Efficient Storage Compression for 3D Regions

Georgia Panagopoulou, Spiros Sirmakessis, Athanasios Tsakalidis

Department of Computer Engineering and Informatics
University of Patras, 26500 Patras, Greece
and
Computer Technology Institute
P.O.Box 1122, 26110 Patras, Greece

Abstract

In this work we present heuristics algorithms for efficient storage compression for 3D regions. Giving a 3D decomposed in parallelepipeds, we want to store it using the least storage information. We present the steps and the experimental results of five algorithms; the first one is a simple, space consuming, approach that works as the upper bound for the storage requirements of the other four algorithms. In brief we present the steps of the algorithm of Franzblau-Kleitman (Franzblau and Kleitman 1984). This algorithm has a very good average performance. The algorithm works well for 2D regions. We produced an invariant of their algorithm for 3D regions. Our contribution is the development of the other three algorithms that have less storage requirements than the algorithm of Franzblau-Kleitman. Using experimental testing procedures (the results are presented in the conclusion) we present the evaluation of these algorithms. From this evaluation the fifth algorithm gives the best performance in every case. This algorithm behaves better for every case compared with the algorithm of (Franzblau and Kleitman 1984) with a near to optimal performance.

1. Introduction

In the recent years, a lot of work has focused on developing applications that can create, maintain and produce views of different objects for cartography and geographical information applications (Samet 1990) and other research areas such as graphics and database storage problems. Due to the nature of these objects, applications are strongly forced to manipulate huge amounts of data. Although current hardware and software tools have the ability to ensure schemes, techniques and tools for handling these amount of data, special care should be taken in order to achieve optimality (Preparata and Shamos 1985). Optimality refers to the usage of computing resources, such as memory and CPU time.

In this paper we present a work done for handling a special problem in methodology and other areas. Assume that we have a 2-dimensional region with edges parallel to X and Y axis. We will try to find the minimum number of rectangles needed to cover this region. This set of rectangles is called the *rectangle cover* of the query region. This problem is essential for use in aerial photography methodology, photolithography, VLSI design, image processing and in any application responsible for handling geometric objects. Moreover, solutions to this problem can guarantee the reduction of the space required to store any region in cartography and GIS applications. By using the minimum number of rectangles to describe a region, we have a modular way to store this region without requiring to store every vertex of the region. This solution was a preliminary result of a project trying to store the internal structure of the human brain (Anogianakis et al, 1992a), (Anogianakis et al, 1992b). In this work, we solve the general problem of describing a 3-dimensional region with edges parallel to *X*, *Y* and *Z* axis, in order to provide solutions for problems that can arise in every application in 2D or 3D space. The problem in 2D can be defined as follows:

Given a 2-dimensional region with edges parallel to X and Y axis find the minimum number of rectangles that cover the given region.

An example of the problem is presented in figure 1.

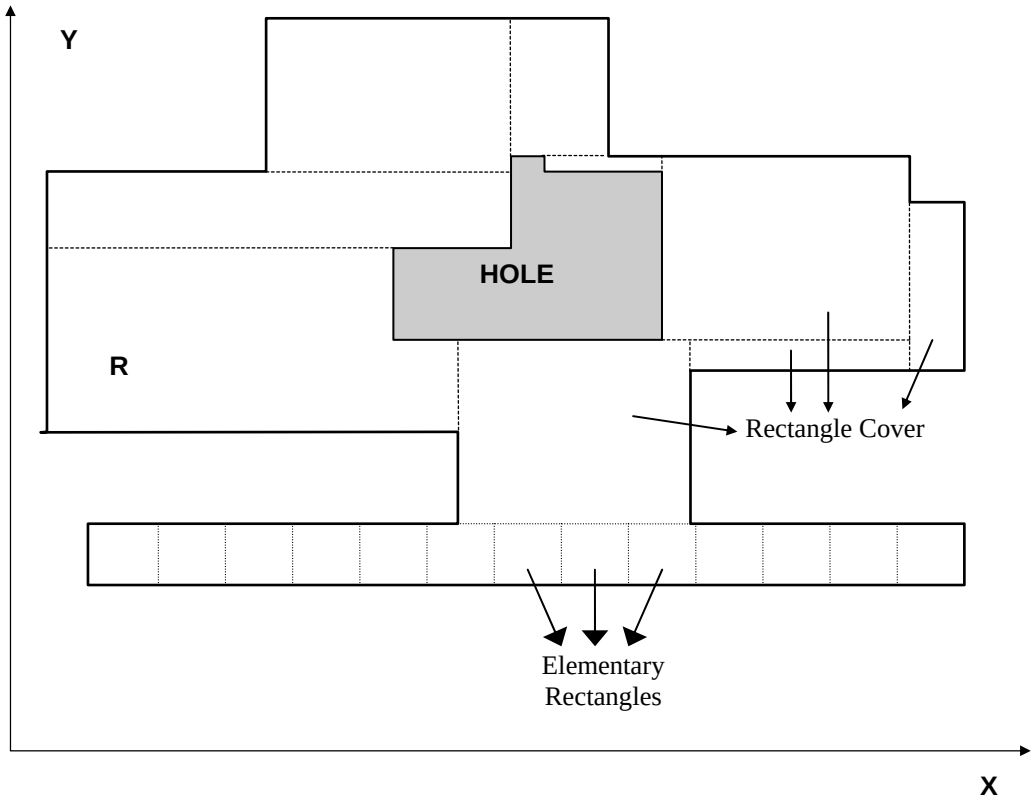


Figure 1. The 2-dimensional case of the problem

In this figure a region R is covered by ten rectangles. This is the optimal solution of the problem. We assume that the query region is described as a collection of elementary rectangles. An elementary rectangle is a square with |X|=|Y|=1.

