SIAM J. COMPUT.
Vol. 23, No. 2, pp. 227–246, April 1994
© 1994 Society for Industrial and Applied Mathematics

A LINEAR-TIME ALGORITHM FOR THE HOMOTOPIC ROUTING PROBLEM IN GRID GRAPHS*

MICHAEL KAUFMANN† AND KURT MEHLHORN‡

Abstract. The paper considers the problem of finding edge-disjoint paths between pairs of vertices in a finite grid graph. The homotopy class for each path to be routed is prespecified. A very fast algorithm that guarantees to find a solution for any solvable homotopic routing problem is given.

Key words. algorithms, homotopic routing, edge-disjoint paths, VLSI-theory

AMS subject classifications. 68Q25, 68U05

1. Introduction.

We give a linear time algorithm for the homotopic routing problem in grid graphs.

Problem: Homotopic Routing Problem in Grid Graphs (HRP)

Input: A grid graph R and nets $q_1, \ldots, q_k$.

Output: Pairwise edge-disjoint grid paths $p_1, \ldots, p_k$ such that p_i is homotopic to q_i, $1 \le i \le k$, or an indication that no such paths exist. □

The *planar rectangular grid consists* of vertices $\{(x, y); x, y \in \mathbb{Z}\}$ and edges $\{((x, y), (x', y')); |x - x'| + |y - y'| = 1\}$. A *grid graph* $R = (V, E)$ is a finite subgraph of the planar rectangular grid. We call a bounded face F of R trivial if it has exactly four vertices on its boundary and nontrivial otherwise. We muse M to denote the set of nontrivial bounded faces together with the unbounded face F_{ext} and $\mathcal{O}$ to denote the union of the interiors of the faces in M. A nontrivial face is also called a hole.

A path p is a continuous function $p : [0, 1] \to \mathbb{R}^2 - \mathcal{O}$. A path p is called a *net* if $\{p(0), p(1)\} \subseteq V \cap \partial\mathcal{O}$ where $\partial\mathcal{O}$ is the boundary of $\mathcal{O}$. Two paths p and q are *homotopic*, denoted $p \sim q$, if there is a continuous function $F : [0, 1] \times [0, 1] \to \mathbb{R}^2 - \mathcal{O}$ such that $F(0, x) = p(x)$ and $F(1, x) = q(x)$ for all $x, 0 \le x \le 1$, and $F(t, 0) = p(0)$ and $F(t, 1) = p(1)$ for all $t, 0 \le t \le 1$. A path p is called a *grid path* if $p(x)$ belongs to R for all x.

Figure 1 gives an example of an HRP. For the algorithmic treatment, we assume that the nets $q_1, \ldots, q_k$ are grid paths and use n to denote the number of vertices of R plus the total number of edges in the paths q_i. The integer n is called the size of the HRP. We use $\mathcal{N}$ to denote the set $\{q_1, \ldots, q_k\}$ of nets.

THEOREM 1. *Let $P = (R, \mathcal{N})$ be an even bounded HRP of size n.*

(a) *P is solvable if and only if* $\text{fcap}(X) \ge 0$ *for every cut X.*

(b) *In time $O(n)$ one can decide whether P has a solution and also construct a solution if it does.* □

Part (a) of this theorem was shown in [KM2] and later extended by [Sh]. The paper [KM2] also presents an $O(n^2)$ algorithm. In the present paper we give a new algorithm with linear running time.

A *cut* C is a simple path in $\mathbb{R}^2 - \mathcal{O} - V$ with its endpoints in $\partial\mathcal{O}$. The *capacity* $\text{cap}(C)$ of a cut C is the number of intersections with edges of R. If C is a cut and p is a path then $\text{cross}(p, C)$ is the number of intersections with edges p and C and $\text{mincross}(p, C) =$

*Received by the editors February 13, 1989; accepted for publication (in revised form) June 10, 1992. This work was supported by the DFG, Sonderforschungsbereich 124, Teilprojekt B2, VLSI Entwurf und Parallelität.

†Institut für Informatik, Universität Tübingen, Sand 13, D-72076 Tübingen, Germany.

‡MPI für Informatik, Universität des Saarlandes, Im Stadtwald 15, D-66123-Saarbrücken 11, Germany.

min{cross (q, D); $q \sim p$, $D \sim C$}. Finally, the *density* dens(C) of cut C is defined by

$$\text{dens}(C) = \sum_{p \in \mathcal{N}} \text{mincross}(p, C)$$

and the *free capacity* fcap (C) is given by

$$\text{fcap}(C) = \text{cap}(C) - \text{dens}(C).$$

A cut C is *saturated* if fcap$(C) = 0$ and *oversaturated* if fcap$(C) < 0$. An HRP is *even* if fcap(C) is even for every cut C.

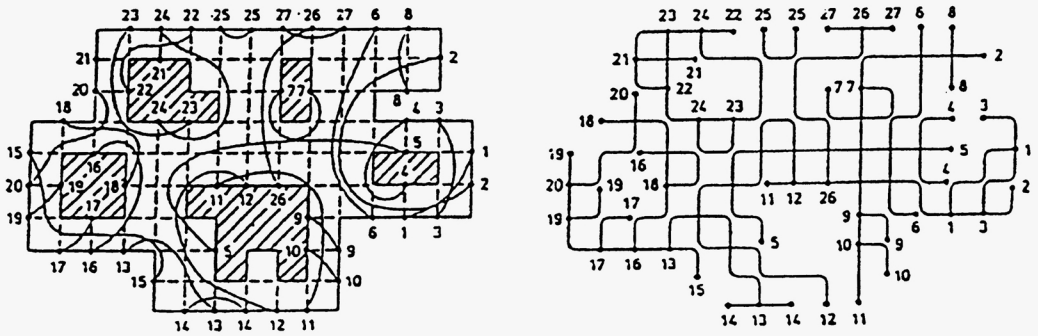

FIG. 1.

Let v be a vertex in R. We denote the degree of v by deg(v) and the number of nets having v as endpoint by ter(v). An HRP is *bounded*, if deg(v) + ter$(v) \leq 4$ for all vertices v, and *weakly bounded* if deg$(v) = 4$ implies ter$(v) = 0$.

There are many previous papers on finding edge-disjoint paths in grid graphs and general planar graphs, e.g., [PL], [F], [MP], [NSS], [KM1], [KM2], [BM], [K], [Sh]. The present paper extends the work in [KM2]. We refer the reader to [KM2] for a discussion of the relationship between VLSI-design and homotopic routing problems.

This paper is organized as follows. In §2 we describe the algorithm and in §3 we prove its correctness. The algorithm is similar in spirit to the algorithm in [KM2], but differs in many details. In particular, its correctness does not follow from [KM2]. In §4 we then describe the linear time implementation of the algorithm. Weinelt [W] has implemented the algorithm; Fig. 1 has been produced by his program.

2. The algorithm. In this section we describe an algorithm for the homotopic routing problem in grid graphs. Recall that we are given a grid graph R and nets $p_1, \ldots, p_k$ and that our goal is to shift the nets $p_1, \ldots, p_k$ into pairwise edge-disjoint grid paths. Our algorithm works iteratively. In each iteration we consider an edge e of R and decide whether to use it for some net and if so for which one. If e is to be used for net p then we choose suitable nets p^1 and p^2 with $p \sim p^1 e p^2$ and replace net p by the three nets p^1, e, and p^2. The edge e is then used to route the net e in the obvious way. Thus, each iteration discards one edge of R and hence there are $O(n)$ iterations. In §4 we show how to implement the algorithm such that each iteration takes amortized time $O(1)$.

For the algorithm we need some further concepts.

DEFINITION 1. *For a path p the canonical representation* can(p) *is the shortest path homotopic to p.*

Note that can(p) is composed of straight-line segments.

DEFINITION 2. (a) *A path p is called a prefix of path q if there is a monotone function $t : [0, 1] \to [0, 1]$ with $t(0) = 0$ such that $p(x) = q(t(x))$ for all x.*

(b) *The reversal p^{-1} of path p is defined by $p^{-1}(x) = p(1 - x)$ for $0 \le x \le 1$.*

(c) *If p and q are paths, then $p \approx q$ if either $p \sim q$ or $p^{-1} \sim q$.*

(d) *For a path p, we define* source$(p) = p(0)$ *and* target$(p) = p(1)$.

