncrossing wires. Hunt and Szymanski [5] describe an $O(n \lg n)$ algorithm, and Nakatsu et al. [6] derive an $O(n(n - p))$ algorithm. Hence, for very small p (close to 1) or very high p (close to n), linear-time algorithms exist; however, as p is not known in advance, choosing the wrong algorithm leads to quadratic worst-case behavior.

In the VLSI design process, the maximum alignment problem has to be solved repeatedly. An efficient algorithm is, hence, highly desirable. It should, furthermore, take advantage of the fact that usually p is quite close to n . In the following, we present an algorithm that combines the advantages of [2,4,5,6] in that it is bounded by $O(n \lg n)$ in the worst case, and that it is linear whenever p is very high or very small. Our algorithm has the same structure as the one given in [2], and it can also be obtained by a modification of [5], yielding a worst-case running time of $O(p + (n - p) \lg(p + 1))$.

3. The alignment algorithm for fixed terminals

In any set of noncrossing wires, e.g., in any optimal solution, the wires can be totally ordered from left to right, according to their terminal numbers. From now on, we will identify the wires with their terminal numbers. By definition, for any set S of $p \geq 1$ noncrossing wires there exists a subset S' of $p - 1$ noncrossing wires with bottom terminals and top terminals to the left of those of the rightmost wire in S . We expect the computation of S to be simple and fast, once we know of such an S' . This observation leads to an algorithm to recursively compute solutions from partial problems of growing size, where the next problem is obtained by adding one more wire to the prob-

lem solved so far. Starting with the empty set of wires, we add the wires, one by one, in the order of the positions of their bottom terminals, from left to right. For the current problem and any achievable number of noncrossing wires we keep track of only one solution (instead of many feasible ones). This is sufficient if we choose the solution that permits us to later find a further noncrossing wire, if there is one, i.e., the one among all feasible solutions with the leftmost top terminal of its rightmost wire. In the example depicted in Fig. 1, considering the set $\{3, 5, 8, 4\}$ of wires, the feasible solutions achieving two wires are $\{3, 5\}$, $\{3, 8\}$, $\{3, 4\}$, and $\{5, 8\}$, and the best of them is $\{3, 4\}$; terminal 4 is the leftmost of the top terminals of the rightmost wires in these solutions. For each achievable number of noncrossing wires we represent the best solution by its rightmost wire and a link to the partial solution of one wire less used in this best solution. In this way, the rightmost wires of the solutions are ordered from left to right with increasing size of the solution, i.e., the rightmost wire of the biggest solution so far is the rightmost of these wires.

Now consider the top terminal t of the next wire w to be processed; the bottom terminal of w is to the right of all bottom terminals seen so far. All partial solutions having their rightmost top terminal to the left of t can be extended by adding w , but for all extensions except the one of the rightmost of these top terminals, say t' , a better solution is already known. The one solution with top terminal t' , however, yields a new best solution when extended by w : if there are wires with top terminals to the right of t in partial solutions, then w replaces the leftmost of them in the corresponding solution, otherwise w is added to the biggest solution so far, giving rise to a new bigger solution. As the biggest solution of each subproblem is

an optimal solution to that subproblem, the algorithm ends up with an optimal solution to the entire alignment problem.

In our example, when wires 3, 5, 8, and 4 have been processed, the best solutions are {3} for one wire, {3, 4} for two wires, {3, 5, 8} for three wires. The top terminals of the rightmost wires of the solutions are 3, 4, and 8. When wire 6 is being processed, it is useless to extend solution {3} to {3, 6} because we already know of the better solution {3, 4} for two wires. It is impossible to extend solution {3, 5, 8}, because adding wire 6 would lead to a set where some of the wires, namely 6 and 8, cross. Solution {3, 4}, however, will be extended giving {3, 4, 6}; this solution will replace {3, 5, 8}.

