Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

Linear-Time Algorithms for Unspecified Routing in Grids *
(Extended Abstract)

Wun-Tat Chan Francis Y.L. Chin

Department of Computer Science
The University of Hong Kong, Hong Kong
E-mail: *wtchan@cs.hku.hk, chin@cs.hku.hk*

Abstract

We give optimal algorithms for an unspecified *channel routing problem which can be formulated as follows: Given a rectangular $p \times q$ grid, two sets of boundary vertices, $\mathcal{S}$ and $\mathcal{T}$, of equal size, i.e., $|\mathcal{S}| = |\mathcal{T}| = n$, the problem is to find n edge-disjoint or vertex-disjoint paths joining vertices in $\mathcal{S}$ with vertices in $\mathcal{T}$. For both edge-disjoint and vertex-disjoint cases, we can test the existence of feasible solution in $O(p+q)$ time and find the disjoint paths in $O(pq)$ time.*

Keywords. Channel Routing, VLSI Routing, Edge-Disjoint Paths, Vertex-Disjoint Paths, Two-Dimensional Grids, Linear-Time Algorithm.

1 Introduction

When the underlying graph of a planar routing problem is a grid, this problem has direct application on VLSI routing. Routing problems have been studied widely by many researchers [4, 7, 9, 10, 13, 14]. Most of the efforts are spent on "specified" routing [4, 7, 9, 10], that is, given a set of nets, $\{(s_1,t_1),(s_2,t_2),\ldots,(s_n,t_n)\}$ where (s_i,t_i) is a net of terminals such that $s_i \in \mathcal{S}$ and $t_i \in \mathcal{T}$ for $1 \leq i \leq n$. The "specified" routing is to find n pairwise disjoint paths joining s_i to t_i, where $1 \leq i \leq n$. There are two different interpretations of pairwise disjoint paths, *edge-disjoint* paths [4, 7, 9, 10] and *vertex-disjoint* paths [13, 14].

Channel routing assumes the region to be routed is in the form of a $p \times q$ rectangle and the terminals of the nets lie on its boundary. Frank [4] gave some necessary conditions for the existence of edge-disjoint paths in the channel. Nishizeki et al. [10], Lai and Sprague [7] applied the conditions and showed that the problem can be solved in $O(N)$ time where $N = pq$.

In this paper, we are interested in an *unspecified* channel routing problem which can be formulated as follows:

> Given a rectangular $p \times q$ grid, two sets of boundary vertices as terminal sets, $\mathcal{S}$ and $\mathcal{T}$, of equal size, i.e., $|\mathcal{S}| = |\mathcal{T}| = n$, the problem is to find n edge-disjoint or vertex-disjoint paths pairing vertices in $\mathcal{S}$ with vertices in $\mathcal{T}$.

In the unspecified routing problem, a vertex in $\mathcal{S}$ can be connected to *any* vertex in $\mathcal{T}$ which forms an one-to-one correspondence between $\mathcal{S}$ and $\mathcal{T}$, while in the specified routing, each vertex in $\mathcal{S}$ has to connect to a specified vertex in $\mathcal{T}$. We shall show in this paper that, for the edge-disjoint and vertex-disjoint problems, we can test the existence of feasible solutions in $O(p+q)$ time and can find the disjoint paths in $O(N)$ time.

Our main contributions of this paper are:

(a) Although the unspecified routing problem seems to be simplier than the specified routing problem and the conditions for testing existence are similar, these two problems are different. For instance, in the edge-disjoint or vertex-disjoint cases, there are examples where there exists no solution for the specified routing problem, but not for the unspecified routing problem (Figure 1).

(b) Unspecified edge-disjoint and vertex-disjoint path routing problems on a grid have not been studied previously.

*The research is partially supported by an RGC grant 338/065/0022.

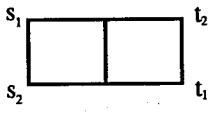

Figure 1: $\mathcal{S} = \{s_1, s_2\}$ and $\mathcal{T} = \{t_1, t_2\}$.

(c) The algorithms given in this paper are simple, easy to implement and time-optimal.

