SCI INSTITUTE
TECHNICAL REPORT

Clustering on Single Refinement Level: Berger-Rigoustos Algorithm

Oren E. Livne

UUSCI-2006-001

Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA

January 23, 2006

Abstract:

We describe the Berger-Rigoustos algorithm for clustering points, and its current implementation for our purposes. The goal is to define rectilinear patches over the set of flagged cells that are indicated by a refinement criterion. We show results for the model problems from the original papers on the algorithm and discuss the required changes to the current algorithm for our purposes.

THE UNIVERSITY OF UTAH

Clustering on Single Refinement Level: Berger-Rigoustos Algorithm

Oren E. Livne *

January 23, 2006

Abstract

We describe the Berger-Rigoustos algorithm for clustering points, and its current implementation for our purposes. The goal is to define rectilinear patches over the set of "flagged cells" that are indicated by a refinement criterion. We show results for the model problems from the original papers on the algorithm and discuss the required changes to the current algorithm for our purposes.

Key words. Clustering, rectangles, signatures, refinement criterion.

1 Goals and Assumptions

A refinement criterion will be developed in ICE and flag certain cells of our computational grid as "cells that need further refinement". Regardless of the criterion, we need to develop the machinery to construct finer-level patches over these flagged cells. For simplicity, we stick in this feasibility study to a 2D domain and 2D boxes. The report is organized as follows. In §1 we present our assumptions and list the computational objectives. In §2 we list the measures by which we measure the algorithm's result. §3 is devoted to a description of the algorithm. §4 contains numerical experiments with the

*SCI Institute, 50 South Central Campus Dr., Room 3490, University of Utah, Salt Lake City, UT 84112.Phone: +1-801-581-4772. Fax: +1-801-585-6513. Email address: livne@sci.utah.edu

algorithm. We try various test cases described in the papers discussing the algorithm. We show both "generic" and "hard" scenarios. §5 discusses the results and suggests the next steps of development.

We will restrict the discussion to

- Rectilinear patches, that is, rectangular boxes that cover the flagged cells.
- Non-overlapping patches.

These are not necessarily natural assumptions: in many cases, the optimal patches (from a numerical viewpoint of storage vs. accuracy of solution, artificial anisotropies in the grids, etc.) are stretched (that we can have in the current setting), rotated, or transformed through a local coordinate system (the latter two are of course not possible). However, for feasibility purposes, and under the constraints of the current working environment, we will assume that, and bear in mind the possible extensions for more complicated patch structures.

For a general set of flagged cells, we focus on obtaining a set of covering boxes. Our ideal objectives for the boxes are as follows.

1. *Objective 1 - Maximum efficiency:* the ratio of the number of flagged cells to the total box area, should be as close to 1 as possible. We would like to minimize the wasted "blank space" by the rectangles: the total work and storage of the patches in the actual solver is proportional to the total box area.
2. *Objective 2 - Minimum box size:* the smallest box should not be less than (say) 4 cells in every direction. Otherwise, there would be a large overhead that would not justify the use of such boxes.
3. *Objective 3 - Maximum box volume:* patches that are 32^3 or $64 \times 64 \times 8$ are equivalent in terms of memory and cost, but we would not want very large box volumes in light of a worse load balancing between processors, in a parallel processing framework.
4. *Objective 4 - Minimum flag distance from box boundary:* the boxes have to be re-generated every (several) time-steps, and the location of the flags move (e.., a shock wave front). We want to keep the same rectangles for as long as possible, and so would not want flagged cells right near the box edges, as they might move out in the next time-step.

5. *Objective 5 - Minimum box mutual-boundary area:* to minimize processor communication, we would like the boxes to have as low mutual edges as possible. One way to indirectly achieve this is by trying to construct more "cubic" boxes than "thin" ones, thereby reducing the edge area of each box (independently of the other boxes' edge area, though).

In the current report we present an algorithm that treats Objective 1, and in part, Objective 5. We maximize efficiency, and the algorithm also tries to construct "cubic" boxes.

2 Indicators

We assess the quality of a constructed set of boxes by the following measures. The measures are naturally related to the objectives listed in §1.

