METHOD FOR FINDING SHORTEST PATH IN PCBs LAYOUT VERIFICATION PROCESS

EXISTING PATENTS RELATED TO THIS INVENTION

Patent title: “Method for identifying short-circuit path in integrated circuit layout verification process”, Publication number: CN103186690 A, Publication date: 3 Jul 2013, Inventors: 王志明, 王国庆, 丁丰庆, 毛凌颖.

GOAL OF THE INVENTION

The invention discloses a method for finding the shortest path between any two points in layouts of Printed Circuit Boards (PCBs).

FIELD OF THE INVENTION

The invention belongs to the technical field of Electronics Design Automation (EDA) or Electronics Computer Aided Design (ECAD). Particularly, it relates to the field of PCB design and verification.

BACKGROUND OF THE INVENTION

EDA is a category of software tools for designing electronic systems such as Printed Circuit Boards (PCBs) and Integrated Circuits (ICs). These tools help designers to manage the complexity of the PCB designs and verify them. When designing PCBs, the distance between two elements should be greater than a threshold to avoid some side effects such as crosstalk and overheating. For measuring that distance, some methods such as finding the shortest path between components in a PCB design are employed. The invention presents a solution for finding the shortest path among components in a PCB layout. Besides, it can be a considered solution for the circuit routing problem.

Finding the shortest path between two elements in a PCB layout can be transformed into a 2D environment shortest path problem. However, because there might exist some holes inside the PCB layout, the problem turns into finding the shortest path in

a polygonal domain. There have many steps involved for finding a shortest path such as decomposing polygonal area, running Dijkstra algorithm, building dual graph…

Polygon triangulation is a common approach to decompose polygonal area (PCB layout) into a set of triangles. There are some algorithms such as ear clipping and monotone to triangulate a polygonal area. Our invention uses constrained Delaunay triangulation algorithm that significantly reduces pathfinding search effort as well as better representing the basic structure of the environment (PCB layout). The complexity of this algorithm is **O(NlogN)** in which **N** is the number of points constructing the PCB layout.

Dijkstra algorithm is an algorithm for finding the shortest path among nodes in a graph. In our invention, this algorithm is used to find a shortest sequence of triangles, called a shortest channel, in the graph of triangles built by the polygon triangulation algorithm stated above. The complexity of Dijkstra algorithm is **O(|E| + VLog|V|)** in which **V** is the number of triangles and **E** is the number of edges in the graph of triangles.

Once we have the shortest channel from the start point to the end point, we can apply funnel algorithm on this channel. The funnel algorithm considers three structures: the *path*, the *apex*, and the *funnel*. The *path* is the series of line segments forming the portion of the shortest path. The *funnel* consists of two series of line segments that represent the area in which all shortest paths to the area are not yet processed. Finally, the *apex* is the point that joins the *path* and the *funnel*. The first *apex* starts at the start point. FIG **6** shows these structures. This algorithm can be performed in linear time **O(T)** in which **T** is the number of triangles in the channel. This number is much smaller than the number of points for constructing the PCB layout.

EFFECTS OF THE INVENTION

The overall complexity of our method is in polynomial time **O(NlogN)**. Therefore, the present method quickly searches the shortest path between any two points in the PCBs layout. It adapts with the complexity of PCB layout so that the required storage space and the running efficiency is high compared to general search algorithms.

HOW GOOD FOR BUSINESS

Verification processes in EDA tools are important for avoiding defects and reducing costs in designs. However, PCBs are becoming increasingly large and complex, typically including millions of individual elements such as ICs, transistors, and logic gates. Therefore, finding a way to complete the verification of the design within the effective time has become the major issue that need to be solved. The present invention can be employed in many kinds of verification processes to reduce the running time for verification while ensuring reliability. In addition, a modified version of our invention can be applied for solving the circuit routing problem.

