BROWN UNIVERSITY
Department of Computer Science
Master's Thesis
CS-91-M19

"User Constraints for Giotto"

by

Sumeet Kaur Singh

User Constraints for Giotto (Project)

By

Sumeet Kaur Singh

Submitted in partial fullfilment of the requirements for the degree of Master of Science in the Department of Computer Science at Brown University
December 1991

Roberto Tamassia (advisor) Robert Tamassia
Dec. 11, 1991

The goal for giotto, which is a software tool that produces and manipulates diagrams, was to show that the readability of diagrams can be achieved through automatic tools. By readability, we mean that the diagram's meaning can easily be understood simply by the way it is drawn.

The automatic layout ability of giotto can be used in several applications. A few of them include a self-standing documentation tool, support for a graphic editor and support for a design tool.

The paper "Automatic Graph Drawing and Readablity of Diagrams", written by Roberto Tamassia, Giuseppe Di Battista, and Carlo Batini, is the original paper describing giotto's algorithm in detail. In the paper, giotto is described as having a couple of features. One of which is the capability of taking into account user defined constraints on the layout. Adding this capability to giotto is the basis of this project.

1 Defining the Constraints

In giotto, graphs are drawn using the grid standard, with a minimum number of bends and crossings for aesthetic as well as readability purposes. We came up with the following three constraints allowing the user some flexibility.

- Constraining the maximum number of bends for an edge
- Constraining the shape of an edge
- Constraining the angle formed by two edges incident to a vertex

These three constraints will be described in greater detail later. First, we need to have some way for the user to define these constraints. This is described in the following two sections.

2 Gds-to-Paren

Originally, giotto took a graph described in gds (giotto data structure) representation. The following is the gds representation of Figure 1.

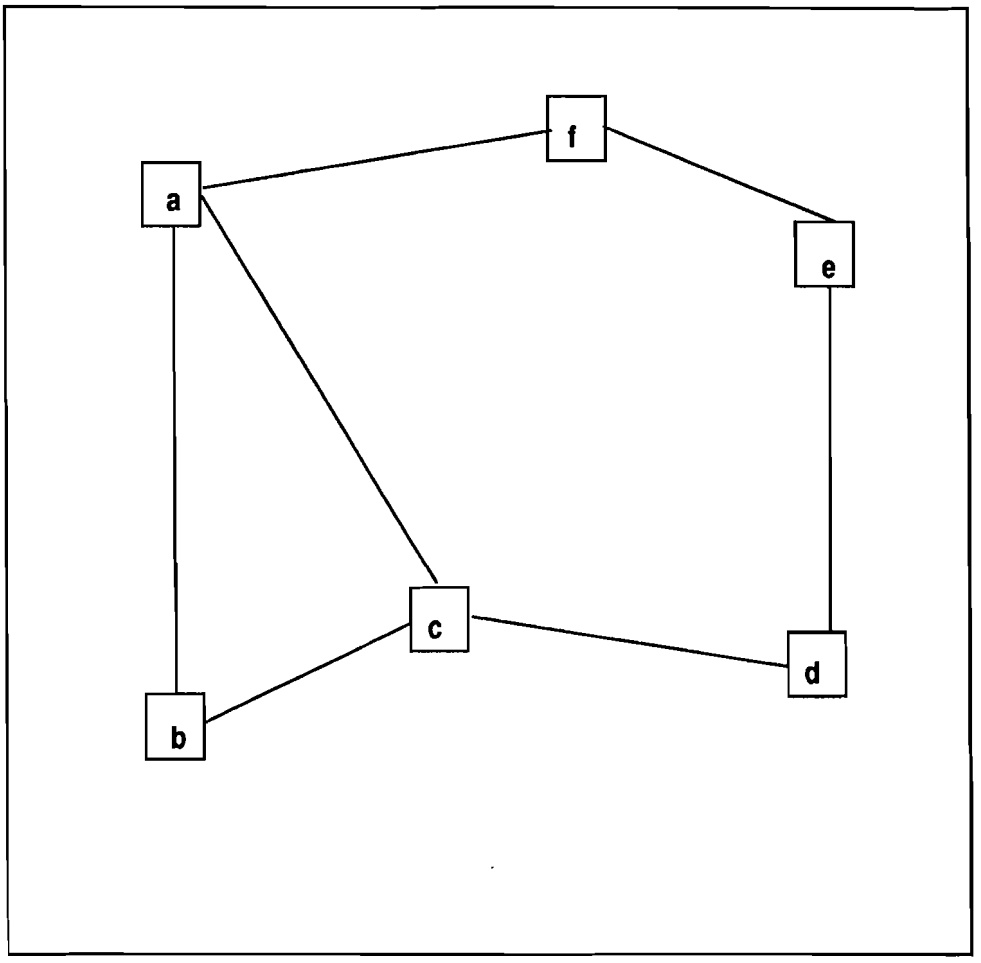

Figure 1:

```
1
6 6 0 0 14



1 0 255 0 1 0 0 0
a
1 0 255 0 2 0 0 0
c
1 0 255 0 3 0 0 0
d
1 0 255 0 4 0 0 0
e
```

```
1 0 255 0 5 0 0 0
f
1 0 255 0 6 0 0 0
b



1 1 53 91 4
1 2 183 280 6
1 3 339 272 10
1 4 343 154 2
1 5 264 46 1
1 6 43 400 3



4 0 -1 -1 14 1 2 0 0
5 0 -1 -1 11 2 1 0 0
1 0 -1 -1 5 3 4 0 0
6 0 -1 -1 7 4 3 0 0
2 0 -1 -1 3 5 6 0 0
6 0 -1 -1 8 6 5 0 0
2 0 -1 -1 13 7 8 0 0
1 0 -1 -1 9 8 7 0 0
3 0 -1 -1 6 9 10 0 0
2 0 -1 -1 12 10 9 0 0
3 0 -1 -1 2 11 12 0 0
4 0 -1 -1 10 12 11 0 0
5 0 -1 -1 4 13 14 0 0
1 0 -1 -1 1 14 13 0 0
```

Naturally, this is quite difficult to read and understand. Therefore, we wrote a piece of software, gds-to-paren, that takes in a graph described in gds representation and returns a representation of the same graph in parenthesized notation, which is clearly much easier to understand. The graph in Figure 1 described in parenthesized notation is as follows.

```
(graph
       (vertex
              (name "a")
              (id 1)
              (x 53.0)
              (y 91.0)
              (incident-edges 7 8 9))
       (vertex
              (name "c")
              (id 2)
              (x 183.0)
              (y 280.0)
              (incident-edges 8 10 11))
       (vertex
              (name "d")
              (id 3)
              (x 339.0)
              (y 272.0)
              (incident-edges 11 12))
       (vertex
              (name "e")
              (id 4)
              (x 343.0)
              (y 154.0)
              (incident-edges 12 13))
       (vertex
              (name "f")
              (id 5)
              (x 264.0)
              (y 46.0)
              (incident-edges 9 13))
       (vertex
              (name "b")
              (id 6)
              (x 43.0)
              (y 400.0)
              (incident-edges 7 10))
       (edge
            (type undirected)
            (id 7)
            (source 1 destination 6))
       (edge
            (type undirected)
            (id 8)
```

```
        (source 1 destination 2))
    (edge
        (type undirected)
        (id 9)
        (source 1 destination 5))
    (edge
        (type undirected)
        (id 10)
        (source 2 destination 6))
    (edge
        (type undirected)
        (id 11)
        (source 2 destination 3))
    (edge
        (type undirected)
        (id 12)
        (source 3 destination 4))
    (edge
        (type undirected)
        (id 13)
        (source 4 destination 5))
)
```

There are several different types of nodes that can be described within the gds structure. The simple node corresponds with our regular definition of a vertex. However cross, decomposition, and expanded nodes exist. If two edges happen to cross, then giotto creates an extra node at the crossing point, called a cross node. This node splits up the edges. Decomposition nodes are created in giotto in order to make all faces rectangular. Since all faces may not be rectangular, giotto places decomposition nodes on edges and creates rectangular faces. These nodes are basically fictitious and do not describe the actual graph, so gds-to-paren ignores these nodes. Expanded nodes on the other hand are very important for giotto. If a vertex is expanded that means that there is more than one node corresponding to the vertex. Basically, a box or skeleton is formed so that more than four edges can be incident to the vertex. Gds-to-paren does handle expanded nodes by describing the skeleton in each vertex definition and the source and destination points for each edge incident to an expanded vertex.

/pro/giotto/paren-notat/documentation/gds-to-paren.tex has more information on gds-to-paren.