(e) *For a point v, $x(v)$ and $y(v)$ denote the x- and y-coordinate of point v, respectively.*

DEFINITION 3. *Let p and q be nontrivial paths with the same source s and let s lie on the boundary of a unique hole F. Then p is said to be* right *of q, if either*

- can$(p) =$ can(q) *or*
- can(p) *is a proper prefix of* can(q) *and there is a hole to the right of* can(q) *at point* target(can(p)) *or*
- can(q) *is a proper prefix of* can(p) *and there is a hole to the left of* can(p) *at point* target(can(q)) *or*
- can(p) *and* can(q) *have no nontrivial common prefix and for every sufficiently small circle K around s there is a counterclockwise scan of K intersecting first ∂F, then* can(p), *then* can(q), *and finally again ∂F or*
- can(p) *and* can(q) *have a maximal common nontrivial prefix r, i.e.,* can$(p) = rp^1$, can$(q) = rq^1$ *and p^1 and q^1 have no nontrivial common prefix, and for every sufficiently small circle K around* target(r) *there is a counterclockwise scan of K intersecting first r, then p^1, and finally q^1.*

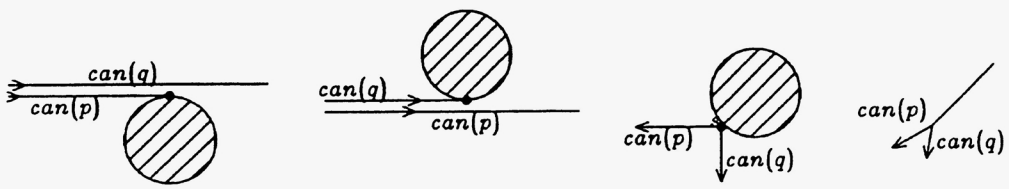

FIG. 2. *An illustration of the four cases in Definition* 3.

Figure 2 illustrates the various cases of Definition 3. The relation *left* is defined analogously. Both relations are clearly transitive.

DEFINITION 4. (a) *A path p is* x-monotone *if x*(target$(p)) \le x$(source(p)) *and for every vertical line L the intersection $p \cap L$ is a segment, i.e., x-monotone paths travel from right to left.*

(b) *The* monotone prefix pref(p) *of a net p is the maximal x-monotone prefix of* can(p).

(c) *Let p and q be nets with a common source s and let s lie on the boundary of a unique hole. Then p is* quasi-right *of q if either* pref(q) *is trivial or* pref(p) *and* pref(q) *are nontrivial and* pref(p) *is right of* pref(q).

Remark. The ordering right played an important role in the algorithm of [KM2]. In this paper we use the ordering quasi-right instead. This is one of the sources for the improved running time.

DEFINITION 5. *A cut C is called* vertical *if it is a vertical straight-line segment.*

The algorithm is given as Program 1. In this program, *top* always denotes the largest y-coordinate of any vertex of R, *Top* is the subgraph spanned by the vertices with y-coordinate equal to *top*, and a *segment* is a connected component of *Top*. An endpoint of a segment is called *exposed* if it is not the terminal of any net. The endpoints of a segment are also called *corners* of the segment. Finally, *Rim* is the closed region above the horizontal line $y = top - 1$. In the formulation of Program 1 we assumed that the lowest numbered case for which the precondition is satisfied is executed, i.e., if e.g., case 4.3 is taken, then cases 1, 2,

3, 4.1, and 4.2 do not apply. In cases 4.2 and 5.1 the concept of rightmost decomposition is used, which we now define.

DEFINITION 6. *Let $e = (b, a)$ with b the left neighbor of a be an edge in row Top and let X be the vertical cut through e. Let C be the set of nets p with* mincross$(p, X) > 0$. *For $p \in C$ an* admissible *decomposition is a triple (p^1, e, p^2) such that $p \approx p^1 e p^2$ and* mincross(p^1, X)+ mincross(p^2, X) = mincross$(p, X) - 1$. *A triple (p^1, e, p^2) is called a* rightmost *decomposition with respect to X if it is an admissible decomposition of a net $p \in C$ and p^2 is quasi-right of q^2 for any admissible decomposition (q^1, e, q^2).*

(1) **while** $R \neq \emptyset$
(2) **do**
(3) case 1: $\exists$ cut X intersecting an edge e in *Top* with cap$(X) = 1$(*operation 1*)
(4) let X be such a cut, let e be the intersected edge, and let p be the unique net with mincross$(p, X) = 1$;
(5) let $p \approx p^1 e p^2$ with mincross$(p^i, X) = 0$ for $i = 1, 2$;
(6) replace p by p^1 and p^2 and discard e;
(7) case 2: $\exists$ segment S consisting of a single vertex b(*operation 2*)
(8) let (b, c) be the vertical edge incident to b and let p be the unique net incident to b; let $p \approx (b, c)p^1$; replace p by p^1 and discard (b, c);
(9) case 3: $\exists$ non-trivial segment S with its right corner b not exposed (*operation 3*)
(10) let p and q be the two nets incident to b with p quasi-right of q and let $e = (b, a)$ be the horizontal edge incident to b;
(11) let $p \approx ep^1$; replace p by p^1 and discard e;
(12) case 4: $\exists$ non-trivial segment S with its right corner b exposed and its left corner not exposed;
(13) case 4.1: $\exists$ net p with can$(p) \subseteq S$(*operation 4.1*)
(14) delete net p and discard all edges in can(p)
(15) case 4.2: the vertical cut X through the horizontal edge $e = (b, a)$ is saturated (*operation 4.2*)
(16) let (p^1, e, p^2) be the rightmost decomposition of any net which crosses X; replace p by p^1 and discard all edges in can(ep^2)
(17) case 4.3: let $p \approx (a, b)p^1$ be a net incident to the left neighbor a of b;
(18) replace p by p^1 and discard edge (a, b) (*operation 4.3*)
(19) case 5: $\exists$ nontrivial segment S with both corners exposed
(20) case 5.1: $\exists$ saturated vertical cut through an edge of S (*operation 5.1*)
(21) let X be the shortest saturated vertical cut through an edge $e = (b, a)$ of S; here b is the right neighbor of a;
(22) let (p^1, e, p^2) be the admissible decomposition of a net with respect to X such that either can $(p^1) \subseteq Rim$ or the decomposition is rightmost;
(23) replace p by p^1 and p^2 and discard e
(24) case 5.2: let a and b be the left and right corner of S(*operation 5.2*);
(25) add nets p and q with $p(\lambda) = q(\lambda) = \lambda a + (1 - \lambda)b$ for $0 \leq \lambda \leq 1$
(26) **od**

PROGRAM 1.

3. Proof of correctness. This section is divided into two subsections. In the first subsection we collect some useful facts about cuts and in the second subsection we argue the correctness of the routing algorithm.

3.1. Some facts about cuts.

DEFINITION 7. *A cut* $C : [0, 1] \to \mathbb{R} - \mathcal{O} - V$ *is called* straight *if the function* C *is linear and it is called* almost-straight *if it is either straight or if the line segment connecting* $C(0)$ *and* $C(1)$ *is contained in the boundary of a nontrivial face, the functions* $C|[0, \frac{1}{2}]$ *and* $C|[\frac{1}{2}, 1]$ *are linear, there is no vertex of* R *contained in the interior of the triangle* $C(0)C(\frac{1}{2})C(1)$, *and no edge of* R *is intersected more than once by* C. *A cut* C *is called* Manhattan *if it consists of horizontal and vertical straight-line segments. For a cut* C, Manhattan(C) *is a Manhattan cut intersecting the same edges as* C.

DEFINITION 8. *A tuple* $(p_1, \ldots, p_k)$ *is called* straight-line decomposition *of* p *if* $\text{can}(p) = p_1 \cdots p_k$ *and each* p_i *is a maximal straight-line segment contained in* can(p). *Each* p_i *is called a straight-line piece of* p.

LEMMA 1. (a) *Let* P *be weakly bounded. Then the cut condition holds if it holds for all almost-straight cuts.*

(b) *Let* P *be bounded. Then* $\text{fcap}(C) > 0$ *for every cut* C *that is almost-straight but not straight. Also, the cut condition holds if it holds for straight cuts.*

(c) *Let* p *be a net and* C *a cut. Then* $\text{cross}(p, C) \equiv \text{mincross}(p, C) \bmod 2$.

(d) *Let* $(p_1, \ldots, p_k)$ *be the straight-line decomposition of* p *and let* C *be a straight cut. Then* $\text{mincross}(p, C) = \sum_{i=1}^{k} \text{cross}(p_i, C)$.

(e) *Let* P *be a bounded problem satisfying the cut condition and let* C *be a saturated Manhattan path in* P. *Then consecutive turns of* C *are in alternate directions.*

(f) *Let* P *be bounded. If there is an oversaturated straight cut in* P, *then there is either an oversaturated horizontal or vertical cut in* P *or an oversaturated straight cut intersecting only edges incident to vertices of degree* 4.

Proof. (a) This is the content of §3 of [Sh].