DRAWINGS OF THE INVENTION

FIG. 1

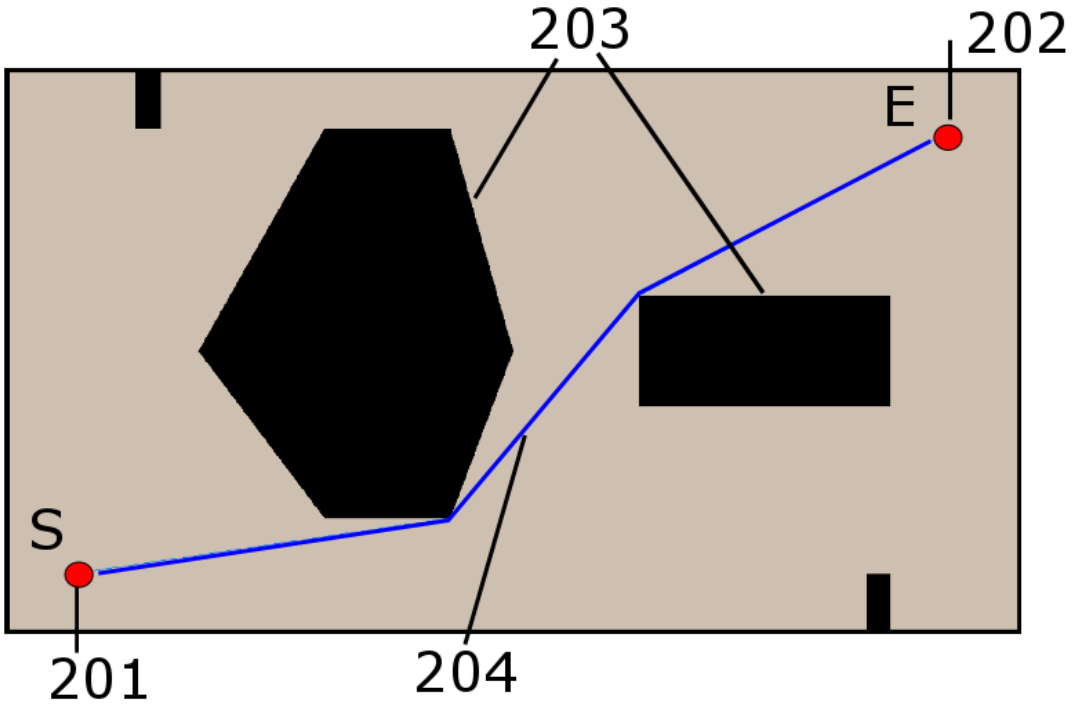

FIG. 2

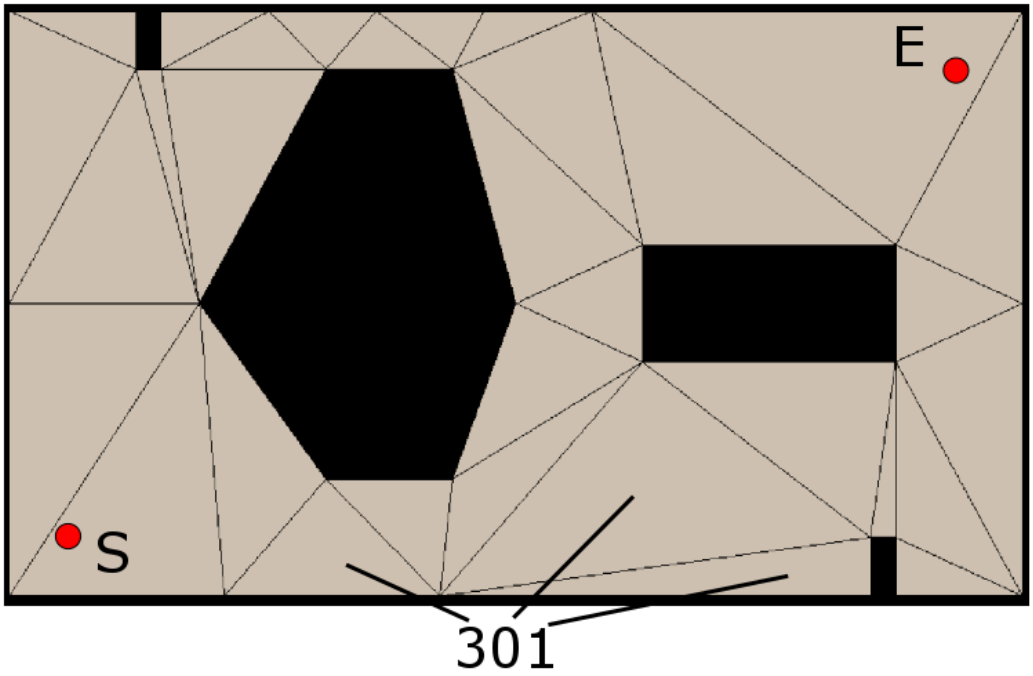

FIG. 3

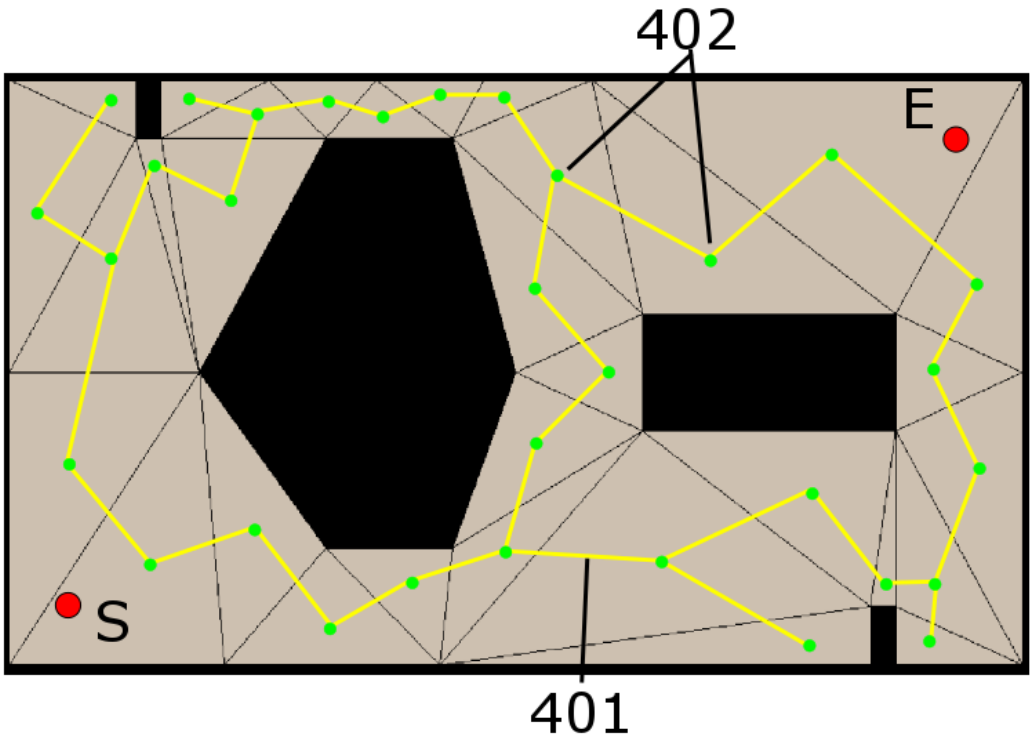

FIG. 4

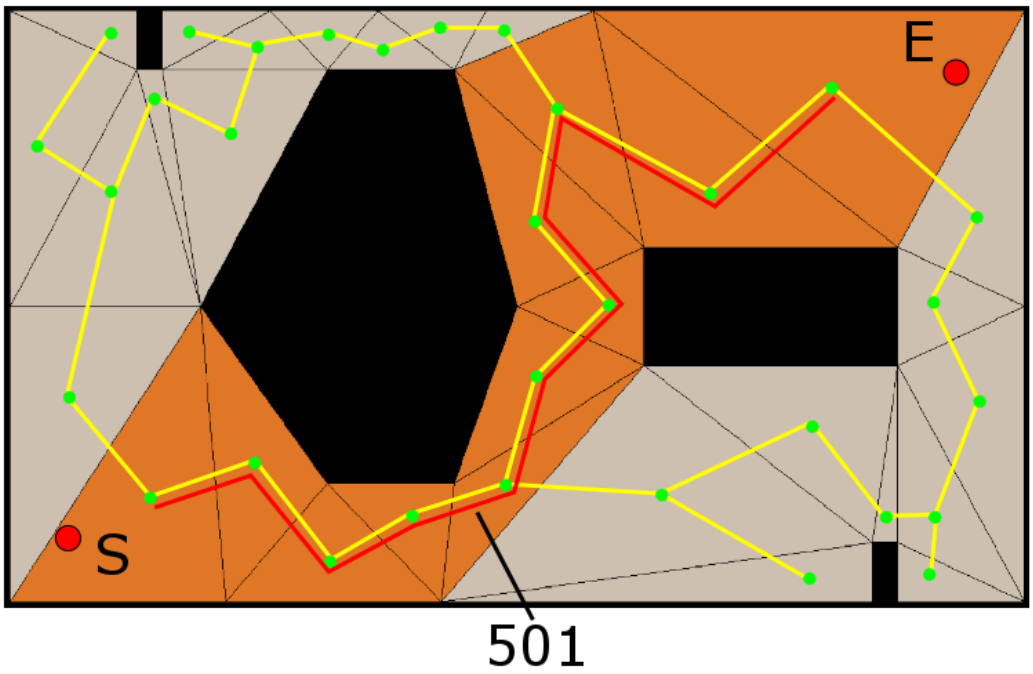

FIG. 5

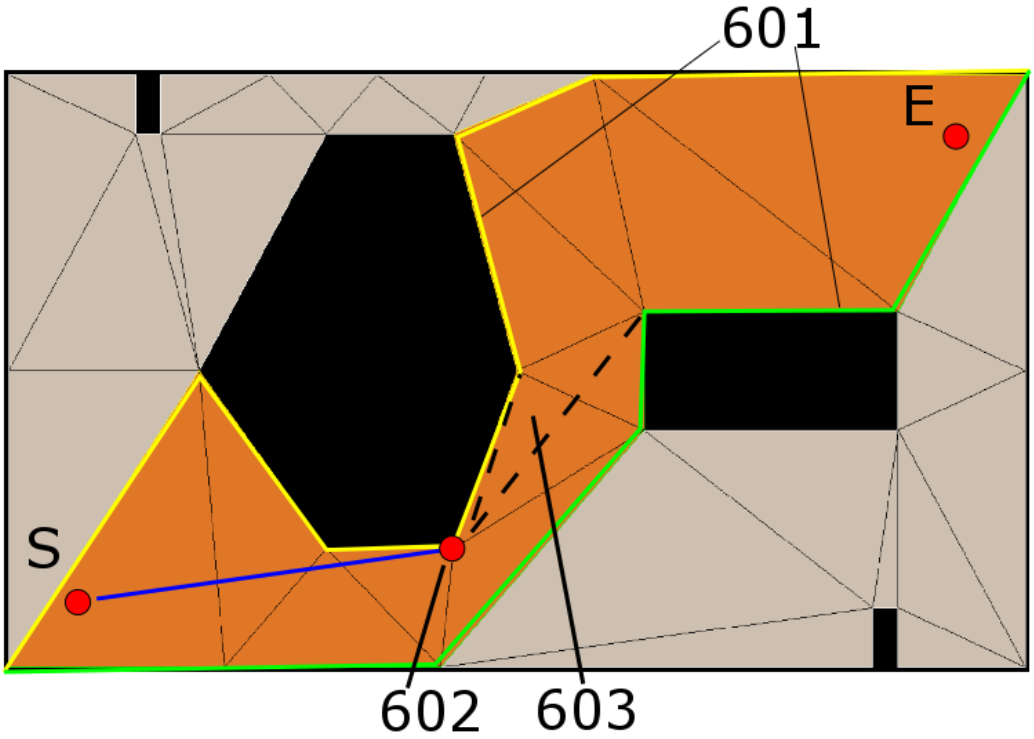