The 3D problem is stated as follows:
Given a 3-dimensional region with edges parallel to X, Y and Z axis find the minimum number of parallelepipeds that cover the given region.

The region can have holes and it is neither needed to be connective, nor convex according to any direction. These parallelepipeds may overlap or not, but their union should cover the whole region and the region should be the union of these parallelepipeds. In other words no prominence is allowed comparing the region and the union of parallelepipeds. We assume that the region is described as a union of unitary cubes, which is always true, because every region can be described by unitary cubes; those are the cubes with |X|=|Y|=|Z|=1 (Figure 2).

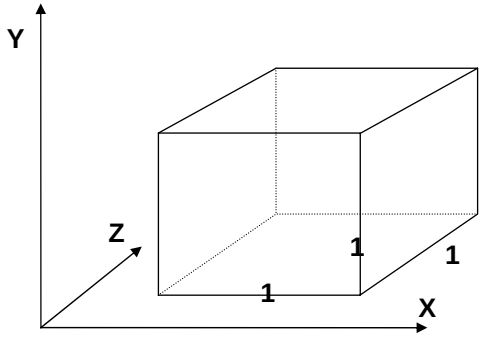
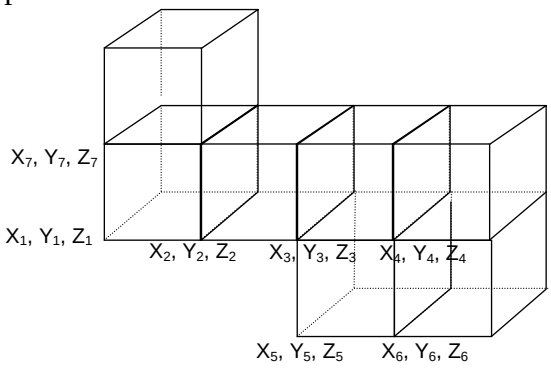


Figure 2. A unitary cube

The algorithms presented assume that the region is stored into a file containing trines of positive integers. The trines are the coordinates of the bottom left corner of every unitary cube describing the region. All trines are sorted firstly by the *y*-coordinate, secondly by the *z* and last by the *x*-coordinate. Any other way of sorting can also be used.

Space reduction is achieved by grouping the unitary cubes into parallelepipeds. Instead of keeping three numbers for every cube (the coordinates of the base), we store for every parallelepiped the coordinates of the bottom left vertex and three more numbers, expressing the height, width and length of the parallelepiped. Figure 3 presents an example of the compression of the storage of a 3D region. The query region is described as a collection of seven unitary cubes. The original file with the coordinates of the seven cubes contains 21 numbers (real or integer). A first approach for compressing the storage information for region can reside in a file containing 12 numbers. As it can be easily seen the compression rate depends on the way the "grouping" of the unitary cubes is done.

In order to have a basis for the comparison and the evaluation of every algorithm, we studied and implemented the algorithm of Franzblau and Kleitman (Franzblau and Kleitman 1984). The algorithm works for the 2D space but we produced an invariant of the same algorithm for the 3D space by extending its definition to the 3-dimensional problem. The algorithm covers the region with overlapping parallelepipeds of unitary thickness. This work gave us a very good, average case, solution in order to evaluate the performance of our algorithms. In order to have an upper bound, that is the worst performance in storage requirements, we implemented a simple, innovative algorithm that solves the same problem. Moving in this direction, we found and implemented four heuristic algorithms. These algorithms solve the problem by trying to combine unitary cubes into larger parallelepipeds.



Description File of the Region	Compressed Description of the Region
X_1, Y_1, Z_1	X_1, Y_1, Z_1
X_2, Y_2, Z_2	4
X_3, Y_3, Z_3	X_5, Y_5, Z_5
X_4, Y_4, Z_4	2
X_5, Y_5, Z_5	X_7, Y_7, Z_7
X_6, Y_6, Z_6	1
X_7, Y_7, Z_7	

Figure 3. Compression of a 3D region.

We present three more algorithms. The first one covers the query region with non overlapping parallelepipeds of unitary thickness. Using a different approach, we present another algorithm that covers the query region with overlapping parallelepipeds of unitary thickness. These two algorithms, like the algorithm of Franzblau and Kleitman, divide the space into stripes of unitary thickness, parallel to the X-Y plane, and they run a grouping procedure once for every stripe. This procedure generates parallelepipeds of unitary thickness that cover the query region. In other words, they reduce the solution of the 3D problem to the solution of k-2D problems, where k is the number of different stripes that the space is divided. Following a quite different approach, we derived a

third algorithm that solves the problem working only in the 3D space. The efficiency of the algorithms has been tested in practice by trying to describe different 3D regions. The number of rectangles (parallelepipeds) needed to cover the region, was the measure of efficiency. The comparison showed that the last algorithm, working actually in 3D space, has the best performance.

This paper is organised as follows. Section 2 presents our simple and innovative algorithm for solving the problem. Section 3 presents a brief description of the algorithm of Franzblau-Kleitman and our invariant that makes it work in 3D space. An algorithm that produces overlapping parallelepipeds of unitary thickness is described in section 4. An invariant of this algorithm, which covers the query region with overlapping parallelepipeds of unitary thickness can be found in section 5. The last algorithm for the 3D case is the theme of section 6. For every algorithm a pseudo-code is given. The last section contains tables with the performance evaluation of every algorithm and a short presentation of our future work. For a more detailed review of the algorithms, the source code of the implementation can be provided on request.

2. Algorithm 1: A Simple Approach

This algorithm is a simple approach to the solution of the problem. Taking advantage of the sorted input (all trines of the input file are sorted firstly by the *y*-coordinate, secondly by the *z* and last by the *x*-coordinate; any other way of sorting can also be used) we divide the query region into *stripes* of unitary thickness, parallel to the *X*-*Z* plane. The stripe is actually a sequence of unitary cubes with the same value in the Z axis. In every stripe we produce non overlapping parallelepipeds of unitary thickness (by the *Y* axis) and unitary height (by the *Z* axis). This means that we create "rows" (sequence of cubes) containing the maximum number of adjacent cubes that their bases have equal *z* coordinates. These cubes construct a parallelepiped of unitary height and thickness. The compression procedure is presented in figure 3.