(b) Let C be almost-straight but not straight. Then the line segment connecting $C(0)$ and $C(1)$ passes through exactly $\text{cap}(C) - 2$ vertices, all of which have degree 3. Hence $\text{dens}(C) \leq \text{cap}(C) - 2$. The second part now follows from part (a).

(c) This is Lemma 7 of [KM2].

(d) Since $p \sim \text{can}(p) = p_1 p_2 \cdots p_k$, we have $\text{mincross}(p, C) \leq \sum_i \text{cross}(p_i, C)$. Let us assume $\text{mincross}(p, C) < \sum_i \text{cross}(p_i, C)$. Then there are (x_1, λ_1), (x_2, λ_2) such that $x_i \neq x_2$, $\lambda_1 \neq \lambda_2$, and $\text{can}(p)|[x_1, x_2] \sim C|[\lambda_1, \lambda_2]$. Since $\text{can}(p)$ is a shortest path homotopic to p and since C is a straight-line segment, we conclude that $\text{can}(p)(x_1)$ and $\text{can}(p)(x_2)$ must lie on the same segment p_i. So p_i is a subpath of C and hence C passes through a vertex of R, a contradiction.

(e) This is Lemma 12a, Claim 1 of [KM2].

(f) This is Lemma 12b of [KM2]. Note that this lemma state in the terminology of that paper that there is either an oversaturated 0-bend or an oversaturated 1-bend cut connecting two concave corners. The latter gives rise to a straight cut intersecting only edges incident to vertices of degree 4. □

3.2. Correctness of the algorithm. All of the actions in our algorithm fall under the following paradigm. For a certain edge e and a certain net p we choose p^1 and p^2 such that $p \approx p^1 e p^2$, replace p by p^1 and p^2 and discard e; operations 4.1, 4.2, and 5.2 can be viewed as repeated application of this basic action. It is convenient to view the basic step as to consist of two substeps.

substep A: replace p by the three nets p^1, e, and p^2.

substep B: remove edge e and net e.

The further outline of the correctness proof is as follows. We first deal with substep B in Lemma 2, then show in Lemma 3 that substep A generates even problems satisfying the degree constraints postulated in the premise of Lemma 3, and finally show in Lemmas 4 to 10 that operations 1 to 5.2 preserve the cut condition. In the proofs of these lemmas we will frequently use part (a) of Theorem 1, i.e., the present paper does not give an alternative proof of part (a).

LEMMA 2. *Let $P = (R, \mathcal{N})$ be an even routing problem satisfying the cut condition, and let e be an edge in the boundary of $\mathcal{O}$ as well as a net in $\mathcal{N}$. Assume further that* $\text{ter}(v) + \deg(v) \leq 4$ *for all vertices except the endpoints of e and* $\text{ter}(v) + \deg(v) \leq 6$ *for the two endpoints of e. Then removing edge e and net e yields an even, bounded problem satisfying the cut condition.*

Proof. Let us use P' to denote the modified problem. P' is certainly bounded. If there is a cut of capacity one through e, then P' is also even and satisfies the cut condition. So let us assume that there is a trivial face F incident to e in P. Let $R' = R - e, \mathcal{N}' = \mathcal{N} - e$ and let Y' be any straight cut in R'. If Y' is also a cut in P, then $\text{dens}'(Y') = \text{dens}(Y')$ and $\text{cap}'(Y') = \text{cap}(Y')$ where dens′ and cap′ are computed with respect to P'. If, on the other hand, Y' ends in F then we can extend Y' to a cut in P with $\text{cap}(Y) = \text{cap}'(Y') + 1$ and $\text{dens}(Y) = \text{dens}'(Y') + 1$. Thus $\text{fcap}'(Y') = \text{fcap}(Y) \geq 0$. □

LEMMA 3. *Substep* A *generates even problems satisfying the degree constraints postulated in the premise of Lemma* 2.

Proof. The claim about the degree contraints is obvious. For the evenness let C be any cut and let $p \sim p^1 e p^2$. Then

$$\begin{aligned}
\text{mincross}(p, C) &= \text{mincross}(p^1 e p^2, C) \\
&= \text{cross}(p^1 e p^2, C) \bmod 2 \qquad \text{by Lemma 1(c)} \\
&= [\text{cross}(p^1, C) + \text{cross}(e, C) + \text{cross}(p^2, C)] \bmod 2 \\
&= [\text{mincross}(p^1, C) + \text{mincross}(e, C) + \text{mincross}(p^2, C)] \bmod 2
\end{aligned}$$

by Lemma 1(c). Thus evenness is preserved. □

LEMMA 4. *Operations* 1 *and* 2 *preserve the cut condition.*

Proof. Obvious. □

LEMMA 5. *Operation* 3 *preserves the cut condition.*

Proof. In operation 3 there are nets p and q incident to the right corner b of a segment with p quasi-right of q. We replace p by e and p^1, where $e = (b, a)$, a is the left neighbor of b, and $p \approx e p^1$. Assume, for the sake of a contradiction, that there is an oversaturated almost-straight cut C in the modified problem. The cut C must go through edge e since otherwise its density is not affected by the action. Also $\text{mincross}(e, C) = 1$ and $\text{mincross}(p^1, C) = \text{mincross}(p, C) \pm 1$. Thus the density of C is increased by at most 2 and hence C must be saturated in the original problem. Thus C must be straight by Lemma 1(b) and $\text{mincross}(p^1, C) = \text{mincross}(p, C) + 1$. Since the original problem has a solution in which then necessarily net q uses edge e, we conclude $\text{mincross}(q^1, C) = \text{mincross}(q, C) - 1$ where $q \approx e q^1$. Let $(p_1, \ldots, p_h)$ and $(q_1, \ldots, q_k)$ be the straight-line decompositions of p and q, respectively. Then p_1 does not intersect C since $p_1 \cdots p_k$ is a shortest path homotopic to p, $\text{mincross}(p, C) = \sum_i \text{cross}(p_i, C)$ by Lemma 1(d), and $\text{mincross}(p^1, C) = \text{mincross}(p, C) + 1$, and q_1 intersects C for analogous reasons.

Case A. $\lfloor x(\text{target}(C)) \rfloor \leq \lfloor x(\text{source}(C)) \rfloor$: Then p_1 is not right of q_1 and hence p is not quasi-right of q.

Case B. $\lfloor x(\text{target}(C)) \rfloor > \lfloor x(\text{source}(C)) \rfloor$: Consider the cut D indicated in Fig. 3. Then D is saturated, since $\text{mincross}(p, D) = \text{mincross}(p, C) + 1$ and $\text{mincross}(q, D) = \text{mincross}(q, C) - 1$. Also, Manhattan (D) makes two consecutive turns into the same direction, a contradiction to Lemma 1(e). □

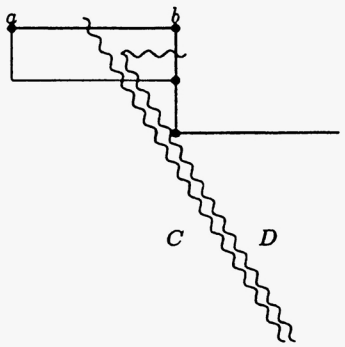

FIG. 3.

LEMMA 6. *Operation* 4.1 *preserves the cut condition. This is even true if the right corner of S is not exposed.*

Proof. In operation 4.1 there is a net p with $\text{can}(p) \subseteq S$. Then $\text{mincross}(p, C) = \text{cross}(\text{can}(p), C)$ for every straight cut C. Hence, deleting the net p and discarding all edges in $\text{can}(p)$ preserves the cut condition. □

LEMMA 7. *Operation* 4.2 *preserves the cut condition.*

Proof. In operation 4.2 the right corner of S is exposed and the left corner of S is not exposed. Furthermore, there is no net p with $\text{can}(p) \subseteq S$ and hence the horizontal cut Y, which cuts off the segment S, is saturated. Note that there are exactly $\text{cap}(Y)$ terminals above segment Y and that each terminal contributes one to the density of Y. Let b be the right corner of S, let $e = (b, a)$ be the horizontal edge incident to b, and let X be the vertical cut through e. In operation 4.2 the cut X is saturated. Let (p^1, e, p^2) be the rightmost decomposition of any net with respect to X.

CLAIM 1. $\text{can}(p^2) \subseteq S$.

