

Algorithms for Computing the Maximum Weight Region Decomposable into Elementary Shapes

Jinhee Chun, Ryosei Kasai, Matias Korman, Takeshi Tokuyama*

Graduate School of Information Sciences, Tohoku University, Japan
(jinhee,ryosei,mati,tokuyama@dais.is.tohoku.ac.jp)

Abstract. Motivated from the image segmentation problem, we consider the problem of finding the maximum weight region with a shape decomposable into elementary shapes in $n \times n$ pixel grid where each pixel has a real valued weight. We give efficient algorithms for several interesting cases. This shows strong contrast to NP-hardness results to find the maximum weight union for the corresponding cases.

1 Introduction

Let \mathbf{P} be an $n \times n$ pixel plane, and consider a family $\mathcal{F} \subset 2^{\mathbf{P}}$ of *pixel regions*. A pixel of \mathbf{P} is the unit square $p(i, j) = [i - 1, i] \times [j - 1, j]$ where $1 \leq i \leq n$ and $1 \leq j \leq n$. The pixel p at the (i, j) position in the grid has a real value $W(p) = W(i, j)$ called the weight of p . We can regard the array $(W(p))_{p \in \mathbf{P}}$ as a real-valued matrix $W = (W(i, j))(1 \leq i, j \leq n)$ where we count the indices of rows from bottom to top. For conveniences' sake, we define $W(0, j) = W(n + 1, j) = W(i, 0) = W(i, n + 1) = 0$ for each i and j . We consider the following *maximum-weight region* problem:

Find a region $R \in \mathcal{F}$ maximizing $W(R) = \sum_{p \in R} W(p)$.

The maximum-weight region problem is considered in several applications such as image processing [1], data mining [2, 3], surface approximation [4, 5, 7], and radiation therapy [5]. The difficulty of the problem depends on the family \mathcal{F} . If $\mathcal{F} = 2^{\mathbf{P}}$, the problem is trivial, since R is obtained as the set of all pixels with positive weights. On the other hand, if \mathcal{F} is the set of all connected regions in \mathbf{P} where we consider usual topology of \mathbf{P} such that each pixel is connected to its four neighbors, the problem is NP-hard [1].

The following is a list of previously known families for which the maximum-weight regions can be computed efficiently (definitions will be

* Partially supported MEXT Grant on fundamental research (B) 18300001.

given later): x -monotone regions, based monotone regions, rectilinear convex regions, staircase convex regions centered at a pixel r (called stabbed union of rectangles in [4, 7]), and digital star-shaped regions [6]. More generally, we can solve the problem if \mathcal{F} can be represented as the family of closures in a graph defined on \mathbf{P} (see [5, 9]), and the above families can be treated in this framework (although it might not lead to the best algorithms).

One natural question is to solve the maximum weight region problem for more general regions. In particular, we consider the problem of finding the maximum weight region constructed from more than one basic regions in families in the above list. One attempt is to compute the maximum weight region represented as a union of basic regions given in the above list. Unfortunately, we have a negative result that it is NP-hard to compute the maximum weight region represented as a union of a based x -monotone region (based monotone region with the x -axis as its base line) and a based y -monotone region, which is considered to be a fundamental combination. It is NP-hard to have any finite ratio approximation algorithm for computing the maximum weight region represented as a union of two digital star-shaped regions with given two centers.

Nevertheless, we can consider a different formulation to have tractable computational problems. In this paper, we consider a region decomposable into basic regions (i.e., represented as a disjoint union of basic regions), and give novel algorithms listed as follows:

- (1). Given k axis parallel base lines, the maximum-weight region decomposable into base monotone regions corresponding to the base lines can be computed in $O(N^{1.5})$ time, where $N = n^2$ is the number of pixels.
- (2). If we consider k base segments instead of lines, we give a FPT algorithm for a special case and an $n^{O(k)}$ algorithm for the general case.
- (3). The maximum weight region decomposable into digital star shaped regions with different centers can be computed in $O(N^3)$ time.

We also show the maximum-weight region decomposable into k staircase convex regions or k rectilinear convex regions can be computed in polynomial time if k is a constant, and also the union problem can be also solved in a similar fashion for these regions.

1.1 Motivating application to image segmentation

Separating an object in an image from its background is a central problem in pattern recognition and computer vision. This operation is commonly called *image segmentation* and many practical methods are proposed in

the literature. Consider a pixel plane \mathbf{P} representing a (say, monochromatic) picture, where each pixel p has a real value $f(p)$ represents the brightness level of the pixel p . The segmented image should be a pixel region with a nice geometric property, and the quality of the segmentation depends on the separation of brightness levels in the image and background. However, it is nontrivial to formulate the image segmentation into a nice optimization problem in the sense of the output quality and computational complexity.

