MATHEMATICAL ENGINEERING
TECHNICAL REPORTS

Generation of Cutter Paths for Hard Material

Shinji IMAHORI, Motoki KUSHIYA,
Takeru NAKASHIMA, Kokichi SUGIHARA

METR 2007–20 April 2007

DEPARTMENT OF MATHEMATICAL INFORMATICS
GRADUATE SCHOOL OF INFORMATION SCIENCE AND TECHNOLOGY
THE UNIVERSITY OF TOKYO
BUNKYO-KU, TOKYO 113-8656, JAPAN

WWW page: http://www.keisu.t.u-tokyo.ac.jp/research/techrep/index.html

Generation of Cutter Paths for Hard Material

Shinji Imahori (The University of Tokyo)

Motoki Kushiya (Sumitomo Wiring Systems, Ltd.)

Takeru Nakashima (Sumitomo Electric Hardmetal Co.)

Kokichi Sugihara (The University of Tokyo)

Abstract: A heuristic method for generating a cutter path for superhard material is presented. The cutter path should satisfy various constraints that come from physical reality in industry; for example, a constraint about undesired projections of the cut-off pieces, and that about vibration and deformation generated by the cutter. A large portion of those constraints is represented by a rooted tree called "force-support tree" in direct and concise manner, and the remaining constraints are implemented in the manipulation of this tree. The resulting method can find a good cutter path very quickly.

Keywords: cutter path, wire cutter, industrial constraints, hard material, force support tree, nearly convex decomposition.

1 Introduction

In order to cut hard material such as rocks and stones, we need superhard material, i.e., diamond. In industry, we usually use artificial diamond for the purpose; more precisely, the polycrystalline diamond (PCD in short) is used. Even artificial diamond is very expensive; it is used only for the most important part of a processing tool. Therefore, there arises a problem of how to cut necessary pieces from a diamond plate.

The problem consists of two stages; packing and cutting. In the packing stage, we want to place pieces in a given area as tightly as possible in order to minimize the unused portion of a PCD plate. The packing problem of irregular shapes has been studied very actively in the last decade [1, 2, 3, 4]. This problem is known to be NP-hard; hence, heuristic and metaheuristic algorithms have been proposed for practical use. In the cutting stage, on the other hand, we want to find an optimal cutter path in the sense that the total length of the path is the smallest; actually the cutter cannot move very quickly and hence we cannot neglect the time for cutting. In this paper, the cutting stage is treated; that is, we consider how to cut pieces when we are given a set of small pieces placed in a PCD plate.

The cutter path generation is a variant of the arc routing problem. Some of the arc routing problems (e.g., the Eulerian path problem, the Chinese postman problem) are solvable in polynomial time; however, many arc routing problems are known to be NP-complete or NP-hard. See a book edited by Dror [5] for many variants of arc routing problems and algorithms for those problems. Our cutting problem has some complicated constraints that come from physical reality in industry (which will be explained in the next section), and seems to be very difficult.

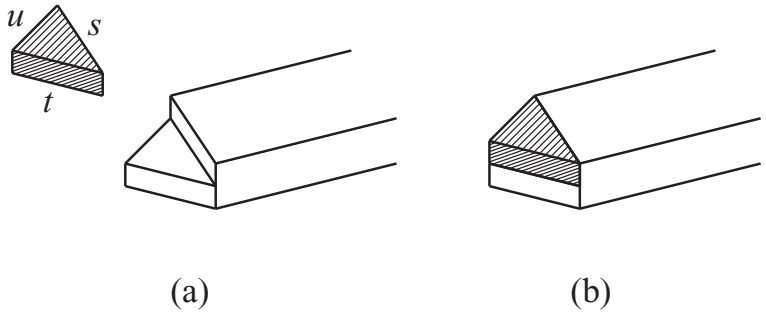

Figure 1: Processing tool made by a superhard cutter piece and a moderately hard supporter: (a) a cutter piece (shaded object) and a supporter; (b) a processing tool consisting of the two parts braced together.

We note that Moreira et al. [6] treated a similar cutting problem to ours; they modelled the problem as a dynamic rural postman problem.

Most of the previous methods tried to formulate the cutting problem as mathematical programming problems [7]. That strategy might be powerful because the methods are general and have a wide range of applications. Moreover, it is possible to use high-performance software tools that have been developed in long history of mathematical programming. However, those approaches have shortcoming because general purpose algorithms are not flexible in utilizing specific characteristics in industrial problems.

In this paper, we take a geometric approach rather than a mathematical programming approach; the constraints are not converted into equalities or inequalities, but they are represented and treated directly as diagrams and the associated adjacency graphs. An advantage of this approach is that we can represent constraints in a straightforward manner, and consequently the search space of the formulated problem is exactly the same as the original; not larger as happens in continuous relaxation, and not smaller as often happens in many heuristic approaches. Furthermore, since the formulation is intuitive, it is easy to introduce heuristic criteria for characterizing subareas of the search area that seems more promising.