Proof. Let $e' = (c, b)$ be the vertical edge incident to b. Consider any solution to the original problem. Then there must be a net, say q, which uses the edges e' and e, i.e., the solution path can be written $q^1 e' e q^2$. Furthermore, since there are exactly $\text{cap}(Y)$ terminals in S, contributing one each to the density, we conclude that $\text{mincross}(q^2, Y) = 0$. Thus $\text{can}(q^2) \subseteq S$. Next observe that p^2 is quasi-right of q^2, since (p^1, e, p^2) is a rightmost decomposition with respect to X, and hence $\text{can}(p^2) \subseteq S$. □

Let P' be the problem obtained by replacing p by p^1 and ep^2. It suffices to show that P' satisfies the cut condition, since the deletion of net ep^2 and all edges in $\text{can}(ep^2)$ is covered by Lemma 6. So assume that there is an oversaturated straight cut C in P'. By Lemma 1(f) we may assume that C is either vertical or horizontal or a straight cut intersecting only edges incident to vertices of degree 4. Since $\text{cross}(ep^2, C) \leq 1$, $p^1 \approx ep^2 p^{-1}$ and hence $\text{mincross}(p^1, C) \leq \text{cross}(\text{can}(p), C) + \text{cross}(ep^2, C) \leq \text{mincross}(p, C) + 1$, we conclude $\text{fcap}(C) = 0$, $\text{fcap}'(c) = -2$, $\text{mincross}(ep^2, C) = 1$, and $\text{mincross}(p^1, C) = \text{mincross}(p, C) + 1$. From $\text{mincross}(ep^2, C) = 1$ we conclude that C must intersect an edge of segment S and hence must be vertical.

Since $\text{mincross}(p, X) > 0$, we can write $\text{can}(p) = r^1 r^2$ where $x(\text{source}(r^2)) = x(b)$, $\text{cross}(r^2, X) = 1$, and the vertical segment s^1 connecting $\text{source}(r^2)$ to b is contained

in R. Then $p^1 \sim r^1 s^1$ and hence $\text{mincross}(p^1, C) \leq \text{cross}(r^1 s^1, C) = \text{cross}(r^1, C) = \text{cross}(\text{can}(p), C) - 1 = \text{mincross}(p, C) - 1$. Thus $\text{fcap}'(C) = \text{fcap}(C)$ for any vertical cut, a contradiction. □

LEMMA 8. *Operation* 4.3 *preserves the cut condition.*

Proof. The precondition is almost as in operation 4.2; however, the vertical cut X through $e = (b, a)$ is not saturated. Let p be a net which has a as a terminal. Let $p \approx (a, b)p^1$ and replace p by (a, b) and p^1.

Assume that there is an oversaturated straight cut C with respect to the modified problem P'; note that P' is bounded. Then C must intersect the edge (a, b), since otherwise its density is not affected by the action, and $\text{fcap}(C) = 0$. Next observe that P' is bounded and hence by Lemma 1(f) we may assume that C is vertical, i.e., $C = X$. This contradicts $\text{fcap}(C) = 0$. □

LEMMA 9. *Operation* 5.1 *preserves the cut condition.*

Proof. Let X be the shortest saturated vertical cut intersecting an edge in S, let $e = (b, a)$ be the edge in S intersected by X with b being the right neighbor of a, and let $p \approx p^1 e p^2$ be a net crossing X such that either $\text{can}(p^1) \subseteq Rim$ or (p^1, e, p^2) is the rightmost decomposition with respect to X. We replace p by p^1, e, and p^2.

The following claim will be useful.

CLAIM 2. *In P there is no saturated nonvertical cut through edge e.*

Proof. Assume that there is such a Manhattan cut C'. By Lemma 1(e) we may assume that C' turns in alternate directions. Assume also that the first segment C'_1 of C' is maximal in length. We now distinguish cases. Assume first that C' has only 1 bend or C'_1 intersects less that $\text{cap}(X) - 1$ edges.

We move the vertical part of C' intersecting e so as to make the cut C' shorter. Then one of the following situations must arise (see Fig. 4). Either we split the cut at some point into several cuts or we move the vertical part beyond the end of segment S. In the first case one of the parts is a saturated vertical cut intersecting S and being shorter than X; in the second case an oversaturated cut is obtained since the corners of S are exposed. In both cases, we have a contradiction. We conclude that C_1 intersects $\text{cap}(X) - 1$ edges and C' has more than 1 bend; i.e., the situation is as shown in Fig. 5.

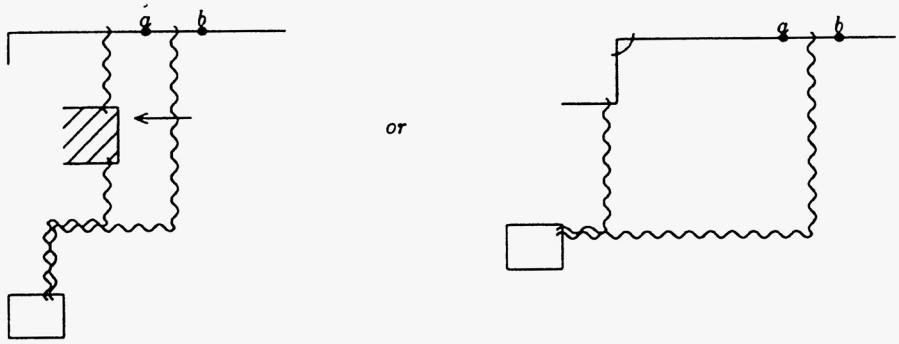

FIG. 4. *Illustrating the first case in the proof of Claim* 2.

We now move the first horizontal part of C' down by one unit, and in this way split C' into several cuts, which are all saturated in P. Hence one of them is oversaturated in P'. But the topmost cut ($= X$) is saturated in P' and the free capacities of all other cuts do not change, since they do not interfere with cut X. □

We now return to the discussion of operation 5.1. Assume that there is an oversaturated cut C with respect to the modified problem P'. Since $\text{mincross}(p, X) > 0$, we can write

$\text{can}(p) = r^1 f r^2$ where $x(\text{source}(f)) = x(b), x(\text{target}(f)) = x(a)$, and the straight-line segments s^1 connecting source(f) to b and s^2 connecting a to target(f) are contained in R. Then $p \sim s^1 r^1$, $p^2 \sim r^2 s^2$, and $f \sim s^1 e s^2$. Also, $\text{mincross}(p^1, C) + \text{mincross}(e, C) + \text{mincross}(p^2, C) \leq \text{cross}(r^1 s^1, C) + \text{cross}(e, C) + \text{cross}(s^2 r^2, C) = \text{cross}(r^1 f r^2, C) + \text{cross}(s^1 e s^2, C) - \text{cross}(f, C) \leq \text{mincross}(p, C) + 2$. We conclude that $\text{cross}(s^1 e s^2, C) = 2$, $\text{fcap}(C) = 0$ and $\text{fcap}'(C) = -2$. Hence C is straight by Lemma 1(b).

Case A. C intersects e: Claim 2 and the fact that $\text{fcap}(C) = 0$ and C is straight imply that C is vertical. Thus $C = X$, a contradiction.

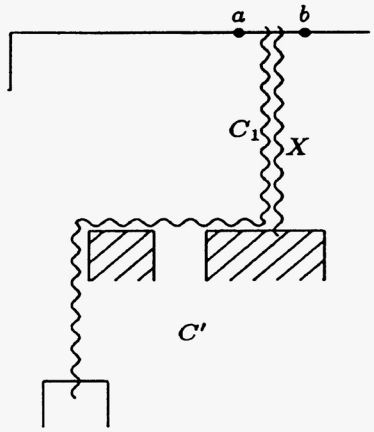

FIG. 5.

Case B. C does not intersect e. Then C must intersect s^1 and s^2 and hence C and X intersect in a single point.

Case B1. $\text{can}(p^1) \subseteq Rim$. Then C must intersect $\text{can}(p^1)$ because otherwise $\text{mincross}(p^1 e, C) + \text{mincross}(p^2, C) = \text{mincross}(p^2, C) \leq \text{cross}(e^{-1}(\text{can}(p^1))^{-1}\text{can}(p), C) \leq \text{cross}(\text{can}(p), C)$, and hence $\text{fcap}'(C) = \text{fcap}(C)$. Let $C' \sim C$ be a Manhattan cut with $\text{cap}(C') = \text{cap}(C)$. Assume first that C' is a straight horizontal cut. Then $C' = Y$, where Y is defined as in Lemma 7, and hence $f\text{cap}(C') \geq 2$, a contradiction. We conclude that C' contains at least one vertical segment. Since $\text{can}(p^1) \subseteq Rim$, C intersects $\text{can}(p^1)$, s^1, and s^2, C is straight, and since the corners of all segments in *Top* are exposed, we may assume without loss of generality (w.l.o.g.) that C' starts with a vertical segment and turns right. Thus if we move that vertical segment by one unit to the left, the capacity of the cut cannot increase. Also, since the corners of all segments are exposed when case 5 applies, the density of the cut cannot decrease and hence the cut stays saturated in P (or even becomes oversaturated in P'). As we move the vertical part further to the left either one of the following three situations must arise (see Fig. 6). Either, we obtain an oversaturated cut in P or a nonvertical saturated cut through e or the cut is split into several cuts at least one of which is saturated in P and oversaturated in P'. In all three cases we have a contradiction; this is obvious in the first case, follows from Claim 2 in the second case, and follows in the third case form the observation that none of the resulting cuts can simultaneously intersect s^1, s^2, and $\text{can}(p^1)$.