An algorithmic description follows in a pseudo-language:

Step 1: Divide the query region into stripes of unitary thickness
Step 2: FOR every stripe DO
  Step 2.1: Divide the stripe into unitary cubes
  Step 2.2: FOR every "row" DO
    take the maximum groups of adjacent cubes until you move to a new plane or the end of file is found.
  Step 2.3: FOR every group DO
    construct a parallelepiped having the coordinates of its base equal to the coordinates of the base of the leftmost cube of the set of the group of adjacent cubes and length equal to the number of the cubes that compose the set.
  Step 2.4: Save this information into the output file

As it can be seen, this is a very simple approach and it was implemented to work as a comparison for more complex algorithms.

3. Algorithm 2: An Invariant of The Algorithm of Franzblau-Kleitman Producing Overlapping Parallelepipeds of Unitary Thickness

This algorithm divides the 3D region into stripes of unitary thickness, like Algorithm 1. Every stripe is divided into convex regions and for every region we run the algorithm of Franzblau-Kleitman that produces overlapping parallelepipeds of unitary thickness (according to the *Y*-axis). Franzblau and Kleitman presented an algorithm in (Franzblau and Kleitman 1984) that can be used to solve the following problem:

Given a polygon with edges parallel to the x and y axes, which is convex in the y direction, find a minimum sized collection of rectangles that cover the polygon and are contained within it.

A *polygon* is defined as a finite subset of an infinite grid of unit squares in the plane. A polygon is vertically (horizontally) *convex* if every column (row) of squares in the polygon is connected. A *rectangle* is a rectangular subset of unit squares that lies entirely inside the polygon. A subset of squares in a polygon is called *independed* if no two squares in the subset are contained within the same rectangle. A collection of possibly overlapping rectangles, the union of which is equal to the query polygonal region is called *rectangle cover*.

In a vertically convex polygon, every maximal connected subset of a row of squares defines an interval. This interval is made by projecting the row of squares onto the axis. Assume S is the set of all intervals from a polygon R generated in this way. Then finding a minimal rectangle cover for R can be reduced to finding a minimum size set of intervals M which generates S (meaning that each interval in S is a union of intervals in M). Conversely, an independent set of squares in R can be showed to correspond to a sequence of intervals in S such that each interval contains at least one point that does not belong to any other interval of this sequence. More formally, a $T \subseteq S$ is independent if the intervals of T can be ordered as I_1, I_2 ...I_n so that each I_k containing points that can not be found in $\bigcup_{j=1}^{k-1} I_j$. If there does not exist such an ordering, T is called *dependent*.

The only information one needs to construct a rectangle cover of a polygon is the set of distinct horizontal intervals determined by the vertical sides of the rectangles. Figure 4 gives a schematic representation of this remark.

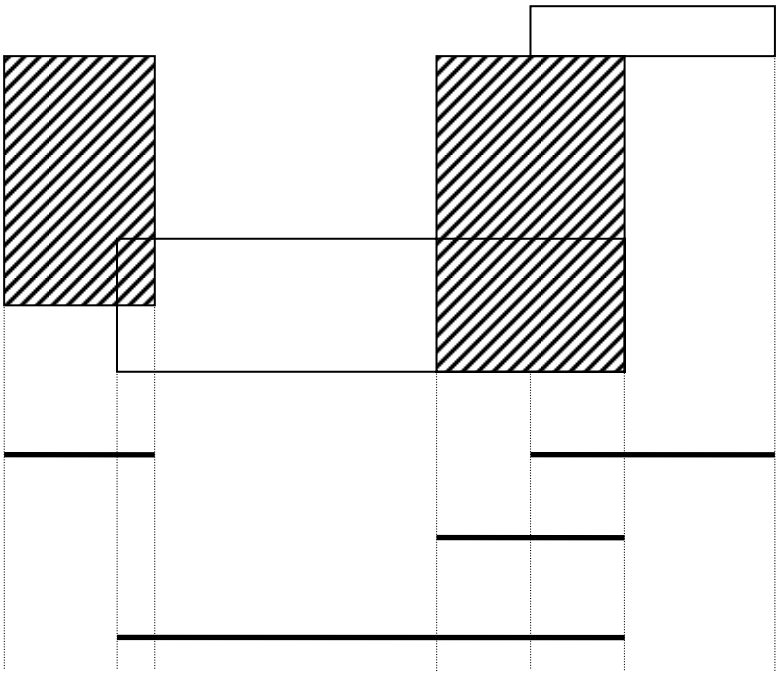


Figure 4. The set of distinct horizontal intervals determined by the vertical sides of the rectangles

The result of calculating the minimal set of intervals that produces any interval of the above kind, ends in a minimal rectangle cover of the given polygon. For example:

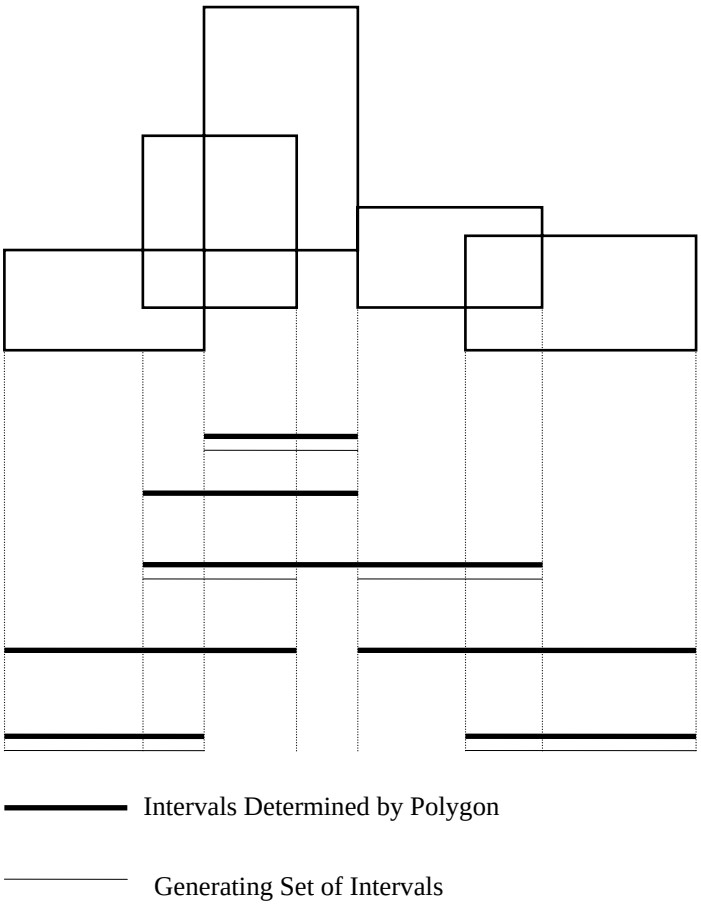


Figure 5. An example for the generating set of intervals

If we try to look at the problem from this point of view, we can describe an algorithm that finds a set of interval which produces S, with the minimum number of elements. From this solution we can easily be leaded to the solution of finding the minimum cover of a region R with rectangles. The time of the algorithm is $O(n^2)$, where n is the number of intervals. This number depends on the number of vertices of the polygon. The basic step of the algorithm is to replace "chains" of overlapping intervals with the intersections of successive couples. It is easy to show that S is dependent if and only if S contains a subset T such that every point in $\cup$T is covered by at least two intervals in T. We call such a collection T of intervals *simply dependent.* We can extend the definition to the sub-intervals $U \subseteq \cup S$ by saying that an interval U is dependent or simply dependent if the set of all intervals with both endpoints contained into U has the corresponding property. We call U minimal simply dependent if no proper subinterval of U is simply dependent. Given S, collection of intervals such that $\cup$S contains a simply dependent interval U, we follow a procedure that constructs a set of generators G from S with ||S||-1 intervals. G is formed by replacing consecutive pairs of maximal intervals U with their intersections.

Reduction Process

Let T be the set of intervals in U (i.e. intervals with both endpoints in U). Let $\{I_1, I_2, ...I_k\}$ be the maximal intervals in T ordered by increasing order of their left endpoint.

If k=1 then $G \leftarrow S-\{I_1\}$.

Else $G \leftarrow [S-\{I_1, I_2, ...I_k\} \cup \{I_1 \cap I_2, I_2 \cap I_3, ..., I_{k-1} \cap I_k\}]$.

Using this reduction process we can built a set of intervals with minimal size that can produce S in this way: if S is a collection of intervals and $U \subseteq \cup S$, let RUS be the collection of intervals obtained by applying the reduction procedure to the intervals of S contained in U. We define the operator D_L

where $D_L S$ is the leftmost minimal simply dependent interval in S. To find a minimum size set of generators we iterate the operator RD_L to construct:

$$RD_L S, (RD_L)^2 S, ...(RD_L)^n S$$

We stop the process when the $(RD_L)^n S$ is independent (it does not contain any simply dependent interval). The validity of the algorithm was proved in the original work of Franzblau-Kleitman in (Franzblau and Kleitman 1984). Doing the same for every 2D region and combining the results for every stripe in the 3D case, a solution to our problem can be achieved (using results from (Edelsbrunner et al. 1984), (Cutting 1984)).

4. Algorithm 3: The non Overlapping Parallelepiped Algorithm

This algorithm, like the previous one, divides the 3D region into stripes of unitary thickness. Every stripe is divided into sequences of cubes and in every stripe we follow the same grouping procedure, as in algorithm 1, that generates a set P of parallelepipeds of maximal length and unitary thickness. This set P of parallelepipeds covers every stripe (this is the same procedure like in algorithm 1). The P is initialised with the parallelepipeds of the row with the minimum *z*-coordinate. For the next rows, we try to enlarge the parallelepipeds produced from the previous initialisation, with the new ones produced in the same way in the first step of the algorithm as in Algorithm 1. We project the front face of every parallelepiped r, from the new rows, onto the *X*-axis and we search for the parallelepipeds of P, that their projections intersect with the projection of r. From all these parallelepipeds, we choose those having the *z*-coordinate of their top face equal to the *z*-coordinate of the base of r. Assume that T is the sub-set of the set P satisfying this condition. All elements of T are sorted by the *x*-coordinate of their left edge. We traverse through the elements of T twice, once from left to right and once towards the opposite direction. During this traversal we test whether or not we can "***embody"*** r into the current parallelepiped t of the T.

Embodiment means that we produce a parallelepiped that is the union of r and the current parallelepiped $t \in P$. If the cardinality of the set $T \cup \{r\}$ is reduced or remains the same, then we embody r into t; otherwise we continue with the next parallelepiped of T.

This test is done in the following way: Assume that I_1 is the projection of r onto *X*-axis and I_2 is the projection of t onto the same axis. We calculate the set $I_1 \cup I_2$ and we subtract the $I_1 \cap I_2$. The result is a set of disjoint intervals with cardinality at most 2. If the cardinality is less than 2, then we can embody r into t. The procedure for inserting r into t follows next: first we calculate the sets $I_{11}=I_1-I_1 \cap I_2$ and $I_{22}=I_2-I_1 \cap I_2$. Due to the previous condition, at least one of the I_{11} or I_{22} is empty, while the other set contains only one interval. Finally, we delete t from the set P and insert into P the parallelepiped satisfying the following conditions:

a) its front face's projection is the $I_1 \cap I_2$,
b) the *z*-coordinate of the base is equal to the *z*-coordinate of the base of t and
c) the height is equal to the height of t plus one.