The remainder of this paper is organized as follows: In Section 2, we list up constraints that should be satisfied by the cutter path. Section 3 constructs a method for subdividing the connected area of a PCD plate for later purpose, and Section 4 introduces a new concept named "force-support tree" which can represent a large part of the constraints in a concise manner. The entire algorithm is constructed in Section 5, and the performance of the algorithm is discussed in Section 6. The concluding remarks will be given in Section 7.

2 Constraints for the Cutter Path

A processing tool for cutting is usually made by two different parts, a superhard cutter piece and a moderately hard supporter called "carbide substrate," braced together, as shown in Figure 1. In this paper, we are interested in cutting these superhard pieces from a PCD plate.

Suppose that, as shown in Figure 2, we are given a PCD plate whose shape is a circular disk, and a packed configuration of small pieces. We consider how to cut off these pieces from the circular disk. We use a wire cutter (more precisely, a wire electric discharge machine) for the purpose. Since the diamond is very hard, cutting speed of the wire cutter is slow, and hence it is important to find the shortest path for the wire cutter. Note that the circular disk in Figure 2

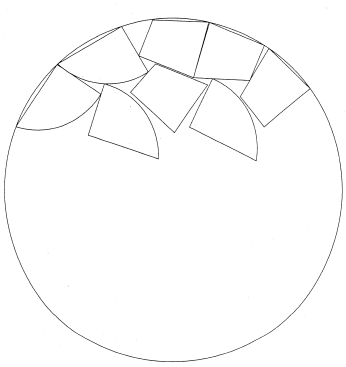

Figure 2: Pieces placed in a circular disk.

contains only a small number of pieces and there is still a large empty region, which is not common in industry. In industry, they usually try to pack as many pieces as possible in a PCD disk. However, we use this simple example throughout the paper in order to avoid unnecessary complication.

Here, we have several assumptions that come from actual situations in industry.

Assumption 1. Each piece forms a simply connected region surrounded by a sequence of edges. The edges are either line segments or circular arcs. If an edge is a circular arc, it protrudes toward outside; that is, the side containing the center of the arc is occupied by the piece and the other side is empty. The number of vertices and edges of each piece is bounded by some constant.

Assumption 2. The pieces are cut by a wire cutter, and the wire has nonzero volume. Hence, the actual cutter path is not an ideal line; instead it has a nonzero width.

Assumption 3. The circular disk is held by a machine while being cut, and hence there is enough area in the unused region at which the disk is held.

When we decide a cutter path on a PCD disk, we should satisfy several constraints that come from physical reality in industry. Hereafter, we list up such constraints. The first constraint comes from the thickness of the wire. In wire cutting, the wire is oriented perpendicular to the PCD disk, and moves along the cutter path. Since the wire has finite volume, the section of the wire is not a point but a circle with a nonzero radius. Therefore, the shapes of the pieces are enlarged by the radius of the wire before they are packed in the disk. However, the shape of a piece actually cut is not the same as the desired shape of the piece in the following sense.

Suppose that we want to cut off the triangle piece shown by solid lines in Figure 3. For this purpose, the triangle is enlarged by the radius of the cutting wire and consequently the center of the wire moves along the broken lines in the figure. Let p_1 be the point at which the wire starts cutting, and assume that the wire moves counterclockwise. Then, the piece is cut off when the center of the wire reaches point p_2 in the figure, at which the wire touches the area occupied by the wire at the initial location. This means that the shaded area in Figure 3 remains uncut, and thus the actually cut off piece is different from the desired piece. We call this remaining portion *projection*. In Figure 4, we can see two types of projections appearing on an edge and at a corner.

Consider the piece shown in Figure 1(a) again. This piece is triangular and has three edges s, t and u. As shown in Figure 1(b), this piece is braced to the supporter along the edge s. In order to brace them tightly, the edge s should have the exactly the same shape as the counter part shape of the supporter. Hence, the projection is not allowed on the edge s. On the other

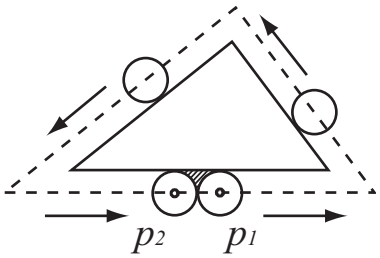

Figure 3: Projection generated by a wire cutter.

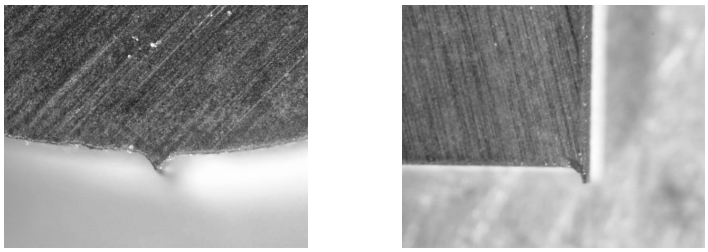

Figure 4: Projections generated on an edge (left) and at a corner (right).

hand, the projection is allowed on the edges t and u because these edges face towered the open space. We note that there is a grinding process after the bracing process, and hence projections on the edges t and u will be removed later. Thus the edges of each piece are classified into *projection-inhibited edges* and *projection-allowed edges* with respect to that piece, and here we have the following constraint.