Case B2. (p^1, e, p^2) is the rightmost decomposition with respect to X. Consider the cuts Z_1 and Z_2 shown in Fig. 7. Let Z_1' be a shortest path homotopic to Z_1. Z_1' is a polygonal path that bends at the corners of some holes. We split Z_1' at these corners and obtain straight cuts $Z_{11}', \ldots, Z_{1k_1}'$ (see Fig. 8). In a similar way we obtain $Z_{21}', \ldots, Z_{2k_2}'$ from Z_2.

CLAIM 3. $\text{fcap}(X) + \text{fcap}(C) \geq \sum_{i=1}^{k_1} \text{fcap}(Z_{1i}') + \sum_{i=1}^{k_2} \text{fcap}(Z_{2i}')$.

Proof. It is clear that $\text{cap}(X) + \text{cap}(C) \geq \sum_{i=1}^{k_1} \text{cap}(Z_{1i}' + \sum_{i=1}^{k_2} \text{cap}(Z_{2i}')$. It therefore

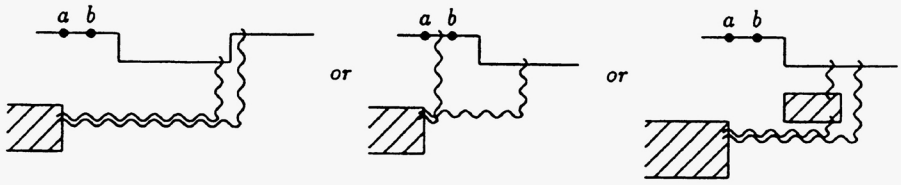

FIG. 6.

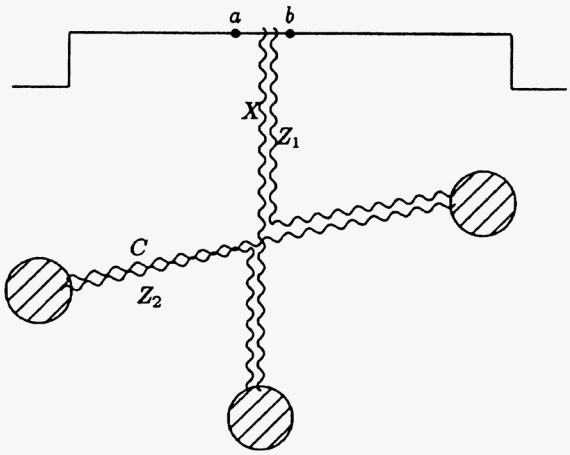

FIG. 7. *X and C produce* Z_1 *and* Z_2.

suffices to show that $\text{cross}(s, X) + \text{cross}(s, C) \leq \sum_{i=1}^{k_1} \text{cross}(s, Z'_{1i}) + \sum_{i=1}^{k_2} \text{cross}(s, Z'_{2i})$ where s is any straight-line piece of a net $q \in \mathcal{N}$. Divide the plane by the lines supporting X and C into four regions as shown in Fig. 9. Then all pieces s contribute the same amount to both sides except those having one endpoint in region D and one endpoint in region F. Such segments contribute 2 to the left hand side and 0 to the right hand side.

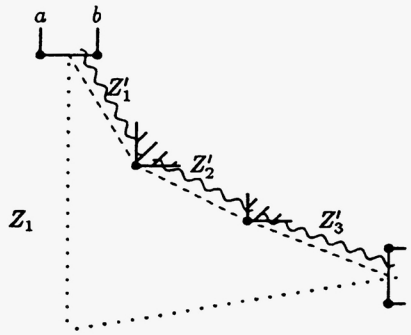

FIG. 8.

We now show that there is no such segment. Assume otherwise. Let q be a net such that $\text{cap}(q) = q'sq''$ contains an elementary piece s, which has its source in F and its target in D. Let $s = s's''s'''$ where $x(\text{source}(s'')) = x(b)$ and $x(\text{target}(s'')) = x(a)$; let l_1 and l_2 be the paths parallel to X and connecting source(s'') with b and a with target(s''), respectively. Then $q \approx q's'l_1el_2s'''q''$, $\text{mincross}(q', X) = \text{mincross}(q's'l_1, X)$ and $\text{mincross}(q'', X) = \text{mincross}(l_2s'''q'', X)$, i.e., $(q's'l_1, e, l_2s'''q'')$ is a decomposition of q with respect to X. Also,

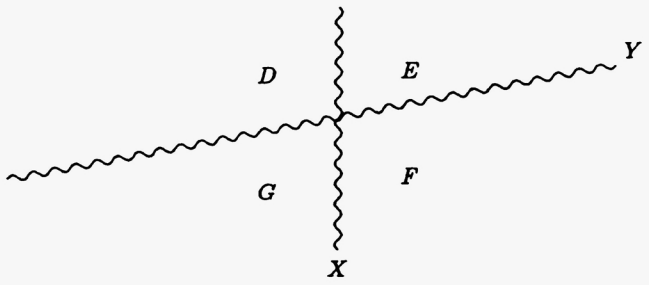

FIG. 9. *The four regions D, E, F, and G.*

$l_2 s''' q''$ is quasi-right of p_2, since the first elementary piece of p_2 ends in G, a contradiction. □

Claim 3 implies that either one of the cuts $Z'_{1i}, 1 \leq i \leq k_1$, or $Z'_{2i}, 2 \leq i \leq k_2$, is oversaturated, a contradiction to the assumption that P satisfies the cut condition, or that all of them are saturated. In the latter case, Z'_{11}, a nonvertical straight cut through e, is saturated, a contradiction to Claim 2. The discussion of case B is now completed and Lemma 9 is shown. □

LEMMA 10. *Operation* 5.2 *preserves the cut condition.*

Proof. When operation 5.2 is applied, both corners of S are exposed and there is no saturated vertical cut through any edge of S. Let a and b be the two corners of S. We add the nets p and q with $p(\lambda) = q(\lambda) = a\lambda + (1 - \lambda)b$ for $0 \leq \lambda \leq 1$. Let P' be the modified problem and assume that there is an oversaturated cut C in P'. Then C must intersect S and C was saturated in P. Since P' is bounded, we may assume that C is vertical by Lemma 1(f), a contradiction. □

4. A linear time implementation. In this section we describe a linear time implementation of our algorithm. In the first part of the section we introduce the required data structures and in the second part we realize the algorithm using these data structures. The first part consists of two interleaved sections: the description of the abstract and the description of the concrete data structure. In the abstract data part we use objects like sequences and sets and operations on these objects and in the concrete part we show how to realize these operations. The concrete part is interleaved with the abstract part.

4.1. The data structure. We first discuss the representation of the routing region. We assume that the vertices and the vertical cuts are numbered 1, 2, 3, ... The numbering of the cuts is such that for each x-coordinate the cuts with the x-coordinate are numbered by consecutive numbers from top to bottom. We identify vertices and cuts with their number. We specify the routing region by storing the neighbors for each vertex.

Concrete. We have four arrays l, r, u, d. For $1 \leq v \leq |V|$, $l[v]$ is the (number of the) left neighbor of v; similarly for r, u, and d. We also have two arrays X and Y that give the coordinates of every vertex. □

Let v be any vertex. Then $L(1, v)(L(2, v)$, respectively) is the closest vertex with the same y-coordinate as v to the left of v, which can be reached from v by a straight-line grid path and which is incident to a vertical boundary edge from below and from above, respectively. $R(i, v), U(i, v)$, and $D(i, v), i = 1, 2$ are defined analogously with left replaced by right, above, and below, respectively.

Concrete. We have another four arrays L, R, U, and D. The array L can be initialized in linear time by scanning each row of the routing region from left to right. Similarly, for R, U, and D. □

We turn to the representation of nets next.