3 Paren-to-Gds

Now that we have decided on describing graphs in parenthesized notation, we have a way for the user to specify his constraints. However, first we need to have a way for the original giotto tool to understand the input graphs. Thus, we created paren-to-gds, which is a piece

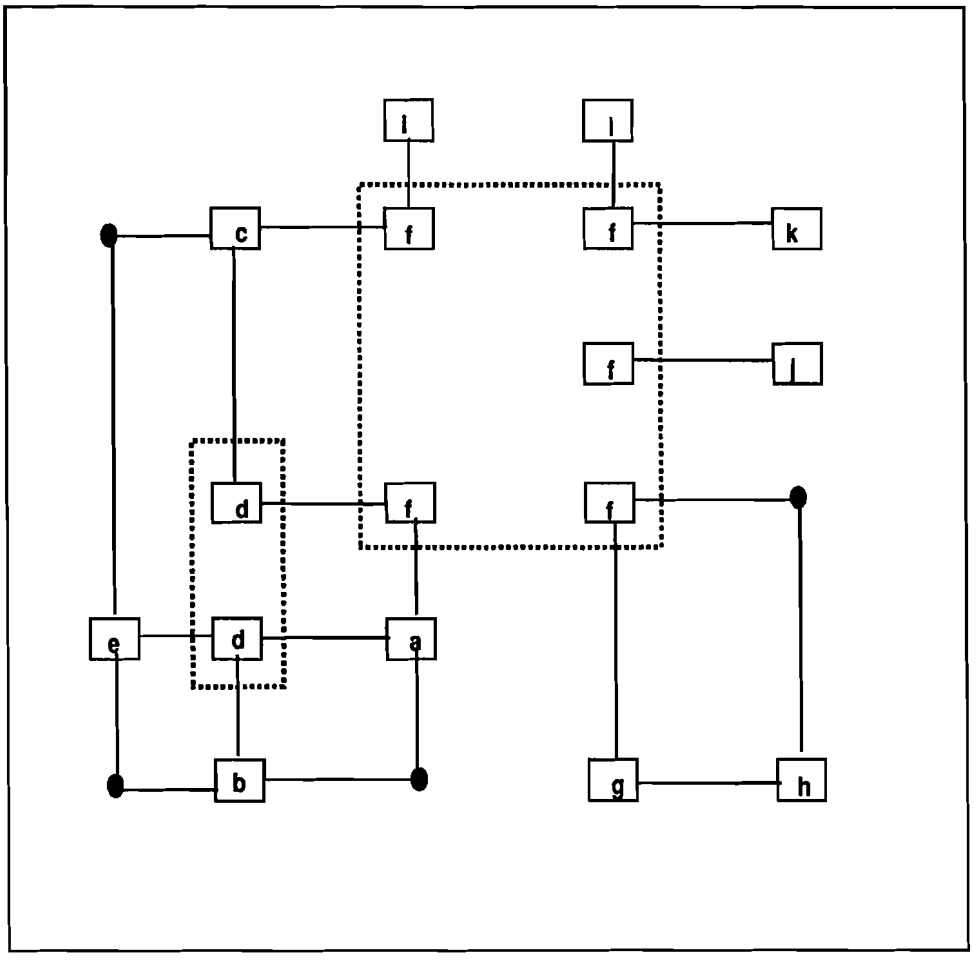

Figure 2:

of software that takes in a parenthesized representation of a graph, and returns the gds representation of the same graph. The following is the parenthesized representation of the graph in Figure 2.

```
(graph
        (vertex
                (name "d")
                (id 1)
                (x 0.0)
                (y -0.5)
                (skeleton (height 1) (width 0))
                (incident-edges 13 14 15 23 24))
        (vertex
               (name "a")
               (id 2)
               (x 1.0)
               (y -1.0)
               (incident-edges 15 16 17))
        (vertex
                (name "f")
                (id 3)
                (x 1.5)
                (y 1.0)
                (skeleton (height 2) (width 1))
                (incident-edges 13 17 18 19 20 26 27 28 29))
        (vertex
               (name "c")
               (id 4)
               (x 0.0)
               (y 2.0)
               (incident-edges 14 20 21))
        (vertex
               (name "e")
               (id 5)
               (x -1.0)
               (y -1.0)
               (incident-edges 21 22 23))
        (vertex
               (name "b")
               (id 6)
               (x 0.0)
               (y -2.0)
               (incident-edges 16 22 24))
        (vertex
               (name "h")
               (id 7)
               (x 3.0)
               (y -2.0)
               (incident-edges 19 25))
```

```
(vertex
       (name "g")
       (id 8)
       (x 2.0)
       (y -2.0)
       (incident-edges 18 25))
(vertex
       (name "i")
       (id 9)
       (x 1.0)
       (y 3.0)
       (incident-edges 26))
(vertex
       (name "j")
       (id 10)
       (x 3.0)
       (y 1.0)
       (incident-edges 27))
(vertex
       (name "k")
       (id 11)
       (x 3.0)
       (y 2.0)
       (incident-edges 28))
(vertex
       (name "l")
       (id 12)
       (x 2.0)
       (y 3.0)
       (incident-edges 29))
(edge
     (type undirected)
     (id 13)
     (source 1 destination 3)
     (source-pt (x 0.0) (y 0.5))
     (destination-pt (x -0.5) (y -1.0)))
(edge
     (type undirected)
     (id 14)
     (source 1 destination 4)
     (source-pt (x 0.0) (y 0.5))
     (destination-pt (x 0.0) (y 2.0)))
(edge
     (type undirected)
     (id 15)
```

```
      (source 2 destination 1)
      (source-pt (x 1.0) (y -1.0))
      (destination-pt (x 0.0) (y -0.5)))
(edge
      (type undirected)
      (id 16)
      (source 2 destination 6)
      (shape
            North 1
             West 1)
      (bends
           ((x 1.0) (y -2.0))))
(edge
      (type undirected)
      (id 17)
      (source 2 destination 3)
      (source-pt (x 1.0) (y -1.0))
      (destination-pt (x -0.5) (y -1.0)))
(edge
      (type undirected)
      (id 18)
      (source 3 destination 8)
      (source-pt (x 0.5) (y -1.0))
      (destination-pt (x 2.0) (y -2.0)))
(edge
      (type undirected)
      (id 19)
      (source 3 destination 7)
      (source-pt (x 0.5) (y -1.0))
      (destination-pt (x 3.0) (y -2.0))
      (shape
            East 1
           North 2)
      (bends
           ((x 3.0) (y 0.0))))
(edge
      (type undirected)

      (source 4 destination 3)
      (source-pt (x 0.0) (y 2.0))
      (destination-pt (x -0.5) (y 1.0)))
(edge
      (type undirected)
      (id 21)
      (source 4 destination 5)
```

```
        (shape
              West 1
             North 3)
        (bends
            ((x -1.0) (y 2.0))))
    (edge
        (type undirected)
        (id 22)
        (source 5 destination 6)
        (shape
             North 1
              East 1)
        (bends
            ((x -1.0) (y -2.0))))
    (edge
        (type undirected)
        (id 23)
        (source 5 destination 1)
        (source-pt (x -1.0) (y -1.0))
        (destination-pt (x 0.0) (y -0.5)))
    (edge
        (type undirected)
        (id 24)
        (source 6 destination 1)
        (source-pt (x 0.0) (y -2.0))
        (destination-pt (x 0.0) (y -0.5)))
    (edge
        (type undirected)
        (id 25)
        (source 7 destination 8))
    (edge
        (type undirected)
        (id 26)
        (source 9 destination 3)
        (source-pt (x 1.0) (y 3.0))
        (destination-pt (x -0.5) (y 1.0)))
    (edge
        (type undirected)
        (id 27)
        (source 10 destination 3)
        (source-pt (x 3.0) (y 1.0))
        (destination-pt (x 0.5) (y 0.0)))
    (edge
        (type undirected)
        (id 28)
```

```
        (source 11 destination 3)
        (source-pt (x 3.0) (y 2.0))
        (destination-pt (x 0.5) (y 1.0)))
    (edge
        (type undirected)
        (id 29)
        (source 12 destination 3)
        (source-pt (x 2.0) (y 3.0))
        (destination-pt (x 0.5) (y 1.0)))
)
```

Now, paren-to-gds, returns the graph in gds representation as follows.

```
      0
     12      21       8       4      54



      1       1      13       0       1       0       0       1
d
      1       0       0       0       2       0       0       0
a
      1      15      14       0       3       0       0       1
f
      1       0       0       0       4       0       0       0
c
      1       0       0       0       5       0       0       0
e
      1       0       0       0       6       0       0       0
b
      1       0       0       0       7       0       0       0
h
      1       0       0       0       8       0       0       0
g
      1       0       0       0       9       0       0       0
i
      1       0       0       0      10       0       0       0
j
      1       0       0       0      11       0       0       0
k
      1       0       0       0      12       0       0       0
l



      3       1       0       0       1
      1       2       1      -1       4
      3       3       1       0       7
      1       4       0       2      11
      1       5      -1      -1      14
      1       6       0      -2      17
      1       7       3      -2      20
      1       8       2      -2      22
      1       9       1       3      24
      1      10       3       1      25
      1      11       3       2      26
      1      12       2       3      27
      3       1       0      -1      28
```

3	3	2	0	32
3	3	1	2	36
3	3	2	1	40
3	3	2	2	43
4	0	1	-2	47
4	0	3	0	49
4	0	-1	2	51
4	0	-1	-2	53

1	0	1
1	0	2
1	0	3
1	0	4
1	0	5
2	3	7
1	0	15
1	0	21

3	2	1	1	2	10	9	0	0
13	3	1	0	3	28	31	1	0
4	1	2	2	1	13	12	0	0
18	5	1	0	5	47	48	0	0
13	2	1	3	6	29	28	0	0
3	4	1	2	4	9	8	0	0
14	4	1	1	8	35	34	1	0
2	2	1	0	9	4	6	0	0
1	1	1	3	10	2	1	0	0
15	6	2	2	7	38	37	1	0
15	1	1	1	12	39	38	0	0
1	3	2	0	13	1	3	0	0
20	4	1	3	11	52	51	0	0
13	7	1	1	15	31	30	0	0
21	4	1	0	16	53	54	0	0
20	3	3	2	14	51	52	0	0
18	4	1	1	18	48	47	0	0
21	7	1	3	19	54	53	0	0
13	5	1	2	17	30	29	0	0
8	8	1	3	21	23	22	0	0
19	4	2	2	20	50	49	0	0
7	4	1	1	23	21	20	0	0
14	8	2	2	22	34	33	0	0