A formal proof of correctness of our algorithm can be carried out in the same way as in [5]. The following algorithmic description is, for the sake of being specific, very close to a real program; a more general formulation in terms of operations alone should be obvious. We think of storing the rightmost wires of the solutions as an ordered sequence w ; $w(1)$, $w(i)$, and $w(last)$ are the first, the i th and the last element in the sequence, respectively; $w(0)$ denotes a dummy element and is used for the purpose of uniform treatment only. The set of links between wires, from wires in bigger solutions to wires in smaller solutions, is denoted by ℓ .

ALGORITHM Maximum Alignment of Fixed Terminals

```

w(0): = 0;
last: = 0;
FOR all bottom terminals of wires t FROM left TO right DO
  IF w(last) < t
    THEN last: = last + 1; i: = last
    ELSE i: = min{j | 1 ≤ j ≤ last, w(j) > t};
  w(i): = t;
  ℓ(t): = w(i - 1);
  {solution is found: trace it back following links}
  FOR all elements w(i) in w FROM w(last - 1) BACK TO w(1) DO
    w(i): = ℓ(w(i + 1))
  {solution is stored in w}

```

END of Algorithm.

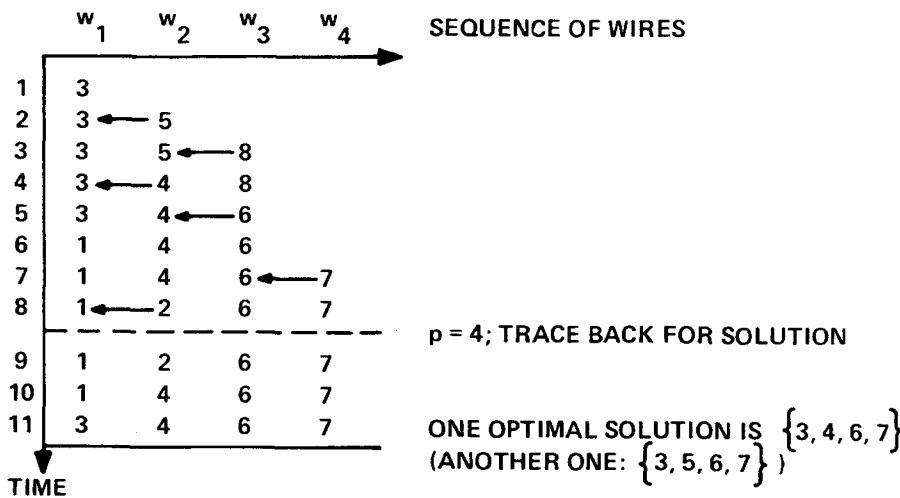


Fig. 2. Illustration of algorithm for fixed terminals.

Let us illustrate the operation of this algorithm with our example by displaying the sequence w after processing each wire as a horizontal row, time being represented at the vertical axis from top to bottom, and the links depicted by arcs between the linked wires (see Fig. 2).

Theorem 1. *A solution to the maximum alignment problem for fixed terminals can be constructed in $O(p + (n - p)\ell g(p + 1))$ time using $O(n)$ space, where n is the total number of wires involved and p is the size of the solution, which is optimal in the worst case.*

Proof. The correctness of the algorithm follows from the previous considerations. When properly implemented, all operations except finding $\min\{j \mid 1 \leq j \leq last, w(j) > t\}$ can be carried out in constant time; the latter can be found by a binary search in $O(\ell g(p + 1))$ time. For all wires increasing the size of the biggest solution, the position in w is found in constant time, and hence only the remaining $n - p$ wires require $O(\ell g(p + 1))$ time each, resulting in the claimed overall runtime. The $O(n)$ bound for the space requirement follows trivially; it becomes $O(p)$ when we do not keep track of the wires constituting the optimal solution, e.g., when we just want to compute the size of the optimal solution. Fredman [2] additionally points out that the maximum number of comparisons performed is given by $n \ell g n - n \ell g \ell g n + O(n)$, which is also a lower bound. \square