DEFINITION 9. (a) *Let p be a net. Then* rightmost p *is a shortest grid path homotopic to p, which is right of any other shortest grid path homotopic to p.* leftmost(p) *is defined analogously.*

(b) *Let p be a net. The monotone decomposition of p is $(p_1, \ldots, p_k)$ where* can(p) $= p_1 \cdots p_k$, *and p_i is either a maximal vertical segment or a maximal subpath not containing a vertical segment.*

(c) *The representative of p is* rep(p) $= p_1' \cdots p_k'$ *where $(p_1, \ldots, p_k)$ is the monotone decomposition of p and*

$$p_i' = \begin{cases} p_i & \text{if } p_i \text{ is vertical,} \\ \text{rightmost } (p_i) & \text{if } x(\text{target}(pi)) < x(\text{source}(p_i)), \\ \text{leftmost}(p_i) & \text{if } x(\text{target}(p_i)) > x(\text{source}(p_i)). \end{cases}$$

(d) *The bend sequence $b_s(p)$ of p is the sequence $v_0, \ldots, v_k$ of vertices such that*

(1) *v_0 and v_k are the terminals of p;*

(2) *$v_1, \ldots, v_{k-1}$ are consecutive bends of* rep(p);

(3) *if $v_0 \notin Rim \cap \partial F_{\text{ext}}$, then v_1 is the first bend of* rep(p) *and if $v_0 \in Rim \cap \partial F_{\text{ext}}$, then v_1 is the first bend not contained in Rim. Similarly, if $v_r \notin Rim \cap \partial F_{\text{ext}}$, then v_{r-1} is the last bend of* rep(p) *and if $v_r \in Rim \cap \partial F_{\text{ext}}$, then v_{r-1} is the last bend not contained in Rim.*

Remark. In the bend sequence of a path we suppress the bends contained in Rim, since these bends appear and disappear as we process the top row. It would therefore be very costly to treat them like the other bends. Figure 10 shows the representative of a net and its bend sequence.

In the input, a net is given as a grid path. The representative of a path can be computed in time proportional to its length by walking along the path and looking for shortcuts using the arrays L, R, U, and D.

This can be seen as follows. In a first walk along the path, a shortest path homotopic to the net is computed (the task is here to remove U-turns which are not forced by a hole).

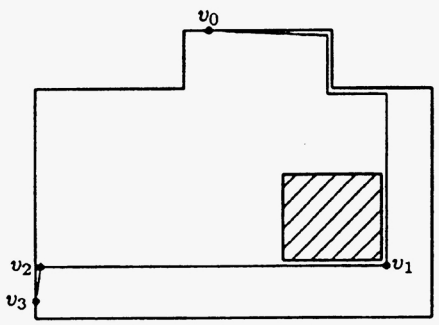

FIG. 10.

In a second walk along the path, the path is decomposed into montone pieces and the pieces are shifted appropriately. All of this takes linear time in the length of the path.

The bend sequence can be computed from the representative by inspecting the initial and final segments of it. Altogether, all representatives and bend sequences can be computed in linear time.

A *crossing* is an intersection between a representative and a vertical cut. Every crossing belongs to two sequences as we describe next.

Let p be a net, let $v_0, \ldots, v_k$ be the bend sequence of p and let $c_1, \ldots, c_m$ be the crossings of rep(p) with vertical cuts in the order in which they occur on rep(p). For $0 \leq i < k$, define the c-sequence of (v_i, v_{i+1}) as the subsequence of crossings that occur between v_i and v_{i+1}.

Let C be any vertical cut. The r-sequence of C is any ordering $c_1, \ldots, c_p$ of the crossings with C satisfying the following three properties: Let $e = (b, a)$, with a the left neighbor of b be the topmost edge intersected by C, and let (p_i^1, e, p_i^2) be an admissible decomposition of net p_i corresponding to crossing c_i, $1 \leq i \leq m$. Then

(1) (*rim property*) There is an integer $k(C) \geq 0$ such that $i \leq k(C)$ implies can $(p_i^1) \subseteq Rim$;

(2) (*ordering property*) If $k(C) < i < j$, then p_i^2 is quasi-right of p_j^2;

(3) (*consistency property*) If D is a vertical cut in the column to the left of C, $k(C) < i < j$ and p_i and p_j also cross D, then $k(D) < i' < j'$ where c_i' is the next crossing of p_i and D and c_j' is the next crossing of p_j and D.

Remark. Initially, we will have $k(C) = 0$, for all C. Then (2) states that the r-sequence of C reflects the ordering quasi-right and (3) states that the orderings for adjacent cuts are consistent with one another. We are not able to maintain (2) and (3) with $k(C) = 0$ for all C. One of the difficulties is case 4.3. In the situation of Fig. 11 the new crossing of p with X would have to be inserted somewhere into the r-sequence of X.

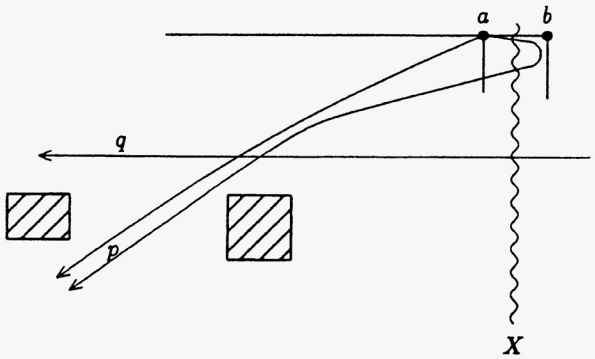

FIG. 11.

We will instead add it to the front of the sequence and increase $k(X)$ by one. In this way (1), (2), and (3) are maintained with small cost. However, the ordering quasi-right in cases 3, 4.2, and 5.1 is now harder to compute. In case 3 we solve this problem by basing the decision on cuts that do not intersect the *Rim* and hence have $k(C) = 0$; in case 4.2 we use the fact that can$(ep^2) \subseteq$ *Top* for the rightmost decomposition (p^1, e, p^2), and in case 5.1 we can always take the first element of the r-sequence. The details are given below.

We call $c_1, \ldots, c_{k(C)}$ the *rim-part* and $c_{k(C)+1}, \ldots, c_m$ the *non-rim-part* of the r-sequence of C.

Concrete. A bend sequence is realized as a doubly linked list (see Fig. 12). The elements of the list are vertices and representatives of c-sequences. Each c-sequence is a doubly linked list of crossings; a crossing is represented by a record to be described below. The representative of a c-sequence points to the first and the last item of the c-sequence. An r-sequence is represented as a singly linked list. There is an array rheads[ ] of list headers for the r-sequences. rhead[i] points to the first element of the r-sequence for cut i. We describe below how this data structure is initialized.

There are several functions that can be applied to crossings and c-sequences. For a crossing c, Findrep(c) returns the representative of the c-sequence to which c belongs, Findcut(c) returns the number of the cut whose r-sequence contains c, Above(c, d) yields true for two crossings c, d belonging to the same r-sequence if c is in front of d in the sequence, Split(c) splits the c-sequence containing the crossing c after c and returns the representatives of the two resulting sequences, Addleft(r, c) and Addright(r, c) add the crossing c to the c-sequence with representative r, Delete(c) deletes the crossing c from the c-sequence containing it, and Access&Split(r, x) splits the c-sequence with representative r after the crossing with x-coordinate x and returns the representatives of the resulting c-sequences.

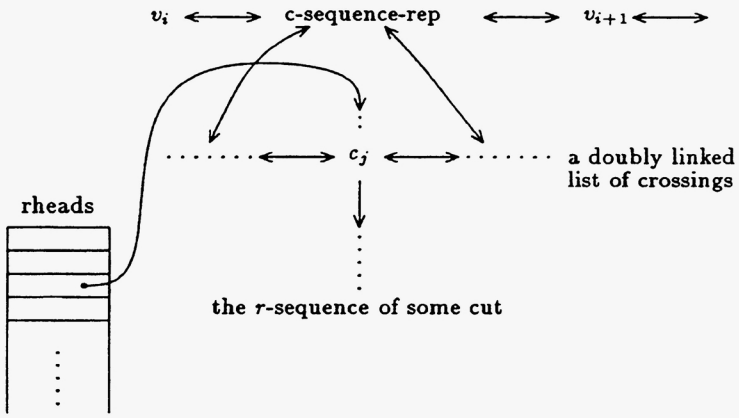

FIG. 12.

Concrete. We realize a crossing as a record consisting of the following fields: Two pointers *csuc* and *cpred* for the c-sequence; a pointer *rsuc* for the r-sequence; an integer *cutnumber*, which is the number of the cut containing the crossing; an integer *rank*; and various other fields, which are used to realize the other operations. We postulate that the rank field increases along any r-sequence and hence Above takes time $O(1)$. Findcut also takes time $O(1)$ by virtue of the cutnumber field. The operations Findrep, Split, Addleft, Addright, Delete, and Access&Split comprise the data type *splittable list*. It is shown in [Schw] that splittable lists can be implemented such that all operations have amortized cost $O(1)$. The implementation combines the level-linked trees of [HMRT] with the split-find data structures of [GT], [IA].