FIG. 6

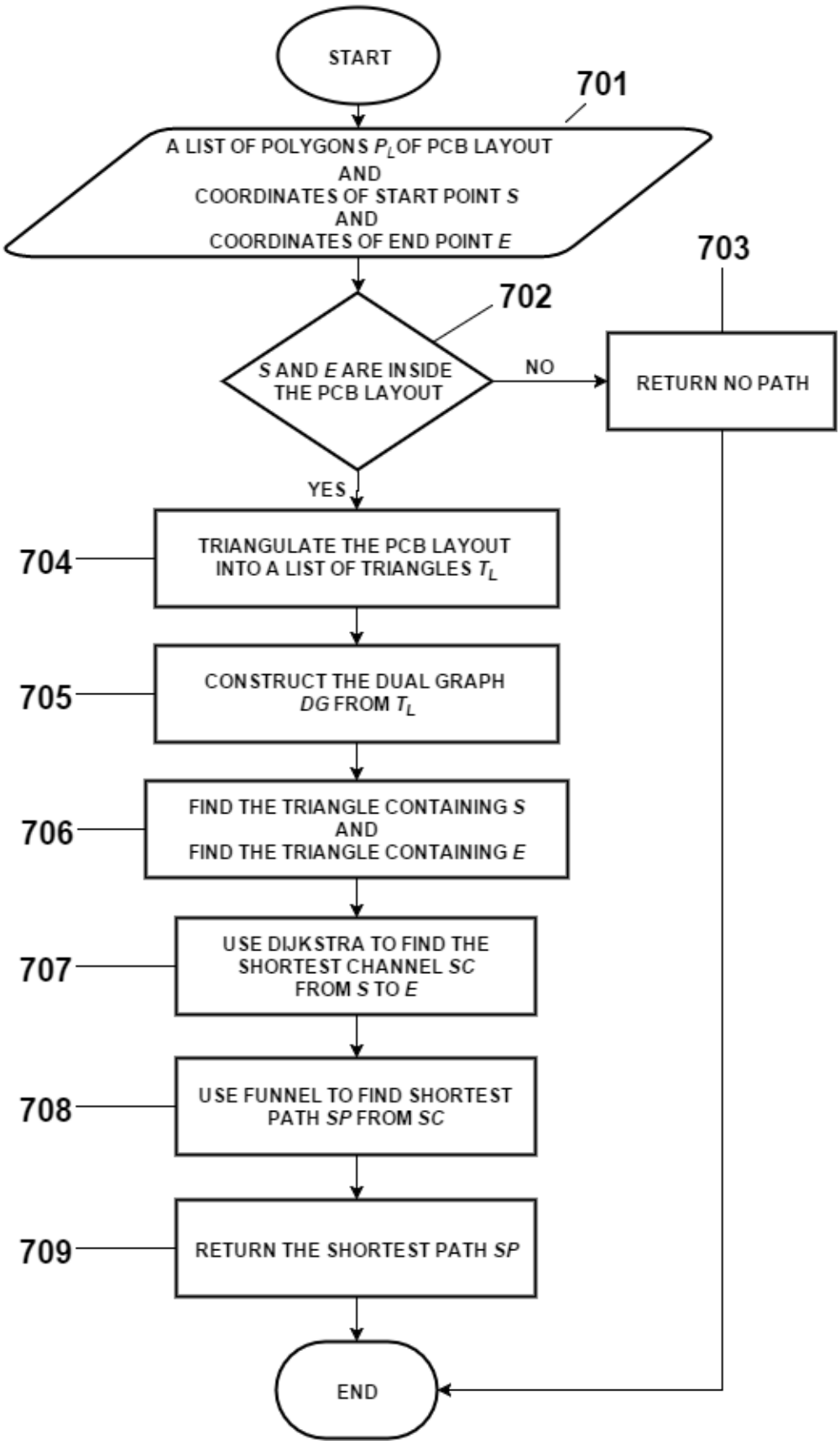
START
701
A LIST OF POLYGONS P_L OF PCB LAYOUT
AND
COORDINATES OF START POINT S
AND
COORDINATES OF END POINT E
703
702
S AND E ARE INSIDE THE PCB LAYOUT
NO
RETURN NO PATH
YES
704
TRIANGULATE THE PCB LAYOUT INTO A LIST OF TRIANGLES T_L
705
CONSTRUCT THE DUAL GRAPH DG FROM T_L
706
FIND THE TRIANGLE CONTAINING S
AND
FIND THE TRIANGLE CONTAINING E
707
USE DIJKSTRA TO FIND THE SHORTEST CHANNEL SC FROM S TO E
708
USE FUNNEL TO FIND SHORTEST PATH SP FROM SC
709
RETURN THE SHORTEST PATH SP
END