Throughout this paper, we shall assume that all the routing problems will be unspecified unless stated otherwise. The edge-disjoint routing problem can be solved by reducing it to a multiple-source, multiple-sink flow problem [8] on a grid network, where all those boundary vertices in $\mathcal{S}$ are *sources* with unit supply, all those boundary vertices in $\mathcal{T}$ are *sinks* with unit demand and every grid edge has unit capacity. As the grid is planar, it can be shown in [8] that this multiple-source, multiple-sink problem on a planar graph can be solved in $O(N^{4/3}logN)$ time when making use of a fast shortest-path algorithm for planar graph [6]. Alternatively, we can reduce a multiple-source, multiple-sink problem to a single-source, single-sink MAX-FLOW problem by connecting the sources to a super-source and sinks to a super-sink. However, this reduction may destroy the planarity of the graph.

The vertex-disjoint routing problem can also be reduced to MAX-FLOW problem. In particular, each grid vertex in the grid is "split" into two and the two vertices are then connected by a directed edge with unit capacity which restricts that only one path can pass through the grid vertex [2]. The reduction also destroys the planarity property of the grid but results in a simple network [11] with unit capacity which leads to an $O(N^{1.5})$ time algorithm for the vertex-disjoint routing problem.

Another relevant unspecified routing problem in a grid was introduced in [12], where the n vertices in $\mathcal{S}$ are inside the grid instead of the boundaries and the set $\mathcal{T}$ includes all the boundary vertices of the grid. Note that the number of vertices in $\mathcal{T}$ is not necessary n in this problem. If the routing paths are restricted to straight lines and vertex-disjoint, an efficient algorithm with $O(n \log n)$ time complexity, independent of the grid size, is given in [1]. However, the general problem is much more difficult and the best known algorithm takes $O(n^3)$ time. In the forthcoming paper [3], we can use the results in this paper to improve the general edge-disjoint and vertex-disjoint routing problem in a grid to $O(n^{2.5})$ time.

The rest of this paper is organized as follows. We propose for the edge-disjoint case and vertex-disjoint case of the unspecified routing problem, in Section 2 and Section 3 respectively, the necessary and sufficient condition for the existence of feasible solutions and method of constructing the set of disjoint paths in the grid if exists. Section 4 concludes this paper.

2 Edge-disjoint routing in grid

2.1 Necessary and sufficient condition

Given a rectangular $p \times q$ grid denoted by $[1 :: p] \times [1 :: q]$, vertices in the grid can be represented by (i, j) for $1 \le i \le p, 1 \le j \le q$ where i is the number of row from the top and j is the number of column from the left. The boundary of the grid consists of a set of vertices $\mathcal{B} = \{(i,j) \mid i = 1,\ i = p,\ j = 1$ or $j = q\}$. In the unspecified routing problem, we have $\mathcal{S}, \mathcal{T} \subset \mathcal{B}$ and $|\mathcal{S}| = |\mathcal{T}|$. A set of n edge-disjoint paths *satisfies* $\mathcal{S}$ and $\mathcal{T}$ if each of the n paths joins a vertex in $\mathcal{S}$ with a vertex in $\mathcal{T}$. WLOG, we assume each of the edge-disjoint paths is directed and starting from a vertex in $\mathcal{S}$ to a vertex in $\mathcal{T}$.

We define a *cut* (*edge cut*) as a set of grid edges whose removal will partition the grid into two components. In particular, an *h-cut*(i) is a horizontal cut which denotes the ith row of grid edges and *v-cut*(j) is a vertical cut which denotes the jth column of grid edges. We further define $hc(i)$ and $vc(i)$ as the *capacities* (number of edges) of *h-cut*(i) and *v-cut*(j) respectively. For example in a rectangular $p \times q$ grid, *h-cut*$(i) = \{((i,b),(i+1,b)) \mid 1 \le b \le q\}$, *v-cut*$(j) = \{((a,j),(a,j+1)) \mid 1 \le a \le p\}$, $hc(i) = q$ and $vc(j) = p$. The *demand of a cut* represents the least number of paths needed to pass through the cut in order to satisfy $\mathcal{S}$ and $\mathcal{T}$. In particular, $hd(i)$ and $vd(j)$ denote the least number of edge-disjoint paths needed to pass through the edges from top to bottom and from left to right in *h-cut*(i) and *v-cut*(j) respectively. If the value of the demand is negative, it represents the least number of paths passing through the cut in the other direction. For instance, $hd(i) = k_1 - k_2$ where k_1 and k_2 are numbers of boundary vertices, above the $(i+1)$th row, in $\mathcal{S}$ and $\mathcal{T}$ respectively. Similarly, $vd(j) = k_3 - k_4$ where k_3 and k_4 are numbers of boundary vertices, on the left of the $(j+1)$th column, in $\mathcal{S}$ and $\mathcal{T}$ respectively. We say,

$$\textit{h-cut}(i) \text{ is } \begin{cases} \text{saturated} & \text{if } |hd(i)| = hc(i) \\ \text{overflowed} & \text{if } |hd(i)| > hc(i) \end{cases}$$

$$\textit{v-cut}(j) \text{ is } \begin{cases} \text{saturated} & \text{if } |vd(j)| = vc(j) \\ \text{overflowed} & \text{if } |vd(j)| > vc(j). \end{cases}$$