15	0	1	0	24	36	39	0	0
16	0	1	3	25	41	40	0	0
17	0	1	3	26	44	43	0	0
17	0	1	0	27	43	46	0	0
2	5	1	1	29	6	5	0	0
6	7	1	0	30	17	19	0	0
5	3	1	3	31	15	14	0	0
1	2	1	2	28	3	2	1	0
19	8	1	1	33	49	50	0	0
8	4	2	0	34	22	23	0	0
3	6	1	3	35	8	7	1	0
16	4	1	2	32	42	41	1	0
17	6	1	1	37	46	45	1	0
3	1	2	0	38	7	10	1	0
4	4	1	3	39	12	11	0	0
9	0	1	2	36	24	24	0	0
10	0	1	1	41	25	25	0	0
14	6	1	0	42	32	35	1	0
17	4	1	2	40	45	44	1	0
11	0	1	1	44	26	26	0	0
16	6	1	0	45	40	42	1	0
15	4	1	3	46	37	36	1	0
12	0	1	2	43	27	27	0	0
6	5	1	3	48	18	17	0	0
2	4	1	2	47	5	4	0	0
7	8	2	0	50	20	21	0	0
14	4	1	3	49	33	32	0	0
4	3	1	1	52	11	13	0	0
5	4	3	0	51	14	16	0	0
6	4	1	1	54	19	18	0	0
5	7	1	2	53	16	15	0	0

For more information on paren-to-gds or the parenthisized notation, one can look at /pro/giotto/paren-notat/documentation/paren-to-gds.tex or /pro/giotto/const-src/Doc/paren-notat.tex.

3.1 Additional Keywords for Constraints

Now that we have a pretty simple way of describing graphs, we need to add a few new keywords to the parenthesized notation describing the user's constraints.

- In constraining the maximum number of bends for an edge, the user should use the keyword "max-bends" within the description of the desired edge. An example would then look as (max-bends 10).
- In constraining the shape of an edge, the user should use the keyword, "fixed-shape" within the description of the desired edge, followed by a number of L's and R's where L stands for a left turn and R stands for a right turn. An example is (fixed-shape L R L L R L R).
- In constraining the angle formed by two edges incident to a vertex, the user simply needs to modify the section containing incident edges in the vertex description. The user should put the angle desired between the edges in parentheses between the two edges. Notice, for this to work the edges listed under incident-edges have to be in clockwise order. The user is responsible for this if he wants to constrain an angle. So, an example would be (incident-edges 7 (90) 8 9 (180)). The user is constraining the angle between edge 7 and 8 to be 90 and the angle between edge 9 and 7 to be 180. The total sum of the angles can not exceed 360.

4 Description of the Constraints

4.1 Maximum number of bends for an edge

In this subsection, we will show different examples of how the user can constrain the maximum number of bends for an edge. Now, consider the graph in Figure 3, with the following parenthesized representation.

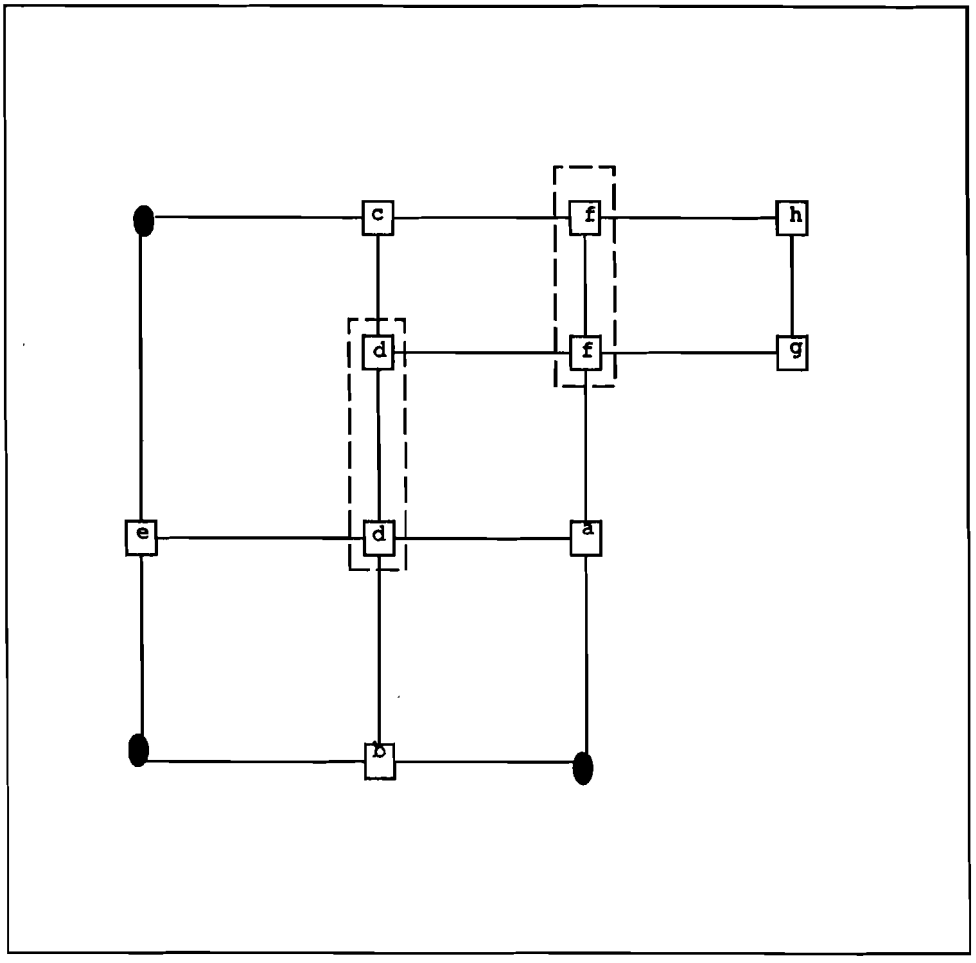

Figure 3:

```
(graph
        (vertex
                (name "d")
                (id 1)
                (x 0.0)
                (y -0.5)
                (skeleton (height 1) (width 0))
                (incident-edges 9 10 11 18 19))
        (vertex
               (name "a")
               (id 2)
               (x 1.0)
               (y -1.0)
```

```
        (incident-edges 11 12 13))
(vertex
        (name "f")
        (id 3)
        (x 1.0)
        (y 0.5)
        (skeleton (height 1) (width 0))
        (incident-edges 9 13 14 15 21))
(vertex
       (name "c")
       (id 4)
       (x 0.0)
       (y 1.0)
       (incident-edges 10 15 16))
(vertex
       (name "e")
       (id 5)
       (x -1.0)
       (y -1.0)
       (incident-edges 16 17 18))
(vertex
       (name "b")
       (id 6)
       (x 0.0)
       (y -2.0)
       (incident-edges 12 17 19))
(vertex
       (name "h")
       (id 7)
       (x 2.0)
       (y 1.0)
       (incident-edges 14 20))
(vertex
       (name "g")
       (id 8)
       (x 2.0)
       (y 0.0)
       (incident-edges 20 21))
(edge
     (type undirected)
     (id 9)
     (source 1 destination 3)
     (source-pt (x 0.0) (y 0.5))
     (destination-pt (x 0.0) (y -0.5)))
(edge
```

```
     (type undirected)
     (id 10)
     (source 1 destination 4)
     (source-pt (x 0.0) (y 0.5))
     (destination-pt (x 0.0) (y 1.0)))


(edge
     (type undirected)
     (id 11)
     (source 2 destination 1)
     (source-pt (x 1.0) (y -1.0))
     (destination-pt (x 0.0) (y -0.5)))
(edge
     (type undirected)
     (id 12)
     (source 2 destination 6)
     (shape
          North 1
           West 1)
     (bends
          ((x 1.0) (y -2.0))))
(edge
     (type undirected)
     (id 13)
     (source 2 destination 3)
     (source-pt (x 1.0) (y -1.0))
     (destination-pt (x 0.0) (y -0.5)))
(edge
     (type undirected)
     (id 14)
     (source 3 destination 7)
     (source-pt (x 0.0) (y 0.5))
     (destination-pt (x 2.0) (y 1.0)))
(edge
     (type undirected)
     (id 15)
     (source 3 destination 4)
     (source-pt (x 0.0) (y 0.5))
     (destination-pt (x 0.0) (y 1.0)))
(edge
     (type undirected)
     (id 16)
     (source 4 destination 5)
     (shape
```

```
                West 1
               North 2)
          (bends
               ((x -1.0) (y 1.0))))
     (edge
          (type undirected)
          (id 17)
          (source 5 destination 6)
          (shape
                North 1
                 East 1)
          (bends
               ((x -1.0) (y -2.0))))
     (edge
          (type undirected)
          (id 18)
          (source 5 destination 1)
          (source-pt (x -1.0) (y -1.0))
          (destination-pt (x 0.0) (y -0.5)))
     (edge
          (type undirected)
          (id 19)
          (source 6 destination 1)
          (source-pt (x 0.0) (y -2.0))
          (destination-pt (x 0.0) (y -0.5)))
     (edge
          (type undirected)
          (id 20)
          (source 7 destination 8))
     (edge
          (type undirected)
          (id 21)
          (source 8 destination 3)
          (source-pt (x 2.0) (y 0.0))
          (destination-pt (x 0.0) (y -0.5)))
)
```

Now, let's constrain an edge. Pick, say, edge 18. Let us change it, to be as follows:

```
(edge
    (type undirected)
    (id 18)
    (source 5 destination 1)
    (source-pt (x -1.0) (y -1.0))
    (destination-pt (x 0.0) (y -0.5))
```

```
(max-bends 2))
```

The graph after Figure 3 is run through giotto becomes the graph in Figure 4. Notice that the edge between nodes e and d (edge 18) has no more than two bends.

