

Problem 4: A Reduced Gridless Routing Problem

Source: Springsoft Inc.

December 20, 2000

I. Introduction

Given a placement of blockages within a bounding box and a set of nets, a gridless router finds an interconnection path for each net in the routing area. A gridless router has more flexibility than a gridded-based router. It can handle variable widths routes with variable spacing design rules.

There are blockages inside a bounding box and the complementary region is for routing interconnection. The blockages are called *block tiles*, and the routing region is divided into *space tiles*, i.e., the routing area is dissected to obtain a set of space tiles by horizontal lines. For example, see Figure 1, B_1 , B_9 , and etc. with thick borders are block tiles. Start (target) tile is the space tile whose border the start (target) point of a net is located on. In Fig.1, (S1, T1) is a pair of start and target points. Similarly (S2, T2) is another pair. Tiles A1 (A2) and A9 (A10) are the start tile and target tile of the first (second) route.

Traditional gridded-based router algorithms, such as maze routers, can be applied to the gridless router problem with some modifications. Lee's algorithm [1] is based on breadth-first search process. It moves one step forward in all possible directions at one time, and selects the nearest point to the target as the next expansion point. The expansion process stops when the target is reached or this route fails. In the gridless routing problem, a tile, rather than a grid, is the expansion unit. The concept of tile expansion is similar to that of a maze router. The searching process can be Depth-First Search or Breadth-First Search. Time efficiency is the advantage of DFS, and BFS has the advantages of better routing quality. Related papers can be found in [2, 3, 4].

Data structure to keep the neighboring relationship of tiles heavily affects the routing speed. Corner Stitching is an approach to organize the data of tiles [5, 6]. It provides quick query and editing operation for tiling system. Each tile contains four pointers, say tr , rt , lb , bl , to point to its neighbors, as shown in Fig. 2. For a tile, say A , its pointer tr points to the topmost tile whose left edge abuts with its right edge, and pointers rt , lb , and bl are defined similarly. Through these four pointers, we can quickly locate a tile by a point, find all neighbors of a tile. For tile $A1$ in Fig. 2, we can traverse all its top neighbors through pointers

rt and *bl*. It is encouraged to invent a new sophisticated data structure for this tool.