Asano et al. [1] proposed an *optimization-based* image segmentation method that gives a robust solution with theoretical guarantee. The general mathematical framework defines a family \mathcal{F} of grid regions, and finds the region $R \in \mathcal{F}$ maximizing an objective function $\Phi(R)$. The function Φ needs a kind of convexity (Asano et al. [1] particularly considered the *intra-class variance*), and solved the problem via a parametric optimization framework, where a probing algorithm finds the optimal solution by solving the following key problem for $O(N)$ different parameter values of θ found during the runtime of the algorithm.

Key problem: Compute the region $R \in \mathcal{F}$ maximizing $\sum_{p \in R} (f(p) - \theta)$.

Once the parameter value θ is given, we can replace $f(p)$ by $W(p) = f(p) - \theta$, and the key problem is maximizing the sum $W(R) = \sum_{p \in R} W(p)$, which is the maximum weight region problem. Segmentation in color pictures can be also reduced to the maximum weight region problem by using a three-dimensional parameter space.

Our positive results imply that several kinds of shapes decomposable into two or more fundamental shapes can be treated in Asano et al.'s framework. This allows a variety of objects to be handled in a robust fashion, and the authors believe that it gives significant advancement to the theoretical aspect of image segmentation problem. On the other hand, the NP-hardness says that it is difficult to segment an object that is an overlay of two highly intersecting basic objects. Thus, the first picture of Figure 1 is difficult to segment, while the second picture is easy to segment since it can be decomposed into two star shaped regions as shown in the rightmost picture.



Fig. 1. Union and decomposition

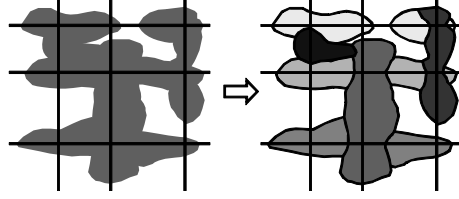


Fig. 2. A feasible region and its decomposition



Fig. 3. A segmented flower

2 Regions decomposable to based monotone regions

A base line of the pixel grid \mathbf{P} is a vertical line $x = i$ or horizontal line $y = i$ where $0 \leq i \leq n$. For a given horizontal base line $\ell : y = i$, its *based monotone region* is the region $\{p(s, j) : 1 \leq j \leq n, g(j) < s \leq f(j)\}$ for a suitable pair of functions $g(j) \leq i \leq f(j)$. In other words, it is a union of segments of columns intersecting the base line. It is a special case of *x-monotone region*, where the intersection with each column can be any connected segment. Note that the j -th column part is empty if $g(j) = f(j)$, thus we do not assume connectivity of the region. The vertical base line case is analogously defined. A based monotone region with the base line $x = 0$ (resp. $y = 0$) is often called *based x-monotone* (resp. *based y-monotone*) region.

A based monotone region is divided by the (horizontal) base line ℓ into the based monotone regions of the upper and lower halfplanes of ℓ : They are the pixel region defined by $\{p(s, j) : 1 \leq j \leq n, i \leq s \leq f(j)\}$ and $\{p(s, j) : 1 \leq j \leq n, g(j) < s \leq i\}$, respectively. The family of base monotone region of the upper and lower half planes of ℓ are denoted by $\mathcal{U}(\ell)$ and $\mathcal{D}(\ell)$, respectively (meaning that each column grows upward and downward from the base line, respectively). Similarly, for a vertical base line m , we define families $\mathcal{L}(m)$ and $\mathcal{R}(m)$ of base monotone region in the left and right halfplanes of m , respectively.

Given a set of k base lines, a region R is called a feasible region if it can be decomposed into base monotone regions with the base lines. Figure 2 shows a feasible region of given six base lines, and Figure 3 gives intuition of a segmentation using four grid boundary lines as base lines.

