Автоматика и телемеханика, № 3, 2021

Управление в технических системах

© 2021 г. **Т.А. МАКАРОВСКИХ, канд. физ. мат. наук (Makarovskikh.T.A@susu.ru),**
А.В. ПАНЮКОВ, д-р. физ. мат. наук (paniukovav@susu.ru)
(Южно-Уральский государственный университет (НИУ), Челябинск)

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДЛЯ ЗАДАЧИ ПОСТРОЕНИЯ ТРАЕКТОРИИ ДВИЖЕНИЯ РЕЖУЩЕГО ИНСТРУМЕНТА ДЛЯ CAD/CAM СИСТЕМ ТЕХНОЛОГИЧЕСКОЙ ПОДГОТОВКИ ПРОЦЕССОВ РАСКРОЯ[1]

Большинство исследований, касающихся траекторий инструмента для режущих машин, посвящены определению траектории при поконтурном вырезании. Современные ресурсосберегающие технологии резки листовых материалов позволяют совмещать контуры вырезаемых деталей, что уменьшает количество отходов материала и сокращает длину резки. Однако совмещение границ вырезаемых контуров является источником ряда ограничений, формализуемых в терминах плоских графов: (1) упорядоченное охватывание, (2) самонепересекающаяся траектория резания. В статье рассмотрены основные структуры данных и алгоритмы, используемые в разрабатываемой CAD/CAM системе технологической подготовки процессов раскроя, допускающей раскройный план с совмещенными контурами, и программное обеспечение, которое для решения задачи маршрутизации по раскройному плану строит гомеоморфный образ графа, решает данную задачу и интерпретирует результаты решения.

Ключевые слова: вырезание из листового материала, раскройный план, совмещение фрагментов контуров деталей, плоский граф, маршрут, алгоритм, структуры данных, программное обеспечение.

DOI: 10.31857/S0005231021030077

1. Введение

Лазерная резка является одной из основных современных технологий, используемых при обработке листового материала, что делает актуальной задачу определения траектории движения режущего инструмента. Задача определения траектории заключается в определении точной последовательности резов. Развитие автоматизации производства привело к появлению технологического оборудования с числовым программным управлением, используемого для резки листовых материалов. Новые технологии позволяют осуществлять вырезание по произвольной траектории с достаточной для практики точностью. Преимуществом при использовании лазерной резки является

[1] Статья выполнена при поддержке Правительства РФ (Постановление № 211 от 16.03.2013 г.), Соглашение № 02.A03.21.0011 и Министерства науки и высшего образования РФ (государственное задание FENU-2020-0022).

минимальность таких показателей, как ширина реза и термические деформации. Целью задачи определения маршрута резки является поиск такого пути режущего инструмента, при котором выполняются условия предшествования, а время, затраченное на вырезание, минимально [1].

Основными ограничениями при лазерной резке являются: (1) все элементы внутренних контуров должны быть вырезаны прежде, чем будет полностью пройден охватывающий их контур (условие OE-охватывания [2]); (2) следует избегать пересечения траектории резки, касания допустимы (NOE-ограничение [1, 3]); (3) в процессе лазерной резки происходит нагревание металлического листа, поэтому необходимо учитывать термальные эффекты [4]; (4) ограничения на расположение точки врезки (построение PPOE-покрытия [5]); (5) общее время, требуемое на выполнение резки, представляющее суммарное время для осуществления всех вырезаний, время на выполнение холостых переходов и время на врезку желательно сокращать.

В [1, 6] приводится классификация задач маршрутизации режущего инструмента и отмечается, что технологии ECP (Endpoint Cutting Problem) и ICP (Intermittent Cutting Problem) за счет возможности совмещения границ вырезаемых деталей позволяют сократить расход материала, длину резки и длину холостых проходов [1]. Проблемы уменьшения отходов материала и максимального совмещения фрагментов контуров вырезаемых деталей решаются на этапе составления раскройного плана.

Несмотря на очевидные преимущества технологий ECP и ICP, в настоящее время большинство отечественных [7–11] и зарубежных [1, 6, 12, 13] публикаций посвящено развитию технологии GTSP (General Travelling Salesman Problem), которая не предполагает совмещение контуров вырезаемых деталей. При использовани технологии GTSP длина траектории будет равна сумме периметров всех контуров, а количество точек врезки — количеству контуров, при этом проблема выполнения отмеченных выше условий оказывается тривиальной.

Для определения последовательности резки фрагментов раскройного плана не используется информация о форме детали, поэтому все кривые без самопересечений и соприкосновений на плоскости, представляющие форму деталей, интерпретируются в виде ребер графа, представляющего гомеоморфный образ раскройного плана, а все точки пересечений и соприкосновений представляются в виде вершин этого графа.

Гомеоморфным образом раскройного плана является плоский граф G с внешней гранью f_0 на плоскости S. Для любой части J графа G (т.е. $J \subseteq \subseteq G$) обозначим через $\mathsf{Int}\,(J)$ теоретико-множественное объединение его внутренних граней (объединение всех связных компонент $S \setminus J$, не содержащих внешней грани). Если J считать пройденной частью маршрута режущего инструмента (очевидно, что J – плоский граф), то $\mathsf{Int}(J)$ интерпретируется как отрезанная от листа часть. Множества вершин, ребер и граней графа J будем обозначать через $V(J)$, $E(J)$ и $F(J)$ соответственно.

Топологическое представление плоского графа G на плоскости S с точностью до гомеоморфизма определяется заданием для каждого ребра $e \in E(G)$ следующих функций [2, 3]: $v_k(e)$, $k = 1, 2,$ — вершины, инцидентные ребру e;

$l_k(e)$, $k = 1, 2$, — ребра, полученные вращением ребра e против часовой стрелки вокруг вершины $v_k(e)$; $r_k(e)$, $k = 1, 2$, — ребра, полученные вращением ребра e по часовой стрелке вокруг вершины $v_k(e)$; $f_k(e)$ — грань, находящаяся справа при движении по ребру e от вершины $v_k(e)$ к вершине $v_{3-k}(e)$, $k = 1, 2$.

Таким образом, используя известные координаты прообразов вершин графа G и размещения фрагментов раскройного плана, являющихся прообразами ребер графа G, любой маршрут в графе G можно интерпретировать как траекторию режущего инструмента.

2. Представление исходных данных в программе

Как правило, раскройный план содержит большие группы однотипных деталей, а для описания размещения детали на раскройном плане достаточно указания величин (x, y, φ), где (x, y) — координаты базовой точки этой детали (обычно это начало координат, к которым привязаны координаты остальных точек), φ – угол поворота детали вокруг ее базовой точки. Поэтому разумно иметь базу данных типовых деталей.

Основными примитивными элементами траекторий маршрута резки являются отрезки прямых и дуги окружностей. Целая окружность представляется как объединение двух дуг с центральными углами φ и $2\pi - \varphi$, $\varphi > 0$. Для идентификации таких примитивов достаточно указать координаты (x_1, y_1) и (x_2, y_2) крайних точек v_1 и v_2 соответственно, а также $\mathrm{tg}(\varphi/4)$, где φ — центральный угол дуги (v_1, v_2) окружности, проходящей от точки v_1 до точки v_2 (очевидно, что для отрезка можно считать $\mathrm{tg}(\varphi/4) = 0$). С формальной точки зрения плоская деталь представляет часть плоскости, ограниченной внешней границей и внутренними границами по числу дыр. Каждая граница представляет замкнутый контур, состоящий из такой последовательности примитивов, что начало следующего совпадает с концом предыдущего.

Для представления данных о детали используется формат JSON [14]. На рис. 1 представлены примеры деталей и их описания. Описание каждой детали содержит ее имя в ключевом поле `partid` и список контуров в поле `paths`. Каждый контур представляет из себя трехэлементный массив, состоящий из двух координат концов и значения $\mathrm{tg}(\varphi/4)$. Декартовы координаты конечной точки одного примитива являются координатами начальной точки следующего. Очевидно, что такое представление декартовых координат позволяет легко определять замкнутые контуры (начальные координаты первого примитива совпадают с конечными координатами последнего). Примеры описаний деталей, состоящих из нескольких контуров, число которых больше единицы, приведено на рис. 1,*а* и 1,*д*.

Первым объектом при таком представлении данных будет определение листа, содержащего раскройный план, а все последующие объекты — описания деталей, координаты их опорной точки и ориентция.

Пример раскройного плана с совмещенными границами контуров деталей представлен на рис. 2,*а*.

Для определения последовательности резки фрагментов раскройного плана не используется информация о форме детали, поэтому все кривые без са-

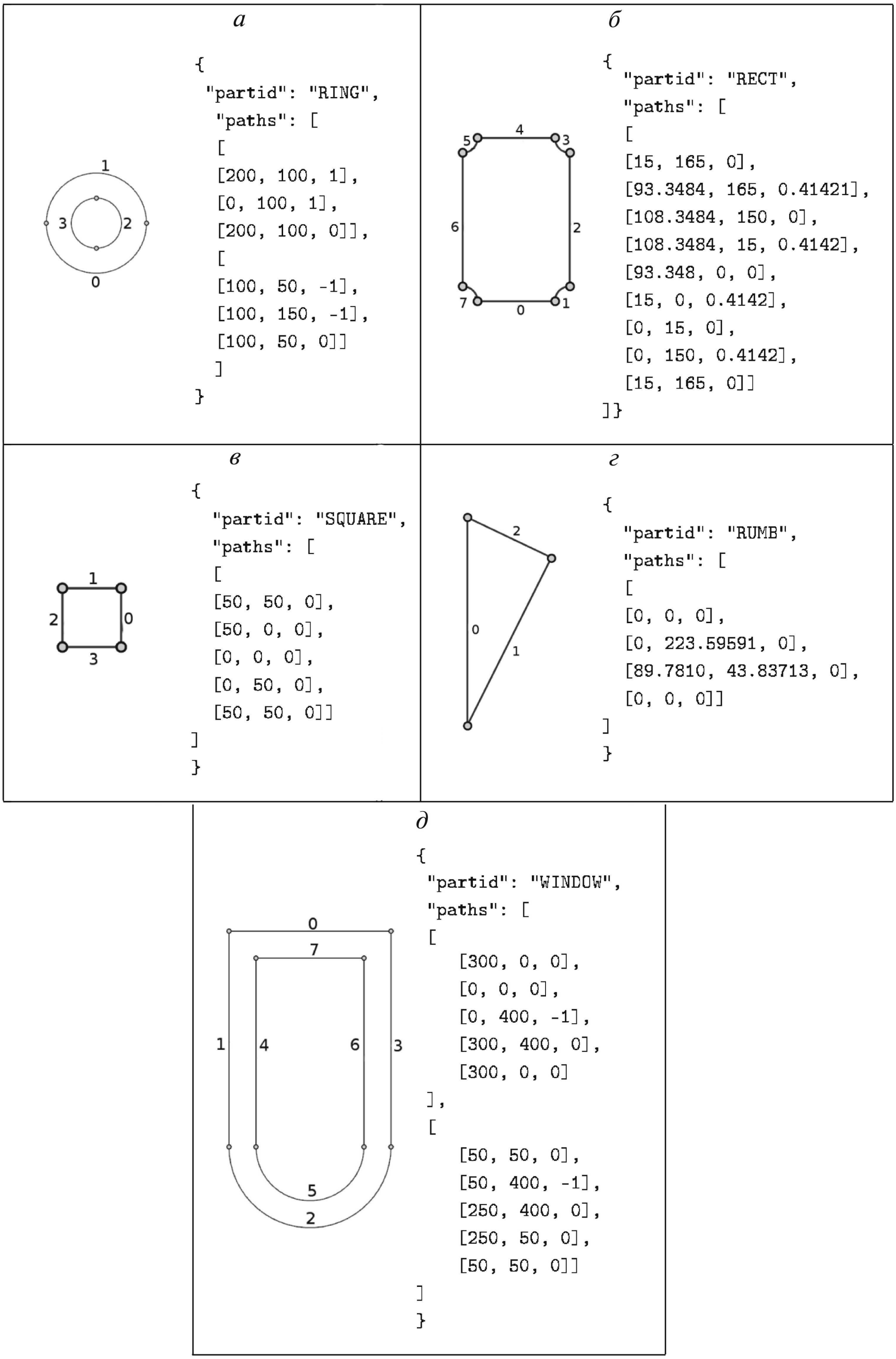
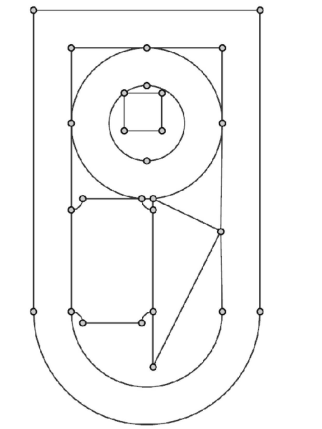


Рис. 1. Примеры и описания деталей, используемых в дальнейшем изложении: *а* – кольцо; *б* – прямоугольник; *в* – квадрат; *г* – треугольник; *д* – окно.

а



б

```
[{
  "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]}
]
```

Рис. 2. Пример: *а* — раскройный план с совмещенными границами; *б* — JSON-код раскройного плана.

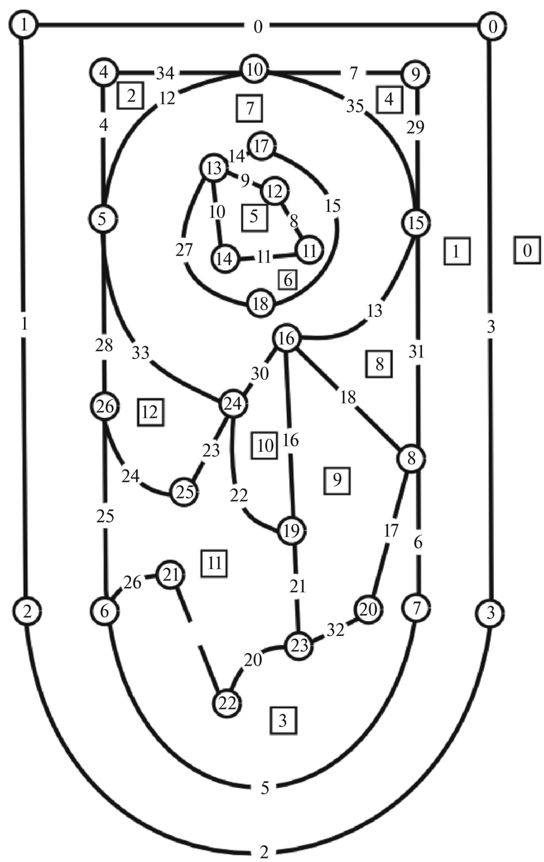


Рис. 3. Пример: гомеоморфный образ раскройного плана с указанием номеров вершин, ребер и граней.

мопересечений и соприкосновений в описании формы деталей можно интерпретировать как ребра графа, а все точки пересечений и соприкосновений — как вершины графа. Для получения представления плоского графа, позволяющего восстановить с точностью до гомеоморфизма исходный раскройный план, необходимо и достаточно в каждой вершине зафиксировать циклический порядок на множестве инцидентных ей ребер. На рис. 3 представлен гомеоморфный образ раскройного плана, изображенного на рис. 2,*а*. Структуры данных, используемые для представления гомеоморфного образа, должны включать всю необходимую информацию для эффективной работы алгоритмов маршрутизации и интерпретации построенных маршрутов [15, 16]. На листинге 1 представлены структуры данных для представления гомеоморфного образа раскройного плана в виде плоского графа.

Листинг 1. Структуры данных для гомеоморфного образа раскройного плана

```
struct  Vert {// Вершина графа
        double x, y; // координаты вершины
        int sv; // номер компоненты связности содержащей вершину
        int rank; // ранг вершины
        double dpth; // глубина вершины (скорректированный ранг)
        int deg; // степень вершины
        int mark; // метка для поиска в ширину в алгоритмах
        };
struct  Edge { // ребро графа
        int rank; // ранг ребра
        int dpth; // глубина ребра (скорректированный ранг)
        int v1, v2; // номера инцидентных вершин в контейнере
                     vector<Vert> V
        double x_0, y_0, r; // координаты центра дуги и ее радиус
        double v1_ang, v2_ang; // центральные углы возможных дуг
        double v1_ta4, v2_ta4; // тангенсы четвертей соответствующих дуг
        int l1,l2,r1,r2; // номера возможных смежных ребер
                     в vector<Edge> E
        int sv; // номер компоненты связности в контейнере
                     vector<Connect> Sv
        int cycle1, cycle2; // номера контуров в контейнере
                     vector<Cycle> C
        int f1, f2; // номера граней, инцидентных ребру,
                     в vector<Face> F
        int mark; // метка для поиска в ширину в алгоритмах
        };
struct  Face { // Грань графа
        int number; // номер грани (детали, отверстия, дополнительной)
        int dpth; // глубина грани
        };
struct  Connect { // Компонента связности
        int v; // одна из вершин компоненты (номер в vector<Vert> V)
        int outcycle; // номер охватывающего контура в vector<Cycle> C
        int up; // смежная охватывающая компонента
                     (номер в vector<Connect> Sv)
        int rank; // ранг компоненты
        Connect ():v(-1),outcycle(-1),up(-1),rank(-1){}; // конструктор
};
```

```
struct  Cycle { // контур внешней или внутренней грани графа
        int v; // номер вершины контура в контейнере vector<Vert> V
        int e; // номер ребра в контейнере vector<Edge> E, инцидентного v
        int f; // номер грани в контейнере vector<Face>
        vector<int> vertexes; // отсортированные в порядке обхода вершины
        vector<int> edges; // ребра цикла, отсортированные в порядке обхода
        Cycle (int vtx, int edg) : v(vtx), e(edg), f(-1) { }; // конструктор
        Cycle ( ) : v(-1), e(-1), f(-1) {}; // конструктор по умолчанию
};
```

Структура `Vert` содержит поля с указанием декартовых координат соответствующей точки на раскройном плане и ряд вспомогательных полей. Эти данные необходимы для интерпретатора маршрута и при заполнении полей структуры `Edge`. Поля структуры `Edge` содержит номера `v1`, `v2` инцидентных вершин, номера `f1`, `f2` инцидентных граней, номера `l1`, `l2`, `r1`, `r2` соседних в циклическом порядке ребер, значения `v1_ta4`, `v2_ta4` величины $\mathrm{tg}(\varphi/4)$, где φ – центральный угол дуг (v_1, v_2) и (v_2, v_1) соответственно, а также вспомогательные величины. Структура `Face` ставит в соответствие объекты на раскройном плане с гранями его гомеоморфного образа. Структура `Connect` используется для идентификации компонент связности гомеоморфного образа раскройного плана. Структура `Cycle` используется для идентификации контуров, являющихся границами граней. Контейнеры для приведенных структур, другие вспомогательные данные, все используемые методы их заполнения и алгоритмы маршрутизации инкапсулированы в класс `DataHolder` [15–17].

3. Маршрутизация в связных графах

Использование плоского графа в качестве гомеоморфного образа модели раскройного плана позволяет формализовать технологические ограничения на порядок вырезания фрагментов плана резки: во-первых, граф G содержит образы всех возможных элементов траектории инструмента; во-вторых, маршрут резки должен удовлетворять условию упорядоченного охватывания, т.е. отрезанная от листа часть не должна требовать дополнительных разрезаний [2]; в-третьих, должны отсутствовать самопересечения траектории резки [3].

Допустимый маршрут формализуется как упорядоченная последовательность OE-цепей, покрывающая граф [2, определения 4–5].

Определение OE-покрытия вполне конструктивно, доказательством этого факта является эффективность алгоритмов, рассмотренных в [2]. Если связный граф G не является эйлеровым, то он содержит $2k$, $k \geqslant 1$, вершин нечетной степени. В этом случае OE-маршрут состоит из k реберно-непересекающихся цепей. Задача построения такого маршрута решается алгоритмом `OE-Router` [18]. При этом в построенном маршруте длина холостых переходов (т.е. переходов между концом текущей цепи и началом следующей цепи) может не быть оптимальной. Если плоский граф G, представляющий образ раскройного плана, не содержит мостов (т.е. ребер, инцидентных одной грани), то возможно построить OE-маршрут, в котором ребра произвольного

Таблица 1. Алгоритмы построения OE-маршрутов

Название алгоритма	Вычислительная сложность
Eulerian OE-cycle (рекурсивный алгоритм) [2]	$O(\|V\|^2)$
Eulerian OE-cycle (алгоритм OE-Cycle) [2]	$O(\|E\| \cdot \log_2 \|V\|)$
OE-Postman Route (алгоритм CPP_OE)	$O(\|E\| \cdot \|V\|)$
OE-Router [18]	$O(\|E\| \cdot \log_2 \|V\|)$
M-OE-Router [2]	$O(\|V\|^2)$

паросочетания M на подмножестве $V_{odd} \subset V(G)$ множества вершин нечетной степени и только они соответствуют холостым движениям. Выбор кратчайшего паросочетания M позволяет найти маршрут с минимальной длиной холостых переходов. Отметим, что связные плоские графы, являющиеся образами раскройных планов, как правило, не содержат мостов. Поэтому если M является кратчайшим паросочетанием, то алгоритм `M-OE-Router` строит маршрут с минимальной длиной холостых переходов.

Если граф G, представляющий гомеоморфный образ раскройного плана, связен и не содержит мостов, то алгоритм `M-OE-Router` точно решает задачу, но требует определения кратчайшего паросочетания. Алгоритм `OE-Router` решает задачу для любого графа G, используя жадную стратегию выбора холостого хода. Полный перечень алгоритмов построения OE-маршрутов для связных графов приведен в табл. 1.

4. NOE-маршрутизация

Задача построения эффективных алгоритмов нахождения непересекающихся цепей в плоских графах является открытой: некоторые попытки решить данную задачу были предприняты в [19]. В [20] предложено решение задачи для плоского связного 4-регулярного графа.

Определение 1. *Эйлеров цикл C в плоском графе G называется самонепересекающимся, если он гомеоморфен циклическому графу $\tilde{G}$, представляющему плоскую жорданову кривую без самопересечений. Циклический граф $\tilde{G}$ может быть получен из графа G с помощью применения $O(|E(G)|)$ операций расщепления вершин.*

Общим случаем является решение задачи построения самонепересекающейся OE-цепи (или NOE-цепи, non-intersecting OE-trail) [21].

Определение 2. *Будем говорить, что цепь является NOE-цепью, если она одновременно является OE-цепью и самонепересекающейся цепью.*

Определение 3. *Систему переходов* [22] *цепи, соответствующую самонепересекающейся цепи, будем называть системой непересекающихся переходов.*

Доказательство того факта, что для системы переходов, соответствующей самонепересекающемуся эйлерову циклу, существует такая начальная вершина и такое конечное ребро, смежное внешней грани, для которых построенный цикл будет OE-циклом, во многом схоже с доказательством теоремы 1 в [20] для 4-регулярного графа G, и представляет алгоритм построения NOE-цепи.

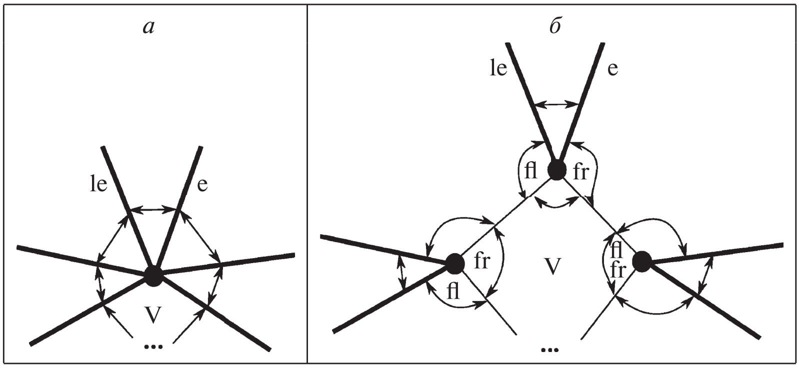


Рис. 4. *а* — Исходные указатели на соседние ребра в расщепляемой вершине. *б* — Расщепление вершины (жирными линиями показаны ребра графа G, тонкими линиями – дополнительные (фиктивные) ребра) и модификация указателей в соответствии с расщеплением.

Для построения самонепересекающейся эйлеровой OE-цепи (или цикла) в плоском эйлеровом графе (в дальнейшем эту цепь будем называть NOE-цепью (non-intersecting OE-chain)), для которой не задано фиксированной системы переходов, можно поступить следующим образом [21].

На множестве вершин $V(G)$ определим булеву функцию

$$\text{Checked}(v) = \begin{cases} \texttt{true}, & \text{если вершина просмотрена;} \\ \texttt{false} & \text{в противном случае.} \end{cases}$$

При инициализации этой функции все вершины из $V(G)$ объявляются как непросмотренные.

Функция `Non-intersecting(`G`)` (алгоритм 1) расщепляет в графе G все вершины $v \in V(G)$, $\deg(v) = 2n$, $n \geqslant 3$, на n искусственных вершин степени 4 и вводит n искусственных ребер, инцидентных полученным после расщепления вершинам и образующим цикл (рис. 4). Для выполнения указанных преобразований необходимо просмотреть все функции $v_k(e)$, $k = 1, 2$, определенные для всех ребер e, и внести требуемые модификации во всю систему кодирования графа. В теле функции используется процедура `Handle` (e, $v_k(e)$, k) (алгоритм 2), которая обрабатывает каждую непросмотренную вершину графа G. Обработка заключается в расщеплении вершины $v_k(e)$ в соответствии с рис. 4,*а* и 4,*б*.

Алгоритм 1. Функция `Non-intersecting (G)`

Require: плоский эйлеров граф G;

Ensure: плоский связный 4-регулярный граф G^*;

for all ($e \in E(G)$) **do** ▷ Просмотреть все ребра графа
    $k = 1$; ▷ Обработать последовательно функции с индексом 1, а затем – 2
    while ($k \leqslant 2$) **do**
        if ($\overline{\text{Checked}(v_k(e))}$) **then** ▷ Если вершина не была обработана ранее
            Handle (e, $v_k(e)$, k); ▷ Обработать вершину
        end if

```
        k++;
    end while
end for
Return G*;
```

Алгоритм 2. Процедура Handle (e,v,k)

```
                                        ▷ Проход 1: Определение степени вершины v
e_first = e;
d = 0;                                                      ▷ Степень вершины
repeat                                                 ▷ Просмотреть все ребра
    le = l_k(e);               ▷ Найти смежное ребро, заданное функцией l_k(e)
    if (v_k(le) ≠ v) then
        REPLACE(le);
    end if
    e = le;                          ▷ Учесть текущее ребро при подсчете степени
    d = d + 1;                                         ▷ и перейти к следующему
until (e = e_first);
                          ▷ Проход 2: Расщепление вершин, степень которых выше 4
if (d > 4) then
    e = e_first; le = l_k (e); fl=new EDGE; fle = fl; e_first = e; e_next = l_k (le);
    repeat
        e = e_next; le = l_k (e); fr = fl; fl=new EDGE; e_next = l_k (le);
        Pointers(e, le, fr, fl);                ▷ Расставить указатели для ребер
    until (l_k(le) = e_first);
    Pointers(e_first, l_k(e_first), fle, fe);   ▷ Расставить указатели для ребер
end if
```

Введенные процедурой `Handle` n искусственных вершин и n искусственных ребер, инцидентных этим вершинам, образуют цикл. В результате обработки всех вершин графа G получим модифицированный граф G^*, являющийся плоским связным 4-регулярным графом. Для G^* можно применить алгоритм `AOE-TRAIL()`, который построит в нем AOE-цепь T^*. Если затем в T^* все искусственные ребра и инцидентные им вершины, полученные при расщеплении вершины v, заменить на v, то получим NOE-цепь T в исходном графе G. Полученная после удаления ребер цепь будет принадлежать классу OE, так как процедура удаления ребер не нарушает порядка следования оставшихся ребер в цепи, что исключает появление цикла, охватывающего еще непройденные ребра.

5. Маршрутизация для несвязных графов

Если раскройный план содержит детали с отверстиями, а также другие детали, расположенные в этих отверстиях, то плоский граф $G = (V(G), F(G), E(G))$, представляющий гомеоморфный образ раскройного плана, оказывается несвязным. Поскольку вырезанные фрагменты содержат прообразы охваченных граней графа, то требования к маршруту резки, гарантирующие выполнение OE-ограничения, можно формализовать в терми-

Таблица 2. Алгоритмы построения OE-маршрутов для несвязных графов

Название алгоритма	Вычислительная сложность
MultiComponent	$O(\|E\| \cdot \log_2 \|V\|)$
Bridging	$O(\|E\| \cdot \log_2 \|V\|)$
DoubleBridging	$O(\|V\|^2)$
FaceCutting	$O(\|E\| \cdot \log_2 \|V\|)$

нах графа $G' = (F(G), V(G), E(G))$, двойственного графу G [2]: необходимо, чтобы порядок обхода граней графа G (т.е. вершин графа G') являлся расширением отношения частичного порядка $\prec$:

$$(f_i \prec f_j) \Leftrightarrow \Big(f_j \text{ принадлежит кратчайшей цепи } T_{G'}^{f_0} \text{ между } f_i \text{ и } f_0\Big),$$

где f_0 — внешняя (бесконечная) грань плоского графа G. Перечень алгоритмов построения OE-маршрутов для несвязных графов приведен в табл. 2. Для построения OE-покрытия в несвязаном графе реализованы следующие подходы: (1) определение допустимого обхода компонент связности графа G; (2) пополнение множества ребер $E(G)$ до множества $E(\tilde{G})$, где $\tilde{G}$ – плоский связный граф.

Первый подход реализуется алгоритмом `MultiComponent`, в котором нахождение искомого OE-маршрута заключается в независимом построении OE-маршрутов для каждой компоненты связности и последующем их объединении в результирующий маршрут в порядке уменьшения рангов компонент связности. Второй подход реализуется алгоритмами `Bridging`, `DoubleBridging` и `FaceCutting` и основан на добавлении мостов в *разделяющих гранях*.

Определение 4 [18]. *Грань $f \in F(G)$ называется разделяющей, если граф $G' \setminus \{f\}$ – несвязный.*

Пусть граф $\tilde{G}$ получен из графа G добавлением между компонентами связности минимального по мощности и по длине множества $\tilde{E}$ мостов, принадлежащих разделяющим граням. Очевидно, что такие ребра в графе G' будут являться ребрами остовного дерева минимального веса. Полученный таким образом граф $\tilde{G}$ является плоским связным графом, для которого можно построить OE-маршрут $M(\tilde{G})$ с помощью алгоритма `OE-Router` [18].

При этом OE-маршрут $M(G)$ строится по маршруту $M(\tilde{G})$ удалением ребер введенного множества $\tilde{E}$. При этом получим множество цепей, представляющих OE-покрытие исходного несвязного графа. Указанный подход реализует алгоритм `Bridging`.

Чтобы обеспечить возможность применения алгоритма `M-OE-Router`, достаточно дополнить граф G до графа $\tilde{\tilde{G}}$ включением множества ребер $\tilde{\tilde{E}} = \cup_{e \in \tilde{E}} \{e_1 = e, e_2 = e\}$, т.е. включением двух дубликатов для каждого ребра $e \in \tilde{E}$. Указанный подход в совокупности с алгоритмом `M-OE-Router` реализует алгоритм `DoubleBridging`.

Теорема 1. Если в каждой компоненте связности G_k графа G степени вершин, инцидентных разделяющим граням графа G, четны, то маршрут с *минимальной длиной дополнительных построений реализуется алгоритмом* `DoubleBridging`.

Доказательство. Очевидно, что обход каждой компоненты связности должен заканчиваться на внешней границе. Если предположить, что обход компоненты связности начинается из вершины, не принадлежащей внешней границе, то завершится данный фрагмент OE-покрытия в вершине нечетной степени, не принадлежащей границе. Поскольку на границе нет вершин нечетной степени, то в соответствии с определением OE-маршрута часть графа, содержащая внешнюю границу, останется непройденной. В оптимальном решении (полученном алгоритмом `M-OE-Router`) компоненты будут связаны парами кратных ребер. Причем, суммарный вес всех связывающих ребер минимален. Теорема доказана.

Очевидно, что длина введенных мостов в алгоритме `DoubleBridging` не меньше удвоенной длины кратчайшего остова в разделяющей грани.

Еще одним способом получения связного графа без мостов является расщепление вершин, инцидентных разделяющей грани, с помощью гамильтонова цикла (алгоритм `FaceCutting`). Такой подход представляется наиболее целесообразным, поскольку, во-первых, на практике размерность задачи коммивояжера сопоставима с оценками степеней соответствующих разделяющих граней (т.е. достаточно низкая); во-вторых, в соответствии с метрикой графа G даже приближенный полиномиальный алгоритм Кристофидеса [23] для задачи коммивояжера строит гамильтонов цикл в разделяющей грани с длиной, не превышающей удвоенной длины кратчайшего остова.

6. Пример

В табл. 3 (левый столбец) приведено оптимальное NOE-покрытие раскройного плана рис. 2 упорядоченной последовательностью OE-цепей. Первая цепь проходит через квадрат и внутреннюю границу кольца. Вторая — по правой стороне прямоугольника, по внешней границе кольца, и завершается вырезание треугольником и прямоугольником. Третья цепь завершает внутренний контур окна. Четвертая проходит по внешнему контуру окна. Код траектории инструмента в терминах JSON представлен в правом столбце табл. 3. Каждая отдельная цепь соответствует определенной части траектории непре-

Таблица 3. Оптимальное NOE-покрытие раскройного плана рис. 2 упорядоченной последовательностью OE-цепей

<table>
<tr><th>Цепь</th><th>JSON-код цепи</th></tr>
<tr><td>1</td><td>2</td></tr>
<tr><td>chain 1:
v17 e14 v13 e9 v12 e8 v11 e11 v14 e10 v13 e27 v18 e15 v17</td><td><pre>{   "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]
]},</pre></td></tr>
</table>

Таблица 3 (окончание)

1	2
chain 2: v23 e21 v19 e16 v16 e30 v24 e33 v5 e12 v10 e35 v15 e13 v16 e18 v8 e17 v20 e32 v23 e20 v22 e19 v21 e26 v6 e25 v26	`{ "partid": "chain_1", "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] ] }`

рывной резки. Она представляет собой последовательность трехэлементных массивов, содержащих координаты текущей начальной точки примитива, а также значение для нее. При вырезании деталей в соответствии с найденной последовательностью цепей соблюдаются все технологические ограничения.

7. Заключение

Технология, допускающая совмещение границ вырезаемых деталей — современная ресурсосберегающая технология резки. Известны алгоритмы маршрутизации, когда на маршрут движения режущего инструмента одновременно наложены следующие технологические ограничения: (1) отрезанная от листа часть не требует дополнительных разрезаний, (2) отсутствуют самопересечения траектории резки.

В статье дано решение проблемы эффективной программной реализации этих алгоритмов: представлены результаты авторов, использованные при разработке функциональных элементов комплекса программ автоматизированной системы технологической подготовки процессов раскроя листового материала.

СПИСОК ЛИТЕРАТУРЫ

1. *Dewil R., Vansteenwegen P., Cattrysse D.* A Review of Cutting Path Algorithms for Laser Cutters // Int. J. Adv. Manuf. Technol. 2016. V. 87. P. 1865–1884. https://doi.org/10.1007/s00170-016-8609-1
2. *Макаровских Т.А., Панюков А.В., Савицкий Е.А.* Математические модели и алгоритмы маршрутизации для САПР технологической подготовки процессов раскроя // АиТ. 2017. № 5. С. 123–140. https://doi.org/10.1134/S0005117917050095

   Makarovskikh T.A., Panyukov A.V., Savitsky E.A. Mathematical Models and Routing Algorithms for CAD Technological Preparation of Cutting Processes // Autom. Remote Control. 2017. V. 78. No. 4. P. 868–882.
3. *Makarovskikh T., Panyukov A.* The Cutter Trajectory Avoiding Intersections of Cuts // IFAC-PapersOnLine. 2017. V. 50. Iss. 1. P. 2284–2289.