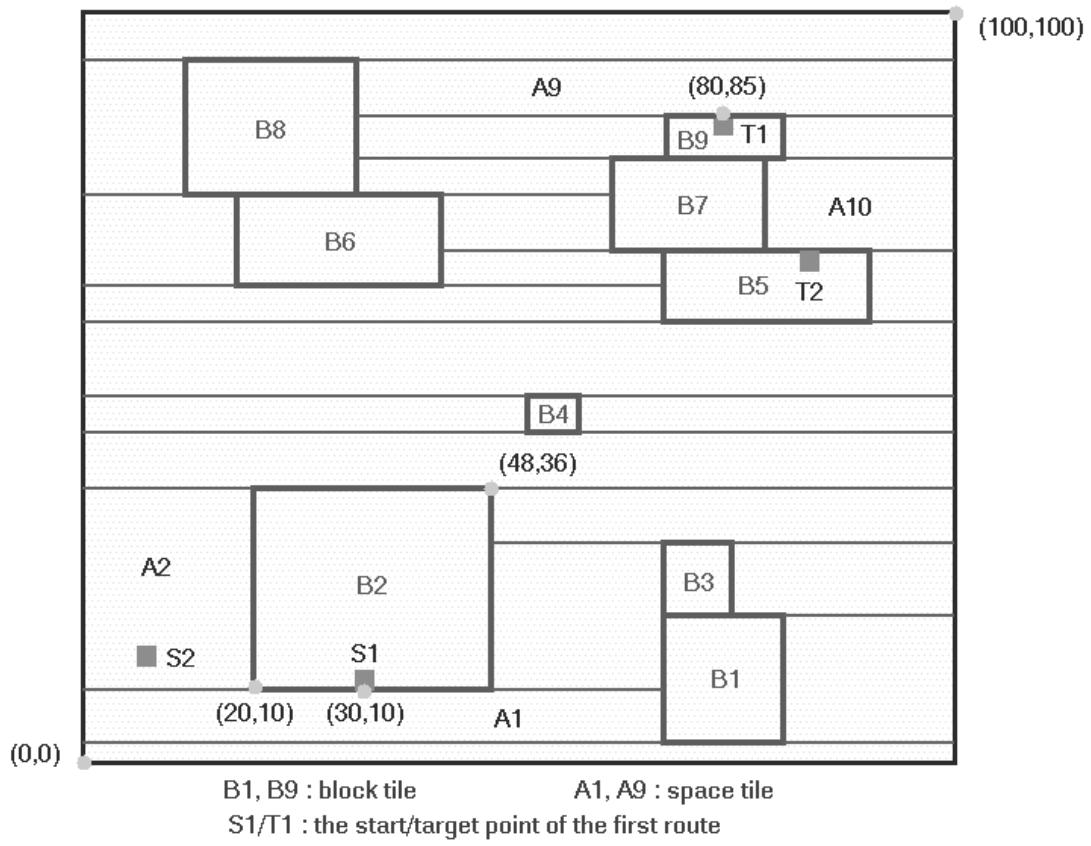
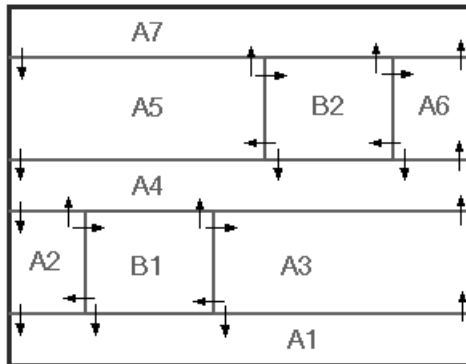
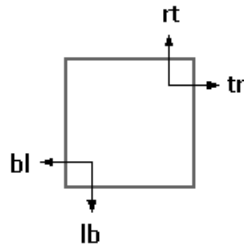


Fig. 1 : A routing tiles example.



A1's top neighbors : A3, B1, A2
A1->rt = A3
A3->bl = B1
B1->bl = A2

Fig. 2 : A Corner Stitching example.

II. Input/Output Specification

Input Format

We shall support two input files for one instance of this problem. One file defines the routing problem, including a routing bounding box, pairs of routing points for all routes, and the block tiles. The other defines all the space tiles. For problem 1, there are two input files, *problem1.blk* and *problem1.spc*. If the tool can produce the space tiles from the first file, it is unnecessary to read the second file. The first file format is as follows:

```
.bBox (point at the left bottom corner) (point at the right top corner)
.route net_name (sx1,sy1) (tx1,ty1)
.route net_name (sx2,sy2) (tx2,ty2)
... More routes
.block_begin
// one line defines a block tile
(point in the left bottom corner) (point in the right top corner)
.block_end
```

Note that the latter route does not need to consider the results of the previous routes. More nets are just to test the router quality with regard to different locations for the same layout.

The file content of Fig. 1 is as follows:

```
.bBox (0,0) (100,100)  
.route netA (30,10) (80,85)  
.route ...  
.block_begin  
(20,10) (48,36)  
... more block tiles  
.block_end
```

The format of the space tile file is as follows:

```
.space_begin  
// one line defines a space tile  
(point in the left bottom corner) (point in the right top corner)  
.space_end
```

Output Format

There are two files, namely *problem1.net* and *problem1.spo* for problem 1, to record the problem output. One records routing path, and the other records all the space tiles. If you do not produce the space tiles automatically, you don't need to output the latter. The format of the routing path file (*problem1.net*) is as follows:

```
// this problem has three routes  
// the first route has n points, the second route fails  
// You don't need to write any comment line in real output  
.net net_name  
(x1,y1) (x2,y2) (x3,y3)...(xn,yn)  
.net net_name  
FAIL  
.net net_name  
(x1,y1) (x2,y2) (x3,y3)  
// remember to output all route results, even those fail to route  
// The first character of each line cannot be a space character and
```

The format of the space tile output file (*problem1.spo*) is as follows:

```
.space_begin  
// each line records one space tile, (x1,y1) is tile's left bottom corner  
// and (x2,y2) is tile's right top corner. Values are separated by one  
// space character.  
// Note that the list order of tiles is from low to high by bottom side and  
// from left to right by left side if tiles have the same bottom side
```

x1 y1 x2 y2

...