FIG. 7A

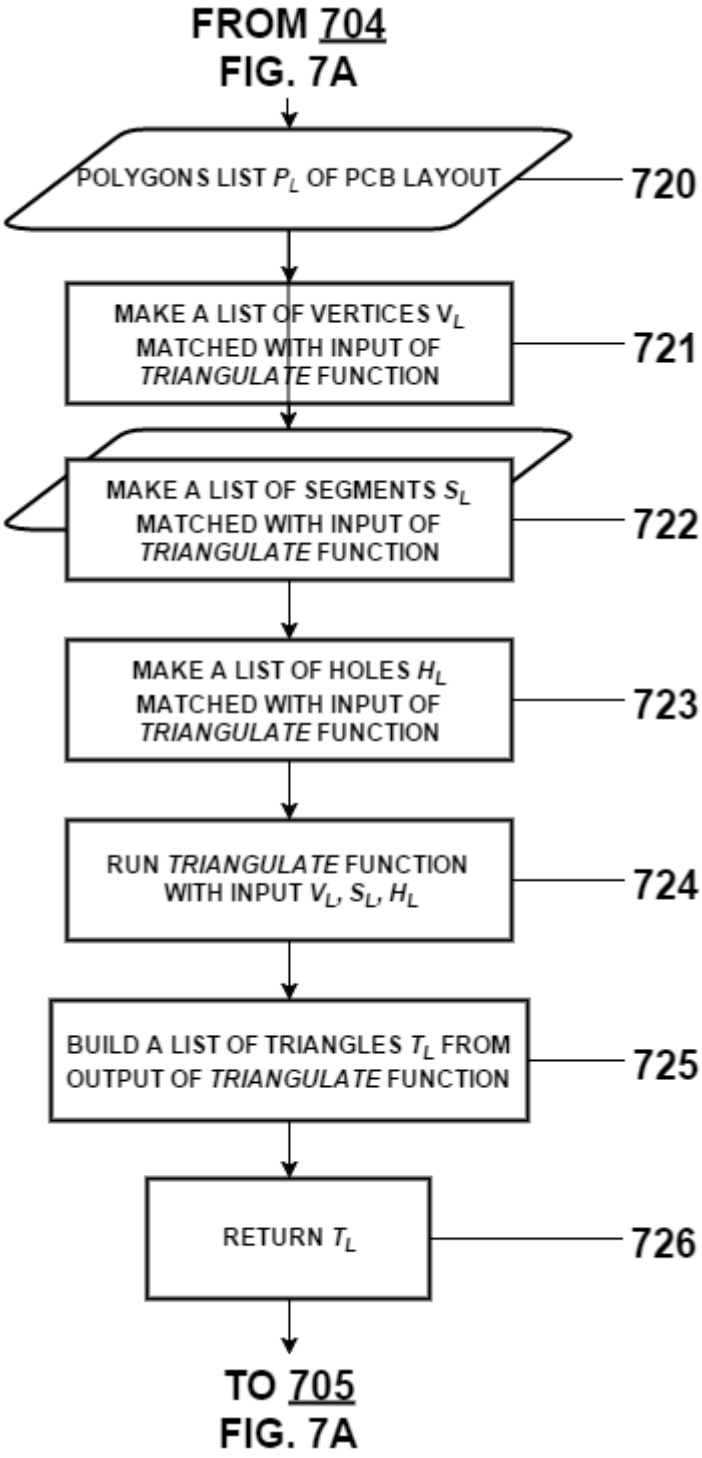
FROM 704
FIG. 7A
POLYGONS LIST P_L OF PCB LAYOUT
720
MAKE A LIST OF VERTICES V_L MATCHED WITH INPUT OF *TRIANGULATE* FUNCTION
721
MAKE A LIST OF SEGMENTS S_L MATCHED WITH INPUT OF *TRIANGULATE* FUNCTION
722
MAKE A LIST OF HOLES H_L MATCHED WITH INPUT OF *TRIANGULATE* FUNCTION
723
RUN *TRIANGULATE* FUNCTION WITH INPUT V_L, S_L, H_L
724
BUILD A LIST OF TRIANGLES T_L FROM OUTPUT OF *TRIANGULATE* FUNCTION
725
RETURN T_L
726
TO 705
FIG. 7A

FIG. 7B

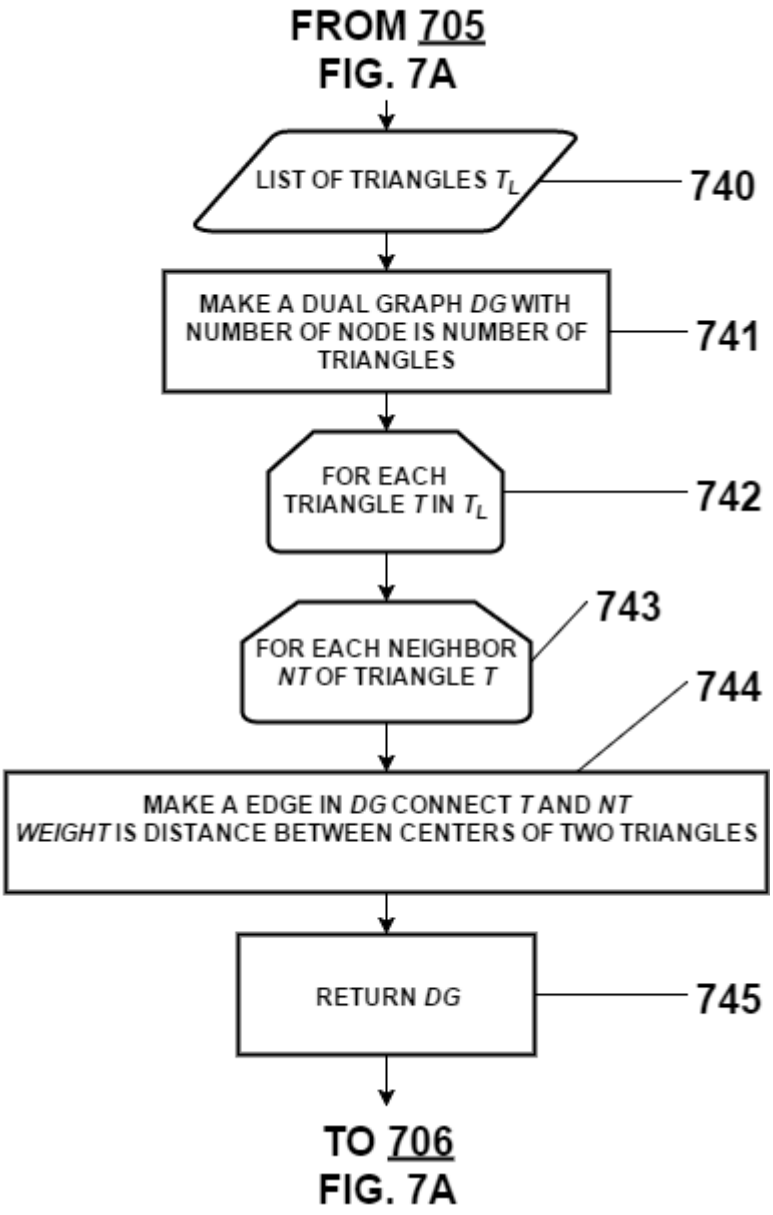
FROM 705
FIG. 7A
LIST OF TRIANGLES T_L
740
MAKE A DUAL GRAPH *DG* WITH NUMBER OF NODE IS NUMBER OF TRIANGLES
741
FOR EACH TRIANGLE *T* IN T_L
742
FOR EACH NEIGHBOR *NT* OF TRIANGLE *T*
743
744
MAKE A EDGE IN *DG* CONNECT *T* AND *NT* *WEIGHT* IS DISTANCE BETWEEN CENTERS OF TWO TRIANGLES
RETURN *DG*
745
TO 706
FIG. 7A