2.1 Room-edge problem

First, we consider a special case (called room-edge problem) where we are given the four (or less) boundary lines of \mathbf{P} as the set of base lines. See Figures 3 and 4. We abbreviate \mathcal{U} , \mathcal{D} , \mathcal{L} and \mathcal{R} for the families of base monotone regions of upper, lower, left, and right halfplanes with respect to

Proof We compute the table $UR(*, *)$ by dynamic programming. We classify the optimal painting of $UR(i, j)$ as shown in figure 5: We have $UR(i, j) = UR(i, j - 1) + U(i, j)$ if $(i, j) \in \mathbf{U}$ (Figure 5, left), while we have $UR(i, j) = UR(i - 1, j) + R(i, j)$ otherwise (Figure 5, center or right). Thus, we have $UR(i, j) = \max\{UR(i, j - 1) + U(i, j), UR(i - 1, j) + R(i, j)\}$. Thus, the dynamic programming using the recursive formula computes the table UR in linear time from precomputed tables U and R . \square

The optimal region attaining $UR(n, n)$ is given by backtracking.

Next, let us consider painting with three colors. We precompute tables UR etc. in $O(N)$ time. Suppose we paint from bottom, top and left edges to maximize $W(\mathbf{U} \cup \mathbf{D} \cup \mathbf{R})$. Let $UDR(j)$ be the maximum weight of coloring the pixels in the first j columns by three colors.

Theorem 2. *We can compute $UDR(j)$ for all $1 \leq j \leq n$ in $O(N)$ time.*

Proof If the j -th column of the region attaining $UDR(j)$ does not intersect the left region \mathbf{R} , we have $UDR(j) = UDR(j - 1) + UD(j)$, where $UD(j)$ is the maximum weight painting of the j -th column from top and bottom. Otherwise, the picture is divided into upper half and lower half by the intersecting row of \mathbf{R} , and each half is painted by two colors. Thus, we have $UDR(j) = \max_{0 \leq i \leq n} \{DR(i, j) + UR(i + 1, j)\}$ for this case. Since we prepared the tables UD , DR , UR , the above formula can be computed in $O(n)$ time for each j . Hence, each new entry of UDR can be computed in $O(n)$ time by taking the maximum of the above two cases, and the table $UDR(*)$ is computed in $O(n^2) = O(N)$ time. \square

Finally, we consider painting by four colors from four edges. If there exists a vertical line $x = j$ separating \mathbf{R} and \mathbf{L} , we can decompose the problem into two instances of the three-color paintings. Thus, the optimal value for this case is computed as $\max_{0 \leq j \leq n} \{UDR(j) + UDL(j + 1)\}$, where $UDL(j + 1)$ is the optimal region of UDL painting of the region to the right of the partition line $x = j$. This type of the solution region can be computed from the arrays UDR and UDL for the three-color paintings in $O(N)$ time. Similarly, we can solve in $O(N)$ time if there exists a separating horizontal line.

Thus, we need to consider the case where the solution does not allow such a partition line. We guess the longest column/row length of each colored region, and decompose the pixel plane into five rectangular parts as shown in the left picture of Figure 6. The center rectangle cannot be painted by any color, since each color is blocked by another region to paint. Moreover, in each other rectangular part, the painting is done by two colors, and can be done independently of the painting of other regions.

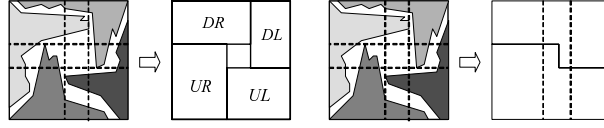


Fig. 6. Decomposition into two-colored rectangles and L-shaped regions

Therefore, we can obtain the optimal four-color painting by combining the two-color paintings of these four rectangular parts. The optimal value is obtained by combining optimal solution of each solution in $O(1)$ time (since tables DL, DR, UL, UR are precomputed) for each possible partitions. Since there are $O(n^4) = O(N^2)$ possible partition patterns, we can solve compute the maximum weight region for the room-edge problem in $O(N^2)$ time with $O(N)$ space.

The time complexity can be improved to $O(N^{1.5})$ if we can use $O(N^{1.5})$ space. We use another decomposition as shown in the right picture of Figure 6, where the pixel grid is decomposed into two L-shaped regions in which each region is painted by using three colors. There are four types of L-shaped regions, each of which is union of two rectangles containing a corner of the grid. Let us focus on the L-shaped region $\mathcal{L}(s, t, u)$ that is a union of two rectangles containing the left-lower corner: the tall rectangle has height s and width u , and the short rectangle has a height t . Other types can be handled analogously.