Constraint 1 (projection constraint). The cutter path should finish cutting of each piece at a projection-allowed edge with respect to the piece.

The second constraint is that the wire cutter should cut each piece completely so that no additional cut is required later. In other words, when a piece is cut off from the circular disk, it must be in its final shape. Thus we have the next constraint.

Constraint 2 (complete-shape constraint). Each piece should be cut off from the PCD disk in its final shape.

The third constraint comes from the thickness of the material (PCD plate). We expect that, when a piece is cut off, it drops from the material. However, a piece sometimes does not drop, but it touches surrounding material and remains hanging. This happens occasionally particularly when a large part of the surrounding area is occupied by remaining material, as shown in Figure 5(a). This situation should be avoided, because the later wire motion might hit and hurt the hanged piece. Thus, there must be large vacant space around a piece, as shown in Figure 5(b), when we cut off the piece. To represent this constraint more formally, we introduce some new concepts.

Let $(e_1, e_2, \ldots, e_k)$ be a circular sequence of edges surrounding a piece counterclockwise. An edge e is said to be *open* if the other side of the edge is vacant, and *closed* if the other side is still occupied with material. The Constraint 2 implies that just before a piece is cut off, the circular sequence of edges is divided into two; the sequence of open edges and that of closed edges. Let e_i and e_j be the first and the last edges of the sequence of closed edges, as shown in

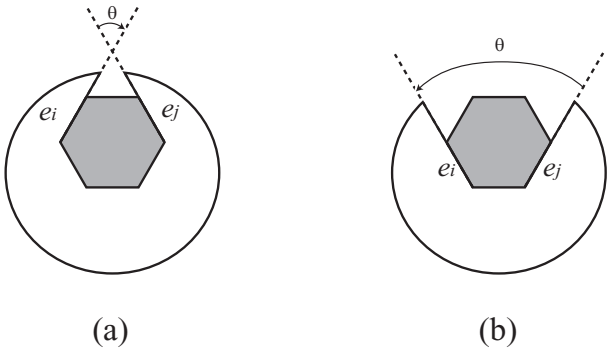

Figure 5: Open angle representing how large part of the surrounding is vacant.

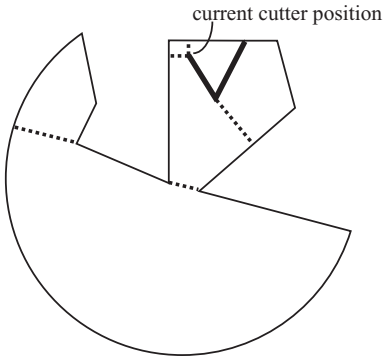

Figure 6: Locally minimal diameters.

Figure 5. We extend e_i and e_j toward the terminal vertices incident to the open edges as shown in the figure. The resulting two half lines form an angle (the angle θ in Figure 5), which we call the *open angle* of the piece. If these two half lines do not cross, as in Figure 5(b), we define the open angle positive, whereas if they cross each other as shown in Figure 5(a), we define the open angle negative. Now, we can express our third constraint in the following way.

Constraint 3 (open-angle constraint). When a piece is cut off, its open angle should be positive.

The next constraint comes from the force generated by the wire cutter. If the wire moves very slowly, it gives almost no force to the material. However, in order to decrease cutting times, the wire moves in its allowed maximum speed, and consequently a certain amount of force is given to the material. Hence the material should be strong enough to support the force and to avoid vibration.

Let p and q be distinct points moving continuously on the boundary of the remaining material, and let $d(p, q)$ denote the Euclidean distance between p and q. The distance $d(p, q)$ is called a *locally minimal diameter* if $d(p, q)$ does not decrease when p and q moves in their neighborhoods.

For example, assume that the material is as shown by solid lines in Figure 6. The locally minimal diameters are generated by line segments represented by broken lines. There are in general many locally minimal diameters. In particularly, when a piece is almost cut, the point at the wire cutter generates locally minimal diameters with some other points. Our fourth constraint can be represented in the following way.

Constraint 4 (force-support constraint). All the locally minimal diameters of the remaining material, except for the diameters generated by the current point of the wire, should be greater than the prespecified threshold d_0.

As we can see in Figure 2, there are unused regions surrounding the pieces. In order to satisfy Constraint 3, vacant space should be generated around pieces, and consequently not only the pieces but also some part of the unused regions should be cut off. Thus, unused regions are partitioned into smaller subregions and some of them are cut off. In this context we need the next constraint.

Constraint 5 (open-angle constraint for unused subregions). When an unused subregion is cut off, its open angle should be positive.

In what follows, we consider how to generate a good cutter path. The input data are a placement of pieces inside a circular region, and the labels of edges, "projection-allowed" or "projection-inhibited." We have to find a cutter path that satisfies all the five constraints.