FIG. 7C

5

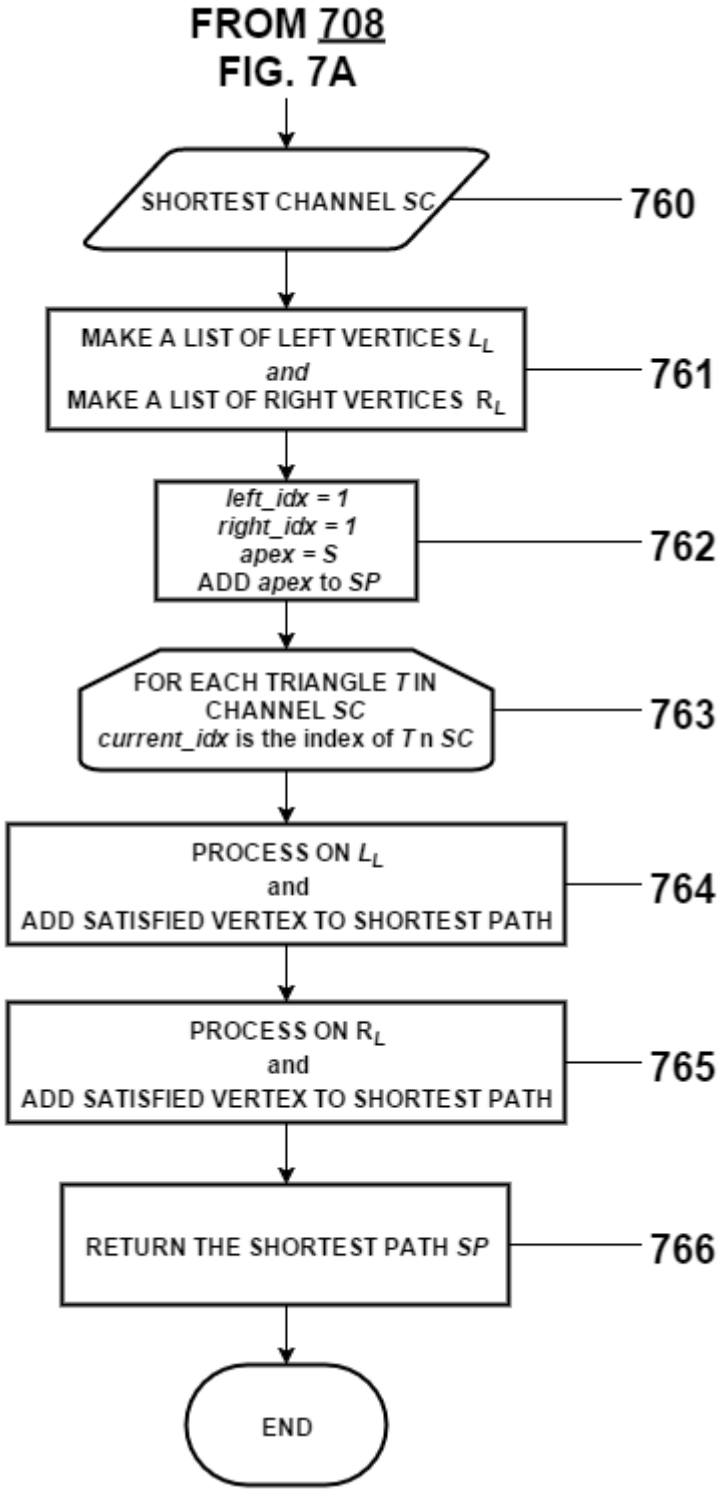
FROM 708
FIG. 7A
SHORTEST CHANNEL SC
760
MAKE A LIST OF LEFT VERTICES L_L
and
MAKE A LIST OF RIGHT VERTICES R_L
761
left_idx = 1
right_idx = 1
apex = S
ADD apex to SP
762
FOR EACH TRIANGLE T IN CHANNEL SC
current_idx is the index of T n SC
763
PROCESS ON L_L
and
ADD SATISFIED VERTEX TO SHORTEST PATH
764
PROCESS ON R_L
and
ADD SATISFIED VERTEX TO SHORTEST PATH
765
RETURN THE SHORTEST PATH SP
766
END