.space_end

III. Problem Statement

This problem is a reduced gridless routing problem. Given (1) a placement of blockages inside a bounding box, and (2) a set of two-point nets, the tool completes the route in the routing area on the plane of the bounding box by finding a path for each net. The design rules, including wire width and shape spacing rule, are ignored, and the routing path is regarded as a path of unit width. The basic unit data is a rectangle. Paths must route inside the space tiles and separate from the boundary by one unit. Each instance of the problem has two input files. One file describes an instance of the routing problem, and the other specifies the space tiles. The tool user can either read the space tiles from the file or ask the tool to generate the space tiles automatically. If the space tiles are generated by the tool, the tool has to output them in a sorted order from low to high according to the positions of their bottom borders. If there are tiles having the same position of bottom borders, they must be permuted from left to right according to the left border. The flow is shown in Fig. 3.

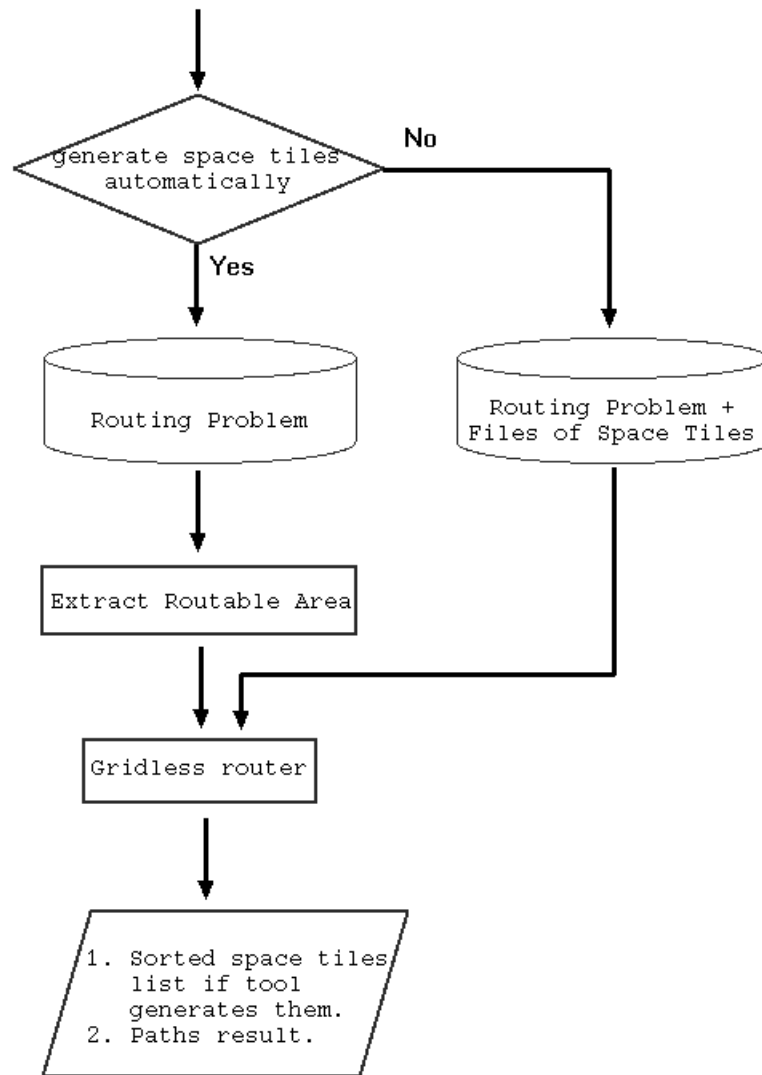


Fig. 3 : The design flow of the gridless routing problem.

We give an example for all IO files in Fig. 4.