We define $F(s, t, u)$ to be the optimal three color painting (from both sides and the bottom edge). We have the following recursive formula

$$F(s, t, u) = \max\{F(s-1, t, u) + R(s, u), UR(s, u) + UL(t, u+1), F(s, t, u-1)\}.$$

The first case is where the s -th row is not penetrated by **U**; thus, the row can only be reached from the left, and painted in the color of **R**. The second case is where the u -th column is not penetrated by the **L**, thus we can cut the shape at the u -th column to have two 2-colored rectangles; one has height s , and the other has t . The third case is where the s -th row is penetrated by **U**, and the u -th column is penetrated by **L**; thus, the u -th column beyond the t -th row is blocked, and prohibit to be colored.

We can compute each new entry of the table $F(*, *, *)$ in $O(1)$ time using the precomputed tables, and hence the time complexity for computing all entries of the three-dimensional array F is $O(n^3) = O(N^{1.5})$. We need $O(N^{1.5})$ space to store it. Analogously, we can compute 3D-tables for other types of L-shaped regions. Then, we can compute the weight $UDLR$ of the maximum-weight four-color painting from them in $O(N^{1.5})$ additional time. Thus, we have the following:

Theorem 3. *The room-edge problem can be solved either $O(N^{1.5})$ time and space or $O(N^2)$ time using $O(N)$ space. If we use only three colors, the problem can be solved in $O(N)$ time.*

2.3 Allowing k base lines.

Now, let us consider the case where we are given k base (either vertical or horizontal) lines. We solve the problem of finding the maximum weight (possibly non-connected) region that can be decomposed into base monotone polygons corresponding to the separating lines. The arrangement of k separating lines decompose the pixel grid into $O(k^2)$ cells that are rectangular subgrids.

Lemma 1. *If we consider the union of the optimal solution of room-edge problem of all cells of the arrangement, it is decomposable into base monotone regions of the separating lines, and attains the maximum weight.*

Proof Suppose that a monotone region $R(\ell)$ based by a line ℓ is cut by another base line ℓ' (parallel to ℓ) into $R_1 \cup R_2$ where R_2 is separated from ℓ by ℓ' . Then, we can replace $R(\ell)$ by R_1 and $R(\ell')$ by $R(\ell') \cup R_2$ to obtain a new pair of based monotone regions, where the new $R(\ell)$ does not intersect ℓ' . In this way, we can reform the decomposition such that its each component do not intersect other parallel base lines. Cutting with partiton lines orthogonal to base lines into subregions does not affect because of definition of base monotone regions. Thus, the union of the solutions of the room-edge problem gives the global solution of the problem with k base lines. \square

For each cell G_t , if it has $O(N_t)$ pixels, the room-edge problem inside the cell can be solved in $O(N_t^{1.5})$ time. Since the summation of N_t over all cells is $O(N)$, the total time complexity of solving the room-edge problem insider all cells is $O(N^{1.5})$. Thus, we have the following theorem.

Theorem 4. *The maximum weight region decomposable into based monotone regions of given k base lines can be computed in $O(N^{1.5})$ time.*

Remark. The time complexity is independent of k . We can even take $k = n$ to have all grid lines as base lines so that every region can be decomposable into base monotone regions; however, the optimal solution is unfortunately the trivial set gathering all pixels with positive weights. Therefore, in practice, we should consider the trade-off between the loss of simplicity of the region family and the gain of the weight. Moreover, if the selection of k base lines is not given, we have ${}_{2n}C_k$ possible combinations.

2.4 Allowing k base segments

Let us consider a segment s of the base line ℓ , and suppose that we only consider the monotone regions R such that the intersection of the closure of R and ℓ is contained in s . Then, we call R has s as its base segment. Given a set of base segments, we consider an analogous problem. Without loss of generality, we assume that the segments are mutually nonintersecting in their interior, since we can refine segments if they intersect. The algorithm given in the previous subsection does not work, since we cannot easily decompose the problem into room problems.

We also consider the restricted problem in which the region can be build only on the right side of each vertical base segment and on the upper side of each horizontal base segment. In the terminology of the room-edge problem, we only use two colors of paints. We show that this restricted problem is fixed-parameter tractable, but only give a much slower algorithm for the general one. Proof is given in the appendix because of space limitation.

Theorem 5. *If we are given k nonintersecting base segments and consider based monotone regions in upside (for horizontal segments) and right side (for vertical segments), the optimal region decomposable to these monotone regions can be computed in $O(k^{O(k)}N^2)$ time. If we use four directions, the optimal region is computed in $O(N^{O(k)})$ time.*

3 Digital star-shaped regions

\mathbf{G} is the graph representing the adjacent relation of pixels of \mathbf{P} . A digital ray system is a rooted spanning tree T of \mathbf{G} such that all leaves are located on the boundary of the grid. A digital star-shaped region R (associated with T) is a rooted subtree of the tree T . The path from root to a pixel is called the digital ray, and the digital star-shape region is characterized a region such that for any pixel in the region the digital ray to the pixel is also in the region.