Note that without changing its worst-case behavior we can tune our algorithm by fitting the search for $\min\{j \mid 1 \leq j \leq last, w(j) > t\}$ to our specific expectations, as long as we maintain its $O(\ell g(p + 1))$ worst-case time bound. For example, when we expect the positions of the bottom terminals of most wires to be close to the positions of their top terminals, it is reasonable to start a binary search for $\min\{j \mid 1 \leq j \leq last, w(j) > t\}$ at $w(last)$, extending the range considered for searching exponentially until we are sure the answer is in the considered range. The following algorithmic description may help clarify this approach (we do not take care to avoid accessing elements outside the proper range in order not to obscure the basic

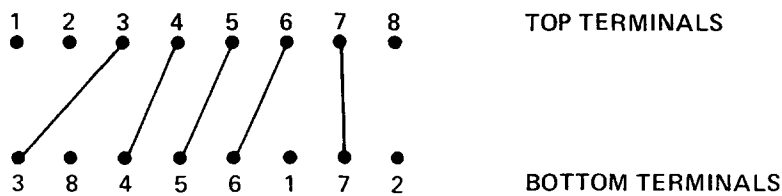
idea):

```
{exponential search for the range from leftend to
  rightend, in which  $\min\{j \mid 1 \leq j \leq last, w(j) > t\}$ 
  lies;}
leftend := last - 1;
WHILE ( $w(leftend) > t$ ) AND ( $leftend > last/2$ ) DO
  leftend :=  $2 \times leftend - last$ ;
IF  $w(leftend) > t$ 
  THEN rightend := leftend; leftend := 1
  ELSE rightend :=  $\lceil (leftend + last)/2 \rceil$ ;
       leftend := leftend + 1
```

The interval from *leftend* to *rightend* is found in at most $O(\ell g(last - leftend))$ steps, and a binary search within this interval will find the solution in at most the same number of steps. Incorporating this method into our algorithm, without deteriorating its worst-case behavior the algorithm runs in linear time whenever $|t - position(t)| \leq c$ for some constant c and all bottom terminals t , independent of p . Other choices for the starting position of a binary search following essentially our approach include starting at a precomputed expected position of the element in question as well as starting at the position of the most recently processed element.

4. The alignment problem for fixed and loose terminals

In a more general pin permutation problem [8], a collection of similar instances of the alignment problem described in the preceding sections naturally occurs in the form of a more general alignment problem. Instead of having all terminals in the bottom row in fixed positions, some terminals' places can be chosen freely within a specified range. The input to the problem is now a sequence of subsequences F_1, F_2, \dots, F_k , $k \geq 1$, of terminals with fixed relative positions such that all terminals in F_i are to the left of those in F_j if $i < j$, and a sequence L of sets L_1, L_2, \dots, L_k of loosely positioned terminals: For each $i = 1, \dots, k$, all terminals in L_i can be arbitrarily permuted and interleaved with those in F_i such that the combined set of terminals $T_i = F_i \cup L_i$ satisfies the condition



$$F_1 = (8, 4), F_2 = (6)$$

$$L_1 = \{3, 5\}, L_2 = \{1, 2, 7\}$$

Fig. 3. An optimal solution to the alignment problem for fixed and loose terminals.

that the terminals in T_i are to the left of those in T_j if $i < j$. A subproblem with fixed terminals is obtained if we choose for each i a permutation of L_i and an interleaving of this permutation with F_i . The objective is to find a subproblem with fixed terminals achieving the highest maximum alignment of all subproblems. For example, for $k = 2$, $F_1 = (8, 4)$, $F_2 = (6)$, $L_1 = \{3, 5\}$, and $L_2 = \{1, 2, 7\}$, the subproblem with fixed terminals as depicted in Fig. 1 achieves an alignment of 4 wires, but with bottom terminal 5 to the right of 4 an alignment of 5 wires, namely wires 3, 4, 5, 6, and 7, can be achieved (see Fig. 3).

Instead of constructing all subproblems and solving them with our fixed terminal alignment algorithm, we will show how the general problem can be solved directly much faster, in fact, as fast as the special problem. Note that the general alignment problem coincides with the special case of fixed terminals whenever $L_i = \emptyset$ for all i . Let us clarify the properties of the problem we will use in our algorithm.