FIG. 7D

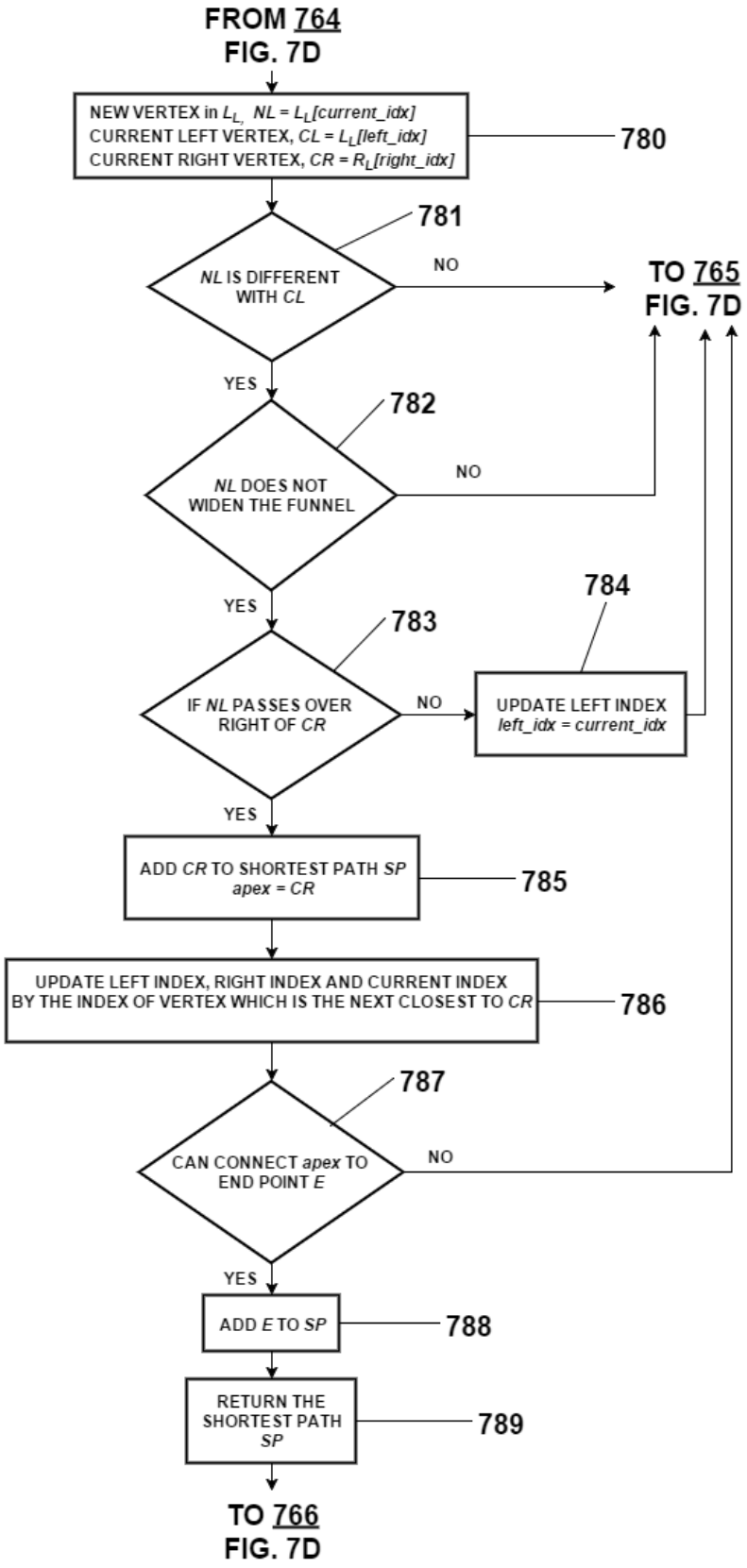
FROM 764
FIG. 7D
NEW VERTEX in L_L, $NL = L_L[current\_idx]$
CURRENT LEFT VERTEX, $CL = L_L[left\_idx]$
CURRENT RIGHT VERTEX, $CR = R_L[right\_idx]$
780
781
NL IS DIFFERENT WITH CL
NO
TO 765
FIG. 7D
YES
782
NL DOES NOT WIDEN THE FUNNEL
NO
YES
784
783
IF NL PASSES OVER RIGHT OF CR
NO
UPDATE LEFT INDEX
left_idx = current_idx
YES
ADD CR TO SHORTEST PATH SP
apex = CR
785
UPDATE LEFT INDEX, RIGHT INDEX AND CURRENT INDEX BY THE INDEX OF VERTEX WHICH IS THE NEXT CLOSEST TO CR
786
787
CAN CONNECT apex TO END POINT E
NO
YES
ADD E TO SP
788
RETURN THE SHORTEST PATH SP
789
TO 766
FIG. 7D

FIG. 7E

BRIEF DESCRIPTION OF THE DRAWING

FIG. **1** is a real PCB layout with many holes **101** inside of it. It also illustrates shortest path **102**, which is generated from our invention.

FIG **2** is a small PCB layout for easily illustrating our invention. It contains holes **203**, start point **201**, and end point **202**.

FIG **3** illustrates the result after using constrained Delaunay triangulation algorithm. The whole layout area is divided into multiple triangles.

FIG **4** illustrates the dual graph of triangles. Nodes **402** are centers of triangles. Edge **401** is done between two nodes if they are two adjacent triangles. The weight of an edge is the distance between centers of two triangles.

FIG **5** illustrates channel **501,** which is the result of Dijkstra algorithm performing on a dual graph.

FIG **6** illustrates components of the funnel algorithm. At this point, the shortest path includes the start point *S* and *apex* **602**. The algorithm is still processing remaining triangles in the channel.

FIG **7A** is a flowchart of the overall method for finding the shortest path in a PCB layout.

FIG **7B** is a flowchart of the method for triangulating the PCB layout.

FIG **7C** is a flowchart of steps for building a dual graph.

FIG **7C** is a flowchart of the funnel algorithm.

