1984

Optimal Placement for Multi-Terminal Nets

Mikhail J. Atallah
Purdue University, mja@cs.purdue.edu

Susanne E. Hambrusch
Purdue University, seh@cs.purdue.edu

Report Number:
84-502

Optimal Placement for Multi-Terminal Nets

Mikhail J. Atallah
Susanne E. Hambrusch

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

November 1984

TR-502

Abstract

In the general multi-terminal placement problem we have to place the terminals of n multi-terminal nets in any of a number of given positions in a channel. The positions for the terminals must be chosen so that the connections made by a routing algorithm (which follows the placement) require a channel of minimum width. While the general multi-terminal placement problem is NP-hard, we show that a number of variants have efficient placement algorithms. For the variant in which the upper row positions of the nets have already been determined, the efficiency of our algorithm is based on a new combinatorial characterization of the optimal placement.

Key Words

Analysis of algorithms, channel routing, data structures, density, design automation, multi-terminal nets, placement.

This work was supported by the Office of Naval Research Contract N00014-84-K-0502.

1. Introduction

A common approach in dealing with computationally intractable problems arising in layout design is to partition them into simpler subproblems that can be handled more efficiently. One such subproblem is the problem of placing and connecting sets of terminals in a channel [HS,R]. In this paper we consider a number of placement problems in a channel in which we are given the freedom of placing each terminal on any one of a number of given positions. The positions for the terminals must be chosen so that the connections made by a routing algorithm following the placement require a channel of minimum width.

Assume that each one of positions $p_1, \cdots, p_s$ on the upper row of a channel and each one of positions $q_1, \cdots, q_m$ on the lower row can be used to place a terminal, $1 \leq p_1 < p_2 < \cdots p_s \leq M$, $1 \leq q_1 < q_2 \cdots < q_m \leq M$. We are given n nets, where the i-th net consists of u_i *entry terminals* and k_i *exit terminals*. We have to place the entry terminals on distinct positions on the upper row, the exit terminals on distinct positions on the lower row, such that the resulting multi-terminal channel routing problem (CRP) can be routed using minimum channel width. We call this problem the Multi-Terminal Placement (MTP) problem. Problems of this nature arise when modules to be connected can be placed on a number of specified positions along a channel, or when a number of terminals on a functional block must be connected to terminals on a driver module and it does not matter which [GCW,P].

It is not hard to show that the MTP-problem defined above is NP-hard. The main contribution of this paper is to present efficient algorithms for two variants of the MTP-problem. In the first variant (which we call MTP1-problem), all the entry terminals have already been positioned, and all the k_i's are equal. We show how to determine an optimal placement of exit terminals in $O(s + (n+m)\log(n+m))$ time. The efficiency of our algorithm is based on a technique which makes use of an elegant and somewhat surprising combinatorial characterization of an optimal placement. The second variant of the MTP-problem for which we give an efficient solution is that where the k_i's are equal, the u_i's are equal, and both entry and exit terminals have to be placed. We call this the MTP2-problem. We present an algorithm

for the MTP2-problem that generates an optimal placement in $O((s+m)\log\hat{d})$ time, $\hat{d} \leq n$ ($\hat{d}$ will be defined shortly).

We now give a more precise definition of the problems considered in this paper, and make some preliminary observations. In the MTP1-problem we are given, in addition to the lower positions $q_1, \cdots q_m$ and an integer k, sets $P_1, \cdots, P_n$, which are a partition of the set $\{p_1, \cdots, p_s\}$ of positions on the upper row, $m \geq kn$. P_i contains the positions of the entry terminals of net i; $P_i = \{p_i^1, p_i^2, \cdots, p_i^{u_i}\}$ with $p_i^1 < \cdots < p_i^{u_i}$, $1 \leq i \leq n$. The problem is to assign each one of the k exit terminals of each net to a lower position, i.e. we must compute sets $Q_1, \cdots, Q_n$ where set $Q_i = \{q_i^1, \cdots, q_i^k\}$ contains the k lower positions assigned to net i, $q_i^1 < \cdots < q_i^k$. The assignment should be such that the multi-terminal CRP consisting of the nets (P_i, Q_i) can be wired using minimum channel width over all other assignments. Since in a number of routing models the width required by a routing algorithm is proportional to the *density* of the CRP [CL, H, PL, RBM], the density is the cost measure to be minimized by our algorithms. The density d of a multi-terminal CRP is defined as follows. Let $x_i = \min\{p_i^1, q_i^1\}$ and $y_i = \max\{p_i^{l_i}, q_i^k\}$, then d is the maximum over all c of the number of pairs (x_i, y_i) for which $x_i \leq c < y_i$ or $x_i > c \geq y_i$ [SP]. Throughout, we use $\hat{d}$ to denote the optimal density, i.e. the smallest density over all assignments.

For the MTP1-problem we can, without loss of generality, assume that every P_i contains at most two upper positions: The leftmost upper position p_i^1 (also called l_i), and the rightmost upper position $p_i^{u_i}$ (also called r_i). Henceforth we consider every P_i to be an *interval* rather than a set, and initially interval P_i extends from position l_i to position r_i. Throughout, we use the words "net i" and "interval i" interchangeably. Assigning to P_i lower positions that are between l_i and r_i does not increase the initial density. Of course, it will not always be possible to assign lower positions in this fashion. See Figure 1.1 for a solution to an MTP1-problem; the intervals are indicated by solid lines, the assignments by dashed lines.

In this paper we present a characterization of the optimal density $\hat{d}$ which is a generalization of the one introduced for 2-terminal nets in [AH]. This

characterization allows us to compute $\hat{d}$ in $O(s+m)$ time. We present an algorithm that, given $\hat{d}$, generates in $O((n+m)\log(n+m))$ time the assignment of lower positions to intervals that results in a CRP of density $\hat{d}$. We also show that for a special case of the MTP1-problem, which includes 2-terminal nets, an optimal assignment can be produced in $O(n+m)$ time.

In the MTP2-problem the positions of both the entry and the exit terminals have not yet been determined, and every net i consists of u entry terminals and k exit terminals, $s \geq un$, $m \geq kn$. The characterization of the optimal density used in the MTP1-problem for precomputing $\hat{d}$ cannot be used in the MTP2-problem, but the freedom of selecting the entry terminal positions does allow us to generate nets of a special structure. We present an algorithm for the MTP2-problem that generates a CRP of optimal density $\hat{d}$ in $O((s+m)\log\hat{d})$ time.

This paper is organized as follows. In Section 2 we define a quantity Δ, which we show to be a lower bound on the best achievable density of any solution to an MTP1-problem. We describe how to compute Δ in $O(s+m)$ time. Sections 3 and 4 describe an assignment algorithm, which, given Δ, determines in $O((n+m)\log(n+m))$ time the k lower positions to be assigned to every interval so that the resulting CRP has density Δ. In section 5 we prove the correctness of our assignment algorithm, thus establishing that the optimal density $\hat{d}$ equals Δ. Section 6 contains the result about the MTP2-problem and the proof of NP-hardness of the MTP-problem.

2. Characterization of the Optimal Density for the MTP1-problem

As discussed in Section 1, we can view the input to an MTP1-problem as consisting of m lower positions and n intervals $P_i = (l_i, r_i)$, where l_i is the left endpoint (which corresponds to the leftmost upper position of net i) and r_i is the right endpoint (which corresponds to the rightmost upper position of net i), $l_i \leq r_i$, $1 \leq i \leq n$. Preprocessing the problem to put it in this form can be done in $O(s)$ time. We now have to assign to every interval P_i k lower positions (which are the positions for P_i's k exit terminals), such that the resulting multi-terminal CRP has minimum density (the density is defined as in Section 1).

We say that interval i *crosses column* x if and only if it contains the point at $x+0.5$. Let $c(x)$ be the number of intervals that cross column x; i.e., the number of intervals P_i with $l_i \le x < r_i$. Let $up(x,y)$ be the number of intervals with $x \le l_i \le r_i \le y$, and let $down(x,y)$ be the number of lower positions that are $\ge x$ and $\le y$. We next define a quantity Δ, which we later show to be equal to the optimal density.

Definition 2.1 $\Delta = \max \{ \max_x (c(x) + up(1,x) - \lfloor down(1,x)/k \rfloor),$

$\max_{x,y} (\lceil c(x) + c(y) + up(x+1,y) - \lfloor down(x+1,y)/k \rfloor) /2 \rceil),$

$\max_x (c(x) + up(x+1,M) - \lfloor down(x+1,M)/k \rfloor) \}.$

Note that for $x=y$ the middle term equals $c(x)$.

Lemma 2.2 $\hat{d} \ge \Delta$.

Proof: We only go through the argument that $\hat{d}$ is no smaller than the middle term in Definition 2.1 (the arguments for the other two terms are similar). The number of intervals between positions x and y that are crossing neither column x nor column y is $up(x+1,y)$. At least $up(x+1,y) - \lfloor down(x+1,y)/k \rfloor$ intervals have to be assigned lower positions that are $\le x$ or $> y$. Thus, at least $c(x)+c(y)+up(x+1,y)-\lfloor down(x+1,y)/k \rfloor$ nets have to cross column x or column y. Dividing this number equally between columns x and y is certainly a lower bound on $\hat{d}$. □

The next lemma states that Δ can be computed efficiently.

Lemma 2.3 If the intervals are given both sorted according the left endpoints and sorted according to the right endpoints, then Δ can be computed in $O(n+m)$ time.

(The proof is straightforward and is omitted.)

3. A Special Case

In this section we describe some of important concepts used in the algorithm for the MTP1-problem, whose complete description is given in the next section. We do so by considering a special version of the MTP1-problem, which is defined next.

Note that, throughout, we assume that the intervals $P_i=(l_i,r_i)$ are given sorted according to their right endpoints; i.e., $r_1<r_2<\cdots<r_n$, $1\le i\le n$.

Definition 3.1 Let P_i and P_j be two intervals. We say that $P_i<P_j$ iff $l_i<l_j$ and $r_i<r_j$. We say that $P_i\subset P_j$ iff $l_j<l_i\le r_i<r_j$ (i.e., interval P_j encloses interval P_i).

Note that if $r_i<r_j$ then either $P_i<P_j$ or $P_i\subset P_j$.

The special case of the MTP1-problem which is considered in this section is when $P_1<P_2<\cdots<P_n$. The algorithm we give determines an assignment achieving density Δ in $O(n+m)$ time. Note that this special case includes 2-terminal nets. The assignment algorithm for the general case of the MTP1-problem (i.e., when $P_i\subset P_j$ is possible) runs in $O((n+m)\log(n+m))$ time, is considerably more complex, and is left till the next section.

A basic operation in our algorithm is the *left* (resp. *right*) *extension* of an interval. When a lower position q_j to the left of interval P_i is assigned to P_i, position q_j becomes the new left endpoint of interval P_i, and we say that the algorithm has performed a left-extension of P_i to position q_j. Intuitively, it 'stretches' the interval so that it now starts at q_j. The effect of such a left-extension is that the density in every column between and including column q_j and the interval's old left endpoint, is increased by 1. A right-extension is defined similarly. However, even though intervals may expand during the assignment algorithm, the functions $c(x)$, $up(x,y)$, and $down(x,y)$ defined in Section 2 are defined on the initial endpoints; i.e., the endpoints before the assignment algorithm started. The relations $<$ and $\subset$ of Definition 3.1 are also relative to the initial situation. During the algorithm, the intervals that cross a column x are those that crossed it initially ($c(x)$ of them) plus those that cross column x because they were extended.

Definition 3.2 Let P_i and P_j be two intervals with $P_i<P_j$. We say that the *ordered property* holds for P_i and P_j if for every lower position q_α assigned to P_i and for every lower position q_β assigned to P_j, $q_\alpha<q_\beta$ holds.

Among all assignments achieving density Δ there exists one in which the ordered property holds for every pair of intervals P_i and P_j with $P_i < P_j$ (the proof of this depends on the fact that $k_i = k_j = k$, and is left to the reader). When $P_1 < \cdots < P_n$ (as is the case in this section), this implies that the k lower positions assigned to an interval P_i can be chosen so that they are consecutive (i.e., the k lower positions assigned to P_i are $q_j, \cdots q_{j+k-1}$ for some j). We call an interval that has k lower positions assigned to it a *satisfied* interval. The algorithm terminates when all n intervals become satisfied.

In the assignment algorithm we simultaneously scan, from left to right, the list of intervals and that of the lower positions. Recall that the intervals are sorted according to their right endpoints and that $q_1 < \cdots < q_m$. Assume the scan is currently at interval P_{up} in the list of intervals and at q_{down} in the list of lower positions. We distinguish two cases:

(i) If $q_{down} < l_{up}$ and density Δ would be exceeded if P_{up} were left-extended to q_{down}, the algorithm continues with P_{up} and q_{down+1}. It is not hard to show that, if density Δ were exceeded, this would happen in column q_{down}, and possibly other columns to the right of column q_{down} (the proof of this is omitted).

(ii) If $q_{down} \geq l_{up}$, or if $q_{down} < l_{up}$ and P_{up} can be left-extended to position q_{down} without causing the density to exceed Δ, then the algorithm assigns the k lower positions, $q_{down}, \cdots, q_{down+k-1}$, to P_{up}. It then sets $up = up + 1$ and $down = down + k$, and continues with the new P_{up} and q_{down}.

In (ii), assigning $q_{down}, \cdots q_{down+k-1}$ to P_{up} would not be possible if there are either not enough lower positions (i.e., $down + k - 1 > m$) or if, in the case that $q_{down+k-1} > r_{up}$, a right-extension of P_{up} to $q_{down+k-1}$ would result in density $> \Delta$. As we will show in Section 5, none of these two failure possibilities can occur.

We next describe how to determine in $O(1)$ time whether or not density Δ would be exceeded if P_{up} were left-extended to q_{down}. To do so, the algorithm uses a variable *dens* which contains the current density of column q_{down}. Note that this density is made up by intervals that crossed column q_{down} initially, and by intervals that cross column q_{down} because they were left-extended. The algorithm also uses a

pointer that is positioned in the list of intervals at the interval P_s with $r_s > q_{down}$ and $r_{s-1} \leq q_{down}$.

If $dens = \Delta$, then no left-extension of P_{up} to q_{down} can be done, and the algorithm updates *dens* before continuing with q_{down+1} and P_{up}; i.e., as long as $r_s \leq q_{down+1}$, it sets $s = s+1$ and $dens = dens - 1$.

If $dens < \Delta$ (i.e., the left-extension can be done), the algorithm sets $dens = dens + 1$ and an updating of variable *dens* similar to the one described above computes the current density in column $q_{down+k-1}$.

Recall that we need a 'density check' only for left-extensions, since we show in section 5 that right-extensions are always safe. This concludes the description of our $O(n+m)$ time algorithm for the MTP1-problem when $P_1 < P_2 < \cdots < P_n$.

4. An Assignment Algorithm for the MTP1-Problem

We now return to the general case of the MTP1-problem in which intervals can enclose other intervals (i.e., $P_i \subset P_j$ is possible). This section describes an algorithm that generates an assignment of density Δ in $O((n+m)\log(n+m))$ time (the correctness of the algorithm is proved in Section 5).

As we pointed out in the previous section, there exists an optimal assignment in which the ordered property holds for every P_i and P_j with $P_i < P_j$. But if $P_i \subset P_j$, this is no longer true, since now it is possible that lower positions that are $< l_i$ or $> r_i$ cannot be assigned to P_i, while they can be assigned to P_j.

Definition 4.1 Let P_i and P_j be two intervals with $P_i \subset P_j$. The *mixed property* holds for P_i and P_j if there exist two lower positions q_α and q_γ assigned to P_j, and two lower positions q_β and q_δ assigned to P_i, and $q_\alpha < q_\beta < q_\gamma < q_\delta$.

There always exists an optimal solution in which, for every pair of intervals, the mixed property does not hold. This can be seen by noting that, if the mixed property holds for P_i and P_j, then we can always assign q_α and q_β to P_j, and q_γ and q_δ to P_i without increasing the density. Doing this repeatedly gives an optimal solution of the desired type. Our algorithm produces a solution in which

- for every pair of intervals $P_i \subset P_j$, the mixed property does not hold, and
- for every pair of intervals with $P_i < P_j$, the ordered property holds.

Our algorithm consists of a procedure MAIN_SCAN, which simultaneously scans, from left to right, the list of the lower positions and that of the intervals (which, we recall, are given sorted according to their right endpoints). This algorithm differs from the one described in Section 3 in a number of ways:

1) When $P_i \subset P_j$, the scan reaches P_i before it reaches P_j. Suppose the algorithm is unable to assign q_l to P_i: Unlike Section 3, we cannot discard q_l from further consideration, since at a later stage the scan may be able to assign it to P_j.

2) In the density check (i.e., deciding whether or not density Δ would be exceeded if interval P_i were left-extended to q_j) the density Δ can now be exceeded in a column between q_j and l_i without being exceeded in column q_j.

3) When the algorithm of Section 3 assigns a lower position q_l to an interval, the assignment is final. Under some circumstances, the algorithm of this section 'steals' lower positions that were already assigned to an interval P_i from P_i and assigns these lower positions to another interval P_j, $i < j$, in order to achieve the optimal density (the situation in which such stealing occurs is discussed later on).

We start by describing how to determine the lower positions that are $\leq r_i$ and that can be assigned to interval P_i (but could not be assigned to intervals reached earlier in the scan). In order to do so, MAIN_SCAN calls a procedure TRY_LEFT. Intuitively, one purpose of TRY_LEFT is to scan the list of still available lower positions from left to right and to find the leftmost q_j so that P_i can be left-extended to q_j without exceeding density Δ. However, in the actual implementation, q_j is determined without performing a scan (since this would be too time-consuming). We later show how position q_j can be found in $O(\log(n+m))$ time. We next describe procedure TRY_LEFT in more detail.

Procedure TRY_LEFT

Input: interval P_i.

Output: an integer μ_i and, if $\mu_i < k$, a lower position q_α.

Effect: TRY_LEFT assigns to P_i as many lower positions $\leq r_i$ as it can without causing the density to exceed Δ. The lower positions assigned to P_i are as far to the left of r_i as possible. Integer μ_i, which it returns, is the number of lower positions P_i still needs in order to be satisfied. The rightmost lower position assigned to P_i by TRY_LEFT is q_α (note that such a q_α exists only if $\mu_i < k$).

Method: In order to do the above, TRY_LEFT first determines the leftmost available lower position q_j, $q_j \leq r_i$, such that P_i can be left-extended to q_j without exceeding density Δ. Note that such a q_j may not exist, or that we may have $q_j \geq l_i$. The integer μ_i to be returned is determined as follows. If no such lower position q_j exists, set $\mu_i = k$ and then return μ_i. Otherwise let k^* be the number of unassigned lower positions between (and including) q_j and r_i. Assign the leftmost $\min(k, k^*)$ of these lower positions to P_i and let q_α be the rightmost one of the lower positions assigned to P_i. TRY_LEFT returns the value $\mu_i = k - min(k, k^*)$ and q_α to MAIN_SCAN. We will later describe how TRY_LEFT can be implemented to run in $O(\log(n+m)+k^*)$ time.

This completes the description of TRY_LEFT.

We continue to outline algorithm MAIN_SCAN. Assume that MAIN_SCAN is currently at interval P_{up} in the list of intervals and at lower position q_{down} in the list of lower positions. No lower positions have yet been assigned to P_{up}, and q_{down} is still available. During MAIN_SCAN, $q_{down} \leq r_{up}$ will hold. Every interval whose right endpoint is to the left of q_{down} has either been satisfied, or, if such an interval still requires lower positions, then it has been put in a set S. Set S is an auxiliary structure used by MAIN_SCAN; and the intervals in S will eventually be satisfied by being assigned lower positions that are $\geq q_{down}$.

The strategy used by MAIN_SCAN is motivated by the following observation. Let P_i be an interval in set S that still requires μ_i lower positions. Assume, for example, that $\mu_i < k$, $r_{up} = q_{down}$ and $P_i \subset P_{up}$. Since it is possible that, in the optimal

solution, interval P_{up} uses lower positions that MAIN_SCAN was unable to assign to P_l, MAIN_SCAN calls TRY_LEFT(P_{up}). Observe that if P_{up} still requires some lower positions after TRY_LEFT (i.e., $\mu_{up}>0$), then the density is minimized and the mixed property avoided by letting P_{up} 'steal' from P_l some (or all) lower positions already assigned to P_l. After this lower position 'redistribution' P_l requires more lower positions than before, while P_{up} is closer to being satisfied (or is already satisfied).

We now give a more formal description of MAIN_SCAN. The entries in set S are pairs (P_l,μ_l), where P_l is an interval that still requires μ_l lower positions in order to be satisfied. Initially set S contains all the intervals P_l with $r_l<q_1$, and $\mu_l=k$. Pointer up is initially positioned at the first interval in the list whose right endpoint is $\geq q_1$, while $down$ is initially equal to 1. Thus, the condition $q_{down}\leq r_{up}$ is satisfied. Assume that, when MAIN_SCAN is at P_{up} and q_{down}, $S = \{(P_{l_1},\mu_{l_1}), (P_{l_2},\mu_{l_2}), \cdots ,(P_{l_h},\mu_{l_h})\}$, where P_{i_1}, ..., P_{i_h} are intervals whose right endpoints are to the left of q_{down} (and thus to the left of r_{up}). The intervals P_{i_1}, ..., P_{i_h} still require (respectively) μ_{l_1}, ..., μ_{i_h} lower positions. P_{i_1} is always the interval of S with the smallest left endpoint among all intervals in S. During the algorithm the following condition holds: Only P_{l_1} can require fewer than k lower positions, all the other intervals in S require exactly k lower positions; i.e, $\mu_{i_1}\leq k$ and $\mu_{l_2} = \cdots = \mu_{l_h} = k$. Note that this condition is satisfied after the initialization of S. The purpose of the above condition is to ensure that we get an assignment in which for every column x at most one interval P_l with $r_l\leq x$ is assigned both lower positions that are $>x$, and lower positions that are $\leq x$. We will show in Section 5 that among all the assignments achieving density Δ there is one with this property.

When MAIN_SCAN is positioned at P_{up} and q_{down} it first decides what to do with lower position q_{down}, and it distinguishes 2 cases:

A1) Set S is empty: P_{up} requires exactly k lower positions, and MAIN_SCAN calls TRY_LEFT(P_{up}). If it returns μ_{up} and q_α with $q_\alpha>q_{down}$, then set $down=\alpha$ (since all the lower positions between and including q_{down} and q_α have been

taken). If, in addition, $\mu_{up} > 0$, then add (P_{up}, μ_{up}) to set S.

A2) Set S is not empty: MAIN_SCAN assigns q_{down} to P_{i_1}. Recall that $r_{i_1} < q_{down}$ holds. (In our proof we will show that this assignment never causes the density to exceed Δ.) MAIN_SCAN then sets μ_{i_1} to $\mu_{i_1} - 1$. If μ_{i_1} is now zero (i.e., P_{i_1} is now satisfied), it removes P_{i_1} from S and choses the interval with the smallest left endpoint currently in S as the new P_{i_1} (choosing the new P_{i_1} in this way helps insure that the generated solution satisfies the ordered property).