1. Efficiency: ratio of the number of flagged cells to the total box area [dimensionless] (Objective 1).
2. Number of boxes (important for parallelism in general).
3. Minimum edge size of a box [cells] (Objective 2).
4. Maximum box volume [cells] (Objective 3).
5. Average box volume (related to Objective 2 and 3 and to load balancing).
6. Minimum flag distance from box edge [cells] (Objective 4).
7. Average ratio of the shorter to the longer side ratio [dimensionless] (Objective 5). Note that this is not the total boundary length to the total box volume; but for a cluster of many boxes, the "localized" measure will be comparable with the desired "global" ratio, divided by the number of boxes. We prefer this localized measure, which is easier to compute and to control with our algorithm. It is also dimensionless – it does not increase with the number of boxes.

3 The Clustering Algorithm

We use the Berger-Rigoustos clustering algorithm (basically, it is a "smart bisection" algorithm). The algorithm has been first purposed in [BR91], including a lot of numerical experiments. An improved version has been suggested in [JBW94]. We use the improved version, with trivial modifications. The numerical experiments of §4 include both the test case battery of [BR91], some hard test cases described therein, and some other examples we came up with to illustrate the strengths and weaknesses of the algorithm. In this section we describe the algorithm's flow, for the 2D case.

3.1 Signatures

Given a continuous function $f(x, y)$, the horizontal and vertical signatures Σ_x and Σ_y are defined as

$$\Sigma_x := \int_y f(x, y)dy$$

$$\Sigma_y := \int_x f(x, y)dx,$$

respectively. For a binary image $\{f_{ij}\}_{i,j}, i = 1, \ldots, m, j = 1, \ldots, n$, these translate into

$$\Sigma_x(i) := \sum_{j=1}^{n} f_{ij}$$

$$\Sigma_y(j) := \sum_{i=1}^{m} f_{ij}$$

The summations clearly extend over the non-zero f_{ij} only. In case of a binary image, Σ_d counts the number of non-zero cells (pixels) along a planar cut in the d-direction. We also use the discrete second derivative of Σ (in each direction),

$$\Delta_d(i) := \Sigma_d(i-1) - 2\Sigma_d(i) + \Sigma_d(i+1),$$

defined except at the first and last cell of a rectangle under consideration.

3.2 The Algorithm

The input for the algorithm is a list of gridpoints flagged as needing refinement. In the description below, we use the `flag` input array as a 2D binary

image. In practice, we may want to use proper indexing to keep only the list of flagged point coordinates in `flag`,

The output is a list of boxes, that is, an $K \times 4$ array, where K is the number of boxes, and each box is designated by (x_1, y_1, x_2, y_2): its lower-left corner (x_1, y_1), and upper-right corner (x_2, y_2). This output array is denoted by `rect`.