Intuitively, if we compute the capacities and demands of all the cut in the grid, we can deduce if there is a set of edge-disjoint path satisfies $\mathcal{S}$ and $\mathcal{T}$. The following lemma shows that the set of cuts *h-cut*(i) and *v-cut*(j) are sufficiently large that can be able to determine if the feasible solution exists.

Lemma 2.1 Given a rectangular $p \times q$ grid and two sets of boundary vertices $\mathcal{S}$ and $\mathcal{T}$ with $|\mathcal{S}| = |\mathcal{T}|$, there exists a set of edge-disjoint paths satisfying $\mathcal{S}$ and $\mathcal{T}$ if and only if all the *h-cut*(i) for $1 \leq i \leq p-1$ and *v-cut*(j) for $1 \leq j \leq q-1$ are not overflowed.

Proof: Only if part: Obviously, if one of the *h-cut*(i) or *v-cut*(j) is overflowed, then the set of edge-disjoint paths satisfying $\mathcal{S}$ and $\mathcal{T}$ cannot exist.

If part: (sketch) We can transform this edge-disjoint routing problem into an integral MAX-FLOW problem by connecting those vertices in $\mathcal{S}$ to the source of the network and those vertices in $\mathcal{T}$ to the sink of the network. By the max-flow-min-cut theorem, if a set of n edge-disjoint paths satisfying $\mathcal{S}$ and $\mathcal{T}$ does not exist, a cut-set in the network will exists with capacity less than n. Moreover, we can show that that cut-set can be transformed to another cut-set with same capacity which contains one of the *h-cut*(i) or *v-cut*(j) which has to be overflowed. □

We have the following theorem directly from Lemma 2.1.

Theorem 2.2 Given a rectangular $p \times q$ grid and two sets of boundary vertices $\mathcal{S}$ and $\mathcal{T}$ with $|\mathcal{S}| = |\mathcal{T}|$, we can determine if there exists a set of edge-disjoint paths in the grid satisfying $\mathcal{S}$ and $\mathcal{T}$ in $O(\min\{p+q, n\})$ time.

Proof: It takes $O(p+q)$ time to compute the demands of all the *h-cut*(i) and *v-cut*(j) in the grid. However, for an *h-cut*(k), if there is no terminal on row k, we have $hd(k) = hd(k-1)$. It is similar for the demands of *v-cut*(j) for $2 \leq j \leq q-1$. Therefore, it is necessary to compute only the demands of *h-cut*(i) for $2 \leq i \leq p-1$ and *v-cut*(j) for $2 \leq j \leq q-1$ when there are terminals on row i and column j respectively. □

2.2 Paths construction

In this section, an efficient algorithm is proposed to find such a set of edge-disjoint paths in the rectangular gird. Lemma 2.1 gives the condition which ensures the existence of a set of edge-disjoint paths satisfying $\mathcal{S}$ and $\mathcal{T}$. In fact, we can easily generalize the rectangular grid in Lemma 2.1 to an L-shape grid. An example of an L-shape grid is shown in the shaded region of Figure 2. The idea of our algorithm is that grid vertices will be considered row by row, one at a time from top to bottom, and each row from left to right. When vertex (i,j) is considered, the sum of values of the flows (paths with direction) through its top edge $((i-1,j),(i,j))$ and its left edge $((i,j-1),(i,j))$ is computed, and then the flows through the its right edge $((i,j),(i,j+1))$ and its down edge $((i,j),(i+1,j))$ will be determined. After the vertex is considered, the edges connected to it and itself are removed from the grid. An L-shape grid is formed. Throughout the algorithm, we make sure that the flow at each vertex is conserved, i.e., the number of paths entering the vertex is equal to the number of paths leaving the vertex, and all the *h-cut*(i) and *v-cut*(j) in the subsequent L-shape grids are not overflowed. This guarantees that there still exists sets of edge-disjoint paths in the subsequent L-shape grids.