Given two root positions r_1 and r_2 and digital ray system T_1 and T_2 we call a region R is (T_1, T_2) -admissible if it is decomposed into a pair of digital star shaped regions R_1 and R_2 associated with them. Let $\mathbf{p}_1(u)$ and $\mathbf{p}_2(u)$ be the digital rays to the pixel u in T_1 and T_2 , respectively. Then, if $u \in R$, either $\mathbf{p}_1(u) \subset R$ or $\mathbf{p}_2(u) \subset R$. We have the following theorem.

Theorem 6. *The maximum weight (T_1, T_2) -admissible region can be computed in $O(N^3)$ time.*

Proof The idea of the proof is similar to the one of Theore 1, although it is considerably more complicated since we define a new ordering (or "coordinate system") of pixels using T_1 and T_2 to make the dynamic programming table. We can assume that none of R_1 and R_2 is the empty set. The initial path in R_1 is defined by $\mathbf{e}_1 = R_1 \cap \mathbf{p}_1(r_2)$. Let z be the terminal of \mathbf{e}_1 , and let $\mathbf{e}_2 = R_2 \cap \mathbf{p}_2(z)$ be the initial path in R_2 . Although we do not know R_1 nor R_2 in advance, we pick a pair of candidates of initial paths \mathbf{e}_1 and \mathbf{e}_2 among $O(n^2)$ possibilities.

We say a pixel v is T_1 -prohivited if the path $\mathbf{p}_1(v)$ contains r_2 . Also, a pixel v' is T_2 -prohibited if the path $\mathbf{p}_2(v')$ contains r_1 . Then, it is observed that T_1 -prohibited pixel cannot be in R_1 , and T_2 -prohibited pixel cannot be in R_2 . The union of the set of all prohibited pixels and the chain $\mathbf{p}_1(z) \cup \mathbf{p}_2(z)$ is named the *separating belt*. If we remove the separating belt, the grid is decomposed into two pixel regions \mathbf{G}_1 and \mathbf{G}_2 . We consider each region separately.

Main Process: Without loss of generality, we consider \mathbf{G}_1 , which is the region above the separating belt, and design a dynamic programming algorithm. Let $J_1 = (v_1, v_2, \dots, v_{m-1})$ be the list of all boundary pixels of \mathbf{G}_1 in the clockwise ordering, and we set v_0 and v_m to be the adjacent (prohibited) leaves to the endpoints of the list. Without loss of generality, the path $\mathbf{p}_1(v_m)$ goes through r_2 and $\mathbf{p}_2(v_0)$ goes through z .

Let $\mathbf{G}(i, j)$ be the set of pixels of \mathbf{G}_1 below or on $\mathbf{p}_1(v_{m-i})$ and $\mathbf{p}_2(j)$ for $0 \leq i, j \leq m$. It is observed that $\mathbf{G}(i, j) \subseteq \mathbf{G}(i', j')$ if $i \leq i'$ and $j \leq j'$. We compute $R(i, j) = W(R \cap \mathbf{G}(i, j))$, starting from $R \cap \mathbf{G}(0, 0) = \mathbf{e}_1 \cup \mathbf{e}_2$.

We consider a subpath \mathbf{b}_1 of $\mathbf{p}_1(v_{m-i})$ and a subpath \mathbf{b}_2 of $\mathbf{p}_2(v(j))$, compute the weight $W_{i,j}(\mathbf{b}_1, \mathbf{b}_2)$ of the maximum weight region Q in $GG(i, j)$ satisfying that $Q \cap \mathbf{p}_1(v_{m-i}) = \mathbf{b}_1$ and $Q \cap \mathbf{p}_2(v_j) = \mathbf{b}_2$. We can observe that if $R \cap \mathbf{p}_1(v_{m-i}) = \mathbf{b}_1$ and $R \cap \mathbf{p}_2(v_j) = \mathbf{b}_2$ for the optimal solution R , then $R(i, j) = W_{i,j}(\mathbf{b}_1, \mathbf{b}_2)$.