The pseudo-code that follows uses the MATLAB notations. It can in fact be readily used as a MATLAB script.

```
threshold   = 0.9;                                      % Lowest rectangle efficiency allowed

%%%%%%%%%% Initialize and find the first rectangle
dim         = length(size(flag))                        % Dimension of the problem
[i,j]       = find(flag);                               % Index arrays [i,j] of flagged cells
box         = [min(i) min(j) max(i) max(j)];            % Bounding box for flagged cells
rect        = box;                                      % List of rectangles, has one rectangle to start with

%%%%%%%%%% Main algorithm: loop over rectangles, process them, and possiblly add more rectangles
k       = 1;                                            % Index of rectangle to be processed

while (k <= size(rect,1))                               % Do until all rectangles have been processed
    r           = rect(k,:);                            % Rectangle parameters
    s           = flag(r(1):r(3),r(2):r(4));            % Flag data of this rectangle
    rect_size   = size(s);                              % Vector containing the size of the rectangle: [size_x,size_y]
    efficiency  = length(find(s))/prod(rect_size);      % Percentage of flagged cells in s

    if (efficiency < threshold)                         % Rectangle not efficient, try to "smartly bisect" it
        %%%%% Compute signatures
        for d = 1:dim
            sig{d} = s;
            for j = 1:dim
                if (j ~= d)                             % Loop over all dimensions except d
                    sig{d} = sum(sig{d},j);             % Sum along the j's dimension
                end
            end
            % sig{d} = sum(s,d);                          % Integrate along the d-direction
            %            sig{d}
        end
        cut_found   = 0;                                % 0 if we haven't found a place to bisect, 1 if we have

        %%%%% Look for a hole: a zero value in one of the signature
        for d = 1:dim
            hole = find(sig{d} == 0);
            if (~isempty(hole))
                % Right now using the first found hole; later could switch to the closest to the center of the rectangle
                cut_found   = 1;
                cut_place   = hole(1)+r(d)-1;
                cut_dim     = d;
                break;
            end
        end

        %%%%% Look for an inflection point: a crossing point for the second derivative of the signature
        if (~cut_found)
            best_place  = -ones(dim,1);                 % Init with -1's
            value       = -ones(dim,1);                 % Init with -1's
            for d = 1:dim
                delta{d} = diff(diff(sig{d}));
                schange_abs = abs(diff(delta{d}));
                schange = zeros(size(schange_abs));
                for i = 1:length(delta{d})-1
                    schange(i) = sign(delta{d}(i)*delta{d}(i+1));
                end
                zero_crossing = find(schange <= 0);
                if (~isempty(zero_crossing))
                    max_crossing_value = max(schange_abs);
                    max_crossing = find(schange_abs == max_crossing_value) + 1;
                    center = length(sig{d})/2 + 0.5;
                    distance_from_center = abs(max_crossing+0.5 - center);
```

```
                best_crossing = max_crossing(find(distance_from_center == min(distance_from_center)));
                best_place(d) = best_crossing(1) + r(d)-1;
                value(d) = max_crossing_value;
            end
        end
        zero_crossing_dims = find(value >= 0);
        if (~isempty(zero_crossing_dims))
            max_crossing_dims = find(value == max(value));
            best_dims = max_crossing_dims(find(rect_size(max_crossing_dims) == max(rect_size(max_crossing_dims))));
            cut_found = 1;
            cut_dim = best_dims(1);                     % If there's more than one dimension, take the first one
            cut_place = best_place(cut_dim);
        end
    end

    %%%%% No holes or inflection points; base rectangle acceptance on its efficiency; bisect if not efficient enough
    if (~cut_found)
        if (efficiency <= 0.5)
            longest_dims = find(rect_size == max(rect_size));
            cut_found = 1;
            cut_dim = longest_dims(1);
            cut_place = floor(rect_size(cut_dim)/2);
        else
            % Rectangle has more than a 0.5 efficiency, accepted
        end
    end

    if (cut_found)
        r1 = r;                                     % Create the "left half"
        r2 = r;                                     % Create the "right half"
        r1(cut_dim+dim) = cut_place;                % The "left half" new coords
        r2(cut_dim)     = cut_place+1;              % The "right half" new coords
        s           = flag(r1(1):r1(3),r1(2):r1(4));% Flag data of this rectangle
        [i,j]       = find(s);
        r1          = [r1(1:2) r1(1:2)]-1 + [min(i) min(j) max(i) max(j)];
        s           = flag(r2(1):r2(3),r2(2):r2(4));% Flag data of this rectangle
        [i,j]       = find(s);
        r2          = [r2(1:2) r2(1:2)]-1 + [min(i) min(j) max(i) max(j)];

        rect        = [rect; r1; r2];               % Add the two halves to the list
        rect(k,:)   = [];                           % Delete rectangle k from list, so now k points to the next one
    else
        k = k+1;                                    % Couldn't find a cut; accept this box and consider the next one
    end
else
    k = k+1;                                        % Rectangle is efficient, consider the next rectangle on the list
end
end
```

4 Numerical Experiments

Each test case is a set of flagged cells in 2D. For each case, we plot the original points and the resulting covering. The tables include some summarizing statistics (indicators, see §2). In all cases, we used a minimum efficiency threshold of .8.

4.1 Generic Test Cases

We first tested the algorithm over the test cases (G1–G4) described in [BR91]. Unfortunately, the data reported there was wrong (the number of flagged cells is even not the one shown in the picture). However, the qualitative picture of box covering is the same here and in that paper.