Let us consider the ith row of vertices in Figure 2 and assume the set of vertices $\{(i,1),\ldots,(i,j-1)\}$ have been considered, $f(a)$, the flow on edge $a = ((i-1,j),(i,j))$, and $f(b)$, the flow on edge $b = ((i,j-1),(i,j))$, would have been determined. Initially, when $i = 1$, $f(a)$ for vertex $(1,j)$ may be $1, -1$ or 0 depending on whether vertex $(1,j)$ is in $\mathcal{S}$, in $\mathcal{T}$ or not in $\mathcal{S}$ nor $\mathcal{T}$. Similarly, $f(b)$ can be defined for $j = 1$. Note that $f(a) > 0$ ($f(b) > 0$) indicates a flow along the direction on edge a (b) while a negative $f(a)$ ($f(b)$) indicates a reverse flow. Consider vertex (i,j) and base on the net flow $f(a)+f(b)$, we can determine the flows $f(c)$ and $f(d)$ on edge $c = ((i,j),(i+1,j))$ and edge $d = ((i,j),(i,j+1))$. The assignment of $f(c)$ and $f(d)$ should ensure that the flow at vertex (i,j) is conserved, and the *v-cut*(j) and *h-cut*(i) in the new L-shape grid, after removing the vertex (i,j), are kept from overflowed. A detailed description of the assignment the flows $f(c)$ and $f(d)$ under the 5 cases of the net flow $f(a)+f(b)$ can be found in Appendix A. The proof of correctness of the assignment is omitted here.

Theorem 2.3 Given a rectangular $p \times q$ grid and two sets of boundary vertices $\mathcal{S}$ and $\mathcal{T}$ with $|\mathcal{S}| = |\mathcal{T}|$, and all the *h-cut*$(i)$ for $1 \leq i \leq p-1$ and *v-cut*(j) for $1 \leq j \leq q-1$ not overflowed, we can find a set of edge-disjoint paths in the gird satisfying $\mathcal{S}$ and $\mathcal{T}$ in $O(N)$ time, where $N = pq$.

Proof: (sketch) We shall prove the correctness of

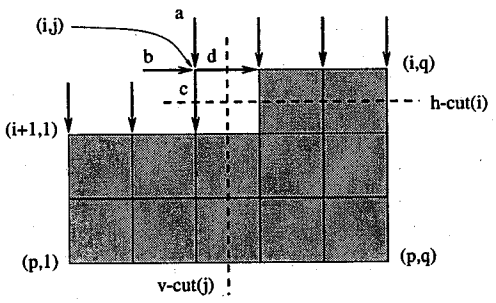

Figure 2: Finding edge-disjoint paths through vertex (i,j).

the algorithm by induction on j, for $1 \leq j \leq q$. Hypothesis: after we consider the vertex (i,j), (1) the *h-cut*(i) and *v-cut*(j) in the new L-shape grid are not overflowed, i.e., $|hd(i)| \leq hc(i)$, and $|vd(j)| \leq vc(j)$ and (2) the flow at vertex (i,j) is conserved. The assignment of flow at each edge takes constant time and thus the whole algorithm takes $O(N)$ time. □

3 Vertex-disjoint routing in grid

3.1 Necessary and sufficient condition

The vertex-disjoint case is similar to the edge-disjoint case. However, considering two sets of cuts, *h-cut*(i) and *v-cut*(j), are not enough. We define a *cut* (*vertex cut*) as a set of vertices in the grid whose removal will partition the grid into two components. The cut can be uniquely defined by specifying two boundary vertices of the grid and by having the least number of vertices in the cut. Suppose the two boundary vertices are u and v, $cut(u,v)$ specifies the cut, $cap(u,v)$ (the *capacity* of the cut) is the number of vertices in $cut(u,v)$, $d(u,v)$ (the *demand* of the cut) is the least number of paths needed to pass through the cut from left to right (or top to bottom) in order to satisfy $\mathcal{S}$ and $\mathcal{T}$. We can see that $d(u,v)$ is actually the difference in the number of boundary vertices in S and in T on the left (upper) part of the cut. Assume the numbers of boundary vertices in $\mathcal{S}$ and $\mathcal{T}$ on the left (upper) part of the cut are k_1 and k_2 respectively excluding u and v, $d(u,v)$ would be $k_1 - k_2 + \delta$, the value of $\delta = |\mathcal{S} \cap \{u,v\}|$ or $-|\mathcal{T} \cap \{u,v\}|$ depends on which of values will lead to a larger absolute value of $d(u,v)$.