3 Partition of Unused Regions into Nearly Convex Subregions

As we have seen, the unused regions surround most of the pieces, and make the open angles of these pieces negative. Hence, we should divide the unused regions into small subregions and cut off some of them to make the open angles of pieces positive.

Let us fix a small positive constant ε; for example $\varepsilon = 0.1$. Let X be a two-dimensional region bounded by a sequence of line segments and circular arcs. We say that X is *almost convex* if all the inner angles are less than or equal to $\pi + \varepsilon$. A vertex whose inner angle is greater than $\pi + \varepsilon$ is called *strongly reflex vertex*.

We partition the unused regions into almost convex subregions. To this end we do the following. Let v be a strongly reflex vertex of region R. Let s be the inner angle at v, and define t by $t = s - (\pi + \varepsilon)$. As shown in Figure 7, we divide the fan-shaped area at v into three fan-shaped areas with angles $t, \pi + \varepsilon - t$ and t in this order, and call the central fan-shaped area the *feasible fan-shaped area*. Suppose that we divide the region R into two smaller regions R_1 and R_2 by cutting R along a line segment that connects v and a point on the boundary of R and that is contained in the feasible fan-shaped area. Then, the strongly reflex vertex v is partitioned into two vertices one belongs to R_1 and the other belongs to R_2, neither of which is strongly reflex. Note that this cut does not generate any new strongly reflex vertex. Hence, the partition of the region R into R_1 and R_2 always decreases the number of strongly reflex vertices at least by one.

Now, an algorithm for partitioning a region into almost convex subregions is described as follows.

Algorithm 1 (partition into almost convex subregions)

Input: A region bounded by a sequence of line segments and circular arcs.
Output: Partition of the region into almost convex subregions.

Procedure:

1. Initialize the storage S so that the given region is the only element of S.
2. If all the regions in S are "almost convex", then report S and stop. Otherwise, choose and delete a region, say R, that is not almost convex from S.
3. Choose a strongly reflex vertex, say v, of R.

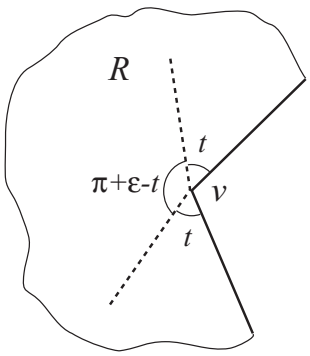

Figure 7: Partition the fan-shaped area at vertex v into three fan-shaped areas.

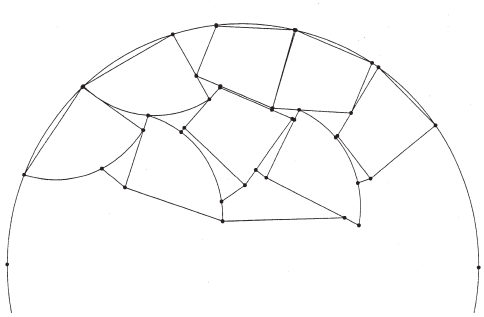

Figure 8: Result of the partition of the unused regions in Figure 2.

4. Find the point p on the boundary of R that is in the feasible fan-shaped area and that is nearest to v.
5. Cut R along the line segment connecting v and p into two subregions and put them into the storage S. Go to Step 2. □

Let m be the number of vertices of the given region, and k be the number of strongly reflex vertices. For each strongly reflex vertex, the cutting line can be found in Step 4 in $O(m)$ time. For each cutting in Step 5, the number of strongly reflex vertices decreases at least by one. Hence, in $O(km)$ time all the strongly reflex vertices disappear. The total computation time to make all unused regions to convex subregions is $O(n^2)$ time.

Figure 8 shows the result of application of a slightly modified version of Algorithm 1 to the unused regions in Figure 2. Actually, Algorithm 1 is modified so that long cuts in the lower area are removed. This is because the lower wide unused area can be used later for other pieces, and hence we want to keep that area as wide as possible.

By using Algorithm 1, we can partition unused regions into almost convex subregions, and thus the circular disk is partitioned into pieces and unused subregions. All the edges introduced by the partition are assigned the label "projection-allowed" with respect to the both side subregions. In what follows we do not distinguish between the pieces and the unused subregions, we call both of them as *parts*.

4 Force-Support Tree

Now we are given a partition of a circular disk into parts (i.e., pieces and unused subregions). Our goal is to find the shortest cutter path that satisfies all the constraints listed in Section 2.

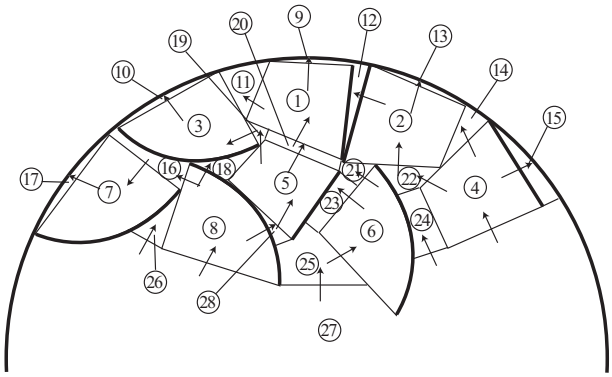

Figure 9: Force-support tree for the parts given in Figure 8; an arrow implies that the tail is a parent node and the head is a child node, and thick lines represent projection-inhibited edges with respect to the associated pieces.