Thus, if we compute $W_{i,j}(\mathbf{b}_1, \mathbf{b}_2)$ for all combinations of $\mathbf{b}_1, \mathbf{b}_2$ and for all (i, j) via dynamic programming, we can compute R . Let q be the first intersecting pixel of $\mathbf{p}_1(v_{m-i})$ and $\mathbf{p}_2(v_j)$ (if there is no intersection, we only consider case (3) below). We consider three cases: (1). $q \in \mathbf{b}_1$, (2). $q \in \mathbf{b}_2$, and (3). otherwise.

If (1) or (3), $W_{i,j}(\mathbf{b}_1, \mathbf{b}_2)$ is the maximum of $W_{i,j-1}(\mathbf{b}_1, \mathbf{b}'_2)$ under the condition that \mathbf{b}'_2 is a subpath of $\mathbf{p}_2(v_{j-1})$ such that $\mathbf{b}_2 \cap \mathbf{p}_2(v_{j-1}) \subseteq \mathbf{b}'_2$. If (2), $W_{i,j}(\mathbf{b}_1, \mathbf{b}_2)$ is the maximum of $W_{i-1,j}(\mathbf{b}'_1, \mathbf{b}_2)$ under the condition that \mathbf{b}'_1 is a subpath of $\mathbf{p}_1(v_{m-i+1})$ such that $\mathbf{b}_1 \cap \mathbf{p}_1(v_{m-i+1}) \subseteq \mathbf{b}'_1$. Thus, we can run the dynamic programming. The time complexity is $O(n^4)$ for each pair $(\mathbf{e}_1, \mathbf{e}_2)$, and $O(n^6) = O(N^3)$ in total. \square

3.1 Union vs. decomposition

We discuss the difference of complexities of the problems for *union* and *decomposition*. The following theorem shows strongly contrast to the positive results in Theorems 1 and 6. An outline of the proof is implicitly given in [8], and is omitted in this version.

Theorem 7. *It is NP-hard to compute the maximum weight union $R = R_1 \cup R_2$ of regions $R_1 \in \mathcal{U}$ and $R_2 \in \mathcal{R}$. Also, the maximum weight union of two star-shaped regions is hard to approximate within any given finite ratio.*

4 Staircase/rectilinear convex regions

A region R is a *staircase convex region* centered at p if R is represented as a union of rectangles containing p , in other words, for each $q \in R$, every L_1 shortest path between p and q is contained in R . A region R is a *rectilinear convex region* if it is both x -monotone and y -monotone; in other words, the intersection of any column or row and R is an interval (or empty). By definition, a staircase convex region is a rectilinear convex region. It is known that the maximum weight staircase convex region for a given p can be computed in $O(N)$ time, and the maximum weight rectilinear convex region can be computed in $O(N^{1.5})$ time [7, 11].

Theorem 8. *If k is a constant, the maximum weight region decomposable into k staircase convex regions can be computed in $O(N^{k+1})$ time. Also, the maximum weight region decomposable into k rectilinear convex regions can be computed in $O(N^{k+1})$ time. For these problems, we can also compute the maximum union of k regions in the same time complexity.*

Proof An intersection of a rectilinear convex region and a horizontal is an interval (or empty). Thus, if we keep track of at most $2k$ endpoints of k intervals on a horizontal line, we can design a sweepline dynamic programming. We keep a table of size $O(n^{2k}) = O(N^k)$, and additional N factor is sufficient for updating the table and keeping the staircase/rectilinear convex property. The union problem can be solved in the same fashion. \square

Therefore, difference of computational complexities of the decomposition problem and the union problem have not been revealed for these cases. However, we conjecture that there is a fixed parameter tractable algorithm for computing the maximum weight region decomposable into k staircase convex region case provided that the k center points are given.

5 Concluding Remarks

As far as the authors know, study of complexity of combinatorial algorithms of segmenting a figure decomposable into k basic objects has just started by this work. Especially, development of FPT algorithms is important and unsolved in many cases. Moreover, if we want to solve the original image segmentation problem in full-automatic manner, we need a mechanism to select the parameter k to have the best segmentation.

The three dimensional extension has a potential application to a variation of *open-pit mining* problem [9, 10]. Each pixel (or voxel, in three dimensions) models a block of mine, and the weight shows the profit to dig the block. If we are given k stem pits and dig orthogonal tunnels from each stem pit without allowing crossing of tunnels, the maximum weight region gives the optimal way of mining.