Although the number of cuts in a grid could be $O(pq)$, we shall show that actually a linear number $O(p+q)$ of cuts are needed to determine whether there exists a set of vertex-disjoint paths satisfying $\mathcal{S}$ and $\mathcal{T}$. In fact there are only two types of cuts needed to be considered, *adjacent* and *opposite*. $cut(u,v)$ is called *adjacent* if it contains two boundary vertices u and v on two adjacent boundaries in which the cut separates a single corner v_c from the grid, and u,v are equally distant from v_c. If u,v are not equally distant from v_c, we can determine if the $cut(u,v)$ is overflowed by consider the adjacent $cut(u,v')$ assuming the distance from u to v_c is less than or equal to the distance from v to v_c and v' is on the same row or column of v. $cut(u,v)$ is called *opposite* if it contains two boundary vertices u and v on two opposite boundaries, and the horizontal or vertical distance between u and v is less than p or q depending on whether u and v lie on the top and bottom boundaries or on the left and right boundaries respectively. If the vertices u and v are on the top and bottom (left and right) boundaries, and the horizontal (vertical) distance between them is more than p (q), we can determine if the $cut(u,v)$ is overflowed by consider the opposite $cut(u,v')$ assuming the vertices v,v' are all on the left or right (upper or lower) side of u and the horizontal (vertical) distance between u and v' is $p-1$ ($q-1$).

Lemma 3.1 Given a rectangular $p \times q$ grid and two sets of boundary vertices S and T with $|\mathcal{S}| = |\mathcal{T}|$, there exists a set of vertex-disjoint paths satisfying $\mathcal{S}$ and $\mathcal{T}$ if and only if $|d(u,v)| \leq cap(u,v)$ for all adjacent and opposite $cut(u,v)$ in the grid.

Proof: (sketch) Similar to the proof of Lemma 2.1, Only if part is straightforward.

If part: assume the contrary, we show that there always exists a minimum cut in the transformed network $\mathcal{N}$ (as given in Lemma 2.1), containing one of the adjacent or opposite $cut(u,v)$ which has its absolute demand value greater than its capacity, i.e., $|d(u,v)| > cap(u,v)$. □

3.2 Computing the cuts with maximum demand

By Lemma 3.1, it is sufficient to compute the capacities and demands of all the adjacent and opposite cuts to determine if $\mathcal{S}$ and $\mathcal{T}$ can be satisfied. For a $p \times q$ grid, the number of adjacent cuts is $O(p+q)$ but there are still $O(pq)$ opposite cuts. For any opposite $cut(u,v)$, since $cap(u,v)$ is fixed to p if u and v are on the top and bottom boundaries, and fixed to q if u and v are on the left and right boundaries, it is sufficient to find the opposite cut with maximum absolute demand in each of the above two cases to determine if there is any overflowed cut.

WLOG assume $p \leq q$, let us describe how we can find the vertical opposite $cut(u,v)$, for u,v on the top and bottom boundaries, with maximum absolute demand. For each top boundary vertex $u = (1,i)$, $1 \leq i \leq q$, we compute the cumulative sum top_u as the difference in the number of boundaries vertices in $\mathcal{S}$ and in $\mathcal{T}$ from the corner vertex $(1,1)$ to $(1,i-1)$. Similarly for each bottom boundary vertex $v = (p,i), 1 \leq i \leq q$, we compute the cumulative sum bot_v as the difference in the number of boundary vertices in $\mathcal{S}$ and in $\mathcal{T}$ from vertex $(2,1)$ to the corner vertex $(p,1)$ and then to $(p,i-1)$. Then the problem of finding the maximum absolute demand among the opposite cuts would be equivalent to finding the pair (u,v) having the maximum absolute value of $top_u + bot_v + \delta$ with $|u-v| \leq p-1$. Since the sequences top_u and bot_v along the top and bottom boundaries are *consecutive* in the sense that the difference between two consecutive members in the sequence is zero or one, the maximum absolute value $top_u + bot_v + \delta$ with $|u-v| \leq p-1$ can be computed in $O(p+q)$ time. This can be done by computing for each vertex $w = (1,i), 1 \leq i \leq q$, the maximum absolute value $top_w + bot_v + \delta$ with $|w-v| \leq p-1$. In general, a sorted list of bot_v for $|w-v| \leq p-1$ is maintained, the maximum absolute value of $top_w + bot_v + \delta$ can then be determined by finding the maximum/minimum bot_v in the sorted list. When vertex $(1,i+1)$ is considered, we only need to "insert" the element $bot_{(p,i+q)}$, "delete" the element $bot_{(p,i-q+1)}$, and then find the new maximum/minimum in the sorted list. Since the sequence is consecutive, the above three operations can be done in constant time. As a result, the maximum absolute value $top_u + bot_v + \delta$, i.e., the maximum absolute demand can be found in $O(p+q)$ time. Finding the horizontal opposite cut $cut(u,v)$ with maximum absolute demand is easy because the vertical distance between u and v is always less than q. Therefore, no "insert" and "delete" is needed.