Lemma 1. *For two sequences, F and F' , of bottom terminals, where F and F' may be interleaved freely, the maximum number of noncrossing wires of all sequences obtained by interleaving F and F' equals the sum of the maximum numbers of noncrossing wires obtained for F and for F' .*

Proof. Consider the subsequences, f and f' , of F and F' , of the bottom terminals of the selected noncrossing wires in any maximum alignment of F and F' . The f and f' can be interleaved without

crossing by forming an increasing sequence of their terminal numbers, and the terminals in $F - f$ and $F' - f'$ can be interleaved into correct positions with respect to those in f and f' , respectively.

On the other hand, for each interleaving of F and F' with interleaved noncrossing wires f and f' , f is also a set of noncrossing wires for F , and f' is also a set of noncrossing wires for F' . \square

Lemma 2. *For a sequence L of bottom terminals which may be permuted freely, the maximum number of noncrossing wires is obtained by sorting the bottom terminals by increasing top terminal number, from left to right.*

Proof. For this ordering, all of the terminals can be connected with noncrossing wires. \square

This implies that if there is only one subsequence of fixed terminals, i.e., the range of the loose terminals' positions is unbounded, the problem can be solved by applying the fixed terminals alignment algorithm to the fixed terminals first and then properly interleaving all loose terminals in between the wires forming the solution.

5. An alignment algorithm for fixed and loose terminals

The problem with loose terminals L_i in ranges of fixed terminals F_i differs from the one with fixed terminals in the fact that in addition a good position for each loose terminal must be chosen.

We use the following two facts to find good positions. First, a loose wire w from L_i cannot be obstructed by a fixed or loose wire w' from $T_i = F_i \cup L_i$, because the loose bottom terminal of w is allowed to take any position relative to the one of w' . Hence, any selection of fixed wires of F_i and any position and selection of loose wires of L_i will be equally favorable for the loose wires within L_i . Second, except within its range, a loose wire does not have any more flexibility than a fixed one: its relative position is fixed with respect to all wires outside its range.

Then, the general alignment algorithm produces a solution in essentially the same way as our fixed terminal alignment algorithm. Note that for constructing partial solutions for growing size the order in which we consider the wires does not matter as long as all best partial solutions are available at all steps. In the fixed terminal case, we have chosen an order of the wires allowing us to accomplish this by adjusting only one partial solution very fast; we reduced the query for the rightmost wire with bottom terminal and top terminal to the left of a given wire to the query for the rightmost wire with top terminal to the left, by identifying the one dimension for bottom terminal positions with the dimension of time. Following this same idea, we choose the following order of wires in the general algorithm: for i increasing from 1 to k , we process the wires in F_i in the order of their bottom terminals, and then the wires in L_i in the order of their top terminals, each from left to right. We maintain a sequence of rightmost wires of partial solutions and links connecting the

wires used in partial solutions as in the case of fixed terminals. By the arguments presented for the fixed terminal alignment problem, the update operations on this sequence are the same as for fixed terminals, except in one case: if a loose wire w from L_i replaces a fixed wire w' from F_i , then $w < w'$, and hence w' can be used to extend the partial solution of which w is the new rightmost wire. If w' has been the rightmost wire of the sequence of rightmost wires of partial solutions, then a new bigger solution is found, otherwise the right neighbor w'' of w' in this sequence is replaced by w' . If, again, w'' belongs to F_i , the replacement process continues. As soon as either the end of the sequence is reached or a wire not belonging to F_i has been replaced, this process stops. In this way, all reasonable interleavings of loose terminals, i.e., interleavings contributing to some partial solution at some stage, are implicitly considered.

Let us illustrate with an example how the algorithm works. Given $k = 2$, $F_1 = (7, 5)$, $F_2 = (3, 9, 6, 8)$, $L_1 = \{4\}$, and $L_2 = \{1, 2\}$, we display the partial solutions obtained over time (see Fig. 4).

