Traversing the Machining Graph

Danny Z. Chen[1⋆], Rudolf Fleischer[2⋆⋆], Jian Li[3], Haitao Wang[1], and Hong Zhu[4⋆⋆⋆]

[1] Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA. E-mail: [dchen,hwang6]@nd.edu

[2] School of Computer Science, IIPL, Fudan University, Shanghai, China. E-mail: rudolf@fudan.edu.cn

[3] Department of Computer Science, University of Maryland, College Park, USA. E-mail: lijian@cs.umd.edu

[4] Software Engineering Institute, East China Normal University, Shanghai, China. E-mail: hzhu@sei.ecnu.edu.cn

Abstract. Zigzag pocket machining (or 2D-milling) plays an important role in the manufacturing industry. The objective is to minimize the number of tool retractions in the zigzag machining path for a given pocket (i.e., a planar domain). We give an optimal linear time dynamic programming algorithm for simply connected pockets, and a linear plus $O(1)^{O(h)}$ time optimal algorithm for pockets with h holes. If the dual graph of the zigzag line segment partition of the given pocket is a partial k-tree of bounded degree or a k-outerplanar graph, for a fixed k, we solve the problem optimally in linear time. Finally, we propose a linear time algorithm for finding a machining path for a pocket with h holes using at most $OPT + \epsilon h$ retractions, where OPT is the smallest possible number of retractions and $\epsilon > 0$ is any constant.

1 Introduction

2D-milling, or *zigzag pocket machining (*ZPM*)*, is an important problem in the manufacturing industry [17, 28, 31, 33]. Either a workpiece is translated under a spinning milling tool, or a cutter is moved across the workpiece. We model the workpiece as an arbitrary planar domain, called a *pocket*. The actual shape of the pocket is not really important for us, so we may just think of a polygon (possibly containing holes). Usually, the cutter (or the workpiece moving below the milling tool) can only cut while moving along a fixed direction, for example, parallel to the x-axis (but it can cut moving in both directions along a line). When it reaches the boundary of the workpiece, it must either move along the boundary to another line parallel to the x-axis (usually the lines are assumed to be equally

⋆ The research of D.Z. Chen was supported in part by a grant from the Shanghai Key Laboratory of Intelligent Information Processing, Fudan University, Shanghai, China, and was partially done while Chen visited the Shanghai Key Laboratory of Intelligent Information Processing at Fudan University. This research was also supported in part by the US National Science Foundation under Grants CCF-0515203 and CCF-0916606.

⋆⋆ This work was supported by grants from the National Natural Science Fund China (nos. 60573025 and 60973026), the Shanghai Leading Academic Discipline Project (no. B114), and the Shanghai Committee of Science and Technology (China grants 08DZ2271800 and 09DZ2272800).

⋆⋆⋆ This work was supported by grants from the National Natural Science Fund China (nos. 60496321 and 60373021) and the Shanghai Science and Technology Development Fund (no. 03JC14014).

spaced with a *zigzag step-over distance* $\delta > 0$, but this technical requirement is not important for us), creating a zigzag movement pattern, or it must jump to another part of the workpiece. A jump requires the cutter to be *retracted*. Since retractions are time-consuming and may have other disadvantages due to technological constraints, the goal is to find a machining path minimizing the number of retractions, under the additional constraint that the cutter must work on any interior part of the workpiece exactly once while it cannot traverse any part of the boundary more than once.

Considerable work has been done on ZPM, see [33] for an extensive survey of the current state-of-the-art. A few algorithms were given in [12, 21], but they did not attempt to minimize the number of tool retractions or to optimize any other criteria. Some heuristic methods were used to reduce the number of tool retractions for general pockets [17, 18]. For pockets with holes, ZPM was shown to be NP-hard by Arkin *et al.* [3] by a reduction from the *Planar 3-Satisfiability Problem* [15, 22]. They also presented a linear time approximation algorithm with at most $5\,OPT+6h$ tool retractions based on a graph model, called the *machining graph*, where OPT is the smallest possible number of retractions and h is the number of holes in the pocket. Tang *et al.* [29] studied a special case when the step-over distance is small with respect to the geometry of the pocket. However, no quantitative measure was given on how small the step-over distance needs to be in order for the solution to be optimal (see [19]).

Tang and Joneja [30] presented a linear time approximation algorithm for ZPM with at most $OPT + h + N_r$ retractions, where N_r is the number of the so-called *reducible blocks* of the pocket. Although the coefficients of OPT and h are both one, the third parameter N_r can be quite large, depending on the shape of the given pocket, the step-over distance, the inclination of the reference line, etc. For example, N_r will usually grow with a larger step-over distance. Thus, the results in [3] and in [30] are not directly comparable. Some practical implications and applications of ZPM were discussed in [17, 29].

It is worth pointing out that although ZPM for pockets with holes is NP-hard [3], for the important case of simply connected pockets (i.e., without holes), only approximation algorithms were previously known [3, 30]. In fact, it has been an open problem to decide whether the simply connected pocket case is NP-hard.

Other optimization criteria for ZPM have also been considered. For example, multi-tool retraction minimization was studied by Arya *et al.* [7]. The problem of minimizing the total length of the tool path was studied by Arkin *et al.* [2]. Algorithms for determining a cutting direction in order to minimize the tool retraction length were given by Kim *et al.* [20]. Some algorithms were designed to optimize the tool path length and the number of tool retractions [1, 27]. A survey of the pocketing requirements can be found in [16]. Tang [28] summarized some recent progress on developing efficient algorithms for several geometric optimization problems in manufacturing.

In this paper, we present significantly improved algorithms for ZPM. Our techniques are different from the previous ones in [3, 30], and our algorithms are superior in theoretical performance. Our algorithms may also be interesting for practical applications (e.g., for k-outerplanar dual graphs, with a small k). More specifically, our results are:

(1) We give an optimal linear time dynamic programming algorithm for simply connected pockets, settling the open question on the complexity of this case.
(2) For pockets with holes, we introduce the concept of the *boundary graph* to remodel the problem, and generalize our dynamic programming approach to optimally solve in time $O(n)$ the cases when the dual planar graph of the pocket is a partial k-tree of bounded degree or a k-outerplanar graph, for any fixed k. Here, we use the algorithmic framework for partial k-trees by Arnborg *et al.* [6].
(3) Combining the ingredients of (1) and (2) with Baker's shifting technique for outerplanar graphs [8], we develop a linear time approximation algorithm for finding a machining path with at most $OPT + \epsilon h$ retractions for a pocket with h holes, for any constant $\epsilon > 0$. This substantially improves the previous approximation solutions [3, 30], and in fact gives a best possible approximation if the output quality is measured in terms of the number of holes.
(4) We give an exact dynamic programming algorithm running in linear plus $O(1)^{O(h)}$ time for a pocket with h holes. This implies that, in particular, ZPM with a logarithmic number of holes is still solvable in polynomia time.

The rest of this paper is organized as follows. In Section 2, we review some definitions from [3] and state the problem formally. In Section 3, we describe an optimal linear time dynamic program for simply connected pockets. In Section 4, we propose an exact algorithm for the problem of pockets with h holes. In Sections 5 and 6, we first introduce the concept of a boundary graph and the MVPC problem, and then present exact linear time algorithms if the dual graph of a pocket with holes is a partial k-tree of bounded degree or a k-outerplanar graph. In Section 7, we present a "best possible" linear time approximation algorithm for the general problem.

2 Preliminaries

We mainly use the terminology from [3]. A *pocket* P is a compact connected planar domain bounded by a contour B. It is *simply connected* (e.g., a simple polygon) if it contains no holes, or *multiply connected* otherwise. For a pocket with h holes, B consists of $h + 1$ unconnected loops (the boundary of the outer face and the boundaries of the h holes). The edges of B can be straight line segments or any types of simple curves.