Theorem 3.2 Given a rectangular $p \times q$ grid and two sets of boundary vertices $\mathcal{S}$ and $\mathcal{T}$ with $|\mathcal{S}| = |\mathcal{T}|$, we can determine if there is a set of vertex-disjoint path satisfying $\mathcal{S}$ and $\mathcal{T}$ in $O(p+q)$ time.

Proof: Checking all the adjacent cuts and finding the vertical and horizontal opposite cuts with maximum absolute demand take no more than $O(p+q)$ time. □

3.3 Paths construction

After computing the maximum demand among cuts involving each grid vertex $(1,j)$ for $1 \leq j \leq q$ and the maximum absolute demand among the horizontal cuts (Section 3.2), and confirming that none of these cuts are overflowed, our next step is to find the vertex-disjoint paths satisfying $\mathcal{S}$ and $\mathcal{T}$. Similar to the construction of edge-disjoint paths, the vertex-disjoint paths are constructed by considering each vertex one by one row-wise from top to bottom. WLOG, let us assume that the flows for vertices in the first $(i-1)$ rows have been determined. The flow $f(a_j)$ through the edges $a_j = ((i-1,j),(i,j)$ for $1 \leq j \leq q$, should have been known and they can be treated as sources and sinks on the top boundary of the $(p-i+1) \times q$ lower subgrid $[i :: p] \times [1 :: q]$. Before considering the vertices in the ith row, we apply the algorithm in Section 3.2 on the $(p-i+1) \times q$ lower subgrid to obtain the maximum demand and capacity among cuts involving each grid vertex (i,j) for $1 \leq j \leq q$. With this information, the direction and amount of flow passing through each vertex (i,j), in particular $f(c)$ and $f(d)$ will be determined. The assignment of $f(c)$ and $f(d)$ has to observe the flow conservation rule and depends on whether the cut involved with vertex (i,j) is saturated or not. If the cut at vertex (i,j) is saturated, we have to ensure that there is flow going through vertex (i,j). Moreover, it can be proved that, in general, the assignment of flow at edge d is always advantageous in our algorithm as long as there is no more than one path passing through vertex $(i,j+1)$. A detailed description of the algorithm can be found in Appendix B.

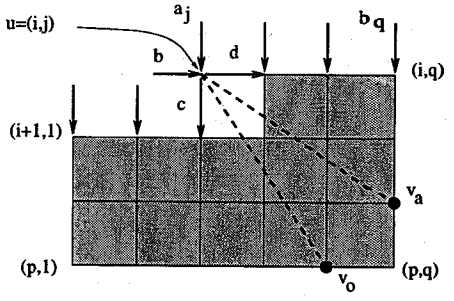

Figure 3: Finding vertex-disjoint path through vertex (i,j).

Theorem 3.3 Given a rectangular $p \times q$ grid and two sets of boundary vertices $\mathcal{S}$ and $\mathcal{T}$ with $|\mathcal{S}| = |\mathcal{T}|$, and the absolute demand value of all adjacent and opposite cuts not greater than their corresponding capac-

ities, we can find a set of vertex-disjoint paths in the grid satisfying $\mathcal{S}$ and $\mathcal{T}$ in $O(N)$ time.

Proof: (sketch) We shall prove the correctness of the algorithm by induction on j, for $1 \leq j \leq q$. Hypothesis: after we consider the vertex (i,j), (1) the absolute demands of the adjacent and opposite cuts involving vertices $(i,j+1)$ and $(i+1,j)$ are not greater than their corresponding capacities. (2) the flow at vertex (i,j) is conserved and at most one path passes through (i,j). The assignment of flow at each edge takes constant time and thus the whole algorithm takes $O(N)$ time. □

4 Conclusion

We have found the necessary and sufficient conditions for the existence of a set of edge-disjoint or vertex-disjoint paths in a rectangular $p \times q$ grid pairing a set of sources with a set of sinks. Based on the conditions, we have devised efficient algorithms to determine its existence in $O(p+q)$ time and to find the disjoint paths in $O(pq)$ time. Related results on specified routing was given in [7, 10]. In those papers, the rectilinear graph does not have to be rectangular grid, in fact, similar results are applicable to convex grids, grids of T-shape or X-shape. Extending our unspecified edge-disjoint routing problem to convex grids and grids of different shapes is straightforward, and we have strong feeling that similar extension of unspecified vertex-disjoint results is also possible.