If $I_{11} \neq \varnothing$, we insert into P the parallelepiped having these properties:

a) its front face's projection is the interval stored in the set I_{11},
b) the *z*-coordinate of its base is the *z*-coordinate of the base of t and
c) its height is equal to the height of t.

If $I_{22} \neq \varnothing$, we reduce r into a parallelepiped, having as its front face's projection the interval stored into I_{22} and we continue walking through the elements of T with the new r. If $I_{22}=\varnothing$, we continue with the next parallelepiped. Figure 6 presents the different cases of the problem. We present the cases in the 2-dimensional space because it is easier to draw the figures than in the case of 3-dimensional space. In 2D space we are referring to rectangles, but these correspond to parallelepipeds in the 3D space. The rectangles that are shadowed correspond to the current

rectangles t of T and the other is the rectangle that we are trying to embody from the set of rectangles of the next row.

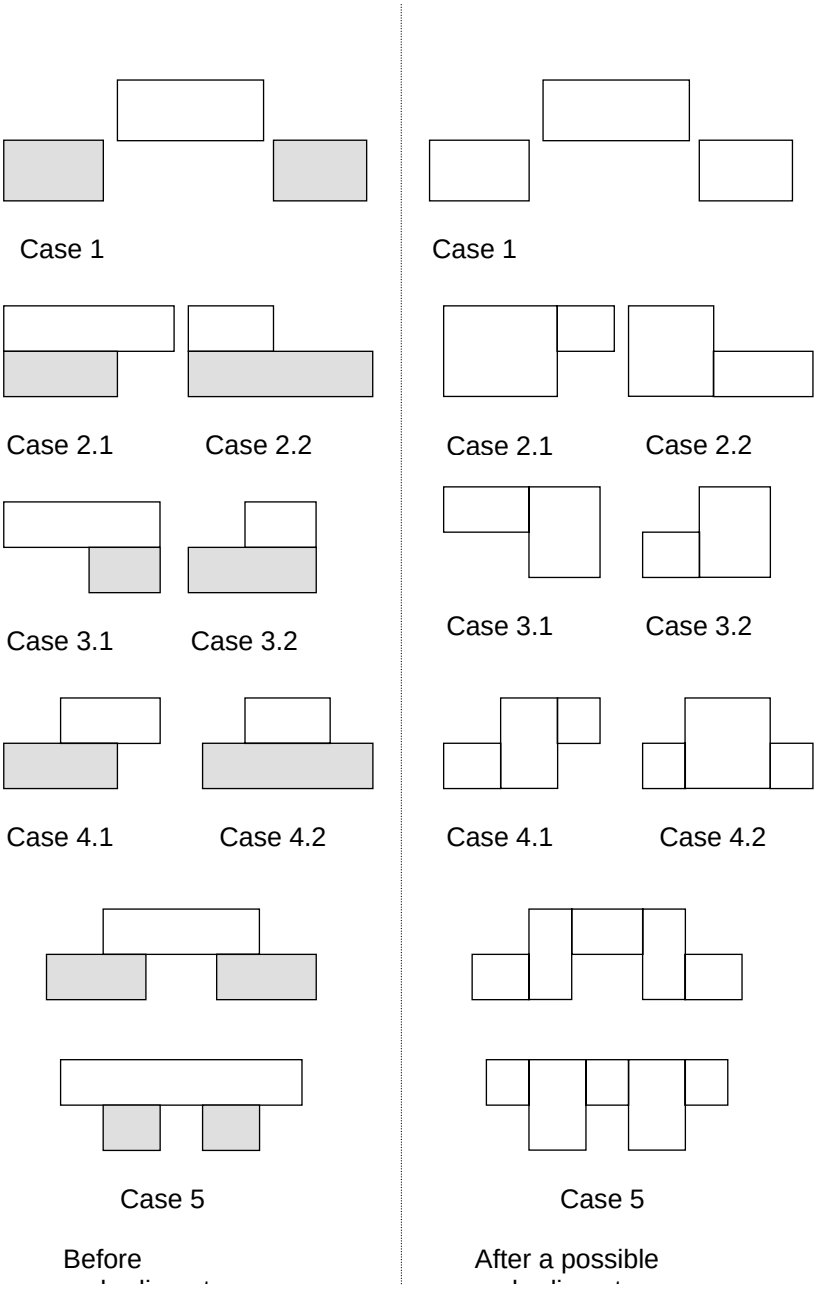


Figure 6. The different cases of the problem

Case 1 is the case where no embodiment can be done since the projections of the rectangles do not intersect. Cases 2 and 3 are similar. The projections of the rectangles intersect and when an embodiment takes place the cardinality of the set $T \cup \{r\}$ remains the same in all cases; it remains 2. Case 4 is the case where the projections of the rectangles intersect but no embodiment can be done, because the cardinality of the set $T \cup \{r\}$ is increased after a possible embodiment; from 2 it becomes 3. Cases like those showed in case 5 are actually transformations of the previous cases. An algorithmic approach of the previously described procedure follows:

Step 1: Divide the query region into stripes of unitary thickness
Step 2: FOR every stripe DO
    Step 2.1: Divide the stripe into unitary cubes

