ISSN 0005-1179, Automation and Remote Control, 2021, Vol. 82, No. 3, pp. 468–480. © Pleiades Publishing, Ltd., 2021.
Russian Text © The Author(s), 2021, published in Avtomatika i Telemekhanika, 2021, No. 3, pp. 123–137.

Software for the Problem of Constructing Cutting Tool Paths in CAD/CAM Systems for Technological Preparation of Cutting Processes

T. A. Makarovskikh[1*] and A. V. Panyukov[1]**

[1]*South Ural State University, Chelyabinsk, 454080 Russia*
*e-mail: *Makarovskikh.T.A@susu.ru, **paniukovav@susu.ru*

Received June 26, 2020; revised April 10, 2020; accepted July 9, 2020

Abstract—Most of the research on tool paths for cutting machines focuses on determining the path for contour cutting. State-of-the-art, resource-efficient sheet metal cutting technologies allow one to match the contours of the cut parts, thus reducing material waste and minimizing the cut length. However, the alignment of the boundaries of the cut contours is the source of a number of constraints formalized in terms of plane graphs: (1) ordered enclosing, (2) nonintersecting cutting path. The article considers the main data structures and algorithms used in the CAD/CAM system developed for technological preparation of cutting processes, which allows cutting plans with combined contours, as well as software that constructs a homeomorphic image of a graph to solve the problem of routing according to the cutting plan, solves this problem, and interprets the solution results.

Keywords: sheet material cutting, cutting plan, matching fragments of part contours, plane graph, route, algorithm, data structures, software

DOI: 10.1134/S0005117921030073

1. INTRODUCTION

Laser cutting is one of the main modern technologies used in the processing of sheet material; this makes the cutting-tool routing problem topical. The problem of determining the path is to determine the exact sequence of cuts. The development of production automation has led to the emergence of numerically controlled processing equipment used for cutting sheet materials. New technologies allow cutting along an arbitrary trajectory with accuracy sufficient for practical purposes. The advantage of using laser cutting is that it minimizes such parameters as cutting width and thermal deformation. The problem of determining the cutting path is aimed at finding a cutting tool route that satisfies the precedence conditions while minimizing the time spent on cutting [1].

The main restrictions in laser cutting are as follows:

1. All elements of the inner contours must be cut out before the enclosing contour is completely traversed (the OE-embracing condition [2]).
2. Cutting path crossings must be avoided, with touchdowns allowed (the NOE-constraint [1, 3]).
3. Thermal effects must be taken into account [4] because the metal sheet heats up during laser cutting.
4. Restrictions on the location of the pierce point (constructing PPOE-cover [5]).
5. It is desirable to reduce the total time required to perform cutting as much as possible, including the total time to complete all cuts, the time spent on idle transitions, and the piercing time.

The papers [1, 6] provide a classification of cutting tool routing problems, and it is noted that ECP (Endpoint Cutting Problem) and ICP (Intermittent Cutting Problem) technologies permit one to reduce material waste, cut length, and length of blank passes due to the ability to align the boundaries of cut parts [1]. The problems of reducing material waste and maximizing the alignment of fragments of the contours of the cut parts are solved at the stage of designing the cutting plan.

Despite the obvious advantages of ECP and ICP technologies, at present a majority of Russian [7–11] and foreign [1, 6, 12, 13] publications are devoted to the development of the GTSP (General Traveling Salesman Problem) technology, which does not imply matching the contours of cut-out parts. When using the GTSP technology, the path length is equal to the sum of the perimeters of all contours, and the number of pierce points is equal to the number of contours, with the problem of satisfying the conditions noted above becoming trivial.

When determining the sequence of cutting fragments of the cutting plan, no information about the shape of the part is used; therefore, all curves without self-intersections and contacts on the plane that represent the shape of the parts are interpreted as the edges of a graph representing the homeomorphic image of the cutting plan, and all points of intersection and contact are represented as vertices of this graph.

A homeomorphic image of the cutting plan is a plane graph G with the outer face f_0 on the plane S. For each part J of the graph G (i.e., for $J \subseteq G$) by $\mathsf{Int}\,(J)$ we denote the set-theoretic union of its inner faces (the union of all connected components $S \setminus J$ not containing the outer face). If we consider J to be the covered part of the cut tool route (it is obvious that J is a plane graph), then $\mathsf{Int}(J)$ is interpreted as the cutoff part of the sheet. The sets of vertices, edges, and faces of J will be denoted by $V(J)$, $E(J)$, and $F(J)$, respectively.

The topological representation of the plane graph G on the plane S is defined up to a homeomorphism by the following functions for each edge $e \in E(G)$ [2, 3]: $v_k(e)$, $k = 1, 2$, are vertices incident to edge e; $l_k(e)$, $k = 1, 2$, are edges obtained by rotating the edge e counterclockwise about the vertex $v_k(e)$; $r_k(e)$, $k = 1, 2$, are the edges obtained by rotating the edge e clockwise about the vertex $v_k(e)$; $f_k(e)$ is the face that is on the right when moving along edge e from vertex $v_k(e)$ to vertex $v_{3-k}(e)$, $k = 1, 2$.