Let n denote the number of pieces. Then, the number of parts is $\mathrm{O}(n)$ since the number of vertices and edges of each piece is bounded by some constant denoted in Assumption 1. Here, we assume that the number of vertices and edges of each part is also bounded by some constant.

A naive method might be to enumerate all the permutations of the parts, to compute the length of the cutter path that cuts the parts in the given order if exists, and to select the shortest one. However this is not practical because the number of the permutations is the exponential order of n.

Actually the problem itself seems very hard in the sense that it is very likely impossible to find the exactly optimal solution in polynomial time of n. Hence, it is reasonable to search for effective heuristics to find a nearly optimal solution.

A simple heuristic method might be to repeat finding the cuttable part nearest from the current location of the wire cutter, and to backtrack if we are stuck. This method also seems impractical because our constraints are very complicated and there is no guarantee of success in finding a possible cutter path within reasonable time.

Therefore, we need some heuristic mechanism in order to restrict the search area into the paths that are feasible with respect to the constraints. For this goal we introduce the next concept. For a given partition of the circular disk into parts, we consider a rooted tree; called *force-support tree*, having the node set N and the arc set A that satisfies the following three conditions:

Condition 1. The node set N is the set of all the parts in the circular disk;

Condition 2. The root corresponds to the largest unused subregion;

Condition 3. If part P_1 is the parent of part P_2, then P_1 and P_2 share an edge whose length is greater than the threshold d_0 (introduced in Constraint 4), and this edge is projection-allowed with respect to the part P_2.

Figure 9 shows an example of the force-support tree for the parts given in Figure 8. The diagram in Figure 9 is deformed in order to make smaller parts more visible. The pieces are numbered from 1 to 8, and the unused subregions are numbered from 9 to 28. The part 27 is the largest unused subregion, and hence it is chosen as the root of the force-support tree. Arrows in the figure represent the tree arcs in such a way the tail represents the parent node and the head represents the child node. The thick lines represent projection-inhibited edges with respect to the associated pieces. Note that no arrow crosses thick lines toward the interior

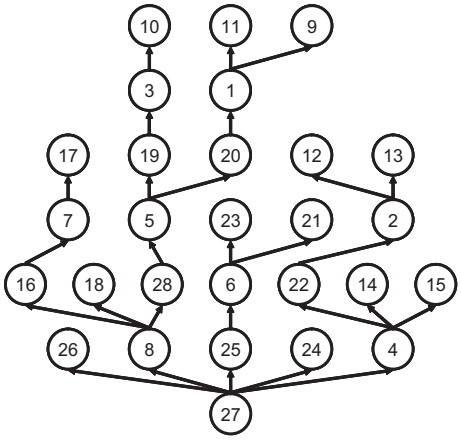

Figure 10: Another representation of the force-support tree in Figure 9.

of the pieces. Some arrows cross thick lines toward outsides the pieces; this does not violate Condition 3, because these edges are projection-inhibited with respect to the parent parts, but not for the child parts. Figure 10 shows a graph structure of the same force-support tree as that in Figure 9. A force-support tree can be constructed by the next algorithm.

Algorithm 2 (force-support tree)

Input: Partition of the circular disk into parts (pieces and unused subregions), and the edge labels, "projection-allowed" and "projection-inhibited," with respect to each side of the edge.

Output: Force-support tree.

Procedure:

1. Find the largest unused subregion, and name it as the root.
2. Put the root node into storage S_1, and put all the other parts into storage S_2.
3. Choose and delete a part P from S_2 that shares an edge with a part P' in S_1 such that the edge is projection-allowed with respect to P and the length of edge is larger than d_0. If there is no such a part in S_2, terminate the procedure with the message "the force-support tree does not exist."
4. Augment the tree by adding P as a child node of P', and put P into S_1.
5. If S_2 is empty, report the tree and stop. Otherwise go to Step 3. □

If there are some feasible force-support trees, our algorithm can find one of them. Here, we have freedom in the choice of a part P in Step 3. If we choose a part arbitrarily, Algorithm 2 runs in $O(n)$ time. One possible criterion to find a good force-support tree is to choose as P the piece with the longest edge. In this case, the storage S_2 is implemented by a heap and consequently requires $O(\log n)$ time for a selection of P; hence total computation time becomes $O(n \log n)$. Another criterion is to choose a piece P that shares an edge with a part P', where P' has the maximum number of child nodes. Using this criterion, a tree with many leaves will be found.