Step 2.2: FOR every "row" DO
take the maximum groups of adjacent cubes until you move to a new plane or the end of file is found.
Step 2.3: FOR every group DO
construct a parallelepiped having the coordinates of its base equal to the coordinates of the base of the leftmost cube of the set of the group of adjacent cubes and length equal to the number of the cubes that compose the set.
Step 2.4: Initialise P with the parallelepipeds of the row with the less *z*-coordinate
Step 2.5: FOR every one of the next rows DO
Step 2.5.1: Assume that r is the next parallelepiped of the row
Step 2.5.2: Compute the set T of the parallelepipeds that corresponds to r.
Step 2.5.3: Assume that t is the next parallelepiped of T (find it by walking through the T from left to right and backwards)
Step 2.5.4: IF the cardinality of $(I_1 \cup I_2 - I_1 \cap I_2) < 2$
THEN embody r into t
IF r non zero
THEN go to Step 2.5.3
ELSE go to Step 2.5.1

Taking under consideration that the input file is sorted, we can follow some tips that are useful for the implementation of the algorithm. We can implement P as a double linked list. Each time we finish with Step 2.5 we can mark the node of the list that corresponds to the right-most parallelepiped of T. In order to calculate the set T produced by the next parallelepiped r of the row at Step 2.5.1, we can work in this way:

We start from the already marked node and we move to the right in the linked list until we meet the first node that contains a parallelepiped that its front face projection intersects with the projection of r's front face. All the parallelepipeds that we crossed by, during this walk, are deleted from the list and are inserted to the output file. Because of the sorted form of the input file, we are convinced that these parallelepipeds will not take part into any new embodies. At the next step we keep moving to the right until we find the last node of the list, that contains a parallelepiped having its front face's projection intersecting with the front face's projection of r. All these parallelepipeds that we visit, are inserted into T. Using this method for calculating T, we traverse the list for every row of parallelepipeds only once.

5. Algorithm 4: The Overlapping Parallelepiped Algorithm; An Invariant of Algorithm 3

This algorithm is actually the same as the previous one. It divides the region into stripes of unitary thickness and for every stripe it produces overlapping parallelepipeds of unitary thickness (by the *Y*-axis). In the first stage the algorithm divides every stripe of unitary thickness into rows of cubes and takes for every row the maximal parallelepipeds, working like Algorithm 1. The set P of parallelepipeds, that cover every stripe, is produced as described below. The set P is initialised with the parallelepipeds having the minimum *z*-coordinate. For the next rows we try to enlarge the parallelepipeds produced from the previous initialisation, with the new ones produced in the same way as in Algorithm 1. We project the front face of every parallelepiped r onto the *X*-axis and we find the parallelepipeds of P that their projections intersect with the projection of r. From all these parallelepipeds, we choose those having the *z*-coordinate of their top face equal to the *z*-coordinate of the base of r. Assume that T is the set of all these parallelepipeds. All elements in T are sorted by the *x*-coordinate of their left edge. We traverse the set T from left to right and we increment by 1 the height of every parallelepiped that its front face's projection is contained into the front face's projection of r. Assume that T_{inc_P} is the set of projections of the front faces of all the parallelepipeds, incremented from the previous action and r_P is the projection of r's front face.

1. If $r_P = \cup T_{inc_P}$, then r has been covered and we move to the next parallelepiped of the row.

2. If $r_p \supset \cup T_{incp}$, assume that $T'_p = T_p - T_{incp}$ (where T_p is the set of the front face's projections of all parallelepipeds of T). If $T'_p \neq \emptyset$, we test to find from all the parallelepipeds those having their projection into T'_p; these *"can embody r"* (a formal definition of this statement is stated in the next paragraph). From the set of parallelepipeds that satisfies this condition we choose only one; the one that produces the smallest number of parallelepipeds after the embodiments. We embody this parallelepiped with r and we execute again step 2 until we cover r, or we can not make any other embodiments. If no new embodiments can be made, we stop and add r into P.

One parallelepiped *can embody r,* if and only if the cardinality of the set $T \cup \{r\}$ remains the same or is reduced after the embody. The procedure, that checks this, follows:

Assume $t_p \in T'_p$. We calculate the interval $r_p \cup t_p$ and we subtract the $r_p \cap t_p$. The result of this action is a set G of disjoint intervals with cardinality at most 2. Let $G = G_1 \cup G_2$, where $G_1 = r_p - r_p \cap t_p$ and $G_2 = t_p - r_p \cap t_p$. For every interval $g \in G_2$ we test if $\exists l \in T_p$ such as $g \subseteq l$ and the *z*-coordinate of the base of the parallelepiped that corresponds to l is less than or equal to the *z*-coordinate of the parallelepiped that corresponds to g. If this is the case then $G_2 \leftarrow G_2 - \{g\}$. For every interval $g \in G_1$ we test if $\exists l \in T_p$ such as l corresponds to parallelepiped that has been increased and $g \subseteq l$. If the condition is true, then $G_1 \leftarrow G_1 - \{g\}$.

If the cardinality of the set $G = G_1 \cup G_2$ remains less than 2 then we state that the result of the test is positive, otherwise it is negative. The procedure followed to *embody* one parallelepiped into another parallelepiped includes these steps:

The cardinality of the set G is at most 1, because embodiments are done when the test is positive. We delete from P the parallelepiped t that will be the basis of the embody and we insert into P the parallelepiped having these properties:

a) its front face projection is the $r_p \cap t_p$,
b) The *z*-coordinate of its base is equal to the *z*-coordinate of the base of t and
c) its height is equal to the height of t plus one.

If $G_2 \neq \emptyset$, then we insert into P the parallelepiped satisfying the conditions:

a) its front face projection is identified with the only interval existed in G_2,
b) it has as its base's *z*-coordinate the *z*-coordinate of the base of t and
c) its height is equal to the height of t.

If $G_1 \neq \emptyset$, we reduce the parallelepiped r in a way that its front face's projection is identified with the only interval existing in G_1. If $G_1 = \emptyset$, then we set r equal to zero.

The algorithm presented step-by-step follows.