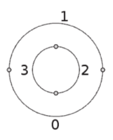
Thus, using the known coordinates of the preimages of the vertices of the graph G and of the fragments of the cutting plan that are the preimages of the edges of the graph G, any route in the graph G can be interpreted as a cutting tool path.

2. REPRESENTATION OF ORIGINAL DATA IN THE SOFTWARE

As a rule, the cutting plan contains large groups of duplicate parts, and to describe the placement of a part on the cutting plan, it is sufficient to indicate the values (x, y, φ), where (x, y) are the coordinates of the base point of this part (usually it is its origin, to which the coordinates of the other points are related), and φ is the angle of rotation of the part around its base point. Therefore, it is wise to have a database of standard parts.

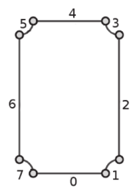
The main primitive elements of cutting route trajectories are segments of straight lines and circular arcs. An entire circle is represented as the union of two arcs with the angles at center φ and $2\pi - \varphi$, $\varphi > 0$. To identify such primitives, it suffices to specify the coordinates (x_1, y_1) and (x_2, y_2) of the extreme points v_1 and v_2, respectively, as well as $\tan(\varphi/4)$, where φ is the central angle of the arc (v_1, v_2) of the circle passing from point v_1 to point v_2 (it is obvious that for a segment we can take $\tan(\varphi/4) = 0$). From the formal point of view, a flat part is part of the plane limited by the outer boundary and inner boundaries in a number equal to the number of holes. Each boundary is a closed contour consisting of a sequence of primitives such that the beginning of each subsequent primitive coincides with the end of the preceding one.

The JSON format [14] is used to represent part-related data. Figure 1 shows examples of parts *a*



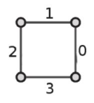
```
{
 "partid": "RING",
  "paths": [
  [
  [200, 100, 1],
  [0, 100, 1],
  [200, 100, 0]],
  [
  [100, 50, -1],
  [100, 150, -1],
  [100, 50, 0]]
  ]
}
```

b



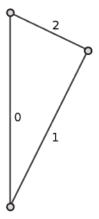
```
{
  "partid": "RECT",
  "paths": [
  [
  [15, 165, 0],
  [93.3484, 165, 0.41421],
  [108.3484, 150, 0],
  [108.3484, 15, 0.4142],
  [93.348, 0, 0],
  [15, 0, 0.4142],
  [0, 15, 0],
  [0, 150, 0.4142],
  [15, 165, 0]]
]}
```

c



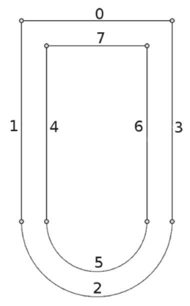
```
{
  "partid": "SQUARE",
  "paths": [
  [
  [50, 50, 0],
  [50, 0, 0],
  [0, 0, 0],
  [0, 50, 0],
  [50, 50, 0]]
]
}
```

d



```
{
  "partid": "RUMB",
  "paths": [
  [
  [0, 0, 0],
  [0, 223.59591, 0],
  [89.7810, 43.83713, 0],
  [0, 0, 0]]
]
}
```

e



```
{
 "partid": "WINDOW",
 "paths": [
 [
    [300, 0, 0],
    [0, 0, 0],
    [0, 400, -1],
    [300, 400, 0],
    [300, 0, 0]
 ],
 [
    [50, 50, 0],
    [50, 400, -1],
    [250, 400, 0],
    [250, 50, 0],
    [50, 50, 0]]
]
}
```

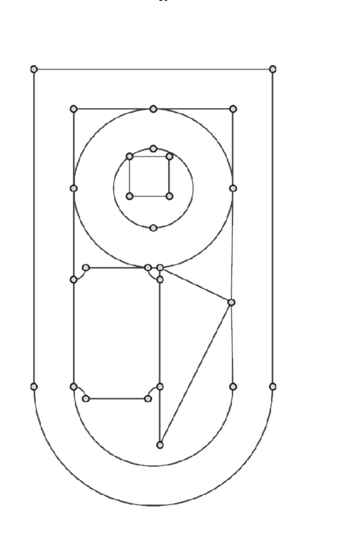
Fig. 1. Examples and descriptions of parts used in the following: (*a*) annulus; (*b*) rectangle; (*c*) square; (*d*) triangle; (*e*) window.

and their descriptions. The description of each part contains its name in the key field `partid` and the list of contours in the field `paths`. Each contour is a three-element array consisting of two coordinates of endpoints and the value of $\tan(\varphi/4)$. The Cartesian coordinates of the endpoint of one primitive are the coordinates of the starting point of the next one. Obviously, such a representation of Cartesian coordinates makes it easy to identify closed contours (the initial coordinates of the first primitive coincide with the final coordinates of the last one). Examples of descriptions of parts consisting of several (more than one) contours are given in Figs. 1a and 1e.

With such a representation of data, the first object is the definition of a sheet containing a cutting plan, with all the subsequent objects being the descriptions of parts, the coordinates of their base points and orientation.

An example of the cutting plan with combined boundaries of part contours is presented in Fig. 2a.

a



b

```
[{
  "partid": "list",
  "paths": [
  [400, 600, 0],
  [400, 0, 0],
  [0, 0, 0],
  [0, 600, 0],
  [400, 600, 0]
]},
{ "partid":"WINDOW", "base":[0, 0, 0]},
{ "partid": "RING", "base": [50, 50, 0]},
{ "partid": "SQUARE", "base": [120, 110, 0]},
{ "partid": "RECT", "base": [50, 250, 0]},
{ "partid": "RUMB", "base": [158.3, 250, 0]}
]
```

Fig. 2. Example: (*a*) cutting plan with matched boundaries; (*b*) JSON-code of cutting plan.

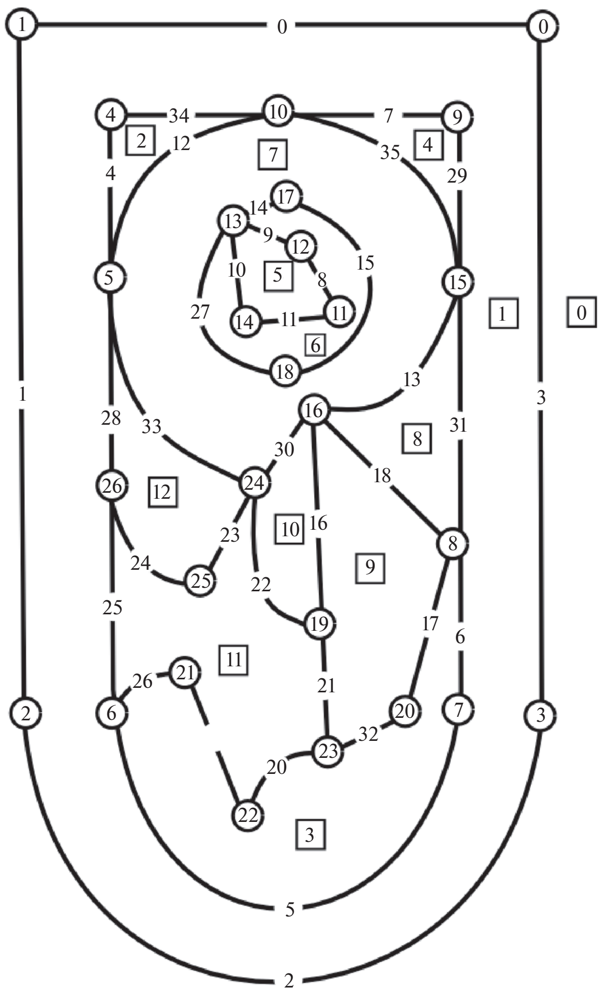


Fig. 3. Example: homeomorphic image of cutting plan with the numbers of vertices, edges, and faces indicated.

No information about the shape of the part is used to determine the sequence of cutting the fragments of the cutting plan; therefore, all curves without self-intersection and contact in the description of the shape of parts can be interpreted as edges of a graph and all point of intersections and touchdowns, as graph vertices. To obtain a representation for the plane graph that makes it possible to reconstruct, up to a homeomorphism, the original cutting plan, it is necessary and sufficient to fix a cyclic order at each vertex on the set of edges incident to it. Figure 3 presents a homeomorphic image of the cutting plan shown in Fig.2a. Data structures used for representing the homeomorphic image must include all information necessary for the effective operation of routing algorithms and for interpreting the routes constructed [15, 16]. Listing 1 gives data structures for representing the homeomorphic image of the cutting plan in the form a plane graph.

Listing 1 (data structures for homeomorphic image of cutting plan).

```
struct Vert { // Graph vertex
   double x, y; // vertex coordinates
   int sv; // number of connected component containing vertex
   int rank; // vertex rank
   double dpth; // vertex depth (adjusted rank)
   int deg; // vertex degree
   int mark; // label for breadth first search in algorithms
   };
struct Edge { // Graph edge
   int rank; // edge rank
   int dpth; // edge depth (adjusted rank)
   int v1, v2; // numbers of incident vertices in container vector<Vert> V
   double x_0,y_0,r; // coordinates of arc center and its radius
   double v1_ang, v2_ang; // central angles of possible arcs
   double v1_ta4, v2_ta4; // tangents of the quarters of corresponding arcs
   int l1,l2,r1,r2; // numbers of possible adjacent edges in vector<Edge> E
   int sv; // number of connected component in container vector<Connect> Sv
   int cycle1, cycle2; // numbers of contours in container vector<Cycle> C
   int f1, f2; // numbers of faces incident to edge in vector<Face> F
   int mark; // label for breadth first search in algorithms
   };
struct Face { // Graph face
   int number; // face (part, hole, additional) number
   int dpth; // face depth
   };
struct Connect { // Connected component
   int v; // one of vertices of component (number in vector<Vert> V)
   int outcycle; // number of enclosing contour in vector<Cycle> C
   int up; // adjacent enclosing component (number in vector<Connect> Sv)
   int rank; // component rank
   Connect () :  v(-1), outcycle(-1), up(-1), rank(-1) {}; // constructor
   };
```

```
struct Cycle { // Contour of outer or inner graph face
   int v; // number of contour vertex in container vector<Vert> V
   int e; // number of edge in container vector<Edge> E incident to v
   int f; // number of face in container vector<Face>
   vector<int> vertexes; // traversal-order sorted vertices
   vector<int> edges; // traversal-order sorted cycle edges
   Cycle (int vtx,int edg) :  v(vtx), e(edg), f(-1) {}; // constructor
   Cycle () :  v(-1), e(-1), f(-1) {}; // default constructor
   };
```

The `Vert` structure contains fields with the specification of the Cartesian coordinates of the corresponding point on the cutting plan and a number of auxiliary fields. These data are necessary for route interpreter and when filling the fields of the `Edge` structure. The fields of the `Edge` structure contain the numbers `v1`, `v2` of incident vertices, the numbers `f1`, `f2` of incident faces, the numbers `l1`, `l2`, `r1`, `r2` of neighboring, in cyclic order, edges, the values `v1_ta4`, `v2_ta4` of the quantity $\tan(\varphi/4)$, where φ is the central angle of arcs (v_1, v_2) and (v_2, v_1), respectively, as well as auxiliary quantities. The `Face` structure puts objects on the cutting plan with the faces of its homeomorphic image. The `Connect` structure is used for identifying the connected components of the homeomorphic image of the cutting plan. The `Cycle` structure is used for identifying contours that are the boundaries of the faces. The containers for the above structures, other auxiliary data, all methods employed for filling them, and routing algorithms are encapsulated in the class `DataHolder` [15–17].

3. ROUTING IN CONNECTED GRAPHS

The use of a plane graph as a homeomorphic image of the cutting plan model allows one to formalize technological constraints on the order of cutting out fragments of the cutting plan: first, the graph G contains the images of all possible elements of the tool path; second, the cutting route must satisfy the ordered enclosing condition, i.e., the part cut off from the sheet should not require additional cuts [2]; third, there should be no self-intersections of the cutting path [3].

An admissible route is formalized as an ordered sequence of OE-chains covering the graph [2, Definitions 4–5].

The definition of an OE-cover is quite constructive, as proved by the efficiency of the algorithms considered in [2]. If the connected graph G is not Eulerian, then it contains $2k$, $k \geqslant 1$, vertices of odd degree. In this case, the OE-route consists of k edge-disjoint chains. The problem of constructing such a route is solved by the `OE-Router` algorithm [18]. Moreover, in the route constructed, the length of idle transitions (i.e., transitions between the end of the current chain and the beginning of the next chain) may not be optimal. If a plane graph G representing the image of the cutting plan contains no bridges (i.e., edges incident to one face), then one can construct an OE-route in which the edges of an arbitrary matching M on the subset $V_{\text{odd}} \subset V(G)$ of the set of odd-degree vertices and only they correspond to idle transitions. Choosing the shortest matching M allows one to find a route with the minimum length of idle transitions. Note that connected planar graphs that are images of cutting plans, as a rule, do not contain bridges. Therefore, if M is the shortest matching, then the `M-OE-Router` algorithm builds a route with the minimum length of idle passes.

If a graph G representing the homeomorphic image of the cutting plan is connected and does not contain bridges, then the `M-OE-Router` algorithm solves the problem exactly but requires determining the shortest matching. The `OE-Router` algorithm solves the problem for any graph G using the greedy idle-course selection strategy. The complete list of algorithms for constructing OE-routes for connected graphs is given in Table 1.

Table 1. Algorithms for constructing OE-routes

Algorithm name	Computational complexity
Eulerian OE-cycle (recursive algorithm) [2]	$O(\|V\|^2)$
Eulerian OE-cycle (OE-Cycle algorithm) [2]	$O(\|E\| \cdot \log_2\|V\|)$
OE-Postman Route (CPP_OE algorithm)	$O(\|E\| \cdot \|V\|)$
OE-Router [18]	$O(\|E\| \cdot \log_2\|V\|)$
M-OE-Router [2]	$O(\|V\|^2)$

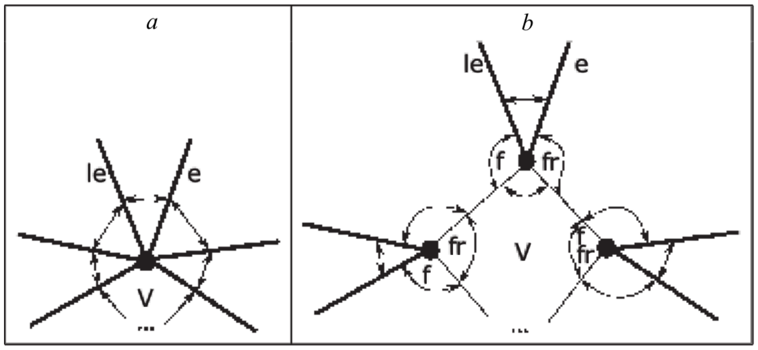


Fig. 4. (a) Original pointers to adjacent edges in split vertex. (b) Splitting of a vertex (bold lines show the edges of the graph G; thin lines are additional (fictitious) edges) and modification of pointers in accordance with splitting.

4. NOE-ROUTING

The problem of constructing efficient algorithms for finding nonintersecting chains in plane graphs remains open. Certain attempts to solve this problem were undertaken in [19]. The paper [20] proposed a solution of the problem for a plane connected 4-regular graph.

Definition 1. The Eulerian cycle C in a plane graph G is said to be nonintersecting (self-avoiding) if it is homeomorphic to a cyclic graph $\tilde{G}$ representing a plane Jordan curve with no self-intersections. The cyclic graph $\tilde{G}$ can be obtained from the graph G by applying $O(|E(G)|)$ vertex splitting operations.

The general case is the solution of the problem of constructing a self-avoiding OE-chain (or NOE-chain, nonintersecting OE-trail) [21].

Definition 2. We say that a chain is an NOE-chain if it is simultaneously an OE-chain and a nonintersecting chain.

Definition 3. The system of chain transitions [22] that corresponds to a nonintersecting chain will be called a system of nonintersecting transitions.

The fact that for a transition system corresponding to a nonintersecting Eulerian cycle there exists an initial vertex and a final edge adjacent to the outer face such that the cycle constructed for them is an OE-cycle is proved in many ways similar to the proof of Theorem 1 in [20] for a 4-regular graph G. This proof is an algorithm for constructing an NOE-chain.

To construct a nonintersecting Eulerian OE-chain (or cycle) in a plane Eulerian graph (in the sequel, this chain will be called an NOE-chain (nonintersecting OE-chain)) for which no fixed system of transitions is given can be constructed in the following manner [21].

On the set of vertices $V(G)$, we define the Boolean function

$$\text{Checked}\,(v) = \begin{cases} \texttt{true} & \text{if the vertex has been checked} \\ \texttt{false} & \text{otherwise.} \end{cases}$$

When initializing this function, all vertices in $V(G)$ are declared as unchecked.

The function `nonintersecting`(G) (Algorithm 1) splits all vertices $v \in V(G)$, $\deg(v) = 2n$, $n \geqslant 3$, in the graph G into n fictive vertices of degree 4 and introduces n fictive edges that are incident to the vertices resulting after splitting and form a cycle (Fig. 4). To perform the above transformations, it is necessary to check all functions $v_k(e)$, $k = 1, 2$, defined for all edges e and introduce the required modifications into the entire graph coding system. The function uses the procedure `Handle`$(e, v_k(e), k)$ (Algorithm 2), which processes each unchecked vertex of the graph G. Processing consists in splitting the vertex $v_k(e)$ in accordance with Figs. 4a and 4b.

Algorithm 1 (function `Nonintersecting`(G)).

```
Require: a plane Eulerian graph $G$;
Ensure: a plane connected 4-regular graph $G^*$;
for all $(e \in E(G))$ do                                  ▷ Check all graph edges
    $k = 1$;                      ▷ Sequentially process functions with index 1 and then 2
    while $(k \leqslant 2)$ do
        if $(\overline{\text{Checked}\,(v_k(e))})$ then     ▷ If the vertex has not been processed earlier
            Handle$(e, v_k(e), k)$;                           ▷ Process the vertex
        end if
        $k++$;
    end while
end for
Return $G^*$;
```

Algorithm 2 (procedure `Handle`(e, v, k)).

```
                                              ▷ Pass 1: determining degree of vertex $v$
$e_{\text{first}} = e$;
$d = 0$;                                                        ▷ Vertex degree
repeat                                                          ▷ Check all edges
    $le = l_k(e)$;                          ▷ Find adjacent edge given by function $l_k(e)$
    if $(v_k(le) \neq v)$ then
        REPLACE$(le)$;
    end if
    $e = le$;                           ▷ Count current edge when calculating degree
    $d = d + 1$;                                        ▷ and pass to the next one
until $(e = e_{\text{first}})$;
                                    ▷ Pass 2: splitting of vertices with degree above 4
if $(d > 4)$ then
    $e = e_{\text{first}}$; $le = l_k\,(e)$; $fl$=new EDGE; $fle = fl$; $e_{\text{first}} = e$; $e_{\text{next}} = l_k\,(le)$;
    repeat
        $e = e_{next}$; $le = l_k\,(e)$; $fr = fl$; $fl$=new EDGE; $e_{\text{next}} = l_k\,(le)$;
        Pointers$(e,\ le,\ fr, fl)$;                   ▷ Arrange pointers for edges
    until $(l_k(le) = e_{\text{first}})$;
    Pointers$(e_{\text{first}},\ l_k(e_{\text{first}}), fle,\ fe)$;   ▷ Arrange pointers for edges
end if
```

The n fictive vertices introduced by the `Handle` procedure and n fictive edges incident to these vertices form a cycle. As a result of processing all vertices of the graph G, we obtain a modified graph G^* that is a plane connected 4-regular graph. We can apply the algorithm `AOE-TRAIL()` to G^*. This algorithm will construct an AOE-chain T^* in it. If we then replace all fictive edges in T^* and the vertices incident to them obtained when splitting the vertex v by v, then we obtain an NOE-chain T in the original graph G. The chain obtained after removing the edges will belong to the OE-class, because the edge removal procedure does not alter the collation order of the remaining edges in the chain; this rules out the emergence of a cycle that encloses the untraveled as yet edges.

5. ROUTING FOR DISCONNECTED GRAPHS

If the cutting plan contains parts with holes, as well as other parts located in these holes, then the plane graph $G = (V(G), F(G), E(G))$ representing the homeomorphic image of the cutting plan turns out to be disconnected. Since the cut out fragments contain the preimages of the enclosed edges of the graph, the requirements for the cutting route that guarantee the validity of the OE-constraint can be formalized in terms of the graph $G' = (F(G), V(G), E(G))$ dual to G [2]: it is necessary that the order of traversing the faces of the graph G (i.e., the vertices of the graph G') be an extension of the partial order relation $\prec$,

$$(f_i \prec f_j) \Leftrightarrow \left(f_j \text{ belongs to the shortest chain } T_{G'}^{f_0} \text{ between } f_i \text{ and } f_0\right),$$

where f_0 is the outer (infinite) face of the plane graph G. The list of algorithms for constructing the OE-routes for disconnected graphs is given in Table 2. The following approaches are implemented to construct an OE-cover in a disconnected graph:

1. Determining the admissible traversal of the connected components of the graph G.
2. Completion of the set of edges $E(G)$ to the set $E(\tilde{G})$, where $\tilde{G}$ is a plane connected graph.

The first approach is realized in the algorithm `MultiComponent`, in which finding the desired OE-route amounts to an independent construction of OE-routes for each connected component and their subsequent merger into the resulting route in the order of decreasing ranks of the connected components. The second approach is realized by the algorithms `Bridging`, `DoubleBridging`, and `FaceCutting` and is based on adding bridges in *separating faces*.

Definition 4 [18]. A face $f \in F(G)$ is said to be separating if the graph $G' \setminus \{f\}$ is disconnected.

Let the graph $\tilde{G}$ be obtained from the graph G by adding bridges belonging to the separating faces between the connected components of the cardinality- and length-minimum set $\tilde{E}$. Obviously, such edges in the graph G' will be the edges of the minimum-weight spanning tree. The graph $\tilde{G}$ thus obtained is a plane connected graph for which one can construct an OE-route $M(\tilde{G})$ using the algorithm `OE-Router` [18].

In this case, the OE-route $M(G)$ is constructed based on the route $M(\tilde{G})$ by removing the edges of the introduced set $\tilde{E}$. In this case, we obtain a set of chains representing an OE-cover of the original disconnected graph. This approach is realized by the algorithm `Bridging`.

Table 2. Algorithms for constructing OE-routes for disconnected graphs

Algorithm name	Computational complexity
MultiComponent	$O(\|E\| \cdot \log_2\|V\|)$
Bridging	$O(\|E\| \cdot \log_2\|V\|)$
DoubleBridging	$O(\|V\|^2)$
FaceCutting	$O(\|E\| \cdot \log_2\|V\|)$

To ensure the possibility of applying the algorithm `M-OE-Router`, it is sufficient to complement the graph G to the graph $\tilde{\tilde{G}}$ by including the set of edges $\tilde{\tilde{E}} = \cup_{e\in\tilde{E}}\{e_1 = e, e_2 = e\}$, i.e., by including two replicas for each edge $e \in \tilde{E}$. This approach, in conjunction with the algorithm `M-OE-Router`, is realized by the algorithm `DoubleBridging`.

Theorem 1. *If the degrees of vertices incident to the separating faces of the graph G in each connected component G_k of the graph G are even, then the route with the minimum length of additional constructions is realized by the algorithm* `DoubleBridging`.

Proof. It is obvious that the traversal of each connected component must end at the outer face. If we assume that the traversal of the connected component starts from a vertex that does not belong to the outer face, then this fragment of the OE-cover ends at a vertex of odd degree that does not belong to the face. Since there are no vertices of odd degree on the boundary, in accordance with the definition of the OE-route, the part of the graph containing the outer face will remain untraveled. In the optimal solution (obtained by the `M-OE-Router` algorithm), the components will be connected by pairs of multiple edges. Moreover, the total weight of all connecting edges is minimal. The proof of the theorem is complete. ■

It is obvious that the length of the bridges introduced in the `DoubleBridging` algorithm is at least twice the length of the shortest spanning tree in the separating face.

Another way to obtain a connected graph without bridges is to split the vertices incident to the separating face using a Hamiltonian cycle (the `FaceCutting` algorithm). This approach seems to be the most expedient, because, first, in practice, the dimension of the traveling salesman problem is comparable to the estimates of the degrees of the corresponding separating faces (i.e., rather low); secondly, in accordance with the metric of the graph G, even the approximate polynomial Christophides algorithm [23] for the traveling salesman problem constructs a Hamiltonian cycle in the separating face with a length at most twice the length of the shortest spanning tree.

6. EXAMPLE

Table 3 (the left column) shows the optimal NOE-cover of the cutting plan in Fig. 2 by an ordered sequence of OE-chains. The first chain goes through the square and the inner border of the annulus. The second chain goes along the right-hand side of the rectangle and along the outer edge of the annulus, and the cutting ends with the triangle and rectangle. The third chain completes the inner outline of the window. The fourth chain runs along the outer contour of the window. The toolpath JSON-code is given in the right column of Table 3. Each individual chain corresponds to a specific part of the continuous cutting path and is a sequence of three-element arrays containing the coordinates of the current starting point of the primitive, as well as the value for this point. All technological constraints are observed when cutting parts in accordance with the found sequence of chains.