References

1. T. Asano, D. Z. Chen, N. Katoh, T. Tokuyama, Efficient Algorithms for Optimization-Based Image Segmentation. *Int. J. Comput. Geometry Appl.* 11(2), pp. 145-166 (2001).
2. T. Fukuda, Y. Morimoto, S. Morishita, T. Tokuyama, Data Mining Using Two-Dimensional Optimized Association Rules: Scheme, Algorithms, and Visualization. *SIGMOD Conference 1996*, pp. 13-23.
3. T. Fukuda, Y. Morimoto, S. Morishita, T. Tokuyama, Data Mining with optimized two-dimensional association rules. *ACM Trans. Database Syst.* 26(2), pp. 179-213 (2001).
4. D. Z. Chen, J. Chun, N. Katoh, and T. Tokuyama. Efficient algorithms for approximating a multi-dimensional voxel terrain by a unimodal terrain. *Proc. COCOON'04, LNCS 3106*, pp. 238-248, 2004.
5. D. Z. Chen, X. S. Hu, S. Luan, X. Wu, C. X. Yu, Optimal Terrain Construction Problems and Applications in Intensity-Modulated Radiation Therapy. *Algorithmica* 42(3-4), 265-288 (2005).
6. J. Chun, M. Korman, M. Noellenberg, T. Tokuyama, Consistent Digital Rays, *Proc. 24th ACM SOCG*, pp.355-364, 2008.
7. J. Chun, K. Sadakane, T. Tokuyama. Efficient Algorithms for Constructing a Pyramid from a Terrain. *Proc. JCD CG02, LNCS 2866(2003)*, pp.108-117.
8. J. Chun, M. Korman, M. Nöllenburg and T. Tokuyama, Hardness of the Maximum Union-of-Closures Problem, Preprint (preliminarlily reported in WAAC2008). <http://www.dais.is.tohoku.ac.jp/~mati/2peakNP.pdf>
9. D. Hochbaum, A New-Old Algorithm for Minimum-cut and Maximum-flow in Closure Graphs, *Network* 37(4), pp. 171-193, 2001.
10. H. Lerchs and I. Grossmann, Optimum Design of Open-Pit Mines, *Trans C.I.M.* 68, pp. 17-24, 1965.
11. K. Yoda, T. Fukuda, Y. Morimoto, S. Morishita, T. Tokuyama, Computing Optimized Rectilinear Regions for Association Rules. *Proc. KDD 1997*, pp. 96-103

Appendix. Proof of Theorem 5

See Figure 7 for the input base segments and an example of feasible region. We draw vertical lines from each endpoint of base segments until it intersects another base segment (including their endpoints) or a boundary edge. This decomposes the grid into rectangles (a special case of *trapezoidal map* used in computational geometry). We can cut each horizontal base segment into fragments such that each fragment is a lower edge of a rectangle given above. By replacing the set of horizontal segments by the set of fragments, we assume that the above condition holds for each input base segment, and the rectangle $\text{Rect}^v(s)$ that has s as the lower edge is called a vertical component rectangle. We artificially consider the lower boundary edge of the grid as a base segment so that pixel is decomposed into vertical component rectangles. Similarly, we define the horizontal component rectangle $\text{Rect}^h(t)$ that has a vertical base segment t as its left edge. We can show that the number of component rectangles is at most $4k$.

A vertical component rectangle $\text{Rect}^v(s)$ intersects some horizontal component rectangles, and cut into smaller rectangles called *cell* by the edges of them (Figure 8). Horizontal component rectangles are also cut into cells. Note that a cell is a union of rooms, and the number of cells is $O(k^2)$. Let c^+ , c^- , c^ℓ , and c^r be the cells adjacent to c that are above, below, to the left, and to the right of c , respectively.

Let R_{opt} be the optimal region. We fix a decomposition of R_{opt} into components corresponding to base lines. This decomposition is not unique, but we can select one and fix it such that the component $R_{\text{opt}}(s)$ for the base segment s lies in its component rectangle.

Consider a cell $c = \text{Rect}^v(s) \cap \text{Rect}^h(t)$. We say that c is vertical terminal (resp. horizontal terminal) if no column of $R_{\text{opt}}(s)$ (resp. row of $R_{\text{opt}}(t)$) penetrates c and intersects c^+ (resp. c^r). Because of noncrossing property of $R_{\text{opt}}(s)$ and $R_{\text{opt}}(t)$, we have the following:

Lemma 2. *Every cell is either vertical terminal or horizontal terminal.*

Let $\text{col}(c)$ be the smallest column index such that the column of $R_{\text{opt}}(s)$ penetrates c and intersects c^+ . We set $\text{col}(c) - 1$ to be the rightmost column index of c if c is a vertical terminal cell. Similarly, $\text{row}(c)$ is the smallest row index such that the row of $R_{\text{opt}}(t)$ penetrates c and intersects c^r . We set $\text{row}(c) - 1$ to be the top row index of c if c is a horizontal terminal cell.