Step 1: Divide the query region into stripes of unitary thickness
Step 2: FOR every stripe DO
  Step 2.1: Divide the stripe into unitary cubes
  Step 2.2: FOR every "row" DO
    take the maximum groups of adjacent cubes until you move to a new plane or the end of file is found.
  Step 2.3: FOR every group DO
    construct a parallelepiped having the coordinates of its base equal to the coordinates of the base of the leftmost cube of the set of the group of adjacent cubes and length equal to the number of the cubes that compose the set.
  Step 2.4: Initialise P with the parallelepipeds of the row with the less *z*-coordinate
  Step 2.5: FOR every one of the next rows DO
    Step 2.5.1: Assume that r is the next parallelepiped of the row; $r_{start} = r$
    Step 2.5.2: Compute the set T of the parallelepipeds that corresponds to r.
    Step 2.5.3: $\forall t_p \in T_p$ DO
      IF $t_p \subseteq r_p$

THEN increment the height of t by 1.

IF $r_p = \cup T_{incP}$

THEN go to Step 2.5.1.

Step 2.5.4: Calculate the $T'_P = T_P - T_{incP}$.

IF $T'_p = \varnothing$,

THEN $P = P \cup \{r_{start}\}$

Go to Step 2.5.1.

Step 2.5.5: Find $t_p \in T_p$ that produces the set G with the minimum cardinality.

IF $||G|| < 2$

THEN embody r with t.

IF r non zero

THEN go to Step 2.5.2.

ELSE go to Step 2.5.1.

ELSE $P = P \cup \{r_{start}\}$

Go to Step 2.5.1.

6. Algorithm 5: The 3D non Overlapping Parallelepipeds Algorithm

Unlike the algorithms described so far, Algorithm 5 produces non-overlapping parallelepipeds of thickness greater than 1. The way it works in the space reminds the way that Algorithm 3 works in the plane. So we can say that it is a generalisation of Algorithm 3 in the 3-dimension. In the first stage, the algorithm divides the 3-dimensional area into layers of unitary thickness and executes Algorithm 3 for every layer, constructing groups of parallelepipeds of unitary thickness. The set D of the parallelepipeds that cover the 3-dimensional area is constructed step-by-step, through a procedure that is described below.

The initial values of the set D are the parallelepipeds of the layer with the lowest y-coordinate (as they were created after the execution of Algorithm 3 in this layer). Next, for any following layer L, we act as follows: every one of the parallelepipeds r, created after the execution of Algorithm 3 in layer L, is projected onto the X-Z plane. For each one of these parallelepipeds we find the corresponding projections of set D's parallelepipeds it crosses. Between these projections we choose these, which correspond to parallelepipeds of D, whose *y*-coordinate of the back face is equal to the *y*-coordinate of the front face of r. Assume that T is a set containing these parallelepipeds of D that correspond to the projections that were chosen in the previous step. It should be mentioned that the set T is not ordered, we traverse set T until we find the first parallelepiped $t \in T$ that can "embody" r. If such t is not found, then we put r in an "auxiliary" set B and we proceed to the next parallelepiped r of L. Otherwise we proceed with the embodiment and we repeat this step in the part of r that is left after the embodiment, until no other embodiment can be done. If the set is not empty after the end of this procedure, we put this in set B. At the end of this procedure, we continue working with set B. For every one parallelepiped $b \in B$ we define the set T_b of the parallelepipeds of D, that b crosses. We traverse T_b until we find the first parallelepiped $t_b \in T_b$ that can embody b. If there is not such a t_b, we leave b and we go on with the next parallelepiped b of B; otherwise we proceed with the embodiment and repeat that step with the part of b that is left after the embodiment until no other action can be made. If a part of it is left unused, after the end of this procedure, we place that part in set B.

The question that arises here is how many times we should traverse the whole set B. The answer is that we stop traversing set B either when it is made $\varnothing$ or when we find out that during the traversal, no embody has taken place. Then we set $D \leftarrow D \cup B$ and we proceed with the next layer.

We will try to describe when we embody a parallelepiped r (or b) with a parallelepiped d of D, without using formal mathematical symbols. We should mention that this embodiment happens only when the projection r_p (b_p) in the *X*-*Z* plane of r (or b) has at least two common angles with the corresponding projection of d. If this is the case, then:

- if r has prominence, then increase the height of d by 1 and transform r in a way that its projection is $r_p \cup d_p - r_p \cap d_p$ (the same stands for b).
- if d has prominence, then remove d from D and add into D the parallelepiped having the following properties:
  a) its projection is the $r_p \cap d_p$ ($b_p \cap d_p$),
  b) the *y*-coordinate of its front face is equal to the coordinate of d's front face and
  c) its height is the height of d plus 1.
  Add into D the parallelepiped having the following properties:
  a) its projection is the $r_p \cup d_p - r_p \cap d_p$ ($b_p \cup d_p - b_p \cap d_p$),
  b) the *y*-coordinate of its front face is equal to the coordinate of d's front face and
  c) its height is the height of d.
  As r (b) has been covered, get the next parallelepiped from L (remove b from B).

A more formal description of the algorithm follows:

Step 1: Divide the query region into layers of unitary cubes.
Step 2: In every layer execute Algorithm 3.
Step 3: Initialise D with the parallelepipeds of the layer, having the lowest y-coordinate.
Step 4: FOR any one of the rest layers, DO:
Step 4.1: r=next parallelepiped of the layer.
Step 4.2: find the set T that corresponds to r.
Step 4.3: $\forall t \in T$ do:
Step 4.3.1: t=next parallelepiped of T.
Step 4.3.2: IF cardinality $(r_p \cup d_p - r_p \cap d_p) < 2$
THEN embody r with t.
IF $r \neq 0$
THEN go to **Step 4.2**.
Step 4.4: Traverse B.
Step 4.5: WHILE $B \neq 0$ AND embodiments have occurred during the traversal of B,
go to **Step 4.4**.
Step 4.6: IF $B \neq 0$
THEN $D \leftarrow D \cup B$.