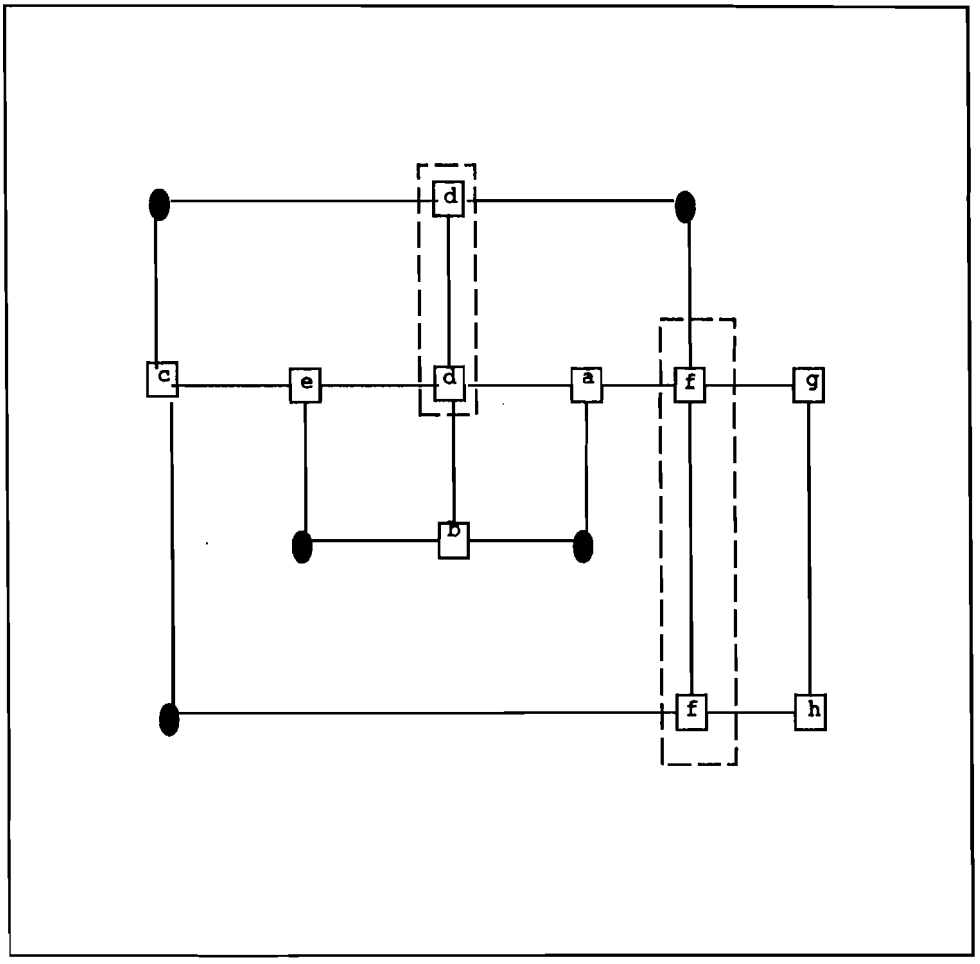

Figure 4:

Keeping the above constraint, we notice that the edge between f and c has one bend. Let us change that edge, edge 15 to be as follows:

```
(edge
  (type undirected)
  (id 15)
  (source 3 destination 4)
  (source-pt (x 0.0) (y 0.5))
  (destination-pt (x 0.0) (y 1.0))
  (max-bends 0))
```

Now after running the new graph through giotto, we get the graph in Figure 5. Edge 15, the edge between f and c, no longer has a bend.

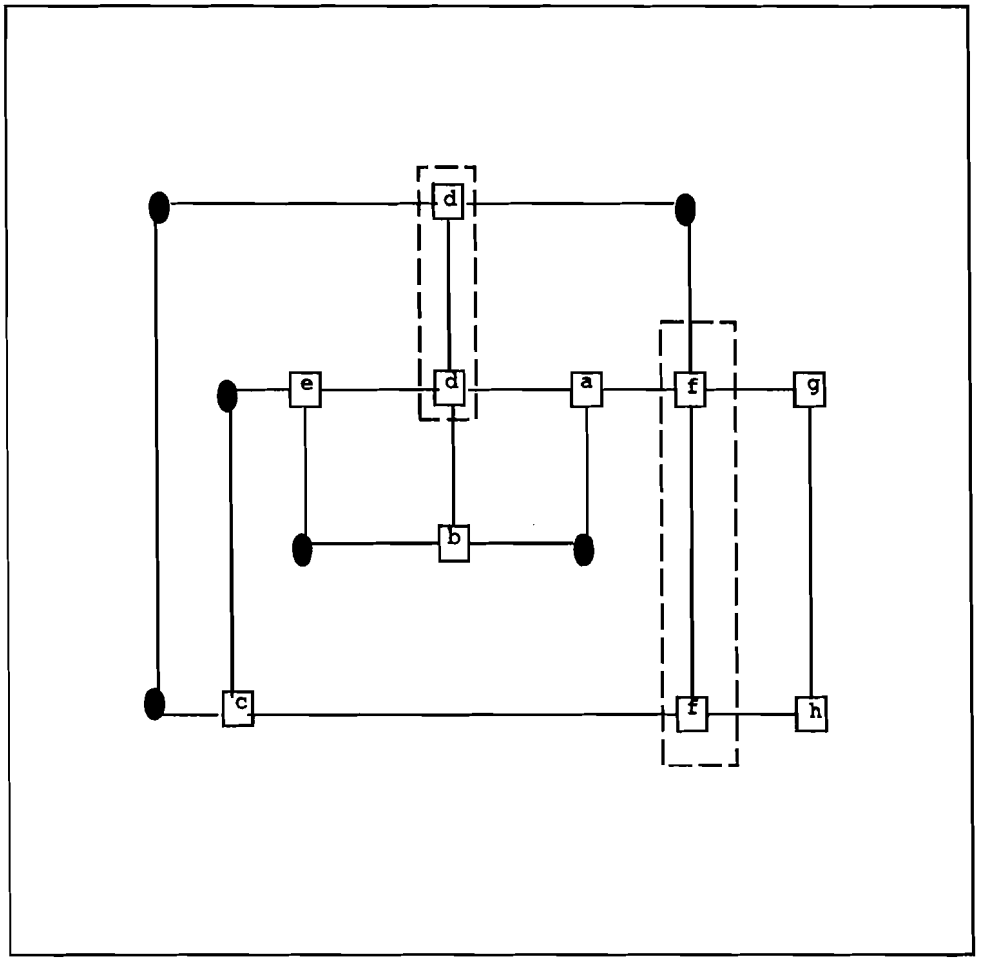

Figure 5:

Finally, let's alter an edge that already has a shape. In this case, giotto ignores the shape and finds its own layout according to the constraint. Let's change edge 12 to be as follows:

```
(edge
  (type undirected)
  (id 12)
  (source 2 destination 6)
  (shape
        North 1
        West 1)
  (bends ((x 1.0) (y -2.0)))
  (max-bends 0))
```

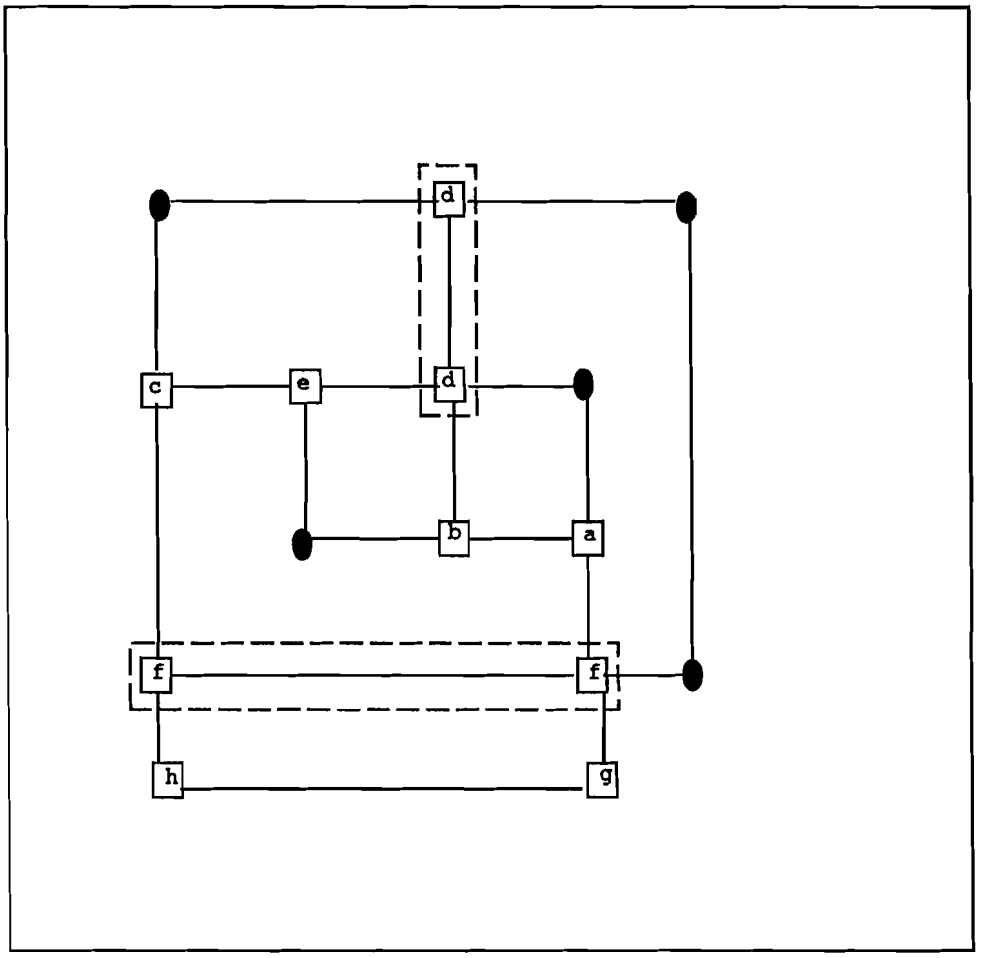

Figure 6:

We now get the graph in Figure 6. Notice that all the constraints are satisfied, especially that the edge between a and b has no bends.