Assume that a force-support tree is given. Suppose that we cut off a part that corresponds to one of leaf nodes in such a way that the edge corresponding to the tree arc is cut most lately,

and that no other tree arc is cut. This cutting satisfies Constraints 1, 2 and 4 automatically, as can be seen in the following way. First, this cutting satisfies Constraint 4 because all the other tree arcs are kept uncut and their lengths are greater than d_0 (Condition 3). Secondly, the tree arc is a projection-allowed edge with respect to the child part (Condition 3), and hence the latest cut of this edge satisfies Constraint 1. Finally, only one part is cut off, and hence Constraint 2 is satisfied.

Note that the use of a force-support tree does not restrict the search area for the cutter path, because every cutter path subject to Constraints 1, 2, 3, 4 and 5 is associated with a force-support tree. Indeed we can construct the corresponding force-support tree from the cutter path in the following way. For each part P, let e be the edge that cut most lately, let P' be the part that is in the other side of e, and generate an arc from a parent node P' to a child node P. Thus we get a rooted tree, which is nothing but a force-support tree.

5 Cutter Path Generation

According to the basic strategies considered in the previous sections, we can construct a heuristic algorithm for generating a good cutter path that satisfies all the five constraints. A rough description of the entire algorithm is as follows.

Algorithm 3 (cutter path)

Input: Nonoverlap placement of n pieces in a circular disk, and the distinction of edge types, i.e., "projection-inhibited" or "projection-allowed" with respect to each piece.

Output: Cutter path that satisfies Constraints 1, 2, 3, 4 and 5.

Procedure:

1. Partition the unused regions into nearly convex subregions.
2. Generate a force-support tree. If the tree cannot be generated, stop the procedure with the message "force-support tree cannot be generated."
3. Initialize the cutter position as a point on the boundary of the disk, and initialize the cutter path as empty.
4. Among the leaves of the force-support tree that satisfy the open-angle constraint, find the one that is nearest to the current position of the wire cutter. If there are no such leaves, stop the procedure with the message "cutter path cannot be found."
5. Augment the cutter path by adding the shortest path from the current cutter position to the leaf node, and by adding the path cutting the piece corresponding to the leaf node. Update the current cutter position to the terminal point of the augmented path.
6. Remove the leaf from the force-support tree.
7. If the force-support tree consists of only the root node, report the cutter path and stop. Otherwise go to Step 4. □

The time complexity of Algorithm 3 is $O(n^2)$. There are two cases where the algorithm fails in generating the cutter path. The first case is that the force-support tree cannot be generated at Step 2. In this case, we have to change the packing pattern of the pieces, and to try again.

The second case is that all the leaves of the force-support tree have negative open angles at Step 4. In that case, we should generate another force-support tree. However, another option to continue our algorithm is to choose the leaf node with the maximum open angle. This is because in many cases a piece drops (instead of being hung) even if the open angle is negative.

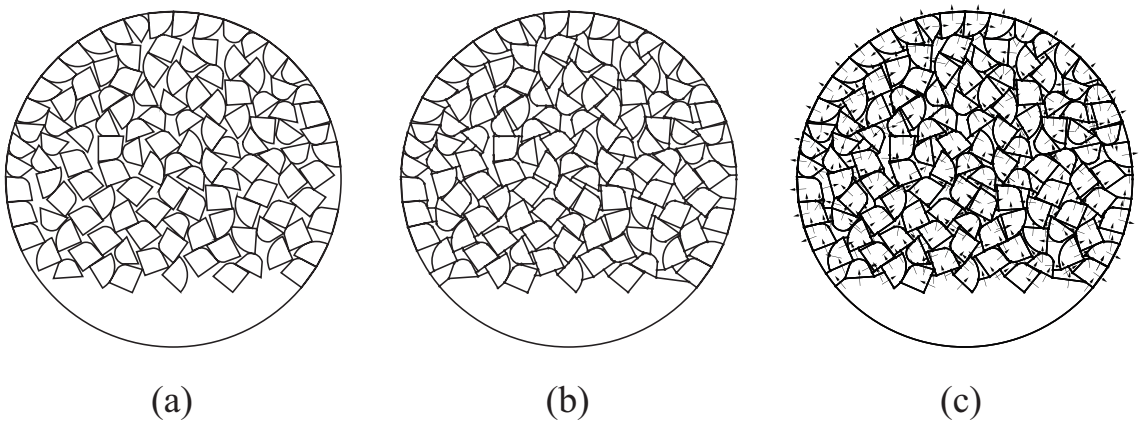

Figure 11: An example of the data used in our experiment: (a) a placement of 120 pieces; (b) partition of the unused regions; (c) force-support tree.

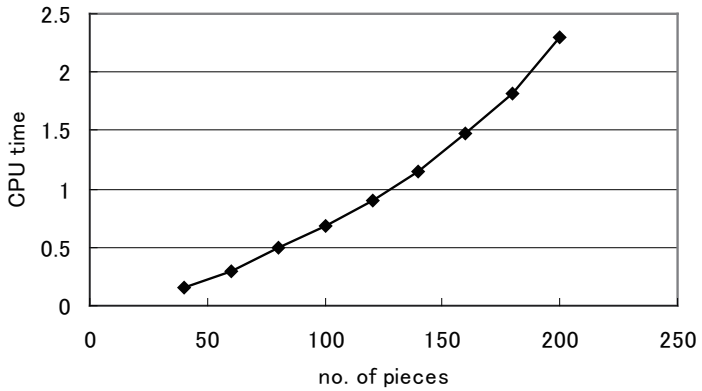

Figure 12: CPU time for generating the cutter path.