We are now ready to describe the initialization of the data structure. We number the cuts from left to right and for each x-coordinate from top to bottom. The c-sequences are easily computed in linear time. The r-sequences are computed as follows. We scan the layout region from left to right. When we reach a certain column, the r-sequences for all cuts to the left of the column are already computed, and ranks are assigned to all crossings in these r-sequences. We now show how to compute the r-sequences for all cuts immediately to the right of the column in time proportional to the length of the column (see Fig. 13).

Consider any cut C to the right of the current column. We first divide the set of crossings with C into three parts: the U-, D- and L-parts; the U-part and D-part consist of all crossings with nets that either terminate in the current column on a module that touches the current column from the right and lies above C or below C, respectively, or that have the next crossing with a cut above C or below C, respectively; and the L-part consists of all other crossings. The r-sequence of C can then be obtained by concatenating an appropriate ordering of the U-part with an appropriate ordering of the L-part with an appropriate ordering of the D-part.

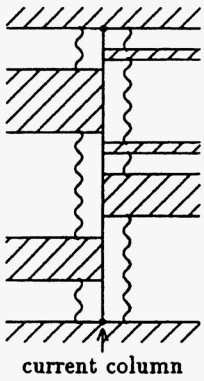

FIG. 13.

We show how to order the L-parts. We first construct a list L consisting of the topmost vertex of the current column, followed by the r-sequence of the first cut to the left of the column, followed by the terminals on the first module touching the current column from the left, followed by the r-sequence of the second cut, ..., and then number the elements of this list in increasing order. We then go through the L-parts of all cuts C and generate for each crossing c with C, say with rep(p), a pair consisting of

- the number of the cut C
- the rank in L of the crossing or terminal adjacent to c in rep(p).

We sort these pairs by bucket sort in time proportional to the length of the column. This gives us the desired ordering of the L-parts, the U- and D-parts can be ordered in a similar way. The ranks are now assigned by numbering the crossings in each q-sequence in increasing order. Altogether we have now shown how to compute the r-sequences of all cuts in linear time.

We finally store for each cut its free capacity (array freecap[ ]). We also have an array satcut[1...] of pointers. The element satcut[i] points to a doubly linked list of all saturated vertical cuts of length i intersecting row *top*. Each saturated cut points to its position on the satcut lists (array satpos[...]). We also keep the nonempty entries of the array *satcut* in a linked list (see Fig. 14). This finishes the description of the data structure.

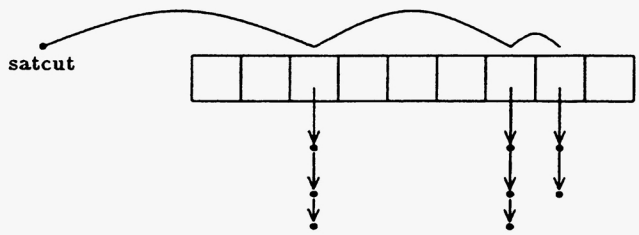

FIG. 14.

4.2. The algorithm. We work through the routing region from top to bottom. Suppose that we are just beginning to process row *top*. Let us also assume that we have the data structure described in §1 available.

Step 1 (*Cuts of capacity one*). A cut of capacity one is necessarily saturated and hence the cuts of capacity one intersecting row *top* are all contained in the list satcut[1]. Let C be any cut in satcut[1], let (a, b) be the edge in row top intersected by C, and let c be the crossing

on the r-list of C. Let c be on the c-sequence of bends u and v. Note that we can find u and v in time $O(1)$ using operation Findrep. We split (operation Split) the c-sequence of u and v at the crossing c and introduce a and b as new terminals (see Fig. 15).

At this point we have created two nets, say p_1 and p_2, with terminals in row top. If either u or v was a terminal of its net then the bend sequences of p_1 and p_2 are correctly computed by the splitting process. If neither u or v was a terminal, then we have to update the bend sequences of p_1 and p_2 as follows. Follow the bend sequence until a vertex, say w, which is not in Rim is encountered, delete the vertices before w and concatenate the appropriate c-sequences. If l vertices were deleted then $l + 1$ c-sequences have to be concatenated. We concatenate these sequences by adding (operation Add) the items on the second, third, ..., sequence to the first sequence one by one. Note that a crossing can be added at most once to a c-sequence because it belongs to a c-sequence incident to a terminal after the addition. Hence the total cost of additions is linear.

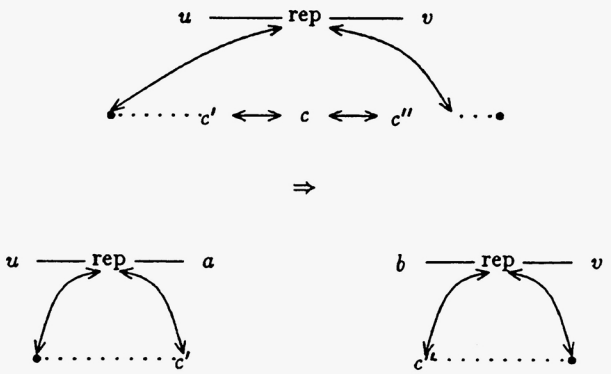

FIG. 15.

Step 2 (*No cuts of capacity one*). All cuts of capacity one are processed. We scan row top from left to right and construct four sets of segments. The set RS, LS, and FS, contains all segments where the right corner is not exposed, the right corner is exposed but the left corner is not exposed, and both corners are exposed, respectively. The set TS contains all segments consisting of a single vertex.

We now discuss cases 2 to 5 of the routing algorithm.

Case 2 applies whenever $TS \neq \emptyset$. Let $S \in TS$ be a segment, let b be the single vertex in S, let p be the net incident to b, and let a be the lower neighbor of b. We replace b by a on the bend sequence of p and delete the edge $e = (b, a)$. This takes constant time.

Case 3 applies whenever $TS = \emptyset$ and $RS \neq \emptyset$. Let S be any segment in RS, let b be the right corner of S, let a be the left neighbor of b, and let p and q be the two nets with terminal b. Let v_1 and w_1 be the vertices following b on the bend sequences of p and q, respectively.

Case A. At least one of the nets p and q leaves the column of b to the left. This can be checked in time $O(1)$ by inspecting the first crossing of p and q. We may assume w.l.o.g. that p leaves the column of b to the left. If q does not then p is quasi-right of q. So let us suppose that q also leaves the column of b to the left. If v_1 and w_1 are different or v_1 and w_1 are equal and at least one is a terminal or at least one of the nets makes a left turn at v_1, then it is easy to decide whether p is quasi-right of q using the coordinates of v_1 and w_1. So let us assume that v_1 and w_1 are equal and both nets make a right turn at v_1. In this case both nets enter the bend point v_1 from above. Let c and d be the crossings following v_1 on p and q, respectively, and let C be the cut containing c and d. Then C does not intersect row top and hence p is

quasi-right of q if c is above d in the r-sequence of C. In either case we have shown that the test whether p is quasi-right of q takes time $O(1)$.

Let w.l.o.g. p be quasi-right of q. We replace b by a on the bend sequence of p and delete (operation Delete) the first crossing from the first c-sequence of p. We also replace b by its lower neighbor on the bend sequence of q. Finally, we mark (array mark) the cut, say X, through the edge (a, b) as processed, add it to the set of marked cuts, delete X from its satcut-list, if it is on one, and insert it into the satcut-list of one smaller index. All of this takes time $O(1)$.

Remark. The mark on cut X indicates that X does not start in row *top* anymore. The mark bit will be used in case 5. The set of marked cuts is used to unmark the marked cuts again when row *top* is completely processed.

If p terminates in b, then we remove S from RS and add it to LS or FS whatever is appropriate (a segment is stored as a pair of vertices and hence this is an $O(1)$ decision). If p does not terminate in b, then S stays in RS.

Case B. p and q leave the column of b to the right. Let c and d be the first crossings of p and q and let C and D be the cuts crossed. If $C \neq D$, then it is easy to decide whether p is quasi-right of q. We may assume w.l.o.g. that p is quasi-right of q in this case. If $C = D$, then p is quasi-right of q and q is quasi-right of p. We may assume w.l.o.g. that either c belongs to the rim-part of the r-sequence of C or that both c and d belong to the non-rim-part and d is above c.

We replace b by a on the bend sequence of p, add the crossing of p and X, where X is the vertical cut through edge (a, b), as first element of the rim-part of the r-sequence of X if c belongs to the rim-part of C and as first element of the non-rim-part otherwise (this preserves the ordering property and the consistency property) and add the crossing to the appropriate c-sequence. We also decrease the free capacity of X by two. If X becomes saturated then we add X to the appropriate satcut-list and mark X. The appropriate satcut-list is determined by linear search in time proportional to the length of X. Since every cut becomes saturated at most once, the total cost of adding saturated cuts to satcut-lists is linear.