4.2 Fixing the shape of an edge

In this subsection, we will show different examples of how the user can constrain an edge to have a fixed shape. Consider again the graph in figure 1. Now, let us change the description of edge 12, the edge between d and e to be as follows:

```
(edge
  (type undirected)
  (id 12)
  (fixed-shape L L R R L R)
  (source 3 destination 4))
```

We want the edge between d and e to make 2 left turn, 2 right turns, then another left turn, followed by another right turn in that order.

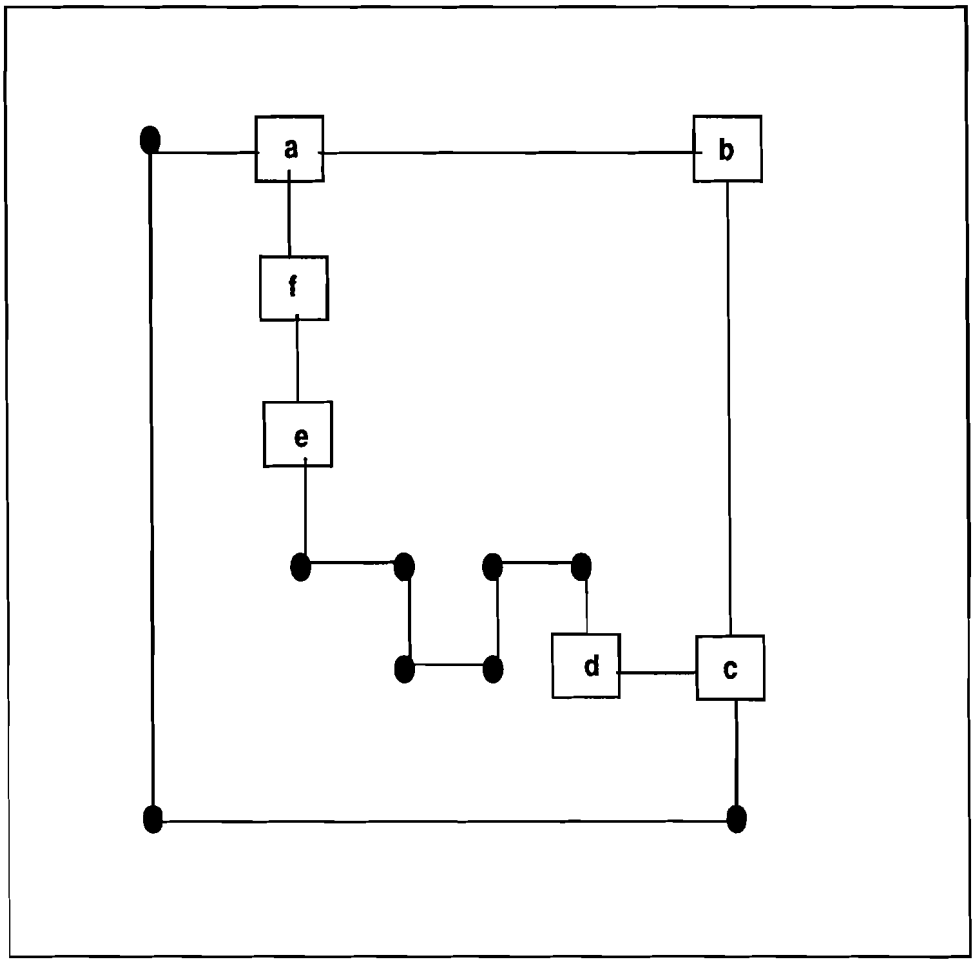

Figure 7:

We now get the graph in Figure 7.

Now, let's consider the graph in Figure 8 with the following parenthesized notation. We want to add a fixed-shape to an edge with a shape already. Giotto ignores the shape and adheres to the fixed shape.

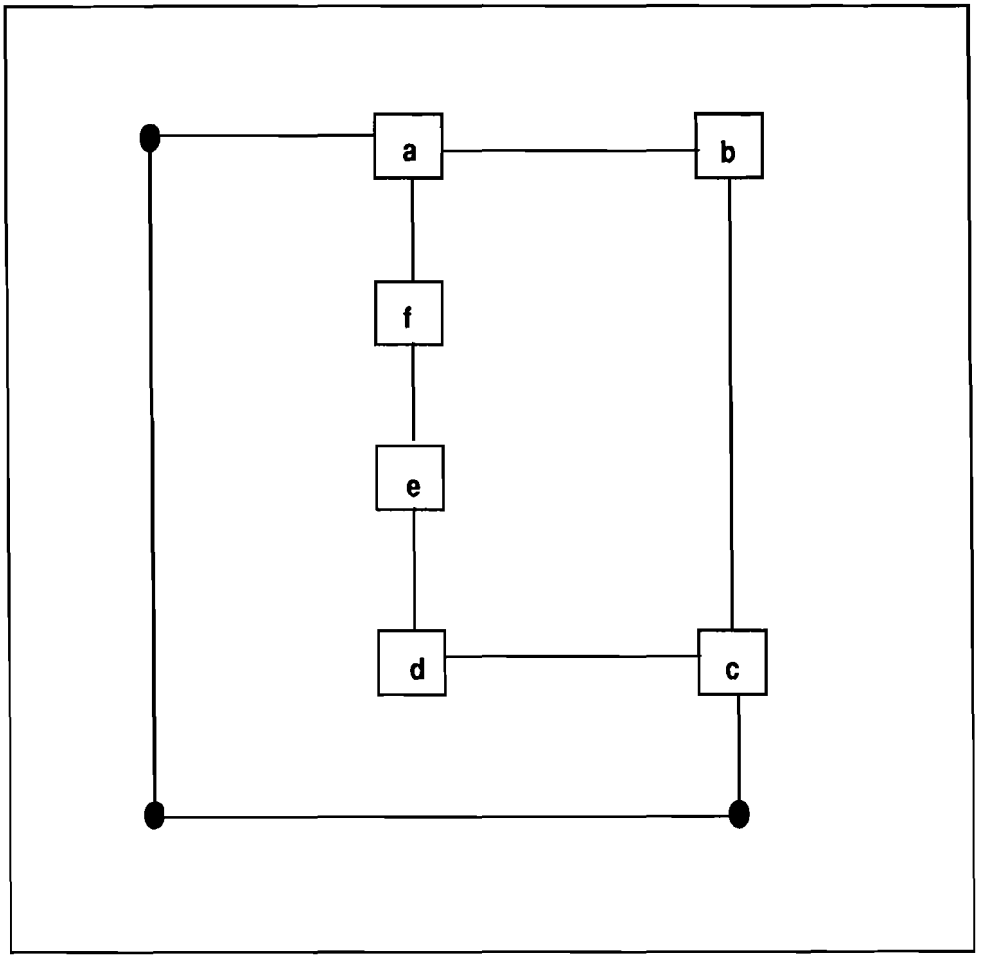

Figure 8:

```
(graph
        (vertex
                (name "a")
                (id 1)
                (x 0.0)
                (y 0.0)
                (incident-edges 7 8 9))
        (vertex
                (name "c")
                (id 2)
                (x 1.0)
                (y -3.0)
                (incident-edges 9 10 11))
        (vertex
```

```
        (name "d")
        (id 3)
        (x 0.0)
        (y -3.0)
        (incident-edges 11 12))
(vertex
        (name "e")
        (id 4)
        (x 0.0)
        (y -2.0)
        (incident-edges 12 13))
(vertex
        (name "f")
        (id 5)
        (x 0.0)
        (y -1.0)
        (incident-edges 7 13))
(vertex
        (name "b")
        (id 6)
        (x 1.0)
        (y 0.0)
        (incident-edges 8 10))
(edge
     (type undirected)
     (id 7)
     (source 1 destination 5))
(edge
     (type undirected)
     (id 8)
     (source 1 destination 6))
(edge
     (type undirected)
     (id 9)
     (source 1 destination 2)
     (shape
            West 1
           North 4
            East 2
           South 1)
     (bends
          ((x -1.0) (y 0.0))
          ((x -1.0) (y -4.0))
          ((x 1.0) (y -4.0))))
(edge
```

```
        (type undirected)
        (id 10)
        (source 2 destination 6))
    (edge
        (type undirected)
        (id 11)
        (source 2 destination 3))
    (edge
        (type undirected)
        (id 12)
        (source 3 destination 4))
    (edge
        (type undirected)
        (id 13)
        (source 4 destination 5))
)
```

We now change edge 9, the edge between a and c, to be as follows:

```
(edge
    (type undirected)
    (id 9)
    (source 1 destination 2)
    (shape
        West 1
        North 4
        East 2
        South 1)
    (bends
        ((x -1.0) (y 0.0))
        ((x -1.0) (y -4.0))
        ((x 1.0) (y -4.0)))
    (fixed-shape R L L))
```

We now get the diagram in Figure 9. Notice that the original shape and bends were ignored. A final point to remember is that if the user constrains the maximum number of bends as well as fixes the shape of an edge and the fixed-shape has more bends than the maximum number of bends, then giotto returns the graph according to the fixed-shape, but returns an error message warning the user.