References

[1] Y. Birk and J.B. Lotspiech. *A Fast Algorithm for Connecting Grid Points to the Boundary with Nonintersecting Straight Lines,* Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms (1991) 465-474.

[2] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications,* North-Holland, Amsterdam (1977).

[3] W.T. Chan and F.Y.L. Chin. *Efficient Algorithms for Finding Non-Intersecting Paths in Grids,* Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (1997), to appear.

[4] A. Frank. *Disjoint Paths in a Rectilinear Grid,* Combinatorica 2(4) (1982), 361-371.

[5] L.R. Ford and D.R. Fulkerson. *Flows in Networks,* Princeton University Press, Princeton, NJ.

[6] P. Klein, S. Rao and M. Rauch. *Faster Shortest-Path Algorithm for Planar Graphs,* Proceedings of the 26th Annual ACM Symposium on Theory of Computing (1994) 27-37.

[7] T-H. Lai and A. Sprague. *On the Routability of a Convex grid,* Journal of Algorithm 8 (1987) 372-384.

[8] G.L. Miller and J. Naor. *Flow in Planar Graphs with Multiple Sources and Sinks,* Proceedings of the 30th IEEE Symposium on Foundations of Computer Science (1989) 112-117.

[9] K. Mehlhorn and F.P. Preparata. *Routing through a Rectangle,* J. ACM 33(1) (1986) 60-85.

[10] T. Nishizeki, N. Saito and K. Suzuki. *A Linear-Time Routing Algorithm for Convex Grids,* IEEE Trans. Comput.-Aided Design 4(1) (1985) 68-76.

[11] C.H. Papadimitrious and K. Steiglitz. *Combinatorial Optimization: Algorithm and Complexity,* Englewood cliffs, NJ: Prentice Hall, (1982).

[12] V.P. Roychowdhury, J. Bruck and T. Kailath. *Efficient Algorithms for Reconfiguration in VLSI/WSI Arrays,* IEEE Trans. on Computers 39(4) (1990) 480-489.

[13] H. Ripphausen-Lipa, D. Wagner, K. Weihe. *The Vertex-Disjoint Menger Problem in Planar Graphs,* Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (1993) 112-119.

[14] A. Schrijver. *Finding k Disjoint Paths in a Directed Planar Graph,* SIAM Journal of Computing 23(4) (1994) 780-788.

Appendix A: Algorithm for constructing edge-disjoint path

Here are the cases when paths are going through the vertex (i,j), i.e., assigning flows $f(c)$ and $f(d)$ for edges c and d respectively. $f(a)$ and $f(b)$ are the flows in the vertex (i,j) which have been determined. $hc(i), vc(j)$ and $hd(i), vd(j)$ are the corresponding capacities and demands of the *h-cut*(i) and *v-cut*(j) respectively. $hd'(i)$ and $vd'(j)$ are the demands in the new L-shape grid after assigning flows $f(c)$ and $f(d)$. These flow functions on the grid edges are graphically represented in Figure 2.

1. $f(a)+f(b)=2$:
   $f(c)\leftarrow 1, f(d)\leftarrow 1$
   $(vd'(j)\leftarrow vd(j)-1, hd'(i)\leftarrow hd(i)-1)$,

2. $f(a)+f(b)=-2$:
   $f(c)\leftarrow 1, f(d)\leftarrow -1$
   $(vd'(j)\leftarrow vd(j)+1, hd'(i)\leftarrow hd(i)+1)$,

3. $f(a)+f(b)=0$:

   (a) $vd(j)=vc(j)$ or $hd(i)=-hc(i)$:
   $f(c)\leftarrow -1, f(b)\leftarrow 1$
   $(vd'(j)\leftarrow vd(j)-1, hd'(i)\leftarrow hd(i)+1)$,

   (b) $vd(j)=-vc(j)$ or $hd(i)=hc(i)$:
   $f(c)\leftarrow 1, f(d)\leftarrow -1$
   $(vd'(j)\leftarrow vd(j)+1, hd'(i)\leftarrow hd(i)-1)$,

   (c) Otherwise:
   $f(c)\leftarrow 0, f(d)\leftarrow 0$
   $(vd'(j)\leftarrow vd(j), hd'(i)\leftarrow hd(i))$,