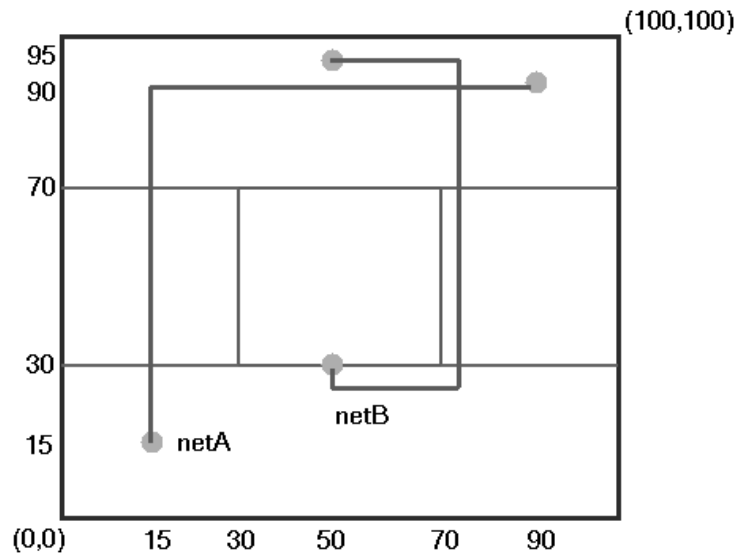


Fig. 4: A routing problem and its solution.

Input files:

[*problem1.blk*]:

```
.bBox (0,0) (100,100)
.route netA (15,15) (90,90)
.route netB (50,30) (50,95)
.block_begin
(30,30) (70,70)
.block_end
```

[*problem1.spc*]:

```
.space_begin
(0,0) (100,30)
(0,30) (30,70)
(70,30) (100,70)
(0,70) (100,100)
.space_end
```

Output files :

[*problem1.net*]

```
.net netA
(15,15) (15,90) (90,90)
.net netB
(50,30) (50,29) (71,29) (71,95) (50,95)
```

Note that *netB* walks along a block tile by separating one unit from its boundary.

[*problem1.spo*]

.space_begin

0 0 100 30

0 30 30 70

70 30 100 70

0 70 100 100

.space_end

IV. Advanced Features

Compare the results by applying DFS and BFS.

V. Language/Platform

1. Language: C or C++.
2. Platform: SUN OS/Solaris or PC DOS/Windows.

VI. Evaluation

The score will be given based on the wire length and the number of corners induced by found paths, time and memory requirements, and whether your tool can generate space tiles automatically. A bonus will be rewarded for comparing DFS and BFS.

VII. Questions

Please report any question regarding this problem to cad@cis.nctu.edu.tw with the subject "CAD Contest: Problem 4." Your question(s) will be answered in two weeks, and the Q&A's will be posted at the contest web site

References

- [1] N. Sherwani. *Algorithms For VLSI Physical Design Automation*. 2nd ed., Kluwer Academic Publishers, Pages 237, 1995.
- [2] A. Margarino, A. Romano, A. DE Gloria, F. Curatelli, and P. Antognetti. A Tile-Expansion Router. *IEEE Transactions on Computer-Aided Design*. VOL. CAD-6, NO. 4, Pages 507-517, July 1987.
- [3] W. L. Schiele, Th. Kruger, K. M. Just, and F. H. Kirsch. A Gridless Router for Industrial Design Rules. In Proc. 27th ACM/IEEE Design Automation Conference. Pages 626-631, 1990.
- [4] M. H. Anold and W. S. Scott. An Iterative Maze Router with Hints. In Proc. 25th ACM/IEEE Design Automation Conference. Pages 672-676, 1988.
- [5] J. K. Ousterhout. Corner Stitching: A Data-Structure Technique for VLSI Layout Tools. *IEEE Transactions on Computer-Aided Design*. VOL. CAD-3, NO. 1, Pages 87-100, January 1984.
- [6] N. Sherwani. *Algorithms For VLSI Physical Design Automation*. 2nd ed., Kluwer Academic Publishers, Pages 107, 1995.