4.3 Constraining an angle between two edges incident to a vertex

In this subsection we give various examples of constraining angles. Remember that the user is responsible for making sure that the edges are listed in clockwise order if he wants to constrain an angle.

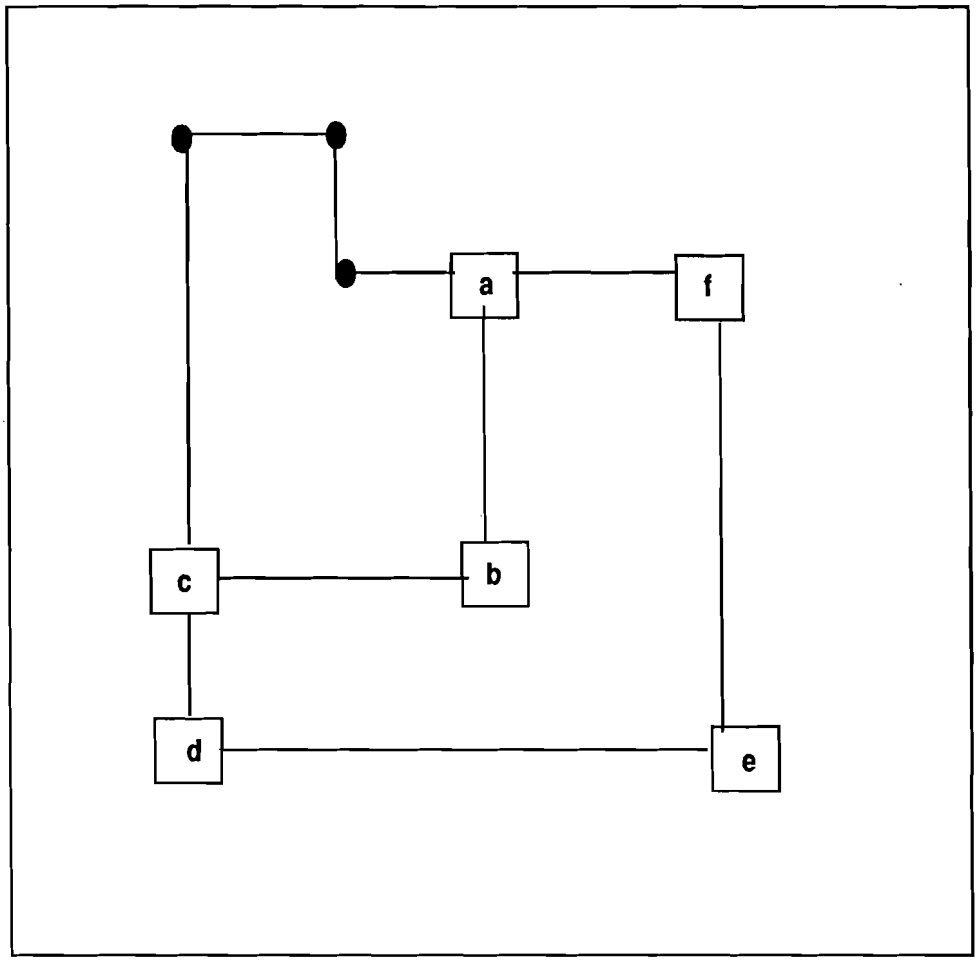

Figure 9:

Consider the graph in Figure 1 again. If we run giotto on the graph as it is, we will get the graph represented in Figure 8. Let us change the angle between the edge from a to c and the edge from a to b to be 90 now instead of 180 as it is. For this, we change the parenthesized notation for vertex a as follows:

```
(vertex
        (name ''a'')
        (id 1)
        (x 53.0)
        (y 91.0)
        (incident-edges 9 8 (90) 7))
```

We now get the graph in Figure 10. Notice the angle constraint is satisfied. The sum of the angles can not be more than 360. If that occurs, giotto returns and error.

We had to do a considerable amount of change in the procedure handling expansion of a node. That is if a node has more than four edges incident to it, the node has to be

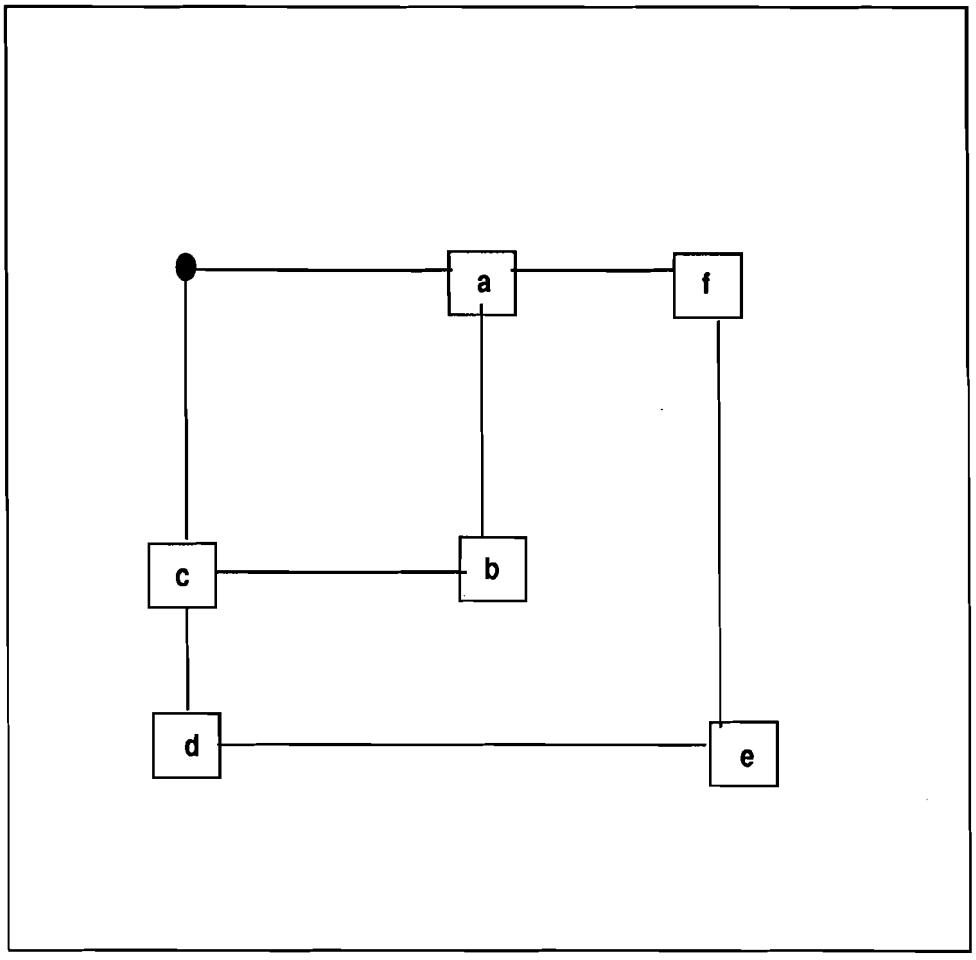

Figure 10:

expanded. Originally, the angles between incident edges were simply fixed. Now, since the user needs to be able to constrain the angles as he pleases, we needed to change this procedure. If you take the graph in Figure 11. Clearly, vertex a needs to be expanded since it has nine incident edges. Figure 11 has the following parenthesized notation.