6 Experimental Evaluation of the Performance

In order to see the performance of the proposed method, we generated cutter paths for many patterns of packed pieces with various sizes. Figure 11 shows an example of the data used in our experiment. Figure 11(a) shows 120 pieces packed in a disk. The reader might feel that the package pattern in this figure still has unnecessarily large space among pieces. Actually this paper concentrates on the method for generating the cutter path; the good-quality packing itself is one of our future problems. Although the quality of package pattern shown in this figure is rather poor, it still requires much time to generate, 10^3 times or more than the time required for computing the cutter path.

Figure 11(b) shows the result of the partition of the surrounding unused area into almost convex subregions, and (c) shows the force support tree where the arrows represent the arcs of the tree such that they cross the associated edges perpendicularly from the parent regions toward the child regions. Since some of the regions are too small, the start and end points of an arrow are not necessarily placed in the corresponding parent and child regions.

Using this kind of data, we measured the CPU time for finding the cutter paths. The computer used for this experiments was Ultra 45 workstation of Sun Microsystems with SunOS

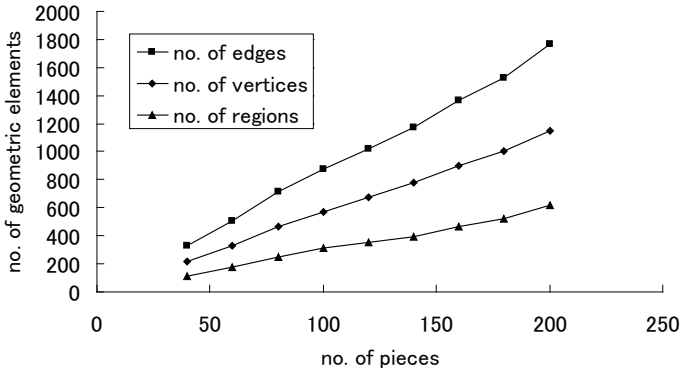

Figure 13: Complexity of the geometric graph composed of the pieces and the nearly convex unused regions.

5.10 and 1024 MB RAM. We generated 10 package patterns for each number of pieces, and computed the cutter paths. The average CPU times are shown in Figure 12, where the horizontal axis represents the number of pieces whereas the vertical axis represents the CPU time in seconds. As we saw in the previous section, the theoretical time complexity of our method is $O(n^2)$. The graph in Figure 12 shows that the actual CPU time obeys this theoretical time complexity.

Figure 13 shows the complexity of the geometric graph consisting of the pieces and the nearly convex subregions. The horizontal axis represents the number of pieces, and the vertical axis represents the complexity of the geometric graph in terms of the number of edges, that of the vertices and that of the regions. These data are also the average of 10 different data for each number of the pieces. The graph shows that all the three numbers grow almost linearly with respect to the number of the pieces.

In order to see the quality of the generated cutter paths, we measured various lengths of the paths. The result is summarized in Figure 14. The horizontal axis represents the number of pieces, and the vertical axes represent the lengths of the paths and their ratios. In this figure, there are five polygonal lines. The upper three polygonal lines represent the length of the paths. The topmost polygonal line represents the total length of the paths along which the cutter moves; in some portion of the path the cutter actually cuts the material (we call it a *cutting motion*), while in the other portion the cutter just moves toward the start point of the next cutting (we call it an *empty motion*).

The second polygonal line from the top represents the length of the cutting motion, and hence the difference of the top and the second polygonal lines corresponds to the empty motion. The third polygonal line from the top represents the total length of the boundaries of the pieces, and hence the difference between the second and the third polygonal lines corresponds to the additional cutting motion for cutting the unused regions into almost convex subregions.

These three polygonal lines are not monotone increasing. This is because we sometimes switched to smaller pieces; otherwise all the pieces cannot be packed into the given disk. Therefore the ratios represented by the other two polygonal lines might be more informative than the absolute values of the lengths.