Consider an arbitrary set $\mathcal{L}$ of non-crossing line segments (it could even be curves) partitioning P into a set of regions. Each line segment connects two points

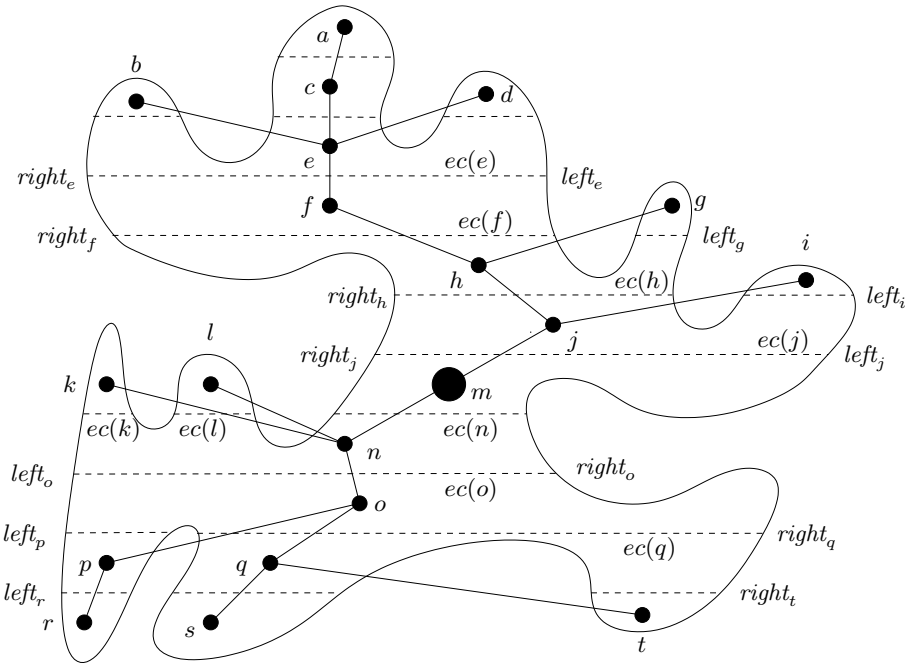

Fig. 1. A simple pocket P_G with horizontal c-edges (dashed) and its dual graph D_P (solid straight edges) rooted at m. Some c-edges are labeled by $ec(x)$, the c-edge on the boundary of $face(x)$ closest to the root. Some c-edge endpoints are labeled by $left_x$ and $right_x$, indicating the first/last encounter with edge $ec(x)$ when walking counterclockwise around B starting at the root.

on B. In 2D-milling, the line segments would be parallel and equally-spaced. Let N be the set of all endpoints of the line segments, and let n denote the size of N. On the node set N, we define the undirected *machining graph* $M_P = (N, E)$, with two types of edges:

- *compulsory* edges (*c-edges*), connecting the two endpoints of a line segment; the c-edges correspond to the line segments in $\mathcal{L}$.
- *non-compulsory* edges (*nc-edges*), connecting two neighboring nodes on B not already connected by a c-edge; the nc-edges correspond to paths on B between neighboring points in N.

Note that every node in N is incident to exactly one c-edge and at most two nc-edges. Fig. 1 shows an example of a pocket in the plane, partitioned into regions by horizontal line segments (c-edges).

Given a machining graph M_P, a *tool traversing path* (or *machining path*) is a collection $\mathcal{P}$ of simple node-disjoint paths in M_P, called *no-retraction paths*, such that every c-edge is traversed exactly once and every nc-edge is traversed at most once.

The machining tool must follow all no-retraction paths of $\mathcal{P}$. When it reaches the end of a path, it *jumps* to an unprocessed no-retraction path. This operation

is called a *retraction*. The number of retractions is one less than the number of no-retraction paths in $\mathcal{P}$. An *optimal* (or minimum) machining path $\mathcal{P}$ minimizes the number of retractions (or equivalently, the number of no-retraction paths in $\mathcal{P}$). If the pocket is multiply connected (i.e., has holes), then finding an optimal machining path is NP-hard [3].

The c-edges and nc-edges of a machining graph M_P induce a planar partition P_G of the pocket P. The faces F of P_G induce the dual graph $D_P = (F, E_d)$ of P_G, see Fig. 1.

If P has holes, D_P may be a multigraph. Note that the interior faces of D_P correspond exactly to the holes of P. Since structural properties like treewidth and k-outerplanarity are usually only defined for simple graphs, we make D_P simple by splitting each edge of D_P into two edges by adding a new node in the middle of the edge, for example at the intersection between the edge and the unique c-edge it is crossing (see Fig. 4). We call these new nodes *c-nodes*, while the original nodes (corresponding to the faces of P_G) are *f-nodes*. Note that adding c-nodes for a simple D_P would have no effect on k-outerplanarity and treewidth.

Throughout this paper, we denote by OPT the minimum number of retractions of all feasible machining paths for P, the number of holes in P by h, the number of nodes in M_P by n, and the number of edges in M_P by m. We also abuse the notation by calling D_P the *dual machining graph* of M_P.

For a graph G, we sometimes denote its node and edge sets by $V(G)$ and $E(G)$, respectively.

3 A Linear Time Algorithm for Simply Connected Pockets

In this section, we optimally solve `ZPM` for a simply connected pocket P in linear time based on dynamic programming. We first discuss some properties of an optimal traversal of M_P and then present our new algorithm. Observe that D_P is a tree if and only if P is a simply connected pocket. Fig. 1 shows an example. For a node $v \in F$, let *face*(v) denote the corresponding face in P_G.

Lemma 1 ([3]). *There exists an optimal machining path $\mathcal{P}$ such that*

1. *each no-retraction path in $\mathcal{P}$ starts and ends with a c-edge, and*
2. *no two nc-edges are traversed consecutively.* □

We will now show how to compute a machining path satisfying the conditions of Lemma 1. We treat $D_P = (F, E_d)$ as a tree T rooted at an arbitrarily chosen node *root* $\in F$. For a node $v \in F$, let T_v denote the subtree of T rooted at v, and let $P_{G/v}$ denote the portion of the partition P_G corresponding to T_v. If $v \neq$ *root*, then the boundary of *face*(v) contains a c-edge, $ec(v)$, of M_P separating *face*(v)

from the face of v's parent in T. This c-edge is closest to the root among all c-edges on the boundary of $face(v)$ and it lies on the boundary of $P_{G/v}$.

Let M_v denote the machining subgraph of $P_{G/v}$, where we require that $ec(v)$ is a c-edge of M_v. See Fig. 1 for an example. We denote the two endpoints of $ec(v)$ by $left_v$ and $right_v$, respectively, where $left_v$ is the endpoint we reach first when we walk counterclockwise along B, starting somewhere in $face(root)$.

We compute optimal machining paths for $P_{G/v}$ for all nodes v of T in a bottom-up manner, starting at the leaves. This will give us a linear time algorithm. In general, an internal v may have k children $v_1, \ldots, v_k$, where we assume that $ec(v), ec(v_1), \ldots, ec(v_k)$ appear counterclockwise along the boundary of $face(v)$. Let $bridge(v_i)$ denote the nc-edge connecting $ec(v_i)$ and $ec(v_{i+1})$, for $1 \leq i < k$.

When we deal with an internal node v we have already (bottom-up) computed the machining paths for the subtrees rooted at $v_1, \ldots, v_k$, respectively. If v is not the root, we must now extend these paths to integrate the edge $ec(v)$. There are a few cases. If $left_{v_1}$ or $right_{v_k}$ is the endpoint of a no-retraction path, we can easily extend this path to include $ec(v)$. If both these points are endpoints of two different paths, we can directly connect both paths via $ec(v)$, thus saving one retraction. If both points are not path endpoints, $ec(v)$ will form a new path by itself. Since we do not know in advance which case can yield an optimal solution for M_P, we must provide for all cases, i.e., we use dynamic programming.

We first compute the best way to combine the different solutions for the subtrees of the children. This is also done for the root of T. Let $M_{i,j} = \bigcup_{\ell=i}^{j} M_{v_\ell} \cup \bigcup_{\ell=i}^{j-1} bridge(v_\ell)$, for $1 \leq i \leq j \leq k$, be the connected portion of M_P formed by $M_{v_i}, \ldots, M_{v_j}$ and the bridges connecting the pieces. We characterize the feasible machining paths for $M_{i,j}$ into five classes.

- $P^0_{i,j}$ contains all machining paths such that no no-retraction path ends at $left_{v_i}$ or $right_{v_j}$;
- $P^l_{i,j}$ contains all paths such $left_{v_i}$ is the endpoint of some no-retraction path but no no-retraction path ends at $right_{v_j}$;
- $P^r_{i,j}$ contains all paths such $right_{v_i}$ is the endpoint of some no-retraction path but no no-retraction path ends at $left_{v_j}$;
- $P^{lr1}_{i,j}$ contains all paths such that some no-retraction path starts at $left_{v_i}$ and ends at $right_{v_j}$;
- $P^{lr2}_{i,j}$ contains all paths such that some no-retraction path ends at $left_{v_i}$ and some other no-retraction path ends at $right_{v_j}$;

Let $h^x(v_i, v_j)$, for $x \in \{0, l, r, lr1, lr2\}$ and $i \leq j$, be the minimum number of retractions among all machining paths in $P^x_{i,j}$. Let $h^x(v_i) = h^x(v_i, v_i)$ denote the minimum number of retractions among all machining paths in $P^x_{i,i}$.

We can in a natural way extend this definition to leaves. If v is a leaf of T, then there exists only one machining path for $P_{G/v}$, namely the edge $ec(v)$ itself. Thus, we can define

$$h^{lr1}(v) = 0$$

and

$$h^0(v) = h^l(v) = h^r(v) = h^{lr2}(v) = \texttt{NULL}\,,$$

where NULL means there is no machining path in this class. If a term NULL appears in an arithmetic expression, the expression has value NULL.