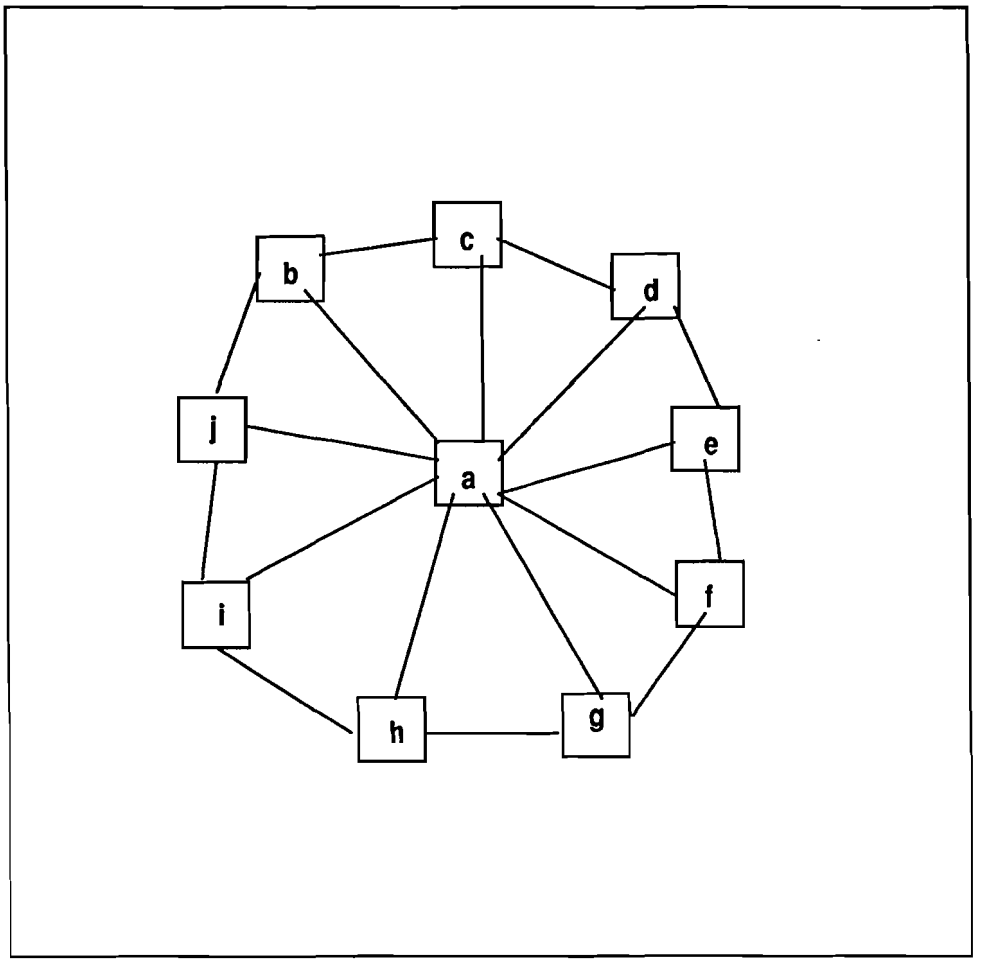

Figure 11:

```
(graph
        (vertex
                (name "a")
                (id 1)
                (x 250.0)
                (y 161.0)
                (incident-edges 11 12 13 14 15 16 17 18 19))
        (vertex
                (name "c")
                (id 2)
                (x 320.0)
                (y 89.0)
                (incident-edges 12 20 21))
        (vertex
```

```
        (name "b")
        (id 3)
        (x 222.0)
        (y 75.0)
        (incident-edges 11 20 22))
(vertex
        (name "j")
        (id 4)
        (x 175.0)
        (y 147.0)
        (incident-edges 19 22 23))
(vertex
        (name "h")
        (id 5)
        (x 159.0)
        (y 247.0)
        (incident-edges 17 24 25))
(vertex
        (name "g")
        (id 6)
        (x 255.0)
        (y 287.0)
        (incident-edges 16 24 26))
(vertex
        (name "f")
        (id 7)
        (x 337.0)
        (y 263.0)
        (incident-edges 15 26 27))
(vertex
        (name "e")
        (id 8)
        (x 373.0)
        (y 216.0)
        (incident-edges 14 27 28))
(vertex
        (name "d")
        (id 9)
        (x 403.0)
        (y 138.0)
        (incident-edges 13 21 28))
(vertex
        (name "i")
        (id 10)
        (x 168.0)
```

```
        (y 202.0)
        (incident-edges 18 23 25))
(edge
     (type undirected)
     (id 11)
     (source 1 destination 3))
(edge
     (type undirected)
     (id 12)
     (source 1 destination 2))
(edge
     (type undirected)
     (id 13)
     (source 1 destination 9))
(edge
     (type undirected)
     (id 14)
     (source 1 destination 8))
(edge
     (type undirected)
     (id 15)
     (source 1 destination 7))
(edge
     (type undirected)
     (id 16)
     (source 1 destination 6))
(edge
     (type undirected)
     (id 17)
     (source 1 destination 5))
(edge
     (type undirected)
     (id 18)
     (source 1 destination 10))
(edge
     (type undirected)
     (id 19)
     (source 1 destination 4))
(edge
     (type undirected)
     (id 20)
     (source 2 destination 3))
(edge
     (type undirected)
     (id 21)
```

```
     (source 2 destination 9))
(edge
     (type undirected)
     (id 22)
     (source 3 destination 4))
(edge
     (type undirected)
     (id 23)
     (source 4 destination 10))
(edge
     (type undirected)
     (id 24)
     (source 5 destination 6))
(edge
     (type undirected)
     (id 25)
     (source 5 destination 10))
(edge
     (type undirected)
     (id 26)
     (source 6 destination 7))
(edge
     (type undirected)
     (id 27)
     (source 7 destination 8))
(edge
     (type undirected)
     (id 28)
     (source 8 destination 9))
)
```

Running this graph through this version of giotto gives us the graph in Figure 12. Now we can constrain angles for expanded vertices. Let us change the description of vertex a to be the following.

```
(vertex
      (name ''a'')
      (id 1)
      (x 250.0)
      (y 161.0)
      (incident-edges 11 (180) 12 (90) 13 14 15 (90) 16 17 18 19))
```

In other words we want the edge between a and b, and the edge between a and c to have angle 180. Also the angle between the edge a to c and the edge a to d should be 90 degrees. Finally, we want to constrain the angle between the edge a to f and the edge a to g to be 90. This new graph is shown in Figure 13.

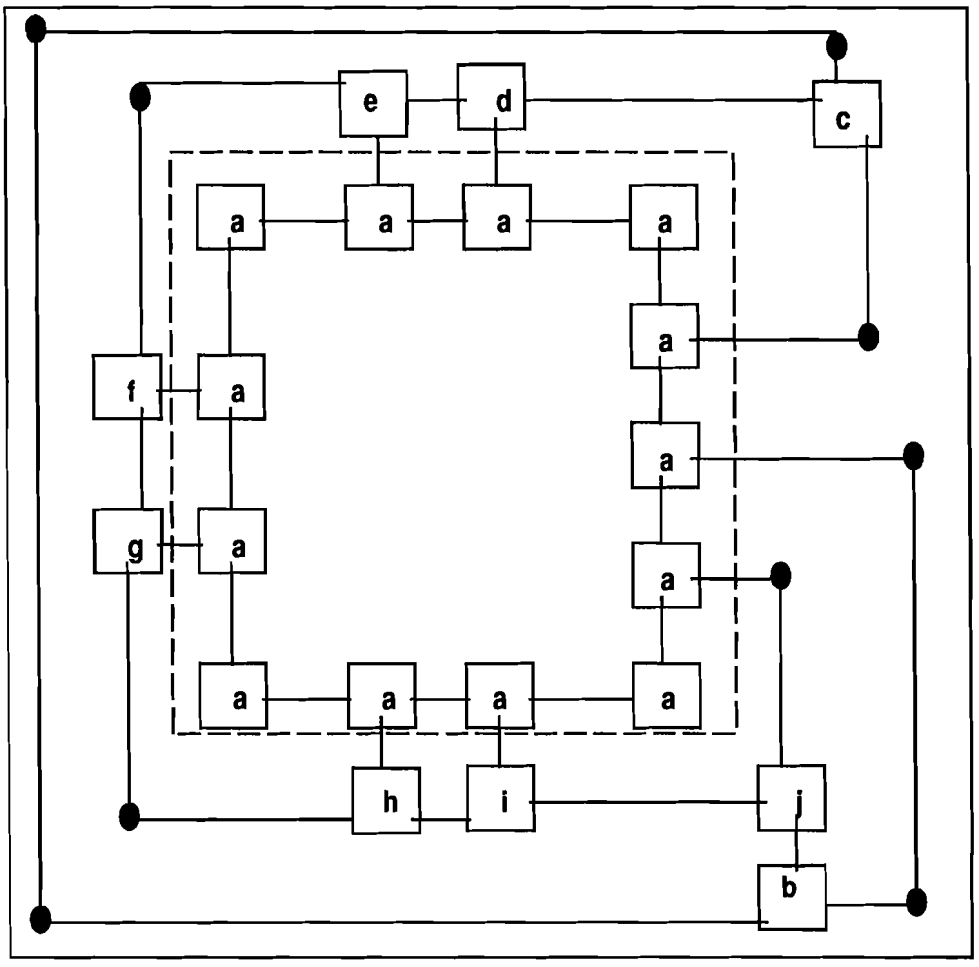

Figure 12:

We can also constrain angles to be zero. In other words, we can force two arcs to be parallel. If a vertex is already expanded, we simply need to move arcs if necessary. However, if a vertex has less than five edges incident to it, but has a zero constraint, we need to expand the vertex anyway.