The lower two polygonal lines represents the ratios of the path lengths. The lowest one represents the ratio of the length of the cutting motion over the total length of the boundaries of the pieces. This ratio is roughly 1.1, independent of the number of pieces. This means that about 10% additional cutting motion is required in order to cut the unused regions into almost

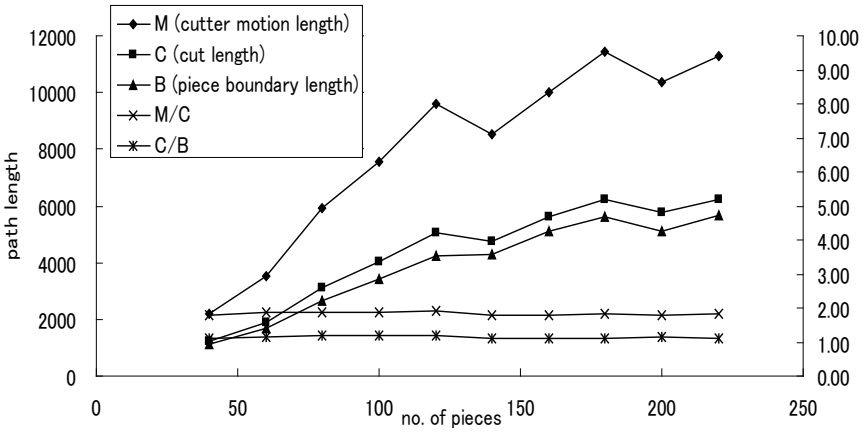

Figure 14: Lengths of the generated cutter paths.

convex subregions.

The second polygonal line from the bottom represents the ratio of the length of the cutter path over the length of the cutting motion. This ratio is around 1.8 to 1.9, independent of the number of pieces. This implies that the cutter need about 80% to 90% of additional empty motion to move from the end point of a cut action to the start point of the next cut action. Note that the cutter can move much faster in its empty motion than in the cutting motion. Hence, this ratio is not so important.

According to these experimental data, we can say that the proposed method can generate a feasible cutter path in $O(n^2)$ time for n pieces, and the resulting path contains about 10% additional cutting motion for cutting the unused region into nearly convex subregions.

7 Concluding Remarks

We have constructed a heuristic method for generating a cutter path for diamond pieces used as the top of cutting tools. The desired cutter path should satisfy several complicated constraints due to physical reality in industry. To achieve our goal we introduced a new and powerful concept named "force-support tree," and search for a cutter path that cuts pieces for the leaf nodes of this tree toward the root node. By this strategy, most of the constraints are automatically satisfied, and consequently we can concentrate only on open-angle constraints.

The total process for cutting small pieces from a PCD plate consists of two stages, packing the pieces into a disk and cutting them apart from the disk. In this paper we solved the latter problem. Hence the next problem we have to attack is the former. Actually, packing required large computational cost when compared with the cutting; if we use some existing algorithms for packing phase and employ the algorithm proposed in this paper for cutting phase.

Another work for future is the stability of the computation against numerical errors. Indeed the packed pieces are placed in highly degenerate positions because they touch each other very often, and hence computation becomes unstable due to degeneracy [8]. A standard method to cope with degeneracy is the symbolic perturbation [9, 10, 11, 12]. However, this technique is useless because most algorithms for packing phase generate degenerate situation intentionally, and hence the degeneracy should not be destroyed. Thus we need some other strategy to make the computation robust against numerical errors.

Acknowledgements

This work is partly supported by the 21st Century COE Programs on Information Science and Technology Strategic Core of the University of Tokyo, and the Grant-in-Aid for Scientific Research (S).

References

[1] E. K. Burke, R. Hellier, G. Kendall and G. Whitwell, "A new bottom-left-fill heuristic algorithm for the 2D irregular packing problem," *Operations Research*, 54 (2006) 587–601.

[2] K. A. Dowsland, S. Vaid and W. B. Dowsland, "An algorithm for polygon placement using a bottom-left strategy," *European Journal of Operational Research*, 141 (2002) 371–381.

[3] J. Egeblad, B. K. Nielsen and A. Odgaard, "Fast neighborhood search for two- and three-dimensional nesting problems," *European Journal of Operational Research*, in press.

[4] A. M. Gomes and J. F. Oliveira, "Solving irregular strip packing problems by hybridising simulated annealing and linear programming," *European Journal of Operational Research*, 171 (2006) 811–829.

[5] M. Dror, *Arc Routing: Theory, Solutions and Applications*, Kluwer Academic Publishers, 2000.

[6] L. M. Moreira, J. F. Oliveira, A. M. Gomes and J. S. Ferreira, "Heuristics for a dynamic rural postman problem," *Computers & Operations Research*, in press.

[7] A. Pott and H. Glaab, "Optimization problems in a semi-automatic device for cutting leather," in *Mathematics - Key Technology for the Future: Joint Projects Between Universities and Industry*, W. Jaeger and H. J. Krebs (eds.), 609–622, 2003.

[8] K. Sugihara, "How to make geometric algorithms robust," *IEICE Transactions on Information and Systems*, E83-D (2000) 447–454.

[9] H. Edelsbrunner and E. P. Mücke, "Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms," *ACM Transactions on Graphics*, 9 (1990) 66–104.

[10] I. Z. Emiris and J. F. Canny, "A general approach to removing degeneracies," *SIAM Journal on Computing*, 24 (1995) 650–664.

[11] R. Seidel, "The nature and meaning of perturbations in geometric computing," *Discrete and Computational Geometry*, 19 (1998) 1–17.

[12] C.-K. Yap, "Symbolic treatment of geometric degeneracies," *Journal of Symbolic Computation*, 10 (1990) 349–370.