If v is an internal node of T and all $h^x(v_i)$ for all children $v_1, \ldots, v_k$ of v are known, then we can iteratively compute the values $h^x(v_i, v_j)$ for all pairs of indices $i < j$. Actually, we only need to compute the values $h^x(v_1, v_2), \ldots, h^x(v_1, v_k)$.

$$\begin{aligned} h^0(v_i, v_{j+1}) = \min\{\ & h^r(v_i, v_j) + h^l(v_{j+1}), && \text{we use } \textit{bridge}(v_j) \\ & h^r(v_i, v_j) + h^0(v_{j+1}) + 1, && \text{we cannot use } \textit{bridge}(v_j) \\ & h^0(v_i, v_j) + h^l(v_{j+1}) + 1, \\ & h^0(v_i, v_j) + h^0(v_{j+1}) + 1\}. \end{aligned}$$

$$\begin{aligned} h^r(v_i, v_{j+1}) = \min\{\ & h^r(v_i, v_j) + h^{lr1}(v_{j+1}), && \text{we use } \textit{bridge}(v_j) \\ & h^r(v_i, v_j) + h^r(v_{j+1}) + 1, && \text{we cannot use } \textit{bridge}(v_j) \\ & h^0(v_i, v_j) + h^{lr1}(v_{j+1}) + 1, \\ & h^0(v_i, v_j) + h^r(v_{j+1}) + 1\}. \end{aligned}$$

$$\begin{aligned} h^l(v_i, v_{j+1}) = \min\{\ & h^{lr1}(v_i, v_j) + h^l(v_{j+1}), && \text{we use } \textit{bridge}(v_j) \\ & h^{lr2}(v_i, v_j) + h^l(v_{j+1}), \\ & h^{lr1}(v_i, v_j) + h^0(v_{j+1}) + 1, && \text{we cannot use } \textit{bridge}(v_j) \\ & h^{lr2}(v_i, v_j) + h^0(v_{j+1}) + 1, \\ & h^l(v_i, v_j) + h^l(v_{j+1}) + 1, \\ & h^l(v_i, v_j) + h^0(v_{j+1}) + 1\}. \end{aligned}$$

$$h^{lr1}(v_i, v_{j+1}) = h^{lr1}(v_i, v_j) + h^{lr1}(v_{j+1}) \quad \text{we use } \textit{bridge}(v_j)$$

$$
\begin{aligned}
h^{lr2}(v_i, v_{j+1}) = \min\{\ & h^{lr2}(v_i, v_j) + h^{lr1}(v_{j+1}), && \text{we use } \textit{bridge}(v_j) \\
& h^{lr1}(v_i, v_j) + h^{r}(v_{j+1}) + 1, && \text{we cannot use } \textit{bridge}(v_j) \\
& h^{lr2}(v_i, v_j) + h^{r}(v_{j+1}) + 1, \\
& h^{l}(v_i, v_j) + h^{lr1}(v_{j+1}) + 1, \\
& h^{l}(v_i, v_j) + h^{lr2}(v_{j+1}) + 1, \\
& h^{l}(v_i, v_j) + h^{r}(v_{j+1}) + 1\} \,.
\end{aligned}
$$

We can then compute the values $h^x(v)$, for all x, as follows. If v is not the root, then

$$
\begin{aligned}
h^0(v) &= h^{lr2}(v_1, v_k) - 1 \,, \\
h^l(v) &= \min_{x \in \{r, lr1, lr2\}} \{h^x(v_1, v_k)\} \,, \\
h^r(v) &= \min_{x \in \{l, lr1, lr2\}} \{h^x(v_1, v_k)\} \,, \\
h^{lr1}(v) &= 1 + \min_{x \in \{0, l, r, lr1, lr2\}} \{h^x(v_1, v_k)\} \,, \\
h^{lr2}(v) &= \texttt{NULL} \,.
\end{aligned}
$$

At the root, we can compute the optimal value for M_P as

$$
\begin{aligned}
OPT(M_P) = \min\{\ & h^{lr2}(v_1, v_k) - 1 \,, \\
& \min\nolimits_{x \in \{0, l, r, lr1\}} \{h^x(v_1, v_k)\}\} \,.
\end{aligned}
$$

Theorem 2. *The dynamic program above computes an optimal machining path for simply connected pockets in linear time.*

Proof. We prove correctness of the computation of $h^x(v_i, v_j)$ by induction for the case $x = lr1$. The other cases are similar. Consider an optimal traversing $p^{lr1}_{i,j+1}$ of $M_{i,j+1}$ satisfying the requirements of $h^{lr1}(v_i, v_{j+1})$ and the properties of Lemma 1.

Let $p^{lr1}(v_i, v_j)$ and $p^{lr1}(v_{j+1}, v_{j+1})$ denote p^{lr1}'s restriction to $M_{i,j}$ and $M_{j+1,j+1}$, respectively. If $p^{lr1}(v_i, v_j)$ is not an optimal traversing path of $M_{i,j}$, then we can substitute it by a traversal with $h^{lr1}(v_i, v_j)$ no-retraction paths for $M_{i,j}$, resulting in fewer no-retraction paths and fewer retractions, contradicting our choice of $p^{lr1}(v_i, v_{j+1})$ as an optimal traversing path for $M_{i,j}$. A similar argument holds for $M_{j+1,j+1}$. This optimal substructure guarantees the correctness of our dynamic program.

We prove correctness of the computation of $h^x(v)$ for the cases $x = 0$ and $x = r$. The other cases are similar.

For $h^0(v)$, note that the minus one occurs because we can link the two different no-retraction paths ending at $left_{v_1}$ and $right_{v_k}$, respectively, into a single path.

For $h^r(v)$, note that a traversing path in $\mathcal{P}^r(v)$ can be obtained in one of three ways, where e denotes the nc-edge connecting an endpoint of $ec(v)$ with an endpoint of $ec(v_1)$.

1. a traversing path in $P^{lr1}(v_1, v_k)$ plus the edges e and $ec(v)$;
2. a traversing path in $P^{lr2}(v_1, v_k)$ plus the edges e and $ec(v)$;
3. a traversing path in $P^{l}(v_1, v_k)$ plus the edges e and $ec(v)$.

□

4 An Exact Algorithm for Pockets with Holes

If P has $h \geq 1$ holes, the dual machining graph D_P is not a tree. However, we can identify $O(h)$ *pivot nodes* that partition D_P into a forest of trees such that each tree is adjacent to at most two pivot nodes. The trees of the forest can be handled similarly as in Section 3. Since each tree has only a constant size interface with the pivot nodes via c-edges, we can test all possible choices for these c-edges of these nodes in $O(1)^{O(h)}$ time. This implies that the problem is still solvable in polynomial time for pockets with $h = O(\log(n + m))$ holes.

We now give the details of the algorithm which is also based on dynamic programming. First, we remove all the c-nodes by merging their two adjacent edges into a single edge. Then, we partition the graph into biconnected components, see Fig. 2 for an example. A *biconnected component* of D_P is a maximal connected induced subgraph that cannot be disconnected by removal of a single edge. If we contract each non-trivial biconnected component (consisting of at least two nodes) into a supernode, we get a tree T. We will handle this tree the same way we handled trees in Section 3, except that we need to do some additional work for the supernodes. We call the smallest subtree of T connecting all the supernodes the *backbone* of T.

The edges of D_P either appear as edges in T (if they are outside a biconnected component), or they do not appear in T (if they are within a biconnected component). We can therefore classify the edges of D_P into three classes. Edges within a biconnected component are *component edges*, edges on the backbone of T are *backbone edges*, and all other edges are *tree edges* which form trees rooted at nodes on the backbone or within a non-trivial biconnected component.

We now explain how to reduce the problem to a problem of size $O(h)$ which can be solved optimally in time $O(1)^{O(h)}$ by brute-force search. In FPT theory, this is called a *kernelization*. A node of D_P is a *pivot node* if it is adjacent to at least three non-tree edges, i.e., backbone or component edges (nodes a, b, c, d, e, f, g in Fig. 2). The pivot nodes divide the subgraph induced by component and backbone edges into simple paths, called *chains*, that start and end at some pivot node (possibly the same node, like the chains at nodes b and d in Fig. 3). Fig. 3 shows

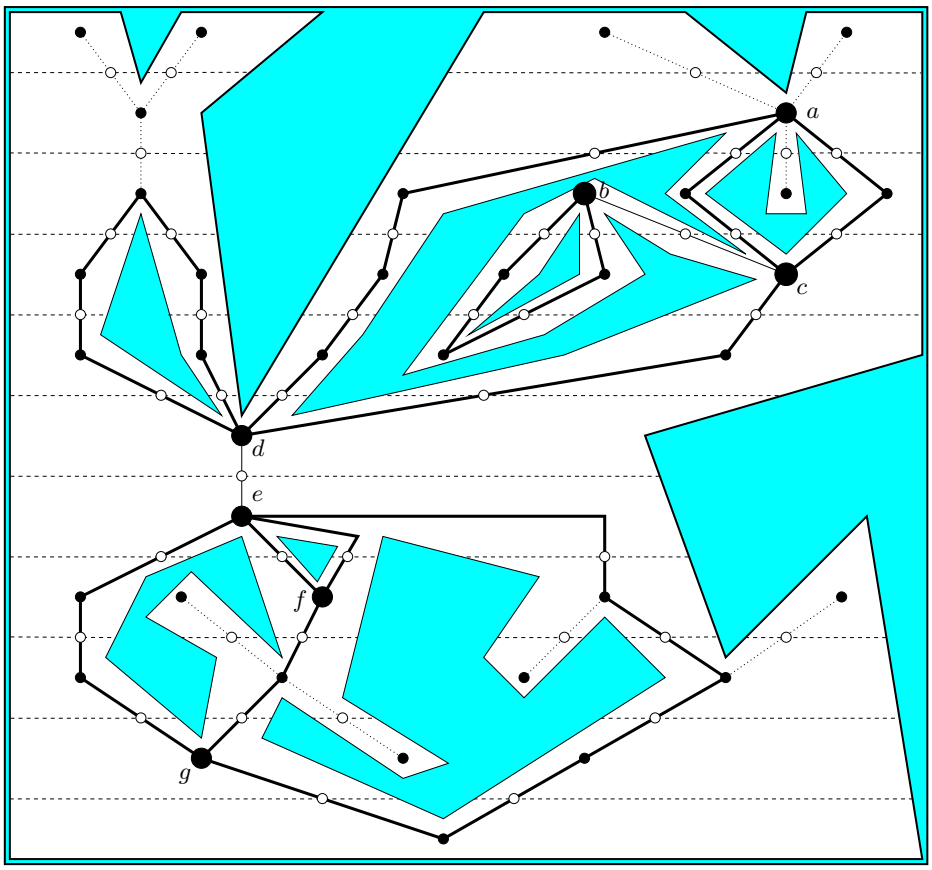

Fig. 2. A pocket with seven holes. The dashed edges are the c-edges. The f-nodes of D_P are black, while the c-nodes are white. The thick solid edges indicate the three biconnected components, the thin solid edges are backbone edges, and the dotted edges are tree edges. The large nodes are the pivot nodes.

the chains of the graph in Fig. 2. Conceptually, we may think of dividing the pivot nodes into several independent nodes, at most one for each adjacent edge, such that each of these new nodes is adjacent to exactly one chain or at least one tree edge (node a in Fig. 3 has one new node adjacent to two tree edges which are neighbors in the cyclic order of edges around a).

Lemma 3. *There are at most* $3h - 3$ *pivot nodes and* $4h - 3$ *chains.*

Proof. There can be at most $2h - 1$ pivot nodes and $3h - 2$ chains within the biconnected components. Either use Euler's formula to bound the number of degree-three nodes in planar graphs, or observe that a biconnected component consisting of a chain around a single hole (as the chain at node b in Fig. 3) induces a single pivot node, while adding more holes to a component adds at most two pivot nodes (start and end nodes of the new chain around the new hole) and at most three chains (the new chain plus splitting an exisiting chain into at most three chains, or splitting two existing chains into two chains each).

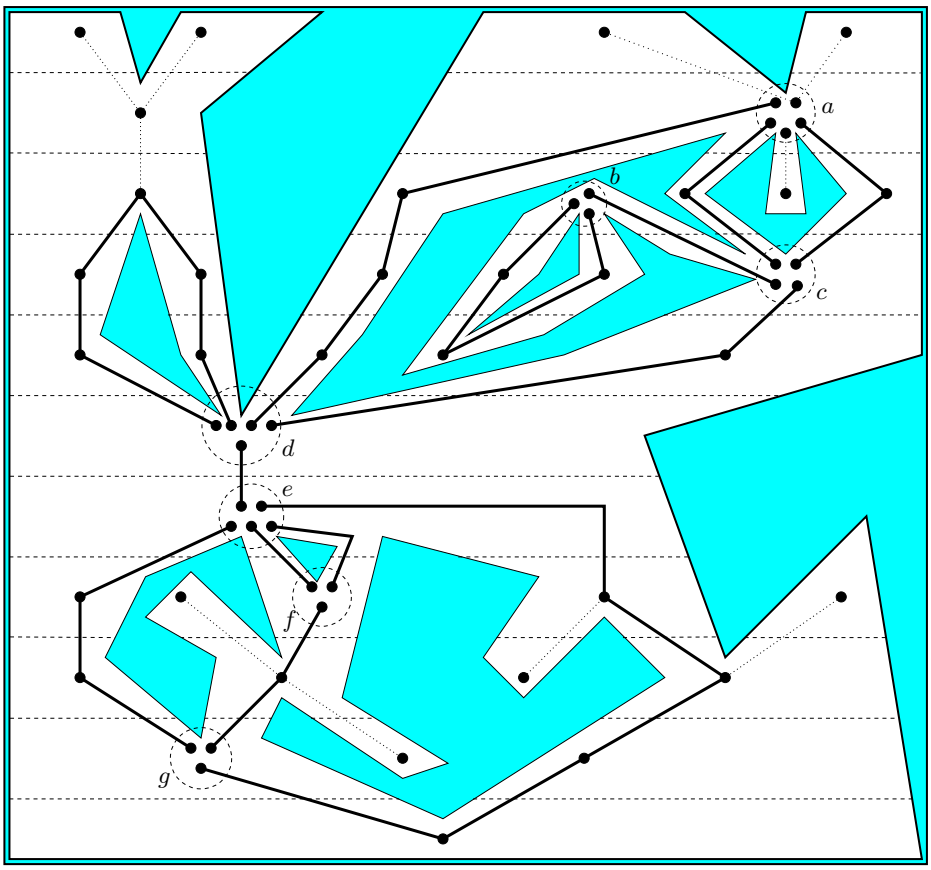

Fig. 3. The pocket in Fig. 2 gives rise to 13 chains (thick solid edges). The dashed circles indicate the pivot nodes.

Since the backbone edges form a tree connecting the at most h biconnected components, there can be at most $h-2$ pivot nodes and $h-1$ chains outside the biconnected components. □

While the trees rooted at the biconnected component and backbone nodes can be treated the same way as in Section 3 (i.e., distinguishing five different classes of no-retraction paths for the interface to the father node), chains are more complex because they have two interfaces to the rest of the graph, one at each endpoint. At each endpoint, we must know for the last c-edge crossed by the chain which of its two adjacent nc-edges are used by the optimal machining path, and whether the two c-edges of both endpoints lie on the same no-retraction path. Thus, the interface for a chain consists of 32 functions h^π, where $\pi \in \{0,1\}^5$ is a five-tuple of Boolean values encoding the five cases just described, giving the minimum number of no-retraction paths according to constraint π. These 32 functions can be computed in a similar way as the five functions h^x in Section 3.

After we have computed all optimal machining paths for all chains for all possible interface combinations, it remains to combine all the chains into a global solution for D_P. Since the graph consisting of pivot nodes and chains has size $O(h)$ by Lemma 3, we can do this in time $O(1)^{O(h)}$ by brute force enumeration of all possible interface combinations.

Theorem 4. *We can find an optimal machining path for a pocket with h holes in time $O(n+m)+O(1)^{O(h)}$.* □

5 Dual Machining Graphs of Bounded Treewidth

In this section we will show how to compute an optimal machining path efficiently on pockets with holes if the dual machining graph D_P has bounded treewidth, generalizing the algorithm for simple pockets in Section 3. We will first review some useful definitions about bounded treewidth and partial k-trees. In Section 5.2, we will then reduce the optimal machining path problem to a related problem, the minimum valid path cover Problem (`MVPC`), which can efficiently be solved if D_P has bounded treewidth.

5.1 Partial k-Trees

In this section we review shortly the definitions of k-trees and bounded treewidth. A good introduction of k-trees, partial k-trees, and treewidth can be found in [14]. We mainly follow the terminology and the dynamic programming framework developed in [6].

A graph $G=(V,E)$ is a *k-tree* if and only if we can obtain it from a k-clique by repeatedly adding nodes connected exactly to a k-clique in the existing graph. A *partial k-tree* is a spanning subgraph of a k-tree. A *tree-decomposition* of G is a pair (X,T), where $T=(I,F)$ is a tree and $X=\{X_i \mid i \in I\}$ is a family of subsets of V with one subset X_i corresponding to each node i of T, such that

(1) $\cup_{i\in I} X_i = V$, that is, every node of G is covered by some X_i,
(2) for every edge $(v,w)\in E$, there is an X_i containing both v and w, and
(3) for each node $v\in V$, the subgraph of T induced by $\{i\in I \mid v\in X_i\}$ is a connected subtree of T.

The *treewidth* of a tree-decomposition (X,T) is $\max_{i\in I}|X_i|-1$. The treewidth of G is the minimum width over all tree-decompositions of G. Any graph of bounded treewidth k is in fact a partial k-tree [24, 32]. Partial k-trees generalize trees, and many classes of graphs are partial k-trees for some constant k. Computing the treewidth of an arbitrary graph [4] is NP-complete (the complexity status for planar graphs is not known), but we can find a width $4k$ tree-decomposition in time $O(n\log n)$ [23]. For planar graphs, we can find a width $\frac{3}{2}k$ tree-decomposition in polynomial time [26] (that paper gives a polynomial

time algorithm for computing the *bandwidth* of a planar graph which is a 1.5-approximation for the treewidth). For the results in our paper, we only need to compute tree-decompositions of trees whose treewidth is known to be constant which can be done in linear time for any graph [10].

Considerable research has been done on the development of polynomial time algorithm for NP-hard problems when instances are restricted to graphs of bounded treewidth. One of the most fundamental results, by Courcelle, states that graph properties defined in monadic second-order logic can be recognized in polynomial time [13]. The result was further extended to include optimization and enumeration problems [5, 11]. Using a dynamic programming approach, many NP-hard graph problems become polynomial-time (usually linear-time) solvable on partial k-trees, for example maximum independent set, graph coloring, and Hamiltonian circuit [6]. We also use dynamic programming to tackle the `MVPC` problem[1]. Although dynamic programming is a simple generic programming method, the actual technical details can be quite different for individual problems and new twists are often needed. We will follow the framework and notations in [6]. We will first briefly sketch the dynamic programming framework, for a more detailed description and for some examples we refer to [6]. We note that dynamic programming on k-trees can be extended to partial k-trees in a straightforward manner.

We can check whether a graph G is a k-tree by repeatedly removing a node of degree k whose neighbors in the remaining graph form a k-clique, until no such nodes remain; then G is a k-tree if and only if the final remaining graph is a k-clique [25], the *root* or *root clique*. This removal process generates a *reduction sequence* of the k-tree: When we remove a node v, we say v becomes a *k-leaf* at this point, and v is said to be a (direct) *descendant* of the k-clique K_v induced by the k neighbors of v. More general, a node v is a descendant of some k-clique K_w if and only if each node adjacent to v is either a member or descendant of K_w. The connected components of the subgraph of G induced by the descendants of K are the *branches* of K. Note that we can easily obtain a tree-decomposition of G with treewidth k from a reduction sequence of a k-tree G, and vice versa.

5.2 The Minimum Valid Path Cover Problem (`MVPC`)

The dual machining graph D_P is a planar graph. In Section 3 we have shown that we can compute the optimal machining path in linear time if D_P is a tree, i.e., if the pocket is a simple polygon without holes. In this section, we will solve the problem by dynamic programming in linear time if D_P is nearly a tree, i.e., if it has bounded treewidth and bounded degree. This algorithm will be a key ingredient for our "best possible" approximation algorithm for the general case in Section 7.

[1] We note that the `MVPC` problem cannot be expressed by a monadic second-order formula of constant size, so we cannot apply Courcelle's result [14].

We reduce the optimal machining path problem to a new path cover problem in a graph related to M_P. Let the *boundary graph* $B_P = (N_B, E_B)$ of M_P be the graph obtained when we contract every c-edge into a single node, deleting self-loops and multiple edges. B_P is a planar graph of maximum degree four (every c-edge is adjacent to two nc-edges at each endpoint) whose edges are exactly the nc-edges of M_P. However, not every path in B_P is a feasible no-retraction path in M_P, because such a path should not use two consecutive nc-edges. This imposes a restriction on the edges we can use when leaving a node we just entered on an nc-edge. We call a pair of edges *consistent* if they can appear consecutively on a no-retraction path. We call a path in B_P *valid* if it corresponds to a feasible no-retraction path in M_P satisfying Lemma 1, i.e., all pairs of consecutive edges are consistent. The *minimum valid path cover problem (*`MVPC`*)* is the problem of finding a smallest set of node-disjoint simple valid paths in B_P such that each node of B_P lies on exactly one path.

In the rest of this section we assume that B_P has bounded treewidth, and that a k-tree with root R is given together with its reduction sequence (it can be computed in linear time [10]). Let $K_{+v} = K_v \cup \{v\}$ denote the $(k+1)$-clique formed by v together with K_v, and let $K_{+v}^{-u} = K_{+v} - \{u\}$, for any node $u \in K_{+v}$. Let B_v denote the set of nodes in the branches of K_v.

We now extend the simple dynamic program from Section 3, where we worked bottom-up in a tree, to k-trees. One difference is that we now work edge-by-edge instead of node-by-node. We follow the general framework in [6]. We maintain a *state* for each K_v. The state information for K_v represents the equivalence classes of the solutions (usually for a slight generalization of the original problem) of the subgraph induced by K_v and all its descendants. This generalizes the classes P^0, P^l, P^r, P^{lr1}, and P^{lr2} we introduced in Section 3. However, in a k-tree, the interface between a node and its descendants is more complicated than in a tree because a k-leaf is connected to a k-clique instead of a single parent node. Still, for fixed k, the number of possible equivalence classes is a (possibly large) constant.

We use an *index set* to denote a state. The index set $C(K_v)$ for K_v classifies solutions to the problem on the subgraph induced by $K_v \cup B_v$, where an index $c \in C(K_v)$ represents an equivalence class of the solutions. The *state value* $s(c, K_v)$ of K_v with index c is the optimum value of the equivalence class of solutions represented by c. The state values generalize the functions h^0, h^l, h^r, h^{lr1}, and h^{lr2} we introduced in Section 3.

The key ingredients of the dynamic program are the definition of the index sets and the update rules for the state values when we compute the state values bottom-up in the reduction sequence of the k-tree. The state values $s(c, K_v)$ for every $c \in C(K_v)$ depend on the state values for all K_{+v}^{-u}, where $u \in K_{+v}$, which reflect each K_{+v}^{-u}'s influence on the state value of K_v for a particular index $c \in C(K_v)$.

Now we define the index set for our algorithm. Let a *ve-pair* be a pair (v, e), where v is an end node of an edge e in the boundary graph B_P. Let $asc(v)$ denote

the edge e in the ve-pair (v, e). We say that two ve-pairs (v_1, e_1) and (v_2, e_2) are *disjoint* if $v_1 \neq v_2$. Intuitively, a no-retraction path can only enter and then leave a K_v on a pair of disjoint ve-pairs.

The state of K_v is indexed by a set of triples (D_v, S_v, I_v), where D_v is a set of mutually disjoint (unordered) pairs of ve-pairs (representing the nodes and edges on which a path enters and leaves K_v), S_v is a set of disjoint ve-pairs (representing a path ending in K_v), and I_v is a set of the 'touched' nodes (internal nodes of a path) in K_v that are disjoint from D_v and S_v. Intuitively, an element in D_v represents the two endpoints in K_v of a traversing path, and an element in S_v represents a single endpoint in K_v of a traversing path. Let $V(D_v) = \{(w_1, w_2) \mid ((w_1, e_1), (w_2, e_2)) \in D_v\}$ and $E(D_v) = \{(e_1, e_2) \mid ((w_1, e_1), (w_2, e_2)) \in D_v\}$ denote the set of node pairs and the set of edge pairs, respectively, corresponding to the ve-pairs in D_v. Similarly, let $V(S_v) = \{w \mid (w, e) \in S_v\}$ and $E(S_v) = \{e \mid (w, e) \in S_v\}$ be the set of nodes and the set of edges, respectively, corresponding to the ve-pairs in S_v.

A *partial solution* of index $c = (D_v, S_v, I_v) \in C(K_v)$ means a set of $|D_v|$ disjoint simple valid paths with both endpoints in K_v, a set of $|S_v|$ disjoint simple valid paths with only one endpoint in K_v, and some other simple valid paths with no endpoint in K_v in the subgraph induced by $K_v \cup B_v$, such that no two consecutive internal nodes on a path are both in K_v and such these paths cover all nodes of B_v. The state value $s((D_v, S_v, I_v), K_v)$ is a positive integer that is the minimum number of valid paths covering $K_v \cup B_v$ under the restrictions of index (D_v, S_v, I_v), or is `NULL` if no such valid paths exist.

The state values $s(c, K_v)$ are initially `NULL` for all c and K_v. To compute the state values for K_v, we update $s((D_v, S_v, I_v), K_v)$, where (D_v, S_v, I_v) arises from a set of $k + 1$ triples $(D_u, S_u, I_u) \in C(K_{+v}^{-u})$, one for every $u \in K_{+v}$, such that the following five conditions are satisfied. The case for computing the root state values is more complicated and will be described later separately. Intuitively, satisfying the five conditions means that we can combine the partial solutions for different K_{+v}^{-u}'s for $u \in K_{+v}$.

(i) $I_u \cap I_w = V(D_u) \cap I_w = V(S_u) \cap I_w = \emptyset$ for $u \neq w$, $u, w \in K_{+v}$.
(ii) Each node pair in $V(D_u)$ occurs at most once.
(iii) Every node in K_{+v} appears at most twice in the multi-set $M = \bigcup_{u \in K_{+v}} (V(D_u) \cup V(S_u))$.
(iv) If a node v appears twice in M, then the two corresponding edges of the ve-pairs with node v must be consistent.
(v) The graph $F = (K_{+v}, \cup_{u \in K_{+v}} E(D_u))$ is acyclic, i.e., F is a set of paths and isolated nodes.

We first define an intermediate tripe (D'_v, S'_v, I'_v) as follows. Consider a path in F. Suppose its two endpoints are v_1 and v_2. If the multiplicities of v_1 and v_2 in M are both one, then $((v_1, asc(v_1)), (v_2, asc(v_2))) \in D'_v$. If only one of v_1 and v_2

has multiplicity one in M, say v_1, then $(v_1, asc(v_1)) \in S'_v$ and $v_2 \in I'_v$. Moreover, I'_v contains the union of the I_u's and the interior nodes of the paths of F.

Next, we compute (D_v, S_v, I_v) from (D'_v, S'_v, I'_v), and then we update the state value $s((D_v, S_v, I_v), K_v)$. It is possible that we obtain more than one (D_v, S_v, I_v) from a single (D'_v, S'_v, I'_v), and we update the state value for every (D_v, S_v, I_v) obtained. Recall that $s((D_v, S_v, I_v), K_v)$ is initially $NULL$. Once we obtain a (D_v, S_v, I_v) from (D'_v, S'_v, I'_v) and a value t_v for it, called the *temporary value*, we replace the value of $s((D_v, S_v, I_v), K)$ by t_v if it is smaller than the current value of $s((D_v, S_v, I_v), K_v)$. After processing all (D'_v, S'_v, I'_v), we will have found the optimum state value for K_v. The temporary value is computed as

$$t_v = \sum_{u \in K_{+v}} s((D_u, S_u, I_u), K^{-u}_{+v}) - \#(\text{elements with multiplicity two in } M) .$$

(D_v, S_v, I_v) is obtained from (D'_v, S'_v, I'_v) in one of the following ways.

(1) If $v \in I'_v$, then $I_v = I'_v - \{v\}$, $D_v = D'_v$, and $S_v = S'_v$.
(2) If $v \in V(S'_v)$, then
  (a) $S_v = S'_v - \{(v, asc(v))\}$, $D_v = D'_v$, and $I_v = I'_v$.
  (b) If there is a node $v_1 \in K_{+v} - I'_v$ adjacent to v, then
    (b.1) If $v_1 \in V(S'_v)$ and $asc(v_1)$ and $asc(v)$ are consistent with (v, v_1), then $S_v = S'_v - \{(v, asc(v)), (v_1, asc(v_1))\}$ and $t_v = t_v - 1$.
    (b.2) If $(v_1, v_2) \in V(D'_v)$ and $asc(v_1)$ and $asc(v)$ are consistent with (v, v_1), then $D_v = D'_v - \{((v_1, asc(v_1)), (v_2, asc(v_2)))\}$, $S_v = S_v - \{(v, asc(v))\} \cup \{(v_2, asc(v_2))\}$, $I_v = I'_v \cup \{v_1\}$, and $t_v = t_v - 1$.
    (b.3) If v_1 is neither in $V(S'_v)$ nor in any pair of $V(D'_v)$ and $asc(v)$ is consistent with (v, v_1), then $S_v = S'_v - \{(v, asc(v))\} \cup \{(v_1, (v, v_1))\}$.
(3) If $(v, v_1) \in V(D'_v)$ for some v_1, then
  (a) $D_v = D'_v - \{((v, asc(v)), (v_1, asc(v_1)))\}$, $S_v = S'_v \cup \{(v_1, asc(v_1))\}$, and $I_v = I'_v$.
  (b) If there is a node $v_2 \in K_{+v} - I'_v$ adjacent to v, then
    (b.1) If $v_2 \in V(S'_v)$ and $asc(v_2)$ and $asc(v)$ are consistent with (v, v_2), then $D_v = D'_v - \{((v, asc(v)), (v_1, asc(v_1)))\}$, $S_v = S'_v \cup \{(v_1, asc(v_1))\}$, and $t_v = t_v - 1$.
    (b.2) If $(v_2, v_3) \in V(D'_v)$ and $asc(v_2)$ and $asc(v)$ are consistent with (v, v_2), then $D_v = D'_v - \{((v, asc(v)), (v_1, asc(v_1))), ((v_2, asc(v_2)), (v_3, asc(v_3)))\} \cup \{((v_1, asc(v_1)), (v_3, asc(v_3)))\}$, $I_v = I'_v \cup \{v_2\}$, and $t_v = t_v - 1$.
    (b.3) If v_2 is neither in $V(S'_v)$ nor in any pair of $V(D'_v)$ and $asc(v)$ is consistent with (v, v_2), then $D_v = D'_v - \{((v, asc(v)), (v_1, asc(v_1)))\} \cup \{((v_1, asc(v_1)), (v_2, asc(v_2)))\}$.
(4) If v is neither in I'_v nor in $V(S'_v)$ nor in any pair of $V(D'_v)$, then
  (a) $D_v = D'_v$, $S_v = S'_v$, $I_v = I'_v$, and $t_v = t_v + 1$.

(b) If v is adjacent to $v_1 \in K_{+v} - I'_v$, then let $asc(v_1) = (v, v_1)$, $I'_v = I'_v \cup \{v\}$. We add v_1 to the multi-set M, test whether conditions (i)–(v) are all satisfied, then obtain (D'_v, S'_v, I'_v), and then compute (D_v, S_v, I_v) and t_v from (D'_v, S'_v, I'_v) in exactly the same way as above.

(c) If v is adjacent to v_1 and v_2 which are both in $K_{+v} - I'_v$, and (v, v_1) and (v, v_2) are consistent with each other, then let $asc(v_1) = (v, v_1)$, $asc(v_2) = (v, v_2)$, and $I'_v = I'_v \cup \{v\}$. We add v_1 and v_2 to the multi-set M and augment F with the edge (v_1, v_2). Further, we test whether conditions (i)–(v) are all satisfied, then obtain (D'_v, S'_v, I'_v), and then compute (D_v, S_v, I_v) and t_v from (D'_v, S'_v, I'_v) in exactly the same way as above.

This finishes the discussion of the non-root case of the `MVPC` algorithm. For the root clique R, we do the same as in the non-root case, but we also perform some additional steps. For an obtained index (D_R, S_R, I_R), we try all possible combinations of the unused edges in the subgraph induced by R. We check the validity (i.e., the consistency of all pairs of adjacent edges) of each path, and decide the final value of t_R. Note that an isolated node in R should be treated as a no-retraction path in the final solution. Since the dynamic program runs in linear time, we have shown the next theorem.

Theorem 5. *If the boundary graph B_P has bounded treewidth, then we can solve* `MVPC` *optimally in linear time.* □

Now we establish a close relation between the treewidth of D_P and B_P.

Lemma 6. *If D_P is a partial k-tree of maximum degree d, then B_P is a partial $(kd + d - 1)$-tree.*

Proof. Suppose $D_P = (F, E_D)$ has a tree-decomposition $(X, T(I, F))$ with $\max_{i \in I} |X_i| \leq k + 1$. We construct a tree-decomposition $(Y, T_B(I_B, F_B))$ of B_P. T and T_{B_P} have the same topology. For a node $i \in I$ representing $X_i \subseteq F$, the corresponding node $Y_i \subseteq E_D$ in I_B is defined as $Y_i = \{e \in E_D \mid e$ has an end node in $X_i\}$. It is easy to verify that T_B is a tree-decomposition of B_P and $\max_{i \in I_B} |Y_I| \leq (k + 1)d$. □

Combining Theorem 5 and Lemma 6, we obtain the main result of this section.

Theorem 7. *If D_P has bounded treewidth and bounded degree, then we can compute an optimal machining path in linear time.* □

6 An Exact Algorithm for k-Outerplanar Dual Graphs

In this section we assume the dual graph D_P is k-outerplanar. To be precise, we always mean that the embedding of D_P as induced by P_G is k-outerplanar, we do not try to find an embedding with smaller outerplanarity parameter. Intuitively, a *k-outerplanar graph* is a planar graph such that nothing remains after we peel off its outer nodes k times.

Lemma 8. *If D_P is k-outerplanar, then B_P is $(2k)$-outerplanar.*

Proof. Given an embedding of $D_P = (F, E_D)$ in the plane, we construct an embedding of B_P. One way to construct B_P from D_P is to connect the c-nodes in F by edges cyclically around each f-node in F, removing self-loops and multiple edges (corresponding to nodes of degree one and two in F, respectively). Obviously, this gives a planar embedding of B_P.

If a node $v \in F$ is lying on the outer layer of the embedding of D_P, then there is an edge $e = (v, w) \in E_D$ whose corresponding node $v_e \in B_P$ is lying on the outer layer of the embedding of B_P. After peeling away the outer layer of B_P, all nodes of B_P corresponding to the edges adjacent to v in D_P have either been peeled off or have been exposed to the new outer face. Hence, peeling off the next layer of B_P will remove all nodes of B_P corresponding to the edges adjacent to v. Thus, B_P has at most twice the number of layers as D_P. □

Since any k-outerplanar graph has treewidth at most $3k - 1$ [9], combining Theorem 5 and Lemma 8 gives the main result of this section.

Theorem 9. *If D_P is k-outerplanar, then we can compute an optimal machining path in linear time.* □

7 An Approximation Algorithm for Pockets with Holes

In this section we give a "best possible" approximation algorithm for the zigzag pocket machining problem for a pocket with h holes. We also treat the outside of P as a hole, the *exterior hole*, we actually have a total of $h + 1$ holes. We basically make the problem easier by merging holes, thus reducing the number of holes, and sometimes even partitioning the pocket into several independent sub-pockets.

If two holes are connected by a c-edge, we can *merge* them by connecting them with a thin corridor along the c-edge. We then update D_P by deleting the c-node corresponding to the c-edge that just vanished and its two incident edges. If these edges had been bridge edges, the pocket will actually be divided into two unconnected pockets. Fig. 6 shows the result of merging four holes of the pocket in Fig. 4 into a single hole.

Lemma 10. *Merging two holes can change the minimum number of retractions by at most one.* □

Proof. Assume we merge two holes along c-edge e. If an optimal machining path p for P ends with e or leaves e on the same side (up or down) as it entered e, then p is still a valid machining path after merging the holes. Otherwise, we must cut p at e, increasing the number of retractions by one. If an optimal machining path p' for the merged holes traverses the new corridor along e only on one side,

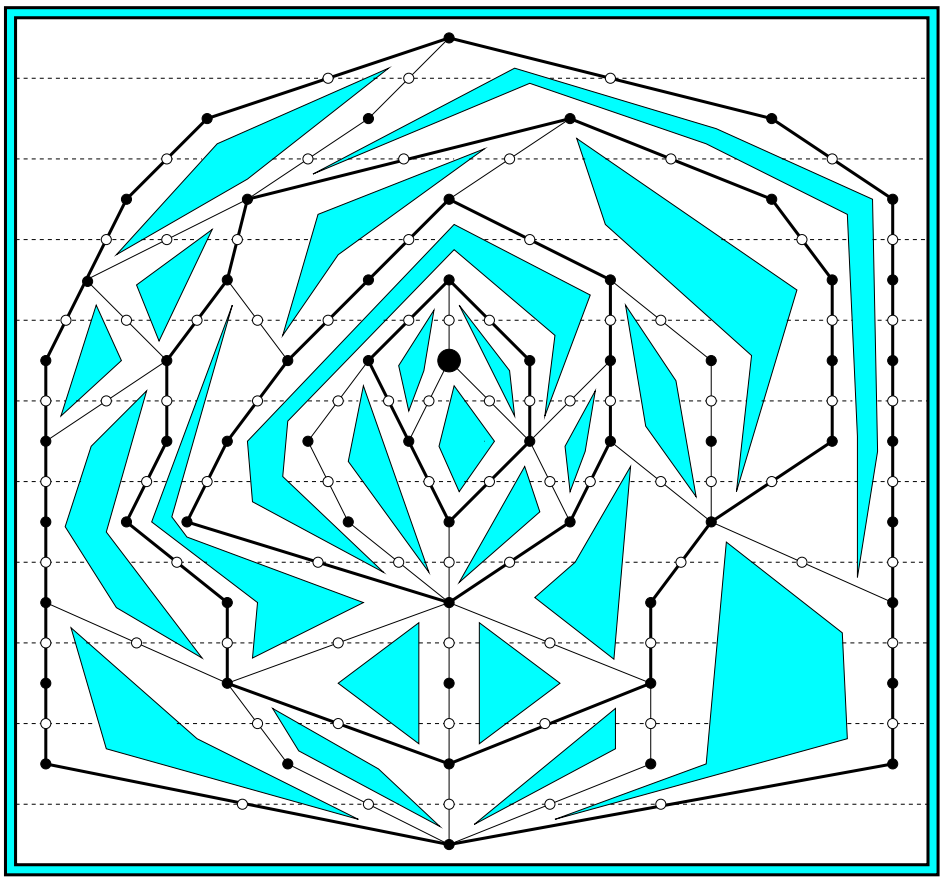

Fig. 4. A pocket with 23 holes. In particular, $h_1 = 9$, $h_2 = 7$, $h_3 = 4$, and $h_4 = 3$. The dashed edges are the c-edges. The f-nodes of D_P are black, while the c-nodes are white. The thick solid edges indicate the four outer layers of D_P, while the large black node in the middle together with its three white neighbors forms the fifth layer. The thin solid edges are the paths connecting nodes on two neighboring layers.

it can be used as a machining path for P (replacing the corridor by the c-edge e). Otherwise, one of the two no-retraction paths using the corridor must be split into two paths, increasing the number of retractions by one. □

Merging the h interior holes with the exterior hole would result in a simple pocket without holes, so we can efficiently compute a machining path with at most $OPT + h$ retractions. To reduce the number of retractions, we use a similar approach as in [8]. The idea is to use merging of holes to partition the original pocket into several unconnected pockets whose dual graphs are k-outerplanar for some small k, because we know how to solve these subproblems efficiently (Theorem 9).

Let k be a fixed integer. Consider the layer structure of D_P we obtain when we repeatedly peel off the nodes on the outer face. Let $D_1 = D_P$ and let L_1 be the set of nodes on the outer face of D_1. For $i = 2, 3, 4 \ldots$, we obtain D_i from

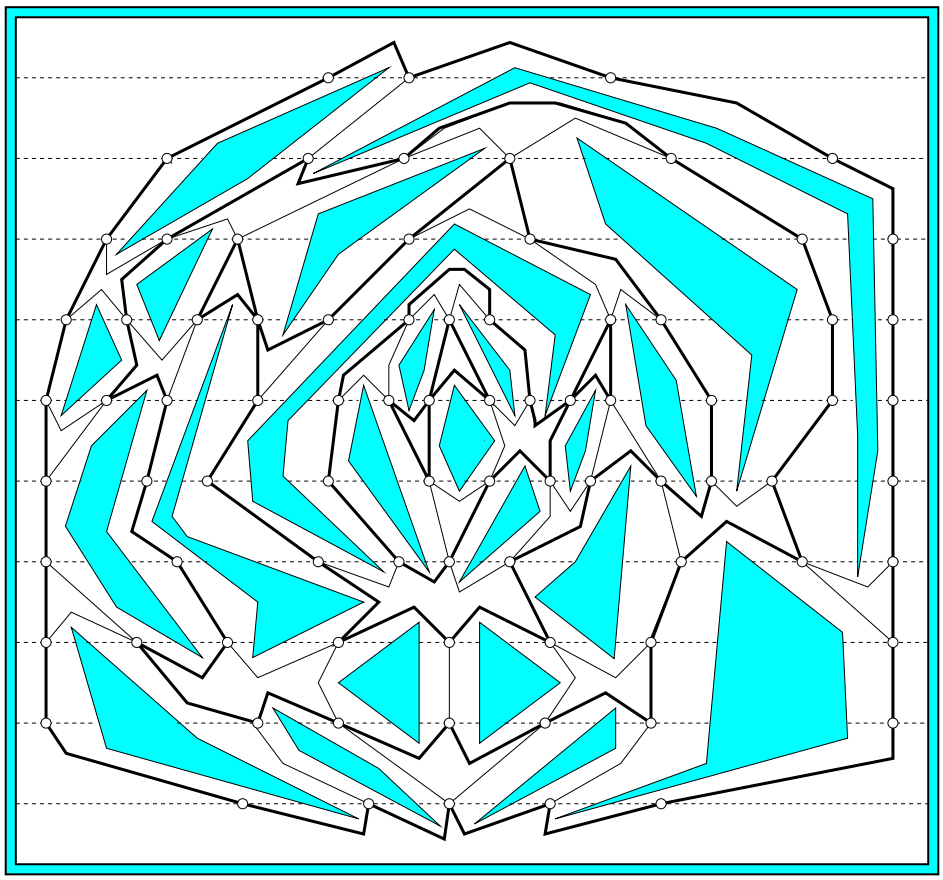

Fig. 5. The MVPC graph corresponding to the pocket in Fig. 4. The thick edges are the five layers of the graph.

D_{i-1} by removing the nodes in L_{i-1} and any incident edges. L_i is then again the set of nodes on the outer face of D_i. We call L_i the i-th *layer* of D_P. Note that a k-outerplanar graph has exactly k layers.

Any two neighboring layers are connected by several node-disjoint paths. Note that the subgraph induced by $R_i = L_i \cup L_{i+1}$ consists of two rings (the layers L_i and L_{i+1}) and node-disjoint paths connecting the two layers. Let h_i denote the number such paths. These paths divide R_i into h_i cells, where each cell corresponds to an interior hole of P. Thus, $\sum_i h_i = h$. See Fig. 4 for an example.

If we merge the h_i holes in R_i into a single ring-shaped hole (see Fig. 6), P_G is partitioned into two unconnected subpockets. If we do this for $R_k, R_{2k}, R_{3k}, \ldots$, we partition P_G into small pockets that are at most $(k+1)$-outerplanar (some parts of the paths we cut may remain as interior nodes of the k-th layer, making this embedding of the subpocket $(k+1)$-outerplanar). We could also do this for $R_{k+q}, R_{2k+q}, R_{3k+q}, \ldots$, for some $q = 0, 1, \ldots, k-1$. Let H_q denote the total number of holes in $R_{k+q}, R_{2k+q}, R_{3k+q}, \ldots$. Then there exists a q such that $H_q \leq \frac{h}{k}$.

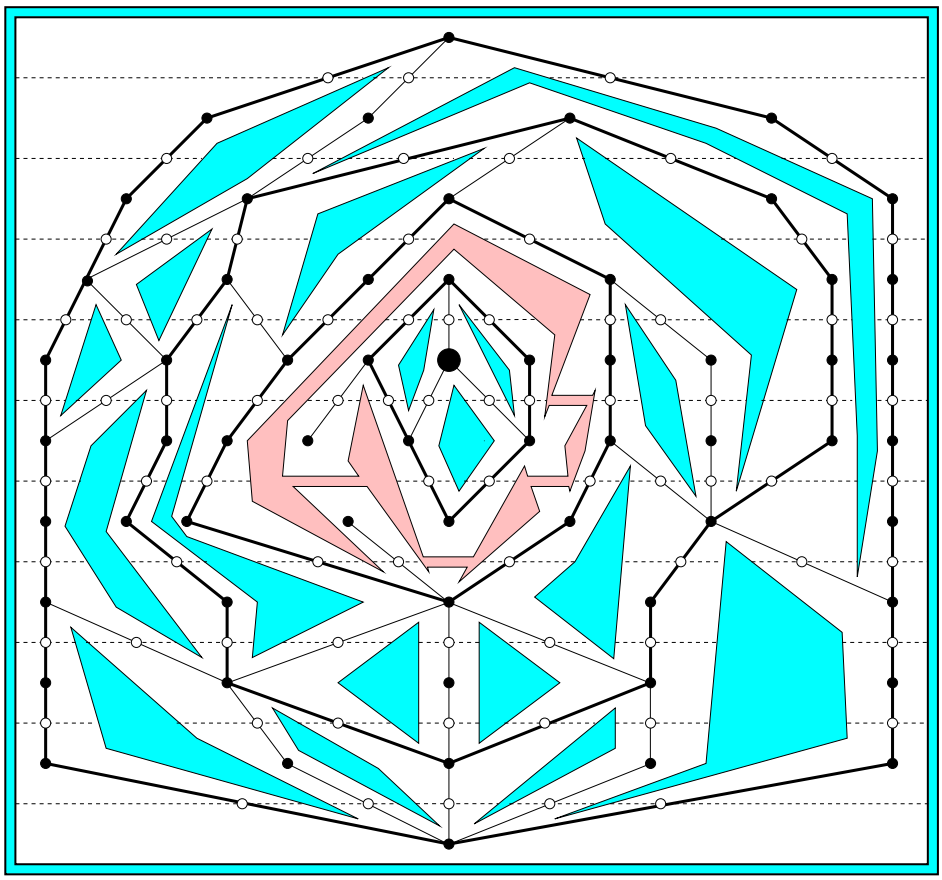

Fig. 6. Considering the pocket in Fig. 4 with $k = 3$, we can separate layers three and four by merging the four holes between the two layers in cyclic order. As a result, we obtain two independent components that are at most 4-outerplanar.

Using the techniques developed in Section 5, we can in linear time compute an optimal solution for each of the $(k+1)$-outerplanar subpockets. The sum of the retractions for all $(k+1)$-outerplanar graphs is at most $OPT + H_q \leq OPT + \frac{h}{k}$ by Lemma 10. Based on Lemma 10, we can then convert these solutions into a solution for P with at most $OPT + \frac{2h}{k}$ retractions. Choosing $k = \frac{2}{\epsilon}$, we have the following result.

Theorem 11. *For arbitrary pockets, we can in liner time compute a machining path with at most $OPT + \epsilon \cdot h$ retractions, for any constant $\epsilon > 0$, where h is the number of holes of the pocket.* □

Note that h is not considered as a constant in the theorem above. Also, note that choosing $k \geq n$ actually means that we solve the original problem directly (because D_P is at most n-outerplanar).

8 Conclusions

We have presented a variety of exact and approximation algorithms for the problem of minimizing the number of retractions in 2D-milling. All our algorithms are based on dynamic programming techniques for trees (or graphs of bounded treewidth) and Baker's shifting technique for dividing a problem into independent subproblems of small treewidth.

In Theorem 11, we gave an algorithm with an additive approximation term. This is a good outcome if the optimal solution has a high cost. However, for the cases with small optimum cost, we may prefer having an approximation algorithm with a multiplicative approximation factor, instead.

Another possible extension of our research may be to study more general cost functions. For example, the cost of a retraction might depend on the locations of the two endpoints of a jump.

References

1. E. M. Arkin, M. A. Bender, E. D. Demaine, S. P. Fekete, J. S. B. Mitchell, and S. Sethia. Optimal covering tours with turn costs. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*, pp. 138–147, 2001, and in *SIAM Journal on Computing*, 35(3):531–566, 2005.
2. E. M. Arkin, S. P. Fekete, and J. Mitchell. Approximation algorithms for lawn mowing and milling. *Computational Geometry: Theory and Applications*, 17:25–50, 2000.
3. E. M. Arkin, M. Held, and C. L. Smith. Optimization problems related to zigzag pocket machining. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'96)*, pp. 419–428, 1996, and in *Algorithmica*, 26(2):197–236, 2000.
4. S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8:277–284, 1987.
5. S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.
6. S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted for partial k-trees. *Discrete Applied Mathematics*, 23:11–24, 1989.
7. S. Arya, S.-W. Cheng, and D. M. Mount. Approximation algorithm for multiple-tool milling. *International Journal of Computational Geometry and Applications*, 11(3):339–372, 2001.
8. B. Baker. Approximation algorithm for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994.
9. H. L. Bodlaender. Some classes of graphs with bounded tree-width. *Bulletin of the EATCS*, 36:116–126, 1988.
10. H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.
11. R. B. Borie, R. G. Parker, and C. A. Tovey. Automatic generation of linear time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7(1-4):555–581, 1992.
12. L. K. Bruckner. Geometric algorithms for $2.5D$ roughing process of sculptured surfaces. In *Proceedings of the Joint Anglo-Hungarian Seminar on Computer-Aided Geometric Design*, 1982.
13. Bruno Courcelle. Graph structure and monadic second-order logic: Language theoretical aspects. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP'08)*, pp. 1–13, 2008.
14. R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer Verlag, New York, 1999.
15. M. R. Garey and D. S. Johnson, *Computers and Intractability — A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.

16. M. K. Guyder. Automating the optimization of 2.5 axis milling. In *Proceedings of the Third International IFIP Conference on Computer Applications in Production and Engineering (CAPE'89)*, 1989, and in *Computers in Industry*, 15(3):163–168, 1990.
17. M. Held. *On the Computational Geometry of Pocket Machining*. Springer Lecture Notes in Computer Science, Vol. 500, 1991.
18. M. Held. A Geometry-based investigation of the tool path generation for zigzag pocket machining. *Visual Computers*, 7(5-6):296–308, 1991.
19. M. Held and E. M. Arkin. Letter to editor: An algorithm for reducing tool retractions in zigzag pocket machining. *Computer-Aided Design*, 32(10):917–919, 2000.
20. B. K. Kim, J. Y. Park, H. C. Lee, and D. S. Kim. Determination of cutting direction for minimization of tool retraction length in zigzag pocket machining. In *Proceedings of the International Conference on Computational Science and Its Applications (ICCSA'03), Part III*, Springer Lecture Notes in Computer Science 2669, pp. 680–689, 2003.
21. T. R. Kramer. Pocket milling with tool engagement detection. *Journal of Manufacturing Systems*, 11(2):114–123, 1992.
22. D. Lichtenstein. Planar satisfiability and its uses. *SIAM Journal on Computing*, 11:329–343, 1982.
23. B. Reed. Finding approximate separators and computing tree-width quickly. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC'92)*, pp. 221–228, 1992.
24. N. Robertson and P. D. Seymour. Graph minors I: Excluding a forest. *Journal of Combinatorial Theory, Series B*, 35:39–61, 1983.
25. D. Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32:597–609, 1970.
26. P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.
27. Y. S. Suh and K. Lee. Neural network modeling for tool path planing of the rough cut in complex pocket milling. *Journal of Manufacturing Systems*, 15(5):273–284, 1990.
28. K. Tang. Geometric optimization algorithms in manufacturing. *Computers-Aided Design and Applications*, 2(6):747–758, 2005.
29. K. Tang, S.-Y. Chou, and L.-L. Chen. An algorithm for reducing tool retractions in zigzag pocket machining. *Computer-Aided Design*, 30(2):123–129, 1998.
30. K. Tang and A. Joneja. Traversing the machining graph of a pocket. *Computer-Aided Design*, 35(11):1023–1040, 2003.
31. H.-P. Wang, H. Chang, R. A. Wysk, and A. Chandawarkar. On the efficiency of NC tool path planning for face milling operations. *Transactions of the ASME, Journal of Engineering for Industry*, 109(4):370–376, 1987.
32. T. V. Wimer. *Linear Algorithms on k-Terminal Graphs*. Ph.D. thesis, Clemson University, Dept. of Computer Science, 1987.
33. Z.-Y. Yao and S. K. Gupta. Cutter path generation for 2.5D milling by combining multiple different cutter path patterns. *International Journal on Production Research*, 42(11):2141–2161, 2001.