TRY_LEFT does not alter the position of pointer *up*, but it possibly advances the position of pointer *down* to position α in case A1. After having executed either case A1 or case A2 described above, MAIN_SCAN checks whether there are any right endpoints of intervals between q_{down} and q_{down+1}. Assume there are l intervals, namely $P_{up}, P_{up+1}, \cdots, P_{up+l-1}$, whose right endpoints fall between q_{down} and q_{down+1}; i.e., $q_{down} \leq r_{up+i} < q_{down+1}$ for $i = 0, \cdots, l-1$. (None of these P_i's is in S.) Each one of these intervals is considered in turn, and for every P_{up+i}, $i = 0,1, \cdots, l-1$, MAIN_SCAN distinguishes 3 cases.

B1) The set S is empty: Call TRY_LEFT(P_{up+i}) and, if the returned value μ_{up+i} is > 0, add (P_{up+i}, μ_{up+i}) to S.

B2) Set S contains an interval P_j with $P_j < P_{up+i}$: Add (P_{up+i}, k) to S. Since P_j is not yet satisfied, TRY_LEFT(P_{up+i}) cannot assign any lower positions to P_{up+i} (otherwise P_j would already have taken them).

B3) Set S is not empty, and $P_j \subset P_{up+i}$ for every interval $P_j \in S$: Call TRY_LEFT (P_{up+i}), and if it returns $\mu_{up+i} > 0$ then do the following. If $\mu_{i_1} = k$, no 'stealing' of lower positions can occur, and the algorithm adds (P_{up+i}, μ_{up+i}) to S. Otherwise, P_{i_1} has been assigned lower positions that are $\leq q_{down}$ and will be assigned at least one which is $> q_{down}$. In order to guarantee that the mixed property does not occur, P_{up+i} 'steals' from P_{i_1} $\min(\mu_{up+i}, k - \mu_{i_1})$ lower positions assigned to P_{i_1}. This causes μ_{i_1} to increase to $\mu_{i_1} + \min(\mu_{up+i}, k - \mu_{i_1})$. If, after assigning these lower positions to P_{up+i}, $\mu_{up+i} > 0$ (in which case $\mu_{i_1} = k$), then add

(P_{up+i},μ_{up+i}) to S as *new* first element. Again, only one element of S, namely (P_{l_1},μ_{l_1}) has $\mu_{l_1} < k$; i.e., has already been assigned some lower positions.

After the intervals P_{up}, $P_{up+1}, \cdots P_{up+l-1}$ have been processed as described above, P_{up+l} is the interval with the smallest right endpoint that is $\geq q_{down+1}$. Set $up = up + l$, $down = down + 1$, and continue MAIN_SCAN with the new P_{up} and q_{down}.

To illustrate how the algorithm works, we describe how it produces the solution shown in Figure 1.1. TRY_LEFT assigns q_1 and q_2 to P_1 and q_3 and q_4 to P_2. It then fails to assign q_5 and q_6 to P_3, assigns q_7 to P_3, and puts $(P_3,1)$ into S. When reaching r_4, q_6 is assigned to P_4 in the TRY_LEFT(P_4), and a stealing of lower positions occurs: q_7 is now assigned to P_4 and $S = \{(P_3,2)\}$. Then, q_8 and q_9 are assigned to P_3. At q_9, TRY_LEFT assigns q_5 to P_5. Finally, q_{10} is assigned to P_5, and q_{11} and q_{12} to P_6.

We now outline how the assignment algorithm described above can be implemented to run in $O((n+m)\log(n+m))$ time. Recall that the input consists of the intervals sorted according to the right endpoints and the list of lower positions, in sorted order. One of the auxiliary data structures used during the algorithm is a 2-3 tree [AHU] T whose leaves contain, in sorted order, the endpoints of the intervals and the available lower positions. Thus, initially, T contain $2n+m$ leaves. The leaves corresponding to lower positions are joined together by a doubly linked list. The nodes of T have, besides the standard 2-3 tree entries, additional entries that allow us to perform in $O(\log(n+m))$ time each one of the following operations.

(*i*) Determine the leftmost available q_j so that P_i can be left-extended to q_j, $q_j \leq r_i$ without exceeding density Δ. Recall that this operation is needed in TRY_LEFT(P_i).

(*ii*) Add/delete the interval (x,y) to/from the tree.

(*iii*) Delete lower position q_j from the tree.

The first operation is implemented by simulating the effect on the density when the interval $(1,l_i)$ is inserted into T. If the density increases to $\Delta+1$, we find the rightmost column y in which the density would be $\Delta+1$, and let q_j be the leftmost available lower position at a position $\geq y+1$. If no such column y exists (i.e., P_i can

be left-extended to position 1 without exceeding density Δ), q_j is the leftmost available lower position currently in the list. It should be clear that, in a 2-3 tree in which every interior node v contains additional entries to record information about the density created by the intervals with both endpoints in v's subtree, this operation can be implemented to run in $O(\log(n+m))$ time. When performing the second operation we need to update entries in the interior nodes of the tree to record the changes in the density caused by the added/deleted interval. Note that when an interval P_i is left-extended (resp. right-extended) to position x, we delete the interval (l_i, r_i) and add the interval (x, r_i) (resp. (l_i, x) in order to maintain a tree of height $O(\log(n+m))$. The details of the implementation of all three operations are straightforward and are left to the reader.

The data structure used to implement set S is a heap in which the interval with the smallest left endpoint currently in S, which is P_{i_1}, is the top element of the heap. Thus, the query 'Is there an interval $P_j \in S$ with $P_j < P_{up+i}$', which is needed in case B2, is answered in constant time by inspecting the top of the heap. Note that if the above query is false and S is not empty, then $P_j \subset P_{up+i}$ holds for every interval $P_j \in S$. Insertion and deletions of intervals are done in $O(\log n)$ time in a heap, and every interval is inserted and deleted at most once. Thus, the $O((n+m)\log(n+m))$ time bound of the assignment algorithm follows. In the next section we prove that this algorithm always generates an assignment of density Δ.