Let $\text{Rect}^v(s, j_1, j_2)$ (resp. $\text{Rect}^h(t, i_1, i_2)$) be the part of $\text{Rect}^v(s)$ (resp. $\text{Rect}^h(t)$) in the column interval $[j_1, j_2)$ (resp. row interval $[i_1, i_2)$). The

union of $\text{Rect}^v(s, \text{col}(c^-), \text{col}(c))$ and $\text{Rect}^h(t, \text{row}(c^\ell), \text{row}(c))$ has a shape of a hook, and called the *responsible hook* of c (see Figure 9).

Lemma 3. *Within each responsible hook, R_{opt} can be computed as the maximum weight region decomposable into the based monotone regions with base segments s and t without considering the effect of other regions.*

A feasible region is decomposed into its parts in responsible hooks (Figure 10). Thus, if we could know the set of responsible hooks, we could compute the optimal regions in all those hooks and combine them to obtain R_{opt} . Unfortunately, the number of possible combination of all responsible hooks is $n^{O(k^2)}$, which is very large. Thus, we consider a dynamic programming approach to avoid the high complexity.

Fortunately, due to lemma 2, every cell is either vertical or horizontal terminal, and we can guess whether each cell is a vertical terminal or a horizontal terminal (or both, included in the second case). There are $2^{O(k^2)}$ possible combinations for this guess. More efficiently, we need to decide which cell is the highest vertical (resp. rightmost horizontal) terminal in each vertical (resp. horizontal) component rectangle. This reduces the number of combinations to $k^{O(k)}$. Suppose that we have the right guess. We construct the *influence graph* whose nodes are cells and a directed edge from c is given to c^+ if it is horizontal terminal, otherwise to c^r . This graph is clearly a directed forest.

Let $W(c, H)$ be the maximum weight of the maximum weight region in the union of ancestors of c (including c) if the responsible hook in c is H . Suppose that the responsible hooks of $W(c^-, H_1)$ and $W(c^\ell, H_2)$ are given for all possible combinations of H_1 and H_2 . Then, we can compute $W(c, H)$ for all possible H in $O(n_c^2 m_c^2)$ time, where n_c and m_c are number of rows and columns of c , respectively.

Now, it is easy to run the dynamic programming to constructing the optimal solution R_{opt} processing cells of the forest starting from source nodes. The time complexity is $O(n^4) = O(N^2)$ for each guess. Thus, the overall time complexity is $O(k^{O(k)} N^2)$. (More precisely, $O((4k)^{4k} N^2)$).

An $n^{O(k)}$ bound for the four-sided case is obtained if we consider all possibilities of decompositions of each component rectangles (like Figure 6). We omit its details in this version.

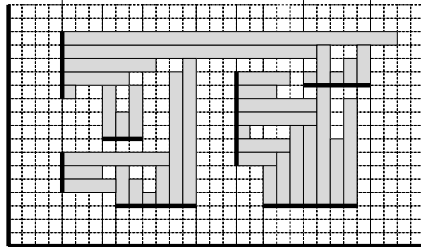


Fig. 7. A feasible region for the two-sided painting with k base segments

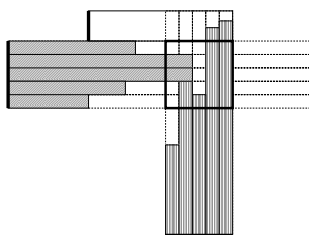


Fig. 8. Two components rectangle intersecting at a cell

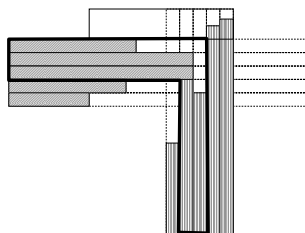


Fig. 9. A responsible hook

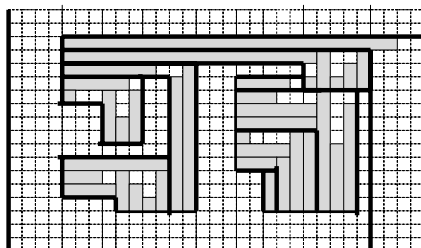


Fig. 10. Responsible hooks (ingoring empty ones)