   https://doi.org/10.1016/j.ifacol.2017.08.226
4. *Li X., Liu Zh., Wang F., Yi B., 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., Panyukov A.* Development of Routing Methods for Cutting out Details // CEUR Workshop Proc. 2018. V. 2098. P. 249–263.

   http://ceur-ws.org/Vol-2098/paper22.pdf
6. *Dewil R., Vansteenwegen P., Cattrysse D., Laguna M., Vossen T.* An Improvement Heuristic Framework for the Laser Cutting Tool Path Problem // Int. J. Prod. Rts. 2015. V. 53. Iss. 6. P. 1761–1776. https://doi.org/10.1080/00207543.2014.959268.
7. *Petunin A., Stylios C.* Optimization Models of Tool Path Problem for CNC Sheet Metal Cutting Machines // IFAC-PapersOnLine. 2016. V. 49. P. 23–28.

   https://doi.org/10.1016/j.ifacol.2016.07.544
8. *Petunin A., Chentsov A.G., Chentsov P.A.* About Routing in the Sheet Cutting // Bulletin of the South Ural State University, Series: Mathematical Modelling, Programming and Computer Software. 2017. V. 10 (3). P. 25–39.

   https://doi.org/10.14529/mmp170303
9. *Chentsov A.G., Grigoryev A.M., Chentsov A.A.* Solving a Routing Problem with the Aid of an Independent Computations Scheme // Bulletin of the South Ural State University, Series: Mathematical Modelling, Programming and Computer Software. 2018. V. 11 (1). P. 60–74. https://doi.org/10.14529/mmp180106
10. *Khachay M., Neznakhina K.* Towards Tractability of the Euclidean Generalized Travelling Salesman Problem in Grid Clusters Defined by a Grid of Bounded Height // Communications in Comput. and Inform. Sci. 2018. V. 871. P. 68–77. https://link.springer.com/chapter/10.1007/978-3-319-93800-4_6
11. *Chentsov A., Khachay M., Khachay D.* Linear Time Algorithm for Precedence Constrained Asymmetric Generalized Traveling Salesman Problem // IFAC-PapersOnLine. 2016. V. 49. P. 651–655. https://doi.org/10.1016/j.ifacol.2016.07.767
12. *Hoeft J., Palekar U.* Heuristics for the Plate-cutting Traveling Salesman Problem // IIE Transactions. 1997. V. 29. P. 719–731. https://doi.org/10.1023/A:1018582320737
13. *Dewil R., Vansteenwegen P., Cattrysse D.* Construction heuristics for generating tool paths for laser cutters // Int. J. Prod. Rts. 2014. V. 52 (20). P. 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. *Макаровских Т.А., Панюков А.В., Савицкий Е.А.* Программное обеспечение для задачи построения траектории движения режущего инструмента // Тр. XVIII-й Междунар. молодежной конф. "Системы проектирования, технологической подготовки производства и управления этапами жизненного цикла промышленного продукта (CAD/CAM/PDM-2018". 2018. С. 172–176.
16. *Макаровских Т.А., Панюков А.В., Савицкий Е.А.* Задача построения движения режущего инструмента: программная реализация // Тр. XIII Всеросс. совещания по проблемам управления (ВСПУ-2019). Под общей редакцией Д.А. Новикова. 2019. С. 2650–2654.
17. *Makarovskikh T.A., Panyukov A.V., Savitskiy E.A.* Software Development for Cutting Tool Routing Problems // Procedia Manufacturing. 2019. V. 29. P. 567–574. https://doi.org/10.1016/j.promfg.2019.02.123
18. *Makarovskikh T.A., Panyukov A.V., Savitskiy E.A.* Mathematical Models and Routing Algorithms for Economical Cutting Tool Paths // Int. J. Prod. Rts. 2018. V. 56 (3). P. 1171–1188. https://doi.org/10.1080/00207543.2017.1401746.
19. *Manber U., Bent S.W.* On Non-intersecting Eulerian Circuits // Discrete Applied Mathematics. 1987. V. 18. P. 87–94. https://doi.org/10.1016/0166-218X(87)90045-X
20. *Макаровских Т.А.* Программное обеспечение для построения A-цепей с упорядоченным охватыванием в плоском связном 4-регулярном графе // Вестн. Южно-Уральского гос. ун-та. Сер. Вычислительная математика и информатика. 2019. Т. 8. № 1. С. 36–53. https://doi.org/10.14529/cmse190103
21. *Макаровских Т.А.* Построение самонепересекающихся OE-маршрутов в плоском эйлеровом графе // Вестн. Южно-Уральского гос. ун-та. Сер.: Вычислительная математика и информатика. 2019. Т. 8. № 4. С. 30–42.
https://doi.org/10.14529/cmse190403
22. *Макаровских Т.А.* О числе OE-цепей для заданной системы переходов // Вестник Южно-Уральского гос. ун-та. Сер.: Математика. Механика. Физика. 2016. Т. 8. № 1. С. 5–12. https://doi.org/10.14529/mmph160101
23. *Garey M.R., Johnson D.S.* Computer and intractability: a guide to the theory of NP-completeness, 1979.

Статья представлена к публикации членом редколлегии А.А. Лазаревым.

Поступила в редакцию 26.06.2019

После доработки 10.04.2020

Принята к публикации 09.07.2020