5. The Assignment Algorithm Achieves Density Δ

In this section we show that the assignment algorithm for the MTP1-problem described in Section 4 always generates a solution of density Δ. We start off with two Lemmas that state properties of the solution generated by the algorithm and the following definition.

Definition 5.1 Let $P_i = (l_i, r_i)$ be the initial interval. Then P_i is to the left of a position x if $x \geq r_i$, and it is to the right of position x if $x < l_i$. A lower position q_j is to the left of x if $x \geq q_j$, and it is to the right of x if $x < q_j$.

Lemma 5.2 For every column x, at most one interval to the left of x is assigned by the algorithm lower positions both left and right of x.

Proof: Suppose to the contrary that P_i and P_j are to the left of x, that P_i is assigned q_α and q_β, $q_\alpha \leq x < q_\beta$, and that P_j $(j \neq i)$ is assigned q_γ and q_δ, $q_\gamma \leq x < q_\delta$. W.l.o.g. we assume that $l_i < l_j$. Assume first that $P_i < P_j$. Then, $q_\alpha \leq x < q_\beta$ and $q_\gamma \leq x < q_\delta$ contradicts the fact that the algorithm maintains the ordered property. (To put it differently, it cannot happen because the algorithm would have assigned q_γ to P_i).

We now consider the case when $P_j \subset P_i$. If $q_\gamma \leq r_i$, then having q_γ being assigned to P_j would have caused a stealing of q_γ by P_i (at B3 in MAIN_SCAN), which contradicts the fact that q_γ is now assigned to P_j. If $q_\gamma > r_i$, then the algorithm would assign k lower positions to P_i before it assigns any to P_j, and thus q_γ would not be assigned to P_j. □

The following Lemma gives another property of the solution produced by the assignment algorithm. Although it is not needed in the proof, it is stated for the sake of completeness.

Lemma 5.3 For every column x, at most one interval to the right of x is assigned by the algorithm lower positions both left and right of x.

Proof: Suppose to the contrary that P_i and P_j are to the right of x, that P_i is assigned q_α and q_β, $q_\alpha \leq x < q_\beta$, while P_j $(j \neq i)$ is assigned q_γ and q_δ, $q_\gamma \leq x < q_\delta$. W.l.o.g. we assume that $l_i < l_j$. We cannot have $P_i < P_j$ because it would contradict the fact that the algorithm maintains the ordered property. So suppose that $P_j \subset P_i$. The algorithm would initially assign q_α and q_γ to P_j. In order to have q_α assigned to P_i at least one stealing process must have occurred. But then the stealing process (at B3 in MAIN_SCAN) would also have taken q_γ away from P_j and given it to P_i, a contradiction. □

Lemma 5.4 Algorithm MAIN_SCAN succeeds in satisfying all intervals without exceeding density Δ.

Proof: We first show that when MAIN_SCAN assigns to interval P_i a lower position q_j which is to the right of P_i (i.e., $q_j > r_i$), density Δ is never exceeded. Suppose to the contrary that it is, and consider the first time during the algorithm it happens. Then, in some column x between r_i and q_j the density was already Δ before we assigned q_j to P_i (if there is more than one such column x then choose the leftmost one). We distinguish two cases: (In either case, we are looking at the situation just before q_j was assigned to P_i.)

Case A: There are no available lower positions to the left of x. Let Γ (resp. Θ) be the set of satisfied intervals that are to the left of x and that were not assigned any lower position to the right (resp. left) of x. Intervals that cross column x have not yet been assigned any lower positions (they have not even been considered yet). By Lemma 5.2 at most one interval to the left of x can be assigned lower positions both left and right of x. If there is such an interval P_s then it is satisfied, and P_i has not been assigned any lower position to the left of x (because of Lemma 5.2). Therefore $down(1,x) \leq k|\Gamma|+k-1$, where the term $k-1$ is due to the fact that P_s may have been assigned up to $k-1$ lower positions to the left of x. Moreover, we then have $\Delta = c(x)+|\Theta|+1$, and $up(1,x) \geq |\Gamma|+|\Theta|+2$. If no such P_s exists (i.e. no interval to the left of x has yet been assigned lower positions both left and right of x) then P_i may have been assigned up to $k-1$ lower positions to the left of x and therefore we again have $down(1,x) \leq k|\Gamma|+k-1$, while $\Delta = c(x)+|\Theta|$ and $up(1,x) \geq |\Gamma|+|\Theta|+1$. In either of the above two sub-cases it is easy to verify that

$$c(x) + up(1,x) - \lfloor down(1,x)/k \rfloor \geq \Delta+1,$$

which contradicts the definition of Δ.

Case B: There are available lower positions left of x. Let q_l be the rightmost one of them. Obviously q_l cannot be below P_i since otherwise it would have been assigned to P_i at an earlier stage of the algorithm. Also note that q_l cannot be between r_i and x since otherwise there is a column between r_i and x where the density is already Δ, contradicting the fact that x is the leftmost such column to the right of r_i. Therefore, $q_l < l_i$ must hold. Since q_l was not assigned to P_i (at an earlier stage of the algorithm), there exists a column y between q_l and l_i with density Δ

(if there is more than one such column then let y be the leftmost one).

We claim that lower positions to the right of y could not have been assigned to intervals whose left endpoint is to the left of y. The proof of this claim follows. First, we observe that no interval crossing column y has been assigned a lower position to the right of y (actually, to the right of q_f), because that would imply that a column $\bar{y}$ between q_f and y has density Δ, which contradicts our choice of y as leftmost. Next, we show that no interval to the left of y has been assigned a lower position to the right of y. To prove this, assume to the contrary that interval P_r is left of y and has been assigned a lower position to the right of y. P_r cannot be to the left of q_f because in that case q_f would have been assigned to it, a contradiction. For the same reason, P_r cannot cross column q_f. P_r cannot be between q_f and l_f either, since that would contradict our choice of y as leftmost (by a now familiar argument). Therefore no such P_r exists, i.e., no interval left of y has been assigned a lower position to the right of y. This completes the proof of the claim.