Notice that for each loose terminal from L_i , in the worst case all fixed terminals from F_i have to be shifted to the right by one position. If we carry out these shifts separately by repositioning element by element, we may need $|L_i| \cdot |F_i|$ steps for each i . However, we can easily avoid this by scanning through the elements of F_i at most once when processing all elements from L_i , because the elements in L_i are processed in sorted order. We

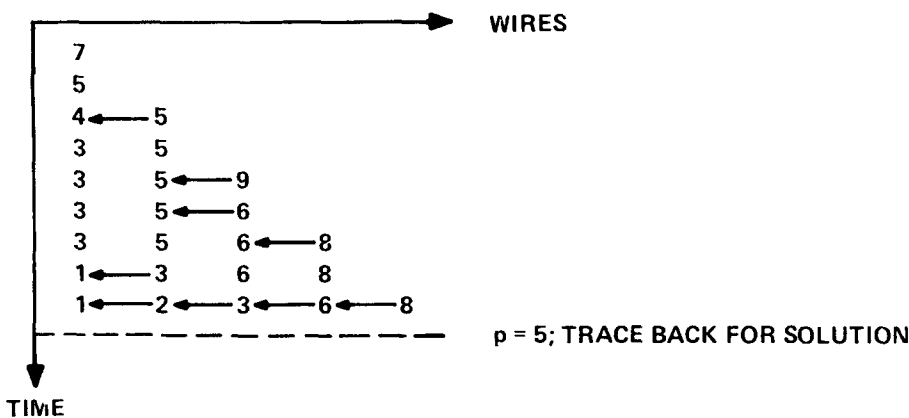


Fig. 4. Illustration of algorithm for fixed and loose terminals.

simply maintain a queue of elements of F_i that still need to be shifted. After processing all elements of F_i , before processing the first element of L_i , the queue is empty. Whenever an element of F_i is replaced in a partial solution, it is appended to the

queue. The elements in L_i and in the queue are processed in sorted order: at each step, we process the smallest of all elements in L_i and the queue. More formally, our algorithm works as follows.

ALGORITHM Maximum Alignment of Fixed and Loose Terminals

```

w(0): = 0;
last: = 0;
queue of wires from fixed terminals qf: = empty;
FOR all i FROM 1 TO k DO {process  $F_i$  and  $L_i$ }
  FOR all bottom terminals of wires  $t$  of  $F_i$  FROM left TO right DO
    IF  $w(last) < t$ 
      THEN  $last: = last + 1; j: = last$ 
      ELSE  $j: = \min\{j \mid 1 \leq j \leq last, w(j) > t\};$ 
     $w(j): = t;$ 
     $\ell(t): = w(j - 1);$ 
  ql: = queue of wires from  $L_i$ , sorted according to the order of their top terminals from left to
  right;
  WHILE ql  $\neq$  empty DO
     $t: =$  first (ql), removed from queue;
    IF  $w(last) < t$ 
      THEN  $last: = last + 1; j: = last$ 
      ELSE  $j: = \min\{j \mid 1 \leq j \leq last, w(j) > t\};$ 
      IF  $w(j)$  comes from  $F_i$  THEN append  $w(j)$  to qf;
     $w(j): = t;$ 
     $\ell(t): = w(j - 1);$ 
    WHILE qf  $\neq$  empty DO {sequence of shifts}
       $t: = \min\{\text{first}(\textit{ql}), \text{first}(\textit{qf})\}$ , removed from queue;
       $j: = j + 1;$ 
      IF  $j \leq last$ 
        THEN IF  $w(j)$  comes from  $F_i$  THEN append  $w(j)$  to qf;
        ELSE  $last: = last + 1;$ 
         $w(j): = t;$ 
         $\ell(t): = w(j - 1)$ 
    {at this point, the solution can be reconstructed following the links :}
  FOR all elements  $w(i)$  in  $w$  FROM  $w(last - 1)$  BACK TO  $w(1)$  DO
     $w(i): = \ell(w(i + 1))$ 
  {the solution is now stored in  $w$ }

```

END of Algorithm.

We conclude the following.

Theorem 2. *A solution to the maximum alignment problem with fixed and loose terminals can be constructed in $O(p + (n - p)\ell g(p + 1))$ time using $O(n)$ space, where n is the total number of wires involved*

and p is the size of the solution, which is optimal in the worst case.

Proof. We sketch only those parts of the argument that are distinct from the proof of Theorem 1. An increase in the size of the biggest partial solution

found so far can result not only from a wire that is directly appended to the sequence, but also from a loose wire that causes a series of shifts of fixed wires to the right. In this case, a search for the loose wire's position may have been carried out. If so, the time for the search is bounded by $O(\ell g|F_i|)$ when the exponential search technique is applied, because the fixed wires being shifted must occupy consecutive positions at the right end of w . If the size of the solution is increased by more than one by shifting wires, then at most one search for a position has taken place, and the other wires have been accumulated in the queue during the scan of wires from F_i . Hence, for each $i = 1, \dots, k$, at most $O(\ell g|F_i|)$ extra steps are carried out for increasing the size of the solution, the sum $\sum_{i=1}^k \ell g|F_i|$ being bounded by $O(n)$. The cost for finding all p wires in the solution is consequently bounded by $O(n + p)$. For all wires not belonging to the solution, we may be forced to carry out a search for the position in the sequence of wires, amounting to $O((n - p)\ell g(p + 1))$ time. The scan through the wires coming from F_i needs at most $|F_i|$ steps, totaling to $O(n)$ for all F_i and sorting the wires of L_i according to their top terminals' positions for each i requires only $O(n)$ steps if the union of all L_i is sorted by bucket sort, and then the sorted sequence is scanned and partitioned into the subsequences L_i . Hence, the overall runtime of the algorithm is as claimed. The space bound follows trivially. Optimality carries over from Theorem 1. \square

6. Discussion

Note that our algorithms implicitly compute all optimal solutions. A compressed representation of all of them, e.g., as a directed acyclic graph, where each node corresponds to a wire, all of them being distinct, and each path from a source node to a sink corresponds to an optimal solution, can easily be constructed in time proportional to the number

of distinct wires in all solutions.

Observe, furthermore, that our algorithms not only combine the advantages of [4,5,6], but also behave linear within a wider range of p , compared to [6], where only while $p \geq n - c$ for a constant c linear behavior is guaranteed; in our case, $p \geq n - n/\ell g n$ is already sufficient.

Finally, it should be pointed out that the maximum independent set problem for permutation graphs studied in [1] can now be solved in $O(p + (n - p)\ell g p)$ time by our algorithm, where p is the output size.

Acknowledgment

We are grateful to an anonymous referee for pointing out Fredman's work to us, and for suggesting improvements in the presentation of the paper.

References

- [1] S. Even, A. Pnueli and A. Lempel, Permutation graphs and transitive graphs, *J. ACM* 19 (3) (1972) 400-410.
- [2] M.L. Fredman, On computing the length of longest increasing subsequences, *Discrete Math.* 11 (1) (1975) 29-35.
- [3] I. Gopal, D. Coppersmith and C.K. Wong, Optimal wiring of movable terminals, *IEEE Trans. Comput.* C-32 (9) (1983) 845-858.
- [4] D.S. Hirschberg, Algorithms for the longest common subsequence problem, *J. ACM* 24 (4) (1977) 664-675.
- [5] J.W. Hunt and T.G. Szymanski, A fast algorithm for computing longest common subsequences, *Comm. ACM* 20 (5) (1977) 350-353.
- [6] N. Nakatsu, Y. Kambayashi and S. Yajima, A longest common subsequence algorithm suitable for similar text strings, *Acta Informatica* 18 (1982) 171-179.
- [7] M.D.F. Schlag, L.S. Woo and C.K. Wong, Maximizing pin alignment by pin permutations, *INTEGRATION, the VLSI journal* 2 (4) (1984) 279-307.
- [8] P. Widmayer and C.K. Wong, Optimizing pin alignment in VLSI circuit layout, *IBM Research Report*, 1984.
- [9] E. Yoffa, P. Hauge, M.D.F. Schlag and C.K. Wong, Improving cascode-switch macro wireability, *IBM Research Report*, 1984.