DETAILED DESCRIPTION OF THE INVENTION

In FIG **7A**, we show the flowchart of the overall method. The method receives a list of polygons P_L as an input **701**. P_L includes points of an outer polygon and points of multiple inner polygons (holes). The start point and end point are given manually, and **702** check whether they are outside the PCB layout or not. If these points are outside the PCB

layout, no shortest path is returned and the method is ended. Otherwise, steps of triangulation the PCB **704** are performed to triangulate the PCB layout into triangles. A list of triangles T_L is returned in these steps, and it is set as an input to construct a dual graph *DG* in **705**. Dual graph *DG* represents triangles as nodes in its graph. So the number of nodes in graph equals to the number of triangles of the T_L. If two triangles are adjacent triangles, an edge will be made between two nodes represented by two triangles. The weight of the edge is the distance between the centers of two triangles. Step **706** determines the nodes in *DG* containing the start point *S* and the end point *E*. A Dijkstra algorithm at **707** is performed to find a shortest path *SC* between the start node and the end node on *DG*. *SC* corresponds to a sequence of triangles from the start point to the end point, which is used to find the shortest path *SP* by the funnel algorithm at **708**. The method is ended when the shortest path *SP* is generated.

FIG **7B** shows the detailed steps to triangulate a PCB layout. Calling *triangulate* function **723** is the main step in the flowchart of this figure. For our implementation, we select constrained Delaunay triangulation algorithm. The input of this algorithm includes three structures: a list of vertices V_L, a list of segments S_L, and a list of holes H_L. They are constructed from polygons list P_L of the PCB layout. The output of the *triangulate* function is a list of triangles T_L whose each element has an identity number, three vertices, and identities of the adjacent triangles.

T_L is the input of the step for constructing a dual graph *DG* that is clearly shown in FIG **7C**. Particularly, for each triangle *T* in T_L, a node is added to *DG*. For each adjacent triangle *NT* of triangle *T*, an edge is added to *DG* to connect *T* and *NT*. The weight of the edge is the distance between two triangles. Finally, the dual graph *DG* is generated and become the input of the Dijkstra algorithm **707**.

FIG **7D** shows a flowchart of the funnel algorithm to illustrate step **708** in FIG **7A**. The shortest channel found in **707** becomes the input of this algorithm. In detail, the channel is a sequence of adjacent triangles from the triangle containing the start point to the triangle containing the end point (as shown in **501** FIG **5**). The shortest path is constructed within this channel. For running the algorithm, a list of left vertices L_L and a list of right vertices R_L are required. They are illustrated in **601** (FIG **6**). In addition, some

variables **762** are set to control the algorithm. *left_idx* variable controls the current index of the left vertices list, and *right_idx* variable controls the current index of the right vertices list. *Apex* is the current vertex in the shortest path and the first apex is the start point. The algorithm iterates through all triangles of the channel (**763**), and *current_idx* variable identifies the current triangle in the iteration. For each triangle, the algorithm starts to process on the L_L first (**764**) and then on the RL (**765**). Every vertex satisfied the algorithm is added to the shortest path *SP* and returned at **766**.

FIG **7E** shows the steps processed on the L_L (**764**). **765** has the same as **764**, but processed on the R_L. The definition of current right vertex C_R, current left vertex C_L and new left vertex N_L are shown in **780**. C_R will be added to *SP* when N_L satisfies three conditions: different with C_L, not make the funnel (**603** in FIG **6**) widen, over the right of C_R. Otherwise, the algorithm skips to process on the L_L and go to **765** for processing on the R_L. After adding C_R to the shortest path *SP*, the apex is updated by C_R. In addition, the current left index, current right index and current index are updated by the index of the vertex which is in R_L and is the next closest to the C_R. At **787**, if the current apex can connect to the end point without intersecting with any part of the channel, the end point is added to *SP*, and the algorithm is ended by returning the *SP*.

THE INVENTION WE WANT TO PROTECT

1. A computer-implemented method determines the shortest path between two points in a PCB layout.

2. The method of claim **1** can be implemented by any high-level programing language such as C++, Java

3. The method of claim **1** comprises steps of:

(A) triangulate the PCB layout into triangles.

(B) construct a dual graph of the triangles.

(C) determine the nodes in the dual graph containing the start and end point.

(D) use Dijkstra algorithm on the dual graph to find a shortest channel from the start point to the end point

(E) use funnel algorithm on the shortest channel to find the shortest path from the start point to the end point.

4. The method of claim **3**, wherein the step (A) comprises steps of:

(A)(1) loop through PCB layout to create three structures which are the inputs of triangulate function. It includes a list of vertices, a list of segments and a list of holes.

(A)(2) perform *triangulate* function and return a list of triangles.

5. The method of claim **3**, wherein step (B) iterates through all triangles. For each triangle:

(B)(1) add a node to the dual graph.

(B)(2) add an edge between the triangle and its adjacent triangle.

(B)(3) compute the weight of the edge.

6. The method of claim **3**, wherein step (D) returns a shortest channel which is a sequence of triangles from the start point to the end point.

7. The method of claim **3**, wherein the step (E) comprises steps of:

(E)(1) build a list of left vertices and a list of right vertices.

(E)(2) iterate through all triangle in the channel and add satisfied vertices to the shortest path. For each triangle, we process vertices on the list of left vertices first and then process vertices on the list of right vertices.

8. In the method of claim **7**, wherein (E)(2), a vertex in the list of right (left) vertices is added to the shortest path if the processed vertex in the list of left (right) vertices satisfies:

(E)(2)(a) different with current left (right) vertex in the list of left (right) vertices.

(E)(2)(b) not make the funnel widen.

(E)(2)(c) pass over the right (left) of the right (left) funnel.

9. In the method of claim **7**, wherein (E)(2), the funnel algorithm is ended when all triangles in the channel are iterated, or the current apex at any iteration can connect to the end point without intersecting with any parts of the channel.