An additional consequence of our choice of y as leftmost is that no interval between y and x was assigned any lower positions left of y (the argument proving this should by now be familiar to the reader and is omitted). These observations imply that every lower position which is $>y$ and $\leq x$ (and which is not available) has been assigned to an interval which is either

(i) a satisfied interval whose k lower positions are $>y$ and $\leq x$, or

(ii) a satisfied interval P_s which has been assigned lower positions both to the right and left of x, or

(iii) interval P_f itself.

Let there be Γ intervals as described in case (i). Because of Lemma 5.2 there can only be one interval P_s as described in case (ii), and cases (ii) and (iii) are mutually exclusive. If case (ii) holds then P_f has not been assigned any position between y and x, whereas if case (ii) does not hold then P_f may have been assigned up to $k-1$ lower positions that are $>y$ and $\leq x$. In either situation we have $down(y+1,x) \leq k|\Gamma|+k-1$. If we let A (resp. B) be the set of intervals between y and x that have been assigned all of their lower positions from the left of y (resp.

right of x) then we have the following:

$$\Delta = c(y)+|A|,$$
$$\Delta = c(x)+|B|+1 \quad \text{if case (ii) holds,}$$
$$\Delta = c(x)+|B| \quad \text{if case (iii) holds,}$$
$$up(y,x) = |A|+|\Gamma|+|B|+2 \quad \text{if case (ii) holds,}$$
$$up(y,x) = |A|+|\Gamma|+|B|+1 \quad \text{if case (iii) holds.}$$

In either case (ii) or (iii) the following holds:

$$c(y) + c(x) + up(y+1,x) - \lfloor down(y+1,x)/k \rfloor \geq 2\Delta+1,$$

which contradicts the definition of Δ. This completes the proof that whenever MAIN_SCAN joins an interval to an lower position to its right then this does not cause the density to exceed Δ.

This in itself does not guarantee the success of MAIN_SCAN since it does not rule out the possibility that MAIN_SCAN may run out of lower positions. We now show that this cannot happen. Assume to the contrary that MAIN_SCAN runs out of lower positions (i.e. *down* becomes $m+1$) at a time when c, $0 \leq c < k$, lower positions have been assigned to P_i. When this happens, all the remaining available lower positions are to the left of P_i. Let q_j be the rightmost one of them (i.e. the rightmost lower position still available). Let x be the leftmost column in which density Δ is exceeded when P_i is left-extended to q_j.

First, we claim that no interval whose left endpoint is to the left of x was assigned any lower position to the right of x. Suppose to the contrary that P_t, $l_t < x$, was assigned a lower position to the right of x. Then we must have $l_t > q_j$ since otherwise P_t would have been assigned q_j, which is not the case. This in turn implies that there is a column $\hat{x}$ between q_j and x whose density is already Δ. This contradicts the fact that x is leftmost. This proves the claim that only intervals to the right of x were assigned lower positions to the right of x.

Our next claim is that no interval to the right of x has been assigned lower positions both left and right of x. To show this, simply note that the existence of such an interval would contradict the fact that x is leftmost.

Let A be the set of satisfied intervals to the right of x that were assigned all of their lower positions from the left of x. Note that $c(x)+|A|=\Delta$. Let B be the set of satisfied intervals that are to the right of x and that were assigned all k of their lower positions to the right of x. We then have

$$c(x) + up(x+1,M) - \lfloor down(x+1,M)/k \rfloor \geq$$
$$c(x) + |A| + |B| + 1 - \lfloor (k|B|+k-1)/k \rfloor = \Delta+1,$$

a contradiction. This completes the proof of Lemma 5.4. □

Corollary 5.5 $\hat{d}=\Delta$.

Proof: Lemma 5.4 implies that $\hat{d} \leq \Delta$. This and Lemma 2.2 together imply that $\hat{d}=\Delta$. □

We are now ready to state the main result about the MTP1-problem:

Theorem 5.6 The placement achieving the optimal density of an MTP1-problem can be computed in $O((n+m)\log(n+m))$ time.

6. Other Variants and NP-Hardness of MTP

In this section we describe other multi-terminal placement problems that can be solved efficiently, and we also show that the most general version of the problem is NP-hard. We first consider the MTP2-problem, in which the i-th net must be assigned u of the s available upper positions and k of the m available lower positions, $1 \leq i \leq n$, so that the resulting CRP has minimum density. We describe a *verification algorithm* that, for a given integer d, decides in $O(s+m)$ time whether or not a solution of density $\leq d$ exists. A solution achieving the optimal density $\hat{d}$ is then obtained using binary search in time $O((s+m)\log\hat{d})$ (by $\log\hat{d}$ applications of the verification algorithm).

We start by making the following observation about the MTP2-problem. If a solution of density d exists, then there always exists one in which for every $i<j$, the rightmost upper position assigned to net i is to the left of the leftmost upper position of net j. Thus, the intervals formed by the entry terminals alone have density 1. This allows us to choose the upper positions assigned to net i so that they are con-

secutive. The verification algorithm for the MTP2-problem is then similar to the algorithm given in Section 3. Assume that the scan is currently at upper position p_{up} and at lower position q_{down}, both terminals have not yet been assigned to a net, and nets $1, \cdots i-1$ have already been satisfied.

- If $q_{down} < p_{up}$, and density d would be exceeded when net i contains p_{up} and q_{down}, continue with q_{down+1} and p_{up}.
- If $q_{down} > p_{up}$, and density d would be exceeded when net i contains p_{up} and q_{down}, continue with q_{down} and p_{up+1}.
- If neither of the above two cases holds, assign the next u upper and the next k lower positions to net i, and continue with p_{up+u}, q_{down+k}, and net $i+1$.

The implementation details and the proof of correctness are left to the reader.

Theorem 6.1 The MTP2-problem can be solved in $O((s+m)\log\hat{d})$ time.