7. CONCLUSIONS

A technology that allows the borders of cut parts to be aligned is a modern resource-saving cutting technology. Routing algorithms are known when the following technological constraints are simultaneously imposed on the cutting tool path:

1. A part cut off from the sheet does not require further cuts.
2. The cutting path has no self-intersections.

The present paper provides a solution of the problem of effective software implementation of these algorithms and presents the authors' results used in the development of functional elements

Table 3. Optimal NOE-cover of cutting plan in Fig. 2 by ordered sequence of OE-chains

Chain	Chain JSON-code
chain 1: v17 e14 v13 e9 v12 e8 v11 e11 v14 e10 v13 e27 v18 e15 v17	`{ "partid": "chain_1", "paths": [ [150.0,100.0,-0.162278], [120.0,110.0,-0.0], [170.0,110.0,-0.0], [170.0,160.0,-0.0], [120.0,160.0,-0.0], [120.0,110.0,-0.720759], [150.0,200.0,-1.0], [150.0,100.0,0] ]},`
chain 2: v23 e21 v19 e16 v16 e30 v24 e33 v5 e12 v10 e35 v15 e13 v16 e18 v80 e17 v20 e32 v23 e20 v22 e19 v21 e26 v6 e25 v26	`{ "partid": "chain_2", "paths": [[150.0,400.0,0.0], [150.0,265.0,-0.0], [150.0,250.0,0.0375], [135.0,250.0,0.374006], [50.0,150.0,0.4142], [150.0,50.0,0.4142], [250.0,150.0,0.4142], [150.0,250.0,-0.0], [250.0,300.0,-0.0], [150.0,475.0,-0.0], [150.0,400.0,-0.414213],[135.0,415.0,-0.0], [65.0,415.0,-0.414213], [50.0,400.0,-0.0], [50.0,265.0,0]]},`
chain 3: v19 e22 v24 e23 v25 e24 v26 e28 v5 e4 v4 e34 v10 e7 v9 e29 v15 e31 v8 e6 v7 e5 v6	`{ "partid": "chain_3", "paths": [ [150.0,265.0,0.414213], [135.0,250.0,0.0], [65.0,250.0,0.414213], [50.0,265.0,-0.0], [50.0,150.0,-0.0], [50.0,50.0,-0.0], [150.0,50.0,-0.0], [250.0,50.0,-0.0], [250.0,150.0,-0.0], [250.0,300.0,-0.0], [250.0,400.0,1.0], [50.0,400.0,0] ]},`
chain 4: v0 e0 v1 e1 v2 e2 v3 e3 v0	`{ "partid": "chain_4", "paths": [ [300.0,0.0,0.0], [0.0,0.0,0.0], [0.0,400.0,-1.0], [300.0,400.0,0.0], [300.0,0.0,0] ]},`

of a complex of programs for an automated system for technological preparation of sheet material cutting processes.

FUNDING

This work was supported by the RF Government, Act no. 211 of March 16, 2013, Agreement no. 02.A03.21.0011 and by the RF Ministry of Science and Higher Education, State Order no. FENU-2020-0022.

REFERENCES

1. Dewil, R., Vansteenwegen, P., and Cattrysse, D., A review of cutting path algorithms for laser cutters, *Int. J. Adv. Manuf. Technol.,* 2016, vol. 87, pp. 1865–1884. https://doi.org/10.1007/s00170-016-8609-1
2. Makarovskikh, T.A., Panyukov, A.V., and Savitsky, E.A., Mathematical models and routing algorithms for CAD technological preparation of cutting processes, *Autom. Remote Control*, 2017, vol. 78, no. 4, pp. 868–882.
3. Makarovskikh, T. and Panyukov, A., The cutter trajectory avoiding intersections of cuts, *IFAC-Papers-OnLine*, 2017, vol. 50, no. 1, pp. 2284–2289. https://doi.org/10.1016/j.ifacol.2017.08.226
4. Li, X., Liu, Zh., Wang, F., Yi, B., and Song, Y., Combining physical shell mapping and reverse-compensation optimisation for spiral machining of free-form surfaces, *Int. J. Prod. Rts.,* 2018. https://doi.org/10.1080/00207543.2018.1512763

5. Makarovskikh, T. and Panyukov, A., Development of routing methods for cutting out details, *CEUR Workshop Proc.,* 2018, vol. 2098, pp. 249–263. http://ceur-ws.org/Vol-2098/paper22.pdf.