For an expanded vertex, consider again the graph in Figure 11. Let us, however, change the description of vertex a to be as follows now.

```
(vertex
      (name ''a'')
      (id 1)
      (x 250.0)
      (y 161.0)
      (incident-edges 11 (180) 12 (0) 13 (0) 14 (0) 15 (90) 16 17 18 19))
```

In this case we want edges 12, 13, 14, and 15 to be parallel. That is the edges a to c, a to d, a to e, and a to f should be parallel to each other. We now get the graph in Figure

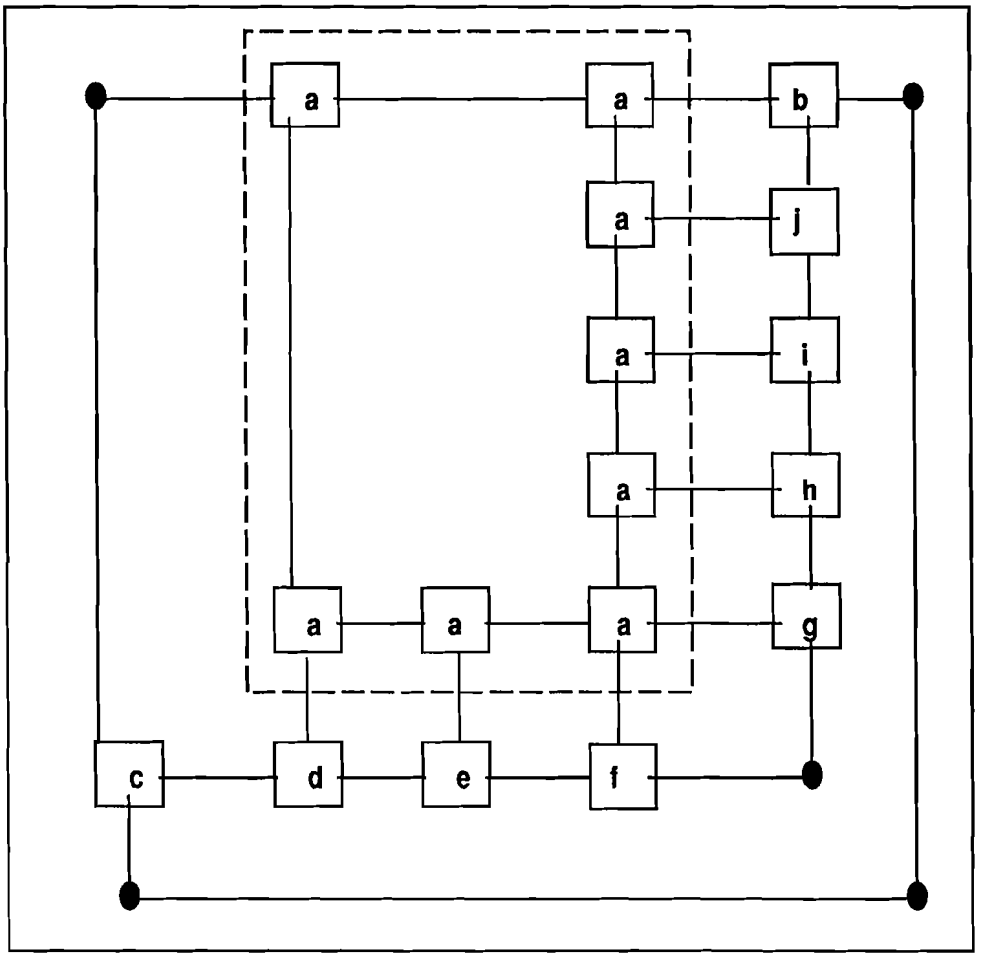

Figure 13:

14. Notice that all the zero angled constraints are satisfied along with the edge a to b and edge a to c having angle 180, and edge a to f and edge a to g having angle 90.

Now, if we want to have zero angle constraints for a vertex that need not be expanded, otherwise, we will need to expand the vertex. Consider again the graph in Figure 1. Let us change the description of vertex a to be as follows.

```
(vertex
      (name ''a'')
      (id 1)
      (x 53.0)
      (y 91.0)
      (incident-edges 7 8 (0) 9))
```

We, now, want the edge a to c and the edge a to f to be parallel. We then expand vertex a and get the graph in Figure 15.

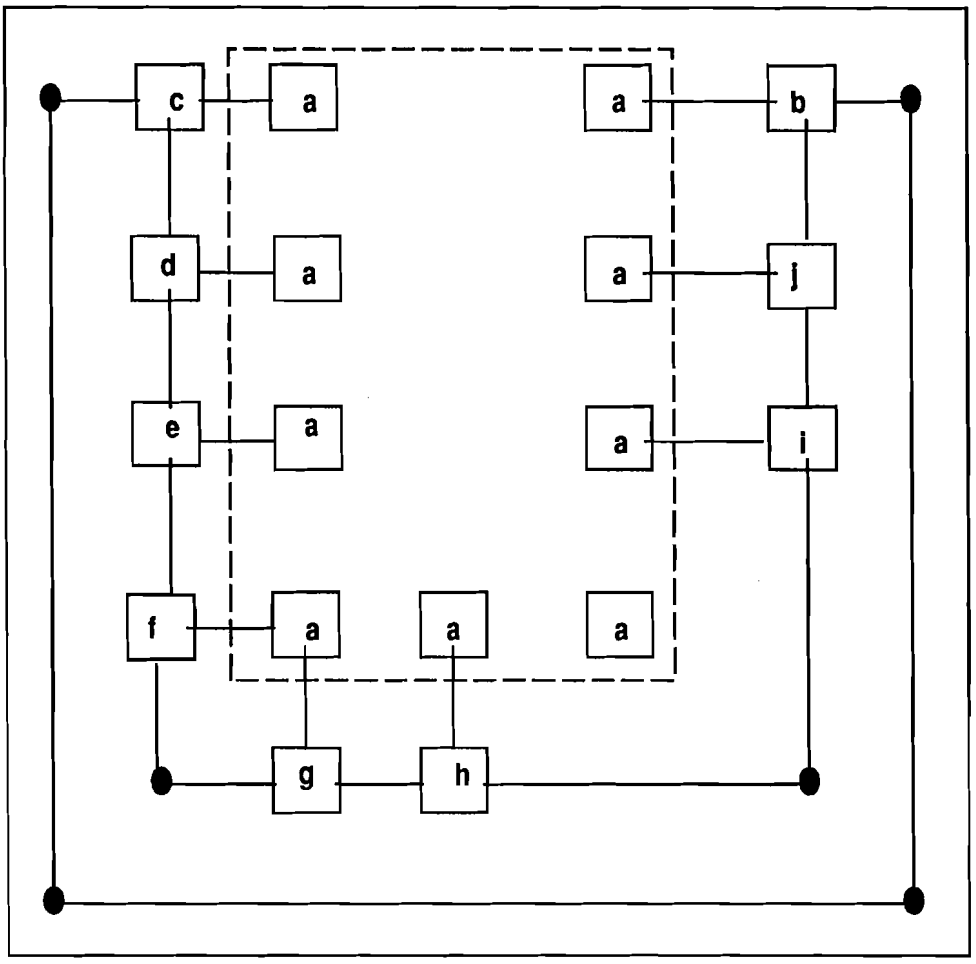

Figure 14:

5 Running Giotto

This constraint version of giotto is located in the directory, /pro/giotto/const-src. To run, simply type: giotto inputfile [outputfile]. The input file should be a graph description in parenthesized notation as shown above. The output file is clearly optional. Giotto returns the graph in gds representation abiding by the constraints enforced by the user.

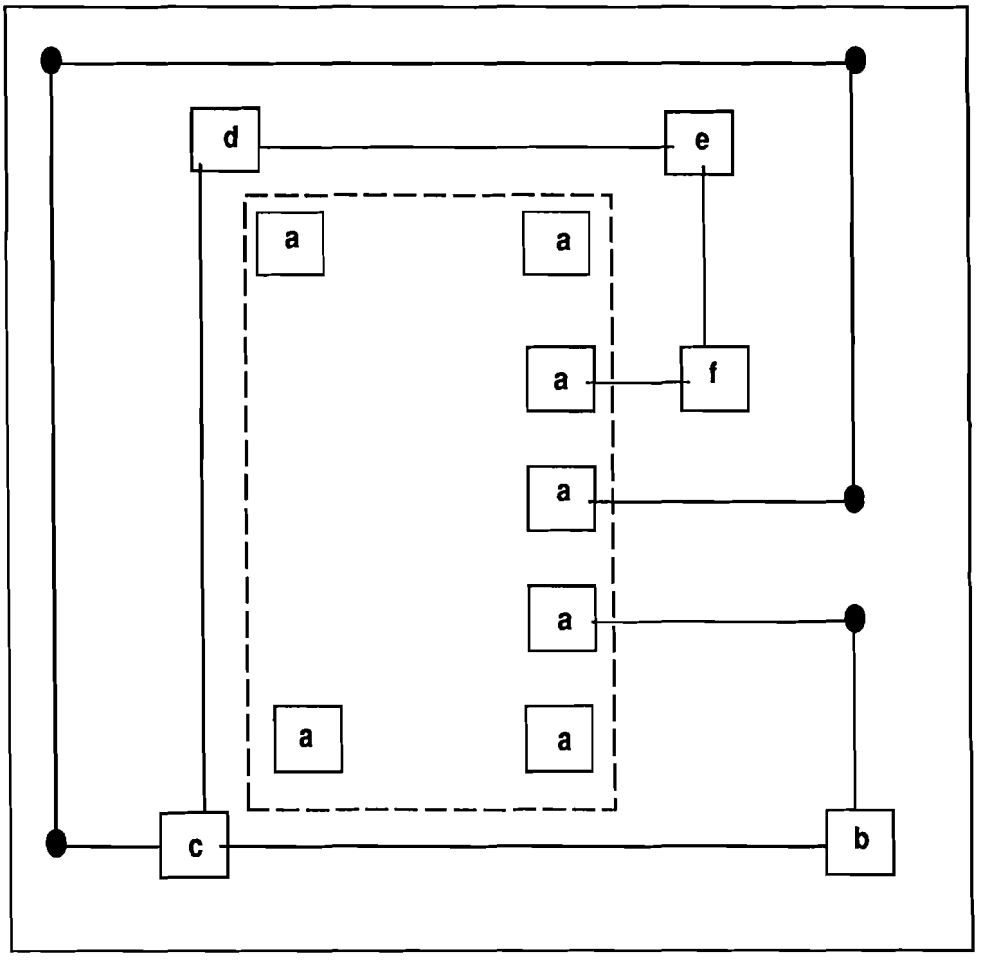

Figure 15: