

Traversing the Machining Graph

Danny Z. Chen^{1*}, Rudolf Fleischer^{2**}, Jian Li²,
Haitao Wang², and Hong Zhu^{2***}

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

² Department of Computer Science and Engineering, Shanghai Key Laboratory of
Intelligent Information Processing, Fudan University, Shanghai, China.
Email: {rudolf,lijan83,wanghaitao,hzhu}@fudan.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, which generalizes to an exact algorithm running in linear plus $O(1)^{O(h)}$ time for pockets with h holes. We generalize our dynamic programming approach to optimally solve in polynomial time pockets with holes whose dual graph (induced by the zigzag line segments) is a partial k -tree of bounded degree or a k -outerplanar graph, for fixed k . Finally, we propose a polynomial time algorithm for finding a machining path with at most $OPT + \epsilon \cdot h$ retractions, where $\epsilon > 0$ is any constant. Our algorithms substantially improve the previously best solutions, and result in the best possible approximation if the output quality is measured in terms of the number of holes.

1 Introduction

2D-milling, or *zigzag pocket machining (ZPM)*, is an important problem in the manufacturing industry [16, 26, 32]. 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. 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 the line). When it reaches the boundary of

* The research of this author was supported in part by a grant from the Shanghai Key Laboratory of Intelligent Information Processing, Fudan University, Shanghai, China and by the US National Science Foundation under Grant CCF-0515203. This work was partially done while the author was visiting the Shanghai Key Laboratory of Intelligent Information Processing at Fudan University, China.

** The work described in this paper was partially supported by a grant from the National Natural Science Fund China (grant no. 60573025).

*** The order of the authors follows the international standard of alphabetic order of the last names. In China, where the first-authorship is a very important aspect of a publication, the order of the authors should be Jian Li, Haitao Wang, Danny Z. Chen, Rudolf Fleischer, and Hong Zhu.

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 spaced with a *zigzag step-over distance* $\delta > 0$; again, this technical requirement is not important to 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 problems), the goal is to find a machining path minimizing the number of retractions, under the additional constraint that the cutter must work on any 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 [32] for an extensive survey of the current state-of-the-art. A few algorithms were given in [9, 19], 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 [15, 16]. 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* [12, 20]. They also presented a linear time approximation algorithm with at most $5 \cdot OPT + 6 \cdot h$ 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. [27] 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 optimal solution to hold (see [17]).

Tang and Joneja [28] 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 1 in [28], the additional parameter N_r can be large in some situations. In fact, the parameter N_r is related to several factors, such as the shape of the given pocket, the step-over distance, the inclination of the reference line, etc. For example, the longer the step-over distance is, the larger may N_r become. Thus, the results in [3] and in [28] are not directly comparable. Some practical implications and applications of ZPM were discussed in [16, 27]. Recently, based on the approaches in [3, 28], Wang et al. [30] gave a linear time approximation algorithm with at most $3 \cdot OPT + 3 \cdot h$ retractions for a pocket with $h \geq 0$ holes.

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, 28, 30]. In fact, it was an open problem to decide whether the simply connected pocket case is NP-hard [3].

Other optimization criteria for ZPM have also been considered. For example, multi-tool retraction minimization was studied in [6]. The problem of minimizing the total length of the tool path was studied in [2]. Some algorithms for determining a cutting direction in order to minimize the tool retraction length were given in [18]. Some algorithms were designed to optimize the tool path length and the number of tool retractions [1, 25]. A survey of the pocketing requirements can be found in [14].

In this paper, we present significantly improved algorithms for ZPM. Our techniques are different from the previous ones [3, 28, 30]. Our algorithms are superior to the existing ones in theoretical performance, and may even be interesting for practical applications (e.g., for k -outerplanar dual graphs, with a small k).

- (1) We give an optimal linear time dynamic programming algorithm for simply connected pockets, settling the open “NP-hardness-or-not” question.
- (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 polynomial time 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 in [5].
- (3) Combining the ingredients of (1) and (2) with the approximation scheme for planar graphs in [7], we develop a polynomial time approximation algorithm for finding a machining path with at most $OPT + \epsilon \cdot h$ retractions for a pocket with h holes, for any constant $\epsilon > 0$. This substantially improves the previous approximation solutions [3, 28, 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 polynomial-time solvable.

The rest of this paper is organized as follows. In Section 2, we review some definitions in [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 first introduce the concept of a boundary graph and the MRPC problem, and then present exact polynomial 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 5, we present a “best possible” polynomial time approximation algorithm for the general problem. In the Appendix, we give the details of the dynamic program for simple connected pockets, and review some facts about k -trees and bounded tree-width.

2 Preliminaries

We mainly use the terminology in [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.

Let $\delta > 0$ be a constant, the so-called *zigzag step-over distance*. Let Δ be another constant. We assume that the given reference line for the machining path is parallel to the x -axis. A line parallel to the x -axis with ordinate y is denoted by $l(y)$. Fig. 1(a) shows a pocket in the plane.

The set $ZL(P)$ of *zigzag lines w.r.t. P* is the set of lines

$$ZL(P) = \{l(y_i) : y_i = \Delta + i \cdot \delta, l(y_i) \cap P \neq \emptyset, \text{ and } i \in \mathbb{Z}\}.$$

The set $ZS(P)$ of zigzag segments induced by P is $ZS(P) = \cup_{\ell \in ZL(P)} P \cap \ell$. Here we make the assumption of general position, i.e., the zigzag lines do not just

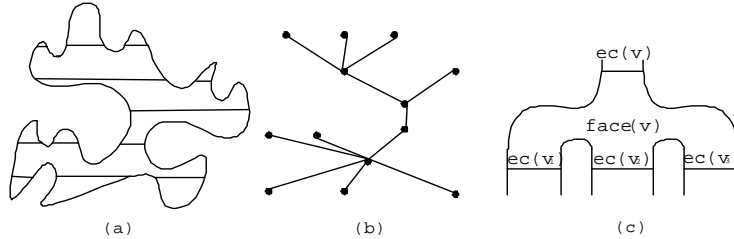


Fig. 1. (a) A simple pocket. (b) Its dual graph is a tree. (c) A face in a simple pocket.

touch B as a tangent. Let N be the set of all endpoints of the zigzag segments in $ZS(P)$. The *machining graph* M_P is an undirected graph on the node set N . We distinguish two types of edges in M_P : (1) *compulsory* edges (*c-edges*) connecting the two endpoints of a zigzag segment; (2) *non-compulsory* edges (*nc-edge*) connecting two neighboring nodes on B .

Throughout the paper, we denote the machining graph of a pocket P by $M_P = (N, E)$. The size of N is denoted by n . Note that every node in M_P is incident to exactly one c-edge and two nc-edges.

Given a machining graph $M_P = (N, E)$, a *tool traversing path* (or *machining path*) is a collection \mathcal{P} of simple vertex-disjoint paths in M_P , called *no-retraction paths*, such that (1) every c-edge is traversed exactly once, and (2) 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, 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(b)).

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 graph of M_P .

3 An Optimal 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. Figs. 1(a) and 1(b) show an example. For a vertex $v \in F$, 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 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 degree-one node $root \in F$. For a node $v \in F$, let T_v denote the subtree of T rooted at v , $P_{G/v}$ 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 . Let M_v denote the machining subgraph of $P_{G/v}$, where we require that $ec(v)$ is a c-edge of M_v . The two endpoints of $ec(v)$ are $left_v$ and $right_v$, where the former one 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. Fig. 1(c) shows the situation for an internal node v with three children. In general, v may have k children v_1, \dots, v_k , where we assume that $ec(v_1), \dots, ec(v_k)$ 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 $1 \leq i < k$. Note that we have already computed the machining paths for the subtrees rooted at v_1, \dots, v_k . Now we must 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 extend this path to include $ec(v)$. If both these points are endpoints of two different paths, we can even 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 of M_P , we must provide for all cases, i.e., we use dynamic programming.

Let $M_{i,j} = \cup_{\ell=i}^j M_{v_\ell} \cup \cup_{\ell=i}^{j-1} bridge(v_\ell)$, for $1 \leq i \leq j \leq k$, be the connected portion of M_P formed by M_{v_i}, \dots, M_{v_j} and the bridges connecting the pieces. We characterize all feasible machining paths for $M_{i,j}$ into five classes. $P_{i,j}^0$ contains all machining paths such that no no-retraction path ends at $left_{v_i}$ or $right_{v_j}$; $P_{i,j}^l$ contains all paths such that a no-retraction path ends at $left_{v_i}$ but no such path ends at $right_{v_j}$; $P_{i,j}^r$ contains all paths such that a no-retraction path ends at $right_{v_j}$ but no such path ends at $left_{v_i}$; finally, $P_{i,j}^{lr}$ contains all paths such that some no-retraction paths end at $left_{v_i}$ and $right_{v_j}$; actually, we must divide the last class further into $P_{i,j}^{lr1}$, where the same no-retraction path connects both points, and $P_{i,j}^{lr2}$, where two different no-retraction paths end at the two points.

Let $h^x(v_i, v_j)$, for $x \in \{0, l, r, lr1, lr2\}$, be the minimum number of retractions among all machining paths in $P_{i,j}^x$. Then the optimal value for M_P is $\min\{h^0(root, root), h^l(root, root), h^r(root, root), h^{lr1}(root, root), h^{lr2}(root, root)\}$.

Initially, for any leaf v of T , $h^{lr1}(v, v) = 0$ and $h^0(v, v) = h^l(v, v) = h^r(v, v) = h^{lr2}(v, v) = \text{NULL}$, 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. It is straightforward to compute these values iteratively for an internal node v and all pairs of indices $i < j$ if the values for all children of v are known. For example,

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

The complete set of recursive formulas can be found in Appendix A.

It remains to compute the values $h^x(v, v)$. Recall v_1, \dots, v_k are children of v .
 $h^{lr1}(v, v) = 1 + \min_{x \in \{0, l, r, lr1, lr2\}} \{h^x(v_1, v_k)\}$,
 $h^{lr2}(v, v) = \text{NULL}$, $h^r(v, v) = 1 + \min_{x \in \{l, lr1, lr2\}} \{h^x(v_1, v_k)\}$, $h^l(v, v) = 1 + \min_{x \in \{r, lr1, lr2\}} \{h^x(v_1, v_k)\}$, and $h^0(v, v) = h^{lr2}(v_1, v_k) - 1$.

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

If P has h holes, the dual graph D_P is no longer a tree. However, we can identify $O(h)$ *pivot nodes* such that the removal of these nodes results in a forest of trees such that each tree is adjacent to a constant number of pivot nodes (for details see the full version of the paper). The trees of the forest can be solved similar as in Section 3. Since each tree has only a constant size interface with the pivot nodes, we can test all possible choices for the c -edges of these nodes in $O(1)^{O(h)}$ time. This implies that the problem is still polynomial-time solvable for pockets with $h = O(\log(n + m))$ holes.

Theorem 3. *We can find an optimal machining path for a pocket with h holes in time $O(n + m) + O(1)^{O(h)}$.* \square

4 The MRPC Problem

Note that even for a pocket with holes, its dual graph is a planar graph embedded in the plane. In this section, we investigate certain types of planar dual graphs. These special cases will be a key to our “best possible” approximation algorithm for the general case in Section 5.

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. Note that B_P is a planar graph of maximum degree four. Its edges are exactly the nc -edges of M_P . 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, i.e., on a path in B_P we cannot leave a node on an arbitrary other edge since some edge is prohibited. We call a path in B_P *valid* if and only if it corresponds to a feasible no-retraction path in M_P . We call pairs of edges that can lie consecutively on a valid path *consistent*. The *minimum restricted path cover problem (MRPC)* is now the problem of finding a set of vertex-disjoint simple valid paths in B_P such that each node of B_P lies on exactly one path. Our goal is to minimize the number of such paths.

We assume that the reader is familiar with the definitions of k -trees and bounded tree-width. A good introduction of k -trees, partial k -trees, and tree-width can be found in [11, 13]. We follow the terminology and the dynamic programming framework developed in [5]. In Appendix B, we review some definitions and facts that we use in this paper. In the rest of this section we assume

that the boundary graph B_P has bounded tree-width, and that a k -tree is given together with its reduction sequence.

For simplicity, we use the following notation: For a vertex v , $K = K(v)$ is the set of neighbors of v when v becomes a k -leaf during the k -tree reduction process. Let $K' = K \cup \{v\}$ and $K^u = K' - \{u\}$, for any vertex $u \in K'$. $B(K)$ is, at all times, the set of vertices in the currently removed branches on K . R is the root clique.

We now extend the simple dynamic program from Section 3, where we worked bottom-up in a tree, to k -trees. We keep a state for each K . This state information for K in some sense represents the equivalence classes of the solutions (usually for a slight generalization of the original problem) of the subgraph induced by K and all its descendants. This is similar with the classes P^0, P^l, P^r, P^{lr1} , and P^{lr2} we classified in Section 3. In a k -tree, the interface between a node and its descendants is more complicated 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 (large) constant.

We use an *index set* to denote the state. The index set $C(K)$ for K is a set of solutions to the problem on the subgraph induced by $K \cup B(K)$. An index $c \in C(K)$ is an equivalence class of the solutions. The value of K with index c , $s(c, K)$, is the optimum value of the solutions represented by c , and we call this value the *state value*. They have a similar meaning as the functions h^0, h^l, h^r, h^{lr1} , and h^{lr2} in Section 3.

If a vertex v is removed, we must update the state value $s(c, K)$ for every $c \in C(K)$. The update is based on the state values for all K^u ($u \in K'$) and reflects each K^u 's influence on the state value of K for a particular index $c \in C(K)$. This procedure is normally called *combining removed branches*, since the removal of v means that we need to combine v with the already removed branches on K .

Now we define the index set for our algorithm. Let a *ve-pair* be a pair (v, e) , where v is an end vertex 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 again leave the interface K on a pair of disjoint ve-pairs.

The state of K is indexed by a set of triples (D, S, I) , where D is a set of mutually disjoint (unordered) pairs of ve-pairs (a path enters and leaves K), S is a set of disjoint ve-pairs (a path ends in K), and I is a set of the 'touched' vertices (internal vertices of a path) in K that are disjoint from D and S . Let

$$D = \{(v_{i1}, e_{i1}), (v_{i2}, e_{i2}) \mid 1 \leq i \leq |D|\}, \text{ and } S = \{(v_i, e_i) \mid 1 \leq i \leq |S|\}.$$

Further, let

$$V(D) = \{(v_{i1}, v_{i2}) \mid 1 \leq i \leq |D|\}, E(D) = \{(e_{i1}, e_{i2}) \mid 1 \leq i \leq |D|\},$$

$$V(S) = \{v_i \mid 1 \leq i \leq |S|\}, \text{ and } E(S) = \{e_i \mid 1 \leq i \leq |S|\}.$$

A partial solution of index $c = (D, S, I) \in C(K)$ means a set of $|D|$ disjoint simple valid paths with both endpoints in K , a set of $|S|$ disjoint simple valid paths with only one endpoint in K , and some other simple valid paths with no endpoint in K in the subgraph induced by $K \cup B(K)$, such that no two consecutive internal vertices of a path are both in K and these paths should cover

all vertices of $B(K)$. The state value $s((D, S, I), K)$ is a positive integer that is the minimum number of valid paths covering $K \cup B(K)$ under the restrictions of index (D, S, I) , or is **NULL** if no such valid paths exist.

The state values $s(c, K)$ are initially **NULL** for all c and K . Upon the removal of v , we update $s((D, S, I), K)$, where (D, S, I) arises from a set of $k + 1$ triples $(D_u, S_u, I_u) \in C(K^u)$, one for every $u \in K'$, such that the following conditions (i)–(v) are satisfied. The case for removing the root needs a little more work and will be described later separately. Intuitively, satisfying these conditions means that we can combine the partial solutions for different K^u 's for $u \in K'$.

- (i) $I_u \cap I_w = \emptyset$ if $u \neq w$, $(\cup_{p \in V(D_u)} \{p\}) \cap I_w = \emptyset$ for $u, w \in K'$, and $V(S_u) \cap I_w = \emptyset$ for $u, w \in K'$.
- (ii) A vertex pair $d \in V(D)$ occurs at most once.
- (iii) Consider the multi-set $M = \bigcup_{u \in K'} (\cup_{p \in V(D_u)} \{p\} \cup V(S_u))$ (multiplicity is counted). Every vertex $u \in K'$ appears at most twice in M .
- (iv) For a vertex v that appears twice in M , consider the two ve-pairs (each contributing a v to M) (v, e_1) and (v, e_2) ; then e_1 and e_2 must be consistent.
- (v) The graph $F = (K', \cup_{u \in K'} V(D_u))$ has no cycle, i.e., F is a set of paths and isolated vertices.

We define (D', S', I') 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 $((v_1, \text{asc}(v_1)), (v_2, \text{asc}(v_2))) \in D'$. If only one of v_1 and v_2 has multiplicity one in M , say v_1 , then $(v_1, \text{asc}(v_1)) \in S'$ and $v_2 \in I'$. Moreover, I' contains the union of the I_u 's and the interior vertices of the paths of F .

Next, we compute (D, S, I) from (D', S', I') , and then update the state value $s((D, S, I), K)$. It is possible that we obtain more than one (D, S, I) from a single (D', S', I') , and we update the state value for every (D, S, I) obtained. Recall that $s((D, S, I), K)$ is initially **NULL**. Once we obtain a (D, S, I) from (D', S', I') and a value for it, called the *temp value*, we replace the value of $s((D, S, I), K)$ by the temp value if it is smaller than the current value of $s((D, S, I), K)$.

(D, S, I) is obtained from (D', S', I') in one of the following ways. Suppose we are currently removing v . Denote the temp value of (D, S, I) by *tempv*.

$$\text{tempv} = \sum_{u \in K'} s((D^u, S^u, I^u), K) - \#(\text{elements with multiplicity two in } M).$$

- (1) If $v \in I'$, then $I = I' - \{v\}$ and $D = D', S = S'$.
- (2) If $v \in V(S')$, then
 - (a) $S = S' - \{(v, \text{asc}(v))\}$ and $D = D', I = I'$.
 - (b) There is a vertex $v_1 \in K'$ that is adjacent to v and $v_1 \notin I'$. Then
 - (b.1) If $v_1 \in V(S')$ and $\text{asc}(v_1)$ and $\text{asc}(v)$ are consistent with (v, v_1) , then $S = S' - \{(v, \text{asc}(v)), (v_1, \text{asc}(v_1))\}$ and $\text{tempv} = \text{tempv} - 1$.
 - (b.2) If $(v_1, v_2) \in V(D')$ and $\text{asc}(v_1)$ and $\text{asc}(v)$ are consistent with (v, v_1) , then $D = D' - \{((v_1, \text{asc}(v_1)), (v_2, \text{asc}(v_2)))\}$, $S = S - \{(v, \text{asc}(v))\} \cup \{(v_2, \text{asc}(v_2))\}$, $I = I' \cup \{v_1\}$, and $\text{tempv} = \text{tempv} - 1$.
 - (b.3) If v_1 is neither in $V(S')$ nor in any pair of $V(D')$ and $\text{asc}(v)$ is consistent with (v, v_1) , then $S = S' - \{(v, \text{asc}(v))\} \cup \{(v_1, (v, v_1))\}$.

- (3) If $(v, v_1) \in V(D')$ for some v_1 , then
- (a) $D = D' - \{((v, asc(v)), (v_1, asc(v_1)))\}$, $S = S' \cup \{(v_1, asc(v_1))\}$, and $I = I'$.
 - (b) There is a vertex $v_2 \in K'$ that is adjacent to v and $v_2 \notin I'$.
 - (b.1) If $v_2 \in V(S')$ and $asc(v_2)$ and $asc(v)$ are consistent with (v, v_2) , then $D = D' - \{((v, asc(v)), (v_1, asc(v_1)))\}$, $S = S' \cup \{(v_1, asc(v_1))\}$, and $tempv = tempv - 1$.
 - (b.2) If $(v_2, v_3) \in V(D')$ and $asc(v_2)$ and $asc(v)$ are consistent with (v, v_2) , then $D = D' - \{((v, asc(v)), (v_1, asc(v_1))), ((v_2, asc(v_2)), (v_3, asc(v_3)))\} \cup \{((v_1, asc(v_1)), (v_3, asc(v_3)))\}$, $I = I' \cup \{v_2\}$, and $tempv = tempv - 1$.
 - (b.3) If v_2 is neither in $V(S')$ nor in any pair of $V(D')$ and $asc(v)$ is consistent with (v, v_2) , then $D = D' - \{((v, asc(v)), (v_1, asc(v_1)))\} \cup \{((v_1, asc(v_1)), (v_2, asc(v_2)))\}$.
- (4) If v is neither in I' nor in $V(S')$ nor in any pair of $V(D')$, then
- (a) $D = D'$, $S = S'$, $I = I'$, and $tempv = tempv + 1$.
 - (b) If v is adjacent to v_1 which is in $K' - I'$, then let $asc(v_1) = (v, v_1)$, $I' = I' \cup \{v\}$. Add v_1 to the multi-set M , test whether conditions (i)–(v) are all satisfied, obtain (D', S', I') , and then (D, S, I) and $tempv$, from (D', S', I') , in exactly the same way as above.
 - (c) If v is adjacent to v_1 and v_2 which are both in $K' - I'$, 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 $I' = I' \cup \{v\}$. Add v_1 and v_2 to the multi-set M and augment F with the edge (v_1, v_2) . Further, test whether conditions (i)–(v) are all satisfied, obtain (D', S', I') , and then (D, S, I) and $tempv$, from (D', S', I') , in exactly the same way as above.

This finishes the discussion of the non-root case of the MRPC 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, S, I) , 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 $tempv$. Note that an isolated vertex in R should be treated as one no-retraction path in the final solution.

Up to now, we assumed that B_P has been known to be a k -tree and is given with a reduction sequence of the k -tree. However, deciding whether a graph has a tree-decomposition with tree-width at most k is NP-complete [4]. Fortunately, we can use an approximation of the tree-width. For a graph $G = (V, E)$ of (unknown) tree-width k , it is possible to compute a tree-decomposition of G of tree-width at most $3k + 2$ in $O((|V| + |E|) \log(|V| + |E|))$ time [22]. Thus, given a boundary graph B_P of bounded tree-width, we can find a tree-decomposition of B_P with at most a constant factor blowup of its tree-width. Then we can compute a reduction sequence of the resulting k -tree for B_P and apply the dynamic program to obtain an optimal solution of ZPM in linear time.

Theorem 4. *If the boundary graph B_P of D_P has bounded tree-width k , then MRPC can be solved optimally in polynomial time. \square*

Lemma 5. *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 $Y_i = \{e \in E_D \mid e \text{ has an end vertex in } X_i\}$. It is easy to verify that T_B is a tree-decomposition of B_P and $\max_{i \in I_B} |Y_i| \leq (k + 1)d$. \square

Theorem 6. *If D_P has bounded tree-width and bounded degree, then we can compute an optimal machining path in polynomial time.*

Intuitively, a k -outerplanar graph is a planar graph such that nothing remains after we peel away its outer vertices k times.

Lemma 7. *If D_P is k -outerplanar, then B_P is $(2k)$ -outerplanar.*

Proof. If $v \in F$ is at the outer layer of the embedding of D_P , then there is an edge $e = (v, v_1)$ whose corresponding vertex $v_e \in B_P$ is at the outer layer of the embedding of B_P . After peeling away the outer layer of B_P , all vertices corresponding to the edges adjacent to v in D_P are either peeled off or exposed to the outer face of the remaining B_P . Hence, the next peeling will remove all vertices of B_P corresponding to the edges adjacent to v . Therefore, after $2k$ peelings, nothing remains in B_P . \square

Lemma 8 [8]. *A k -outerplanar graph has tree-width at most $3k - 1$.* \square

Theorem 9. *If D_P is a k -outerplanar graph, we can compute an optimal machining path in polynomial time.* \square

5 An Approximation Algorithm for Pockets with Holes

This section gives a “best possible” approximation algorithm for the zigzag pocket machining problem for a pocket with h holes. On the dual graph $D_P = (F, E_d)$, we define an *edge-cutting* operation on an edge (v, v_1) at v as follows: Eliminate edge (v, v_1) , create a new node v' , and add the edge (v', v_1) . The corresponding operation on the original machining graph is to divide $face(v)$ into two parts. The boundary of one part will contain the c -edge between $face(v)$ and $face(v_1)$. We denote this new face as $face(v')$. See Fig. 2(a) for an illustration. Note that it may happen that the pocket gets divided into two unconnected regions. Nevertheless, we call the pocket (or both pockets together) after the cut $P/\{v, v_1\}$ and its minimum number of retractions for a machining path $OPT/\{v, v_1\}$. The proof of the following simple lemma is given in Appendix C.

Lemma 10. *If we cut an edge (v, v_1) in D_P at v , then (1) $OPT/\{v, v_1\} \leq OPT + 1$, and (2) from an optimal solution for $P/\{v, v_1\}$ we can obtain a solution for P with at most $OPT/\{v, v_1\} + 1$ retractions.* \square

Our approximation algorithm uses a similar approach as the one in [7]. Let k be a fixed integer. The term “face” below refers to the embedding of D_P . We define the j -th layer L_j as the set of vertices that are removed in the j -th round of removal if we repeatedly remove the outer vertices of the remaining portion

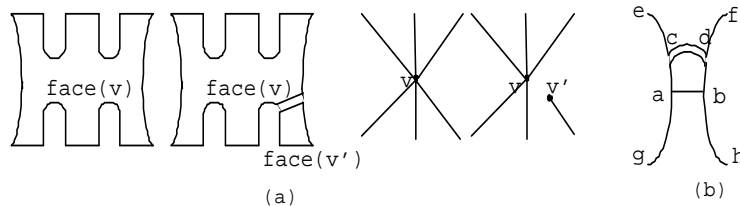


Fig. 2. (a) An edge-cutting operation. (b) Illustrating the proof of Lemma 10.

D_j of D (initially, $D_1 = D$). Let $D(L_j)$ denote the subgraph of D_j induced by the vertices of L_j with the same embedding as D , $O(L_j)$ be the outer face of $D(L_j)$, and $I(L_j)$ be the union of the inner faces of $D(L_j)$. Let h_j be the number of faces in $\mathbb{R}^2 - O(L_j) - I(L_{j+1})$ (the portion between layer j and layer $j + 1$). Since $\bigcup_{j \geq 1} (\mathbb{R}^2 - O(L_j) - I(L_{j+1})) = \mathbb{R}^2 - O(L_1)$, we have $\sum_j h_j = h$.

Lemma 11. *We can disconnect the boundary of $I(L_j)$ from the boundary of $I(L_{j+1})$ by breaking at most $h_j + h_{j+1}$ edges.*

Proof. W.l.o.g., we assume that $I(L_j)$ is connected. Denote the outer boundary vertices of $I(L_j)$ by $B = \{b_1, \dots, b_l\}$, in counterclockwise order; then, $b_i \in L_j$ for $1 \leq i \leq l$. Our proof is based on a construction that proceeds in two phases. In the first phase, consider a vertex v on the outer boundary of $I(L_j)$. If no such vertex is adjacent to some vertex in L_{j+1} , then the lemma holds. Suppose v is adjacent to $v_1 \in L_{j+1}$. If edge (v, v_1) is shared by two different faces, we cut (v, v_1) at v . Let the newly created vertex be $v' \in L_{j+1}$. This operation merges these two faces into one face. Repeat the above operation until it cannot be applied anymore, and let the resulting graph be G_1 .

Consider a connected component C of $D(L_{j+1})$. If C is connected with B , then there must be an edge (b_x, v_C) such that $b_x \in B$ and $v_C \in C$. We claim that cutting (b_x, v_C) at b_x will disconnect B and C in G_1 . Suppose this is not the case. Then there is another edge (b_y, v'_C) connecting B and C , where $b_y \in B$ and $v'_C \in C$. Then, (b_x, v_C) , $\{b_{x+1}, \dots, b_{y-1}\}$, (b_y, v'_C) , and a path from v_C to v'_C in C form a cycle, and (b_x, v_C) , $\{b_1, \dots, b_{x-1}\}$, $\{b_{y+1}, \dots, b_l\}$, (b_y, v'_C) , and a path from v_C to v'_C in C form another cycle. This makes (b_x, v_C) adjacent to two different faces, a contradiction.

In the second phase, we perform the following operation on G_1 . If B is connected with a connected component C of $D(L_{j+1})$ (C contains some nonempty inner faces) by an edge (b_x, v_C) with $b_x \in B$ and $v_C \in C$, then we cut (b_x, v_C) at b_x . Repeat this operation until it cannot be applied anymore on G_1 . From the argument above, we can easily see that these operations disconnect the boundary of $I(L_j)$ from that of $I(L_{j+1})$.

Now we count how many edges have been cut. In the first phase, at most h_j edges are cut. In the second phase, the at most h_{j+1} edges are cut since we cut at most one edge for each connected component C of $D(L_{j+1})$ with some nonempty inner faces. \square

Let q , $0 \leq q < k$, be a constant integer to be chosen later. We disconnect the boundary of $I(L_q)$ from the boundary of $I(L_{q+1})$, the boundary of $I(L_{k+q})$ from the boundary of $I(L_{k+q+1})$, the boundary of $I(L_{2k+q})$ from the boundary

of $I(L_{2k+q+1}), \dots$, by cutting in total at most $\sum_i (h_{ik+q} + h_{ik+q+1})$ edges, by Lemma 11. Choose q to be the integer that minimizes $H_q = \sum_i (h_{ik+q} + h_{ik+q+1})$. By

$$\sum_{i=0}^{k-1} H_i = \sum_{i=1}^{k-1} \sum_j (h_{jk+i} + h_{jk+i+1}) \leq 2 \sum_i h_i = 2h,$$

we have $H_q \leq \frac{2h}{k}$. Moreover, it is easy to see that the resulting graph is a series of $(k+1)$ -outerplanar graphs. Using the techniques developed at the end of Section 4, we compute in polynomial time an optimal solution for each of these $(k+1)$ -outerplanar graphs.

The sum of the retractions for all $(k+1)$ -outerplanar graphs is at most $OPT + H_q \leq OPT + \frac{2h}{k}$ by the first inequality of Lemma 10. Then, by the second part of Lemma 10, we know how to convert these solutions into a final solution with at most $OPT + \frac{4h}{k}$ retractions. Choosing a constant integer value of k such that $\epsilon = \frac{4}{k}$, we have the following result.

Theorem 12. *We can compute in polynomial time a machining path with at most $OPT + \epsilon \cdot h$ retractions, for any constant $\epsilon > 0$.* \square

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. *Proc. 11th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'01)*, pp. 138–147, 2001.
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. *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 and A. Proskurowski. Linear Time Algorithms for NP-hard Problems Restricted for Partial k -trees. *Discrete Applied Mathematics*, 23:11–24, 1989.
6. 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.
7. B. Baker. Approximation Algorithm for NP-complete Problems on Planar Graphs. *Journal of the ACM*, 41(1):153–180, 1994.
8. H. L. Bodlaender. Some Classes of Graphs with Bounded Tree-width. *Bulletin of the EATCS*, 36:116–126, 1988.
9. L. K. Bruckner. Geometric Algorithms for 2.5D Roughing Process of Sculptured Surfaces. *Proc. Joint Anglo-Hungarian Seminar on Computer-Aided Geometric Design*, Budapest, Hungary, Oct. 1982.
10. A. V. Deshmukh, M. M. Barash, and H.-P. Wang. On Selection of Tool Path Orientations for Generating Prismatic Features. Report, IE Dept., Purdue Univ., 1993.
11. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, New York, 1999.
12. 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.

13. Graph-Theoretic Algorithms, CS 762, School of Computer Science, University of Waterloo, Canada. <http://www.student.math.uwaterloo.ca/cs762/>.
14. M. K. Guyder. Automating the Optimization of 2.5 Axis Milling. *Proc. Computer Applications in Production and Engineering*, North-Holland, Oct. 1989.
15. M. Held. A Geometry-Based Investigation of the Tool Path Generation for Zigzag Pocket Machining. *Visual Computers*, 7(5-6):296–308, 1991.
16. M. Held. *On the Computational Geometry of Pocket Machining*. Lecture Notes in Computer Science, Vol. 500, Berlin, Springer-Verlag, 1991.
17. M. Held and E. M. Arkin. Letter to Editor: An Algorithm for Reducing Tool Retractions in Zigzag Pocket Machining. *Comput-Aided Design*, 32(10):917–919, 2000.
18. 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. *Proc. International Conf. on Computational Science and Its Applications*, Lecture Notes in Computer Science, Vol. 2669, pp. 680–689, 2003.
19. T. R. Kramer. Pocket Milling with Tool Engagement Detection. *J. Manufacturing Systems*, 11(2):114–123, 1992.
20. D. Lichtenstein. Planar Satisfiability and Its Uses. *SIAM Journal on Computing*, 11:329–343, 1982.
21. P. V. Prabhu, A. K. Gramopadhye, and H.-P. Wang. General Mathematical Model to Optimizing NC Tool Path for Face Milling of Flat Convex Polygonal Surfaces. *Int. J. Production Research*, 28(1):101–130, 1990.
22. B. Reed. Finding Approximate Separators and Computing Tree-width Quickly. *Proc. 24th Annual ACM Symp. on Theory of Computing*, pp. 221–228, 1992.
23. N. Robertson and P. D. Seymour. Graph Minors I: Excluding a Forest. *Journal of Combinatorial Theory, Series B*, 35:39–61, 1983.
24. D. Rose. Triangulated Graphs and the Elimination Process. *J. Math. Anal. Appl.*, 32:597–609, 1970.
25. Y. S. Suh and K. Lee. Neural Network Modeling for Tool Path Planing of the Rough Cut in Complex Pocket Milling. *J. Manufacturing Systems*, 15(5):273–284, 1990.
26. K. Tang. Geometric Optimization Algorithms in Manufacturing. *Computer-Aided Design and Applications*, 2(6):747–758, 2005.
27. K. Tang, S.-Y. Chou, and L.-L. Chen. An Algorithm for Reducing Tool Retractions in Zigzag Pocket Machining. *Comput-Aided Design*, 30(2):123–129, 1998.
28. K. Tang and A. Joneja. Traversing the Machining Graph of a Pocket. *Computer-Aided Design*, 35(11):1023–1040, 2003.
29. H.-P. Wang et al. On the Efficiency of NC Tool Path Planning for Face Milling Operations. *Trans. of the ASME, J. of Engineering for Industry*, 109(4):370–376, November 1987.
30. H. Wang, J. Li, D. Z. Chen, and H. Zhu. An Approximation Algorithm for Tool Retraction Minimization in Zigzag Pocket Machining. Submitted for publication, 2006.
31. T. V. Wimer. *Linear Algorithms on k-Terminal Graphs*. Ph.D. thesis, Clemson University, Dept. of Computer Science, 1987.
32. Z.-Y. Yao and S. K. Gupta. Cutter Path Generation for 2.5D Milling by Combining Multiple Different Cutter Path Patterns. *Int. J. Prod. Res.*, 42(11):2141–2161, 2001.

A Dynamic Program for Simple Pockets

Here we give the full set of recursive formulas for the dynamic program of Section 3.

$$\begin{aligned}
h^0(v_i, v_{j+1}) &= \min\{ h^r(v_i, v_j) + h^l(v_{j+1}, v_{j+1}), \quad \text{use } \textit{bridge}(v_j) \\
&\quad h^r(v_i, v_j) + h^0(v_{j+1}, v_{j+1}) + 1, \quad \text{can't use } \textit{bridge}(v_j) \\
&\quad h^0(v_i, v_j) + h^l(v_{j+1}, v_{j+1}) + 1, \\
&\quad h^0(v_i, v_j) + h^0(v_{j+1}, v_{j+1}) + 1\}. \\
h^r(v_i, v_{j+1}) &= \min\{ h^r(v_i, v_j) + h^{rl1}(v_{j+1}, v_{j+1}), \quad \text{use } \textit{bridge}(v_j) \\
&\quad h^r(v_i, v_j) + h^r(v_{j+1}, v_{j+1}) + 1, \quad \text{can't use } \textit{bridge}(v_j) \\
&\quad h^0(v_i, v_j) + h^{rl1}(v_{j+1}, v_{j+1}) + 1, \\
&\quad h^0(v_i, v_j) + h^r(v_{j+1}, v_{j+1}) + 1\}. \\
h^l(v_i, v_{j+1}) &= \min\{ h^{rl1}(v_i, v_j) + h^l(v_{j+1}, v_{j+1}), \quad \text{use } \textit{bridge}(v_j) \\
&\quad h^{rl2}(v_i, v_j) + h^l(v_{j+1}, v_{j+1}), \\
&\quad h^{rl1}(v_i, v_j) + h^0(v_{j+1}, v_{j+1}) + 1, \quad \text{can't use } \textit{bridge}(v_j) \\
&\quad h^{rl2}(v_i, v_j) + h^0(v_{j+1}, v_{j+1}) + 1, \\
&\quad h^l(v_i, v_j) + h^l(v_{j+1}, v_{j+1}) + 1, \\
&\quad h^l(v_i, v_j) + h^0(v_{j+1}, v_{j+1}) + 1\}. \\
h^{rl1}(v_i, v_{j+1}) &= h^{rl1}(v_i, v_j) + h^{rl1}(v_{j+1}, v_{j+1}). \quad \text{use } \textit{bridge}(v_j) \\
h^{rl2}(v_i, v_{j+1}) &= \min\{ h^{rl2}(v_i, v_j) + h^{rl1}(v_{j+1}, v_{j+1}), \quad \text{use } \textit{bridge}(v_j) \\
&\quad h^{rl1}(v_i, v_j) + h^r(v_{j+1}, v_{j+1}) + 1, \quad \text{can't use } \textit{bridge}(v_j) \\
&\quad h^{rl2}(v_i, v_j) + h^r(v_{j+1}, v_{j+1}) + 1, \\
&\quad h^l(v_i, v_j) + h^{rl1}(v_{j+1}, v_{j+1}) + 1, \\
&\quad h^l(v_i, v_j) + h^{rl2}(v_{j+1}, v_{j+1}) + 1, \\
&\quad h^l(v_i, v_j) + h^r(v_{j+1}, v_{j+1}) + 1\}.
\end{aligned}$$

B Graphs of Bounded Tree-width

A graph $G = (V, E)$ is a k -tree if and only if we can obtain it from a k -clique by always adding nodes such that their neighborhood is a k -clique. 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 vertex 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 vertex $v \in V$, the subgraph of T induced by $\{i \in I \mid v \in X_i\}$ is a connected subtree of T . The *tree-width* of a tree-decomposition (X, T) is $\max_{i \in I} |X_i| - 1$. The tree-width of G is the minimum width over all tree-decompositions of G .

Any graph of bounded tree-width k is in fact a partial k -tree [23, 31]. Partial k -trees generalize tree, and many classes of graphs are partial k -trees for some constant k .

Using dynamic programming techniques, many NP-hard graph problems become polynomially (usually linearly) solvable on partial k -trees, for example maximum independent set, graph coloring, and Hamiltonian circuit [5].

We can check whether a graph G is a k -tree by repeatedly removing a vertex of degree k whose neighbors in the remaining graph form a k -clique, until no such vertex remains; then G is a k -tree if and only if the final remaining graph is a k -clique K_k [24]. This removal process generates a *reduction sequence* of the k -tree: When we remove a vertex v (we say v becomes a *k -leaf* at this point), v is said to be a *descendant* of its k

neighbors that together form a k -clique. If K is a k -clique, then $v \notin K$ is a descendant of K in the reduction sequence if and only if, when v is being removed, each vertex adjacent to v is either a member of K or a descendant of K . The connected components of the subgraph of G induced by the descendants of K are the *branches* on K . In fact, a reduction sequence of a k -tree and a tree-decomposition of the k -tree with tree-width k imply each other, i.e., given a reduction sequence of a k -tree G , we can easily obtain a tree-decomposition of G with tree-width k , and vice versa. The last remaining k -clique K_k is called the *root* or *root clique*. For a vertex v not in the root clique, let $K(v)$ be the set of neighbors of v when v becomes a k -leaf during the reduction process.

C Proof of Lemma 10

Lemma. 10 *If we cut an edge (v, v_1) in D_P at v , then (1) $OPT/\{v, v_1\} \leq OPT + 1$, and (2) from an optimal solution for $P/\{v, v_1\}$ we can obtain a solution for P with at most $OPT/\{v, v_1\} + 1$ retractions.*

Proof. We refer to Fig. 2(b). To prove (1), consider the traversal of the c-edge (a, b) on an optimal machining path p^* for P . If, on p^* , (a, b) is adjacent to neither (a, e) nor (b, f) , then p^* is a feasible solution for $P/\{v, v_1\}$. If (a, b) is adjacent to only one of (a, e) and (b, f) in p^* , say (a, e) , then we break the no-retraction path of p^* containing (a, b) by removing the nc-edge (a, e) from p^* . This gives a feasible solution for $P/\{v, v_1\}$ with $OPT + 1$ retractions. If (a, b) is adjacent to both (a, e) and (b, f) in p^* , then we break the no-retraction path of p^* containing (a, b) into two no-retraction paths: One is the single edge (a, b) , and the other one uses edges (e, c) , (c, d) , and (d, f) (note that these edges together form an nc-edge in $P/\{v, v_1\}$) to replace (e, a) , (a, b) , and (b, f) in p^* . Again, we obtain a feasible solution for $P/\{v, v_1\}$ with at most $OPT + 1$ retractions.

To prove (2), we construct a feasible machining path p for P from an optimal machining path p' for $P/\{v, v_1\}$ with at most one additional retraction. If p' contains the nc-edge (e, c, d, f) of $P/\{v, v_1\}$, as in Fig. 2(b), then we remove this edge from p' , resulting in a feasible machining path p for P with at most $OPT/\{v, v_1\} + 1$ retractions. In the other case, $p = p'$ is already a feasible solution for P . \square