In the implementation of this algorithm the sets D and B are implemented using doubly-linked lists. As the parallelepipeds of unitary thickness, that are constructed during the execution of Algorithm 3 in every layer are not ordered, we have to implement in a serial way every find-procedure for the above lists. This causes a decrease of the speed of the algorithm.

7. Conclusion-Comparison of the Algorithms

The algorithms have been tested in practice. We used them on files containing regions in 3D space. Every file contained unitary cubes, described with triples of coordinates. We calculated the number of rectangles that every algorithm generates. A small number of rectangles indicate good performance. Table 1 shows the experimental results of the performance of the five algorithms in four different cases (these cases are data representing areas from the cartography of human brain). The number in brackets expresses the percentage of the compression over the simple approach; that is Algorithm 1. In other words we assume that Algorithm 1 is the normal compression on the query regions. The number in brackets will present the percentage of the free space achieved after the usage of each one of the other four algorithms. The last column presents the average performance of its one of the algorithms. As it was expected, the last algorithm, Algorithm 5 has the best performance because it actually works in 3D space. The fourth algorithm, that produces overlapping parallelepipeds of unitary thickness, has the second good performance. The algorithm of Franzblau-Kleitman (algorithm 2) lies in the middle, because it divides the region into x convex regions and these regions are not always the best for all cases.

	Case 1	Case 2	Case 3	Case 4	Average

	706,397 Cubes in 102-198 planes	24,955 cubes in 136-163 planes	505,765 cubes in 106-189 planes	80,357 cubes in 140-172 planes	Reduction
Algorithm 1: A Simple Approach	98,824 (0%)	2,990 (0%)	60,599 (0%)	9,635 (0%)	0%
Algorithm 2: An Invariant of The Algorithm of Franzblau-Kleitman	94,516 (4.35%)	2,621 (12.34%)	53387 (11.9%)	8,668 (10,03%)	9.66%
Algorithm 3: The non Overlapping Parallelepiped Algorithm	85,654 (13.32%)	2,492 (16.65%)	51,437 (15.12%)	8,270 (14.17%)	14.82%
Algorithm 4: The Overlapping Parallelepiped Algorithm	84,066 (14,93%)	2,418 (19.13%)	49,661 (18.05%)	8,119 (15.73%)	16.96%
Algorithm 5: The 3D non Overlapping Parallelepipeds Algorithm	82,853 (16.16%)	2,406 (19.53%)	49389 (18.5%)	7,956 (17.42%)	17.9%

Table 1. The experimental results

Table 2 presents the compression rate of each algorithm for the four cases; that is the percentage of the free space achieved after the usage of its one of the compression algorithms. As it can be easily derived the Algorithm 5 gives the best average compression rate.

	Case 1 706,397 Cubes in 102-198 planes	Case 2 24,955 cubes in 136-163 planes	Case 3 505,765 cubes in 106-189 planes	Case 4 80,357 cubes in 140-172 planes	Average Reduction
Algorithm 1: A Simple Approach	86.01%	88.02%	88.02%	87.05%	87.28%
Algorithm 2: An Invariant of The Algorithm of Franzblau-Kleitman	86.62%	89.5%	89.4%	89.21%	88.68%
Algorithm 3: The non Overlapping Parallelepiped Algorithm	87.885%	90.01%	89.83%	89.7%	89.36%
Algorithm 4: The Overlapping Parallelepiped Algorithm	88.1%	90.31%	90.18%	89.89%	89.62%
Algorithm 5: The 3D non Overlapping Parallelepipeds Algorithm	88.27%	90.36%	90.2%	90.01%	89.71%

Table 2. Compression rates

Although these algorithms are actually heuristic, the performance comparison with the simple approach (algorithm 1) and a near optimal algorithm (algorithm 2, Franzblau-Kleitman) can give us interesting results about the optimality of the last algorithm. This comparison verifies that the last algorithm has an experimental performance better than one of the most powerful algorithm; the algorithm of Franzblau and Kleitman. The proof that the last algorithm is optimal or near optimal is a difficult process. Testing showed that optimal results are achieved using algorithm 5. A formal mathematical proof should be done in order to prove the experimental results and complete our theoretical research. This is going to be our future attempt to prove the optimality of this algorithm.

References

Anogianakis, G., Krotopoulou, K., Spirakis, P., Terpou, D. And Tsakalidis A., 1992, The logical design of the Brain DataBase, Technical Report MB1.1, AIM Project, Magnobrain no A2020.

Anogianakis, G., Krotopoulou, K., Spirakis, P., Terpou, D. And Tsakalidis A., 1992, Design and Implementation Issues of the Brain DataBase, Technical Report MB1.2, AIM Project, Magnobrain no A2020.

Edelsbrunner, H., Leeuwen, J.V., Ottmann, T. and Wood, D., 1984, Computing the Connected Components of Simple Rectilinear Geometrical Objects in d-space, R.A.I.R.O. Theoretical Informatics, 18, 2, pp 171-183.

Franzblau, D. and Kleitman, D. J., 1984, An algorithm for constructing regions with rectangles, independence and minimum generating sets for collections of intervals, in Proc. of the 16th ACM Symp. Theory Computing, pp 167-174.

Gutting, R.H., 1984, An optimal contour algorithm for iso-oriented rectangles, Journal of Algorithms, 5, pp 303-326.

Preparata, F. and Shamos, M.I., 1985, Computational geometry (Spring-Verlag, Berlin).
Samet, P.A., 1990, Applications of spatial data structures (Addison-Wesley, Reading, Mass).
Samet, P.A., 1990, The design and analysis of spatial data structures (Addison-Wesley, Reading, Mass).