Case 4 applies whenever $TS = RS = \emptyset$ and $LS \neq \emptyset$. Let S be any segment in LS. The right corner of S is exposed and the left is not.

Case 4.1: There is a net p with $\text{can}(p) \subseteq S$. Let p be the one with leftmost right terminal.

We can find p as follows. We search through the segment starting in the left corner of S. For every vertex v encountered we inspect the bend sequence of the net incident to v and determine in time $O(1)$ whether v qualifies as vertex t. Thus t can be found in time proportional to its distance from the left corner of S.

We route as shown in Fig. 16. All cuts between s and t are marked and moved to the satcut-list of one smaller index, if saturated. Also, all crossings of net p are removed. Also the terminal is changed for each net incident to vertices between s and t, segment S is removed from LS, segment S' is added to LS, and segment S'' is added to FS.

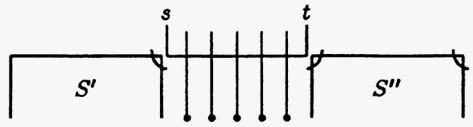

FIG. 16. *Both corners of S'' are exposed and the right corner of S' is exposed.*

Note that the cost of this action is proportional to the length of S' plus the number of edges deleted from the routing region. Since case 4.1 does not again apply to segment S', rather S' is completed by cases 4.2 and 4.3, the total time spent in case 4.1 is linear.

Cases 4.2 and 4.3: There is no net contained in S. Let b be the right corner of S, let a be the left neighbor of b and let X be the vertical cut through (a, b).

Case 4.2: X is saturated. Let p be the quasi-rightmost net across X. We have shown in §3 that p has a terminal s in segment S. We can find p as follows: We start in the right corner of S and walk to the left. For each terminal encountered in the walk we check whether the associated net crosses X by examining its bend sequence. This takes time $O(1)$ per terminal. The first net encountered which crosses X is the desired net p. We route p along the top row (see Fig. 17), route all nets between S and the right corner of S down by one unit, update the data structure as in case 4.1, and put S' on LS.

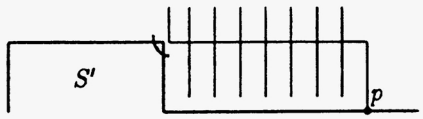

FIG. 17.

Note that case 4.2 or 4.3 applies again to S'. The time required is proportional to the number of edges deleted from the routing region.

Case 4.3: X is not saturated. Let p be the net with terminal a. We route p form a to b and then down by one unit (see Fig. 18).

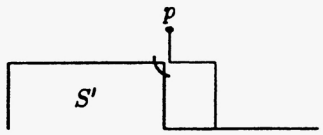

FIG. 18.

If p did not go across X before the action (inspect the first crossing of the net p), then we add a crossing as first element of the rim-part of the r-sequence of X (this preserves the ordering and the consistency property), decrease the free capacity of X by two, and add X to a satcut-list if X became saturated. All of this takes time $O(1)$ if X did not become saturated and time $O(1+$ length of $X)$ otherwise. Thus the total time spent in case 4.3 is linear. Also case 4.2 or 4.3 applies again to segment S'.

Case 5 applies whenever $TS = LS = RS = \emptyset$ and $FS \neq \emptyset$.

We keep a pointer P to the satcut-lists with the following semantics. If the pointer points to list satcut[i] then all cuts in satcut[j], $j < i$, and all cuts preceding the item pointed to in satcut[i] are marked.

We advance P until it points to an unmarked cut. This takes time $O(1)$ per move of P. Note that all cuts added to the satcut-lists during the execution of cases 2, 3, and 4 are marked and hence P never has to be reset. Also the total length of the satcut-list is at most the number of edges in rows top and $top - 1$ and hence the total time spent on advancing P is linear.

Case 5.1: Let X be the saturated cut found, and let (a, b) be the edge in row top intersected by X. Clearly, X is a shortest vertical cut intersecting an edge of the segment containing edge (a, b). Let c be the first crossing on the r-list of C. Let c be on the c-sequence of u and v. Note that we can find u and v in time $O(1)$ using Findrep. Figure 19 shows how the representatives of the nets p^1 and p^2 look like. The bend sequences of p^1 and p^2 can be found as follows.

Let y be the y-coordinate of vertices u and v. Then the bends of p^1 and p^2 are readily determined by inspecting the L- and R-values of all vertices w which lie below a in rows $top, top - 1, \ldots, y$. This takes time $O(1 + top - y)$; since the sum of the lengths of the

representatives of p^1 and p^2 exceeds the length of the representative of p by $2(top - y)$, we can account for the cost by the length increase; note that the total length of the canonical representatives is always bounded by the size of the routing region.

At this point we have computed the bend sequences of p^1 and p^2. We then split the c-sequence of u and v at the appropriate places using the operation Access&Split. This takes amortized time $O(1)$ for each Access&Split and hence the cost is easily accounted for.

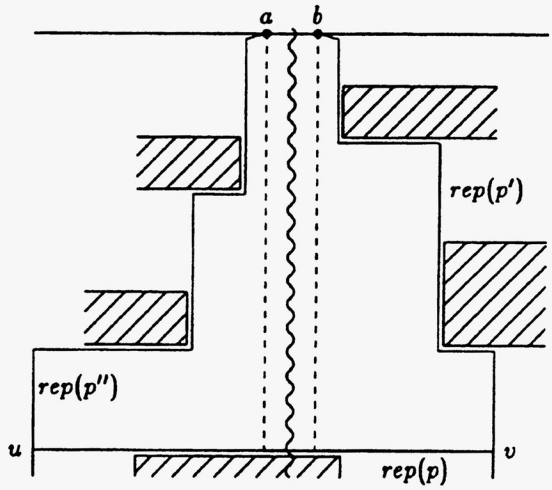

FIG. 19.

Finally, we observe that the ordering and consistency property is preserved.

Case 5.2 (no saturated cut found): Let S be any segment in FS. We route as shown in Fig. 20.

FIG. 20.

All cuts between s and t have their free capacity reduced by two and are added to a *satcut*-list if necessary. This completes the description of the various cases of the routing algorithm.

Suppose now that we processed row top completely. We unmark all cuts using the set of marked nets in time proportional to their number, decrease top by one, go through *Top* and adjust bend-sequences as described in case 1, and are ready for processing the new row top.

This completes the proof of the main theorem.

REFERENCES

[BM] M. BECKER AND K. MEHLHORN, *Algorithms for routing in planar graphs*, Acta Inform., 23 (1986), pp. 163–176.

[F] A. FRANK, *Disjoint paths in a rectilinear grid*, Combinatorica, 2 (1982), pp. 361–371.

[GT] H. N. GABOW AND R. E. TARJAN, *A linear-time algorithm for a special case of disjoint set union*, in Proc. of the 15th SIGACT Symp., 1983, pp. 246–251.

[HMRT] K. HOFFMANN, K. MEHLHORN, P. ROSENSTIEHL, AND R. E. TARJAN, *Sorting Jordan sequences in linear time using level-linked search trees*, Inform. and Control, 68 (1986), pp. 170–184.

[IA] H. IMAI AND T. ASANO, *Dynamic orthogonal segment intersection search*, J. Algorithms, 8 (1987), pp. 1–18.

[K] M. KAUFMANN, *A linear-time algorithm for routing in a convex grid*, IEEE Trans. Computer-Aided Design, (1990), pp. 180–184.

[KM1] M. KAUFMANN AND K. MEHLHORN, *Routing through a generalized switchbox*, Journal Algorithms, 7 (1986), pp. 510–531.

[KM2] ———, *On local routing of two-terminal nets*, J. Combin. Theory Ser. B., (to appear).

[MP] K. MEHLHORN AND F. P. PREPARATA, *Routing through a rectangle*, J. Assoc. Comput. Mach., 33 (1986), pp. 60–85.

[NSS] T. NISHIZEKI, N. SAITO, AND K. SUZUKI, *A linear-time routing algorithm for convex grids*, IEEE Trans. Computer-Aided Design, CAD-4 (1985), pp. 68–76.

[Sh] A. SCHRIJVER, *Edge-disjoint homotopic paths in straight-line planar graphs*, Universität des Saarlandes, preprint.

[Schw] CH. SCHWARZ, *Ein Suchproblem in dynamischen Folgen*, Diplomarbeit, Universität des Saarlandes, 1989.

[W] B. WEINELT, *Homotopische Knock-Knee-Verdrahtung: Eine Linearzeit-Implementierung*, Diplomarbeit, Universität des Saarlandes, 1990.