A result similar to the above Theorem holds when the u_i's are arbitrary and we know in what order the upper row positions are assigned to the nets (i.e., we know to which net the i-th assigned upper row position is given). Here too, we leave the details to the interested reader.

Recall that the MTP-problem is the one where both the u_i's and k_i's are arbitrary, and we are free to assign both upper and lower row positions in order to minimize the density.

Theorem 6.2 The MTP-problem is NP-hard.

Proof: The proof is by a transformation from 3-Partition, which is NP-hard in the strong sense [GJ] and which is defined as follows. Let $A = \{a_1, a_2, \cdots, a_{3q}\}$ with $\sum_{i=1}^{3q} a_i = qB$ and $B/4 < a_i < B/2$. This instance of 3-Partition has a solution iff set A can be partitioned into q disjoint subsets $A_1, \cdots, A_q$ so that $\sum_{a_i \in A_k} a_i = B$, $1 \le i \le q$.

Given set A and the bound B, we construct a corresponding MTP-problem consisting of $4q-1$ nets, as follows. For $1 \le i \le 3q$, net i requires a_i upper positions and no lower positions (i.e., $u_i = a_i$ and $k_i = 0$). For $3q+1 \le i \le 4q-1$, net i requires no upper positions and B lower positions. The channel contains qB upper and $(q-1)B$ lower

positions. Each one of columns $2iB+1, 2iB+2, \cdots, 2iB+B$, $0 \le i \le q-1$, contains an upper position and each one of columns $(2i+1)B+1, (2i+1)B+2, \cdots, (2i+1)B+B$, $0 \le i \le q-2$, contains a lower position. It is not hard to see that 3-Partition has a solution if and only if the corresponding MTP-problem has a solution of density 1. □

References

[AH] M.J. Atallah, S.E. Hambrusch, 'On Terminal Assignments That Minimize the Density', Techn. Report, CSD-TR-468, Purdue University, 1984.

[AHU] A.V. Aho, J.E. Hopcroft, J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[CL] Y.K. Chen, M.L. Liu, 'Three-layer Channel Routing', *IEEE Trans. on CAD*, Vol. cad-2, Nr. 2, pp 156-163, 1984.

[GCW] I.S. Gopal, D. Coppersmith, C.K. Wong, 'Optimal Wiring of Movable Terminals', *IEEE Trans. on Computers*, Vol. c-32, 9, pp 845-858, 1983.

[H] S.E. Hambrusch, 'Channel Routing Algorithms for Overlap Models', to appear in *IEEE Trans. on CAD*, Jan. 1985.

[HS] A. Hashimoto, J. Stevens, 'Wire Routing by Optimizing Channel Assignment within Large Apertures', *Proc. of 8-th Design Aut. Conf.*, pp 155-169, 1971.

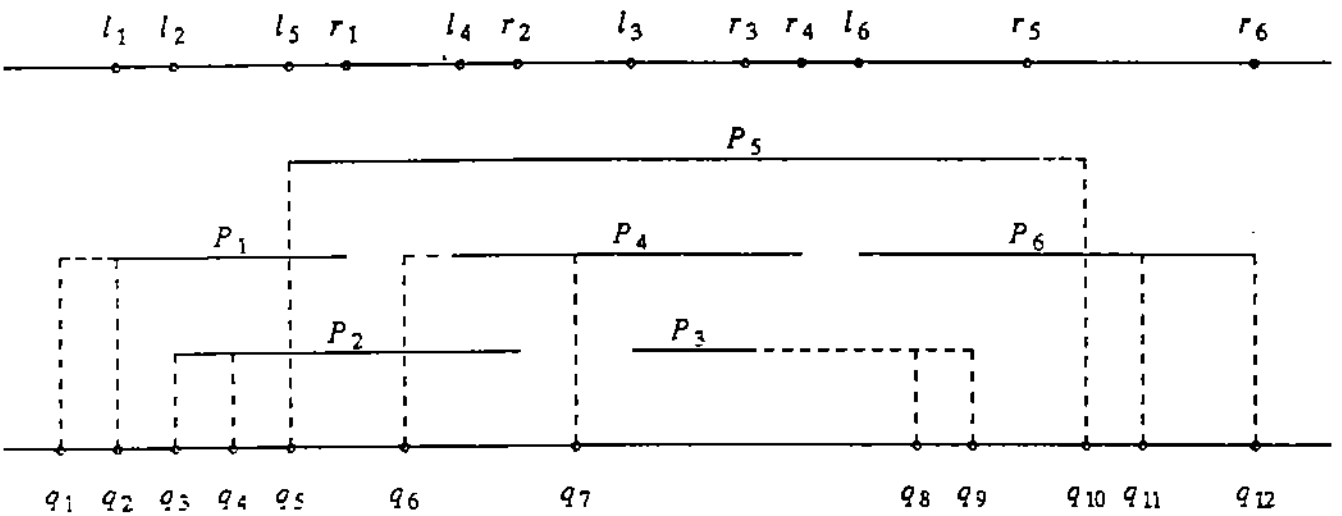
[P] R.Y. Pinter, 'The Impact of Layer Assignment Methods on Layout Algorithms for Integrated Circuits', Ph.D. Thesis, MIT, 1982.

[PL] F.P. Preparata, W. Lipski, 'Three Layers are enough', *Proceedings of the 23rd Annual IEEE Foundations of Comp. Sc. Conf.*, pp 350-357, 1982.

[R] R.L. Rivest, 'The PI - Placement and Interconnect - System', *Proc. of 19-th Design Automation Conf.*, pp 475-481, 1982.

[RBM] R.L. Rivest, A.E. Baratz, G. Miller, 'Provably Good Channel Routing Algorithms', *Proc. of the CMU Conf. on VLSI Syst. and Comp.*, pp 153-159, 1981.

[SP] M. Sarrafzadeh, F.P. Prepatata, 'Compact Channel Routing of Multiterminal Nets', to appear in *Annals of Discrete Mathematics*.



An optimal solution to an MTP1-problem with k=2
Figure 1.1