4. $f(a)+f(b)=1$:

   (a) $vd(j)=vc(j)$ or $hd(i)=-hc(i)+1$:
   $f(c)\leftarrow 0, f(d)\leftarrow 1$
   $(vd'(j)\leftarrow vd(j)-1, hd'(i)\leftarrow hd(i))$,

   (b) Otherwise:
   $f(c)\leftarrow 1, f(d)\leftarrow 0$
   $(vd'(j)\leftarrow vd(j), hd'(i)\leftarrow hd(i)-1)$,

5. $f(a)+f(b)=-1$:

   (a) $vd(j)=-vc(j)$ or $hd(i)=hc(i)-1$:
   $f(c)\leftarrow 0, f(b)\leftarrow -1$
   $(vd'(j)\leftarrow vd(j)+1, hd'(i)\leftarrow hd(i))$,

   (b) Otherwise:
   $f(c)\leftarrow -1, f(d)\leftarrow 0$
   $(vd'(j)\leftarrow vd(j), hd'(i)\leftarrow hd(i)+1)$.

Appendix B: Algorithm for constructing vertex-disjoint path

Here are the cases when path is going through the vertex (i,j), i.e., assigning flows $f(c)$ and $f(d)$ for edges c and d respectively. $f(a_j)$ and $f(b)$ are flows at vertex (i,j) which have been determined. $f(a_{j+1})$ is the flow on edge a_{j+1} at vertex $(i,j+1)$. $cap(u,v_a)$ and $d(u,v_a)$ are the capacity and demand of adjacent $cut(u,v_a)$ respectively. $cap(u,v_o)$ and $d(u,v_o)$ are the capacity and demand of the opposite $cut(u,v_o)$ with maximum absolute demand involving vertex u. These functions are graphically represented in Figure 3.

1. $f(a_j)=1, f(b)=-1$ or $f(a_j)=-1, f(b)=1$:
   $f(c)\leftarrow 0, f(d)\leftarrow 0$.

2. $f(a_j)=f(b)=0$:

   (a) $|d(u,v_a)|<cap(u,v_a)$
   and $|d(u,v_o)|<cap(u,v_o)$:
   $f(c)\leftarrow 0, f(d)\leftarrow 0$.

   (b) $d(u,v_a)=cap(u,v_a)$
   or $d(u,v_o)=cap(u,v_o)$:
   $f(c)\leftarrow -1, \quad f(d)\leftarrow 1$.

   (c) $d(u,v_a)=-cap(u,v_a)$
   or $d(u,v_o)=-cap(u,v_o)$:
   $f(c)\leftarrow 1, f(d)\leftarrow -1$.

3. $f(a_j)=1, f(b)=0$ or $f(a_j)=0, f(b)=1$:

   (a) $|d(u,v_a)|<cap(u,v_a)$
   and $|d(u,v_o)|<cap(u,v_o)$:
   (i) $f(b_{j+1})<1$:
   $f(c)\leftarrow 0, f(d)\leftarrow 1$.
   (ii) $f(b_{j+1})=1$:
   $f(c)\leftarrow 1, f(d)\leftarrow 0$.

   (b) $d(u,v_a)=cap(u,v_a)$
   or ($d(u,v_a)\neq -cap(u,v_a)$
   and $d(u,v_o)=cap(u,v_o)$ and $f(b_{j+1})\neq 1$):
   $f(c)\leftarrow 0, f(d)\leftarrow 1$.

   (c) Otherwise:
   $f(c)\leftarrow 1, f(d)\leftarrow 0$.

4. $f(a_j)=-1, f(b)=0$ or $f(a_j)=0, f(b)=-1$:

   (a) $|d(u,v_a)|<cap(u,v_a)$
   and $|d(u,v_o)|<cap(u,v_o)$:
   (i) $f(b_{j+1})>-1$:
   $f(c)\leftarrow 0, f(d)\leftarrow -1$.
   (ii) $f(b_{j+1})=-1$:
   $f(c)\leftarrow -1, f(d)\leftarrow 0$.

   (b) $d(u,v_a)=-cap(u,v_a)$
   or ($d(u,v_a)\neq cap(u,v_a)$
   and $d(u,v_o)=-cap(u,v_o)$ and $f(b_{j+1})\neq -1$):
   $f(c)\leftarrow 0, f(d)\leftarrow -1$.

   (c) Otherwise:
   $f(c)\leftarrow -1, f(d)\leftarrow 0$.