6. Dewil, R., Vansteenwegen, P., Cattrysse, D., Laguna, M., and Vossen, T., An improvement heuristic framework for the laser cutting tool path problem, *Int. J. Prod. Rts.*, 2015, vol. 53, no. 6, pp. 1761–1776. https://doi.org/10.1080/00207543.2014.959268

7. Petunin, A. and Stylios, C., optimization models of tool path problem for CNC sheet metal cutting machines, IFAC-PapersOnLine, 2016, vol. 49, pp. 23–28. https://doi.org/10.1016/j.ifacol.2016.07.544

8. Petunin, A., Chentsov, A.G., and Chentsov, P.A., About routing in the sheet cutting, *Bull. South Ural State Univ. Ser.: Math. Model. Program. Comput. Software*, 2017, vol. 10, no. 3, pp. 25–39. https://doi.org/10.14529/mmp170303

9. Chentsov, A.G., Grigoryev, A.M., and Chentsov, A.A., Solving a routing problem with the aid of an independent computations scheme, *Bull. South Ural State Univ. Ser. Math. Model. Program. Comput. Software*, 2018, vol. 11, no. 1, pp. 60–74. https://doi.org/10.14529/mmp180106

10. Khachay, M. and Neznakhina, K., Towards tractability of the Euclidean generalized travelling salesman problem in grid clusters defined by a grid of bounded height, *Commun. Comput. Inf. Sci.,* 2018, vol. 871, pp. 68–77. https://link.springer.com/chapter/10.1007/978-3-319-93800-4_6.

11. Chentsov, A., Khachay, M., and Khachay, D., Linear time algorithm for precedence constrained asymmetric generalized traveling salesman problem, *IFAC-PapersOnLine,* 2016, vol. 49, pp. 651–655. https://doi.org/10.1016/j.ifacol.2016.07.767

12. Hoeft, J. and Palekar, U., Heuristics for the plate-cutting traveling salesman problem, *IIE Trans.*, 1997, vol. 29, pp. 719–731. https://doi.org/10.1023/A:1018582320737

13. Dewil, R., Vansteenwegen, P., and Cattrysse, D., Construction heuristics for generating tool paths for laser cutters, *Int. J. Prod. Rts.,* 2014, vol. 52, no. 20, pp. 5965–5984. https://doi.org/10.1080/00207543.2014.895064

14. Crockford D.The Application/json Media Type for JavaScript Object Notation (JSON), Internet Engineering Task Force, 2006. https://www.rfc-editor.org/info/rfc4627.

15. Makarovskikh, T.A., Panyukov, A.V., and Savitskiy, E.A., Software for the problem of constructing cutting tool motion path, *Tr. XVIII-i Mezhdunar. molodezhnoi konf. "Sistemy proektirovaniya, tekhnologicheskoi podgotovki proizvodstva i upravleniya etapami zhiznennogo tsikla promyshlennogo produkta (CAD/CAM/PDM-2018)"* (Proc. XVIII Int. Youth Conf. "Design Systems for Industrial Technological Preparation and Control over Stages of an Industrial Product Lifecycle (CAD/CAM/PDM-2018)") (2018), pp. 172–176.

16. Makarovskikh, T.A., Panyukov, A.V., and Savitskiy, E.A., Cutting tool routing problem: software implementation, in *Tr. XIII Vseross. soveshchaniya po problemam upravleniya (VSPU-2019)* (Proc. XIII All-Russia Meeting on Control Problems (VSPU-2019)) Novikov, D.A., Ed., 2019, pp. 2650–2654.

17. Makarovskikh, T.A., Panyukov, A.V., and Savitskiy, E.A., Software development for cutting tool routing problems, *Procedia Manuf.,* 2019, vol. 29, pp. 567–574. https://doi.org/10.1016/j.promfg.2019.02.123

18. Makarovskikh, T.A., Panyukov, A.V., and Savitskiy, E.A., Mathematical models and routing algorithms for economical cutting tool paths, *Int. J. Prod. Rts.,* 2018, vol. 5, no. 3, pp. 1171–1188. https://doi.org/10.1080/00207543.2017.1401746

19. Manber, U. and Bent, S.W., On non-intersecting Eulerian circuits, *Discrete Appl. Math.*, 1987, vol. 18, pp. 87–94. https://doi.org/10.1016/0166-218X(87)90045-X

20. Makarovskikh, T.A., Software for constructing A-circuits with ordered enclosing in a plane connected 4-regular graph, *Vestn. Yuzhn.-Ural. Gos. Univ. Ser. Vychisl. Mat. Inf.*, 2019, vol. 8, no. 1, pp. 36–53. https://doi.org/10.14529/cmse190103

21. Makarovskikh, T.A., Constructing nonintersecting OE-routes in a plane Eulerian graph, *Vestn. Yuzhn.-Ural. Gos. Univ. Ser. Vychisl. Mat. Inf.,* 2019, vol. 8, no. 4, pp. 30–42. https://doi.org/10.14529/cmse190403

22. Makarovskikh, T.A., On the number of OE-trails for a fixed transition system, *Vestn. Yuzhn.-Ural.Gos. Univ. Ser. Mat. Mekh. Fiz.,* 2016, vol. 8, no. 1, pp. 5–12. https://doi.org/10.14529/mmph160101

23. Garey, M.R. and Johnson, D.S., *Computers and Intractability: a Guide to the Theory of NP-Completeness*, San Francisco: W.H. Freeman, 1979.

This paper was recommended for publication by A.A. Lazarev, a member of the Editorial Board