Table 1: Generic Test Cases: Final Statistics

Case	Flagged Cells	Total Box Volume	Effi- -ciency	# Boxes	Min. Edge	Max. Volume	Avg. Box Volume	Avg. Box Edge Ratio
G1	614	668	.919	11	2	128	60.7	.336
G2	623	683	.912	13	2	288	52.5	.564
G4	245	281	.872	11	1	110	25.5	.761
G5	233	264	.883	10	1	104	63.8	.638

4.2 Hard Test Cases

The paper [BR91] discusses some cases where the algorithm fails to obtain an optimal covering. We include these cases, along with some other examples that are considered "hard to address":

H1 : Two rectilinear blocks in diagonal constellation. Without the direct bisection step, we would put one bounding box over both of them, reducing the efficiency to .5.

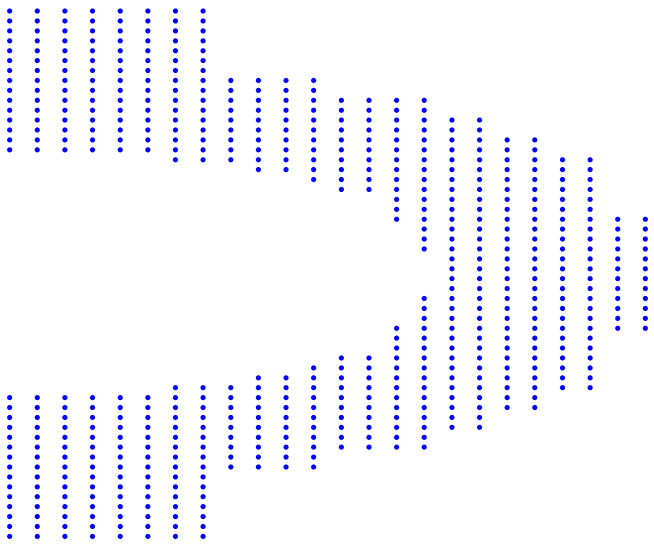
H2 : The "Fig. 18" example from [BR91]. The algorithm fails to find an optimal dissection at the first step, thereby leading to three boxes instead of two.

H3 : red-black grid with a lot of "holes" in it. The algorithm finds it confusing to adapt to the .8 efficiency threshold.

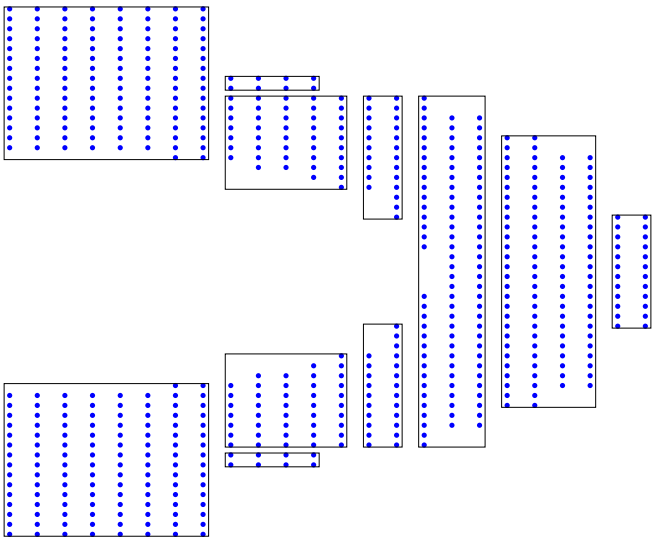
H4 : (suggested by Steve and Todd) randomly (Gaussian) distributed points around the center point of the grid shown. The standard deviation is 3 cells in every direction.

Table 2: Hard Test Cases: Final Statistics

Case	Flagged Cells	Total Box Volume	Effi--ciency	# Boxes	Min. Edge	Max. Volume	Avg. Box Volume	Avg. Box Edge Ratio
H1	82	84	.976	3	1	42	28	.571
H2	108	108	1.00	2	6	54	54	.667
H3	61	81	.753	41	9	9	2.0	1.00
H4	66	81	.815	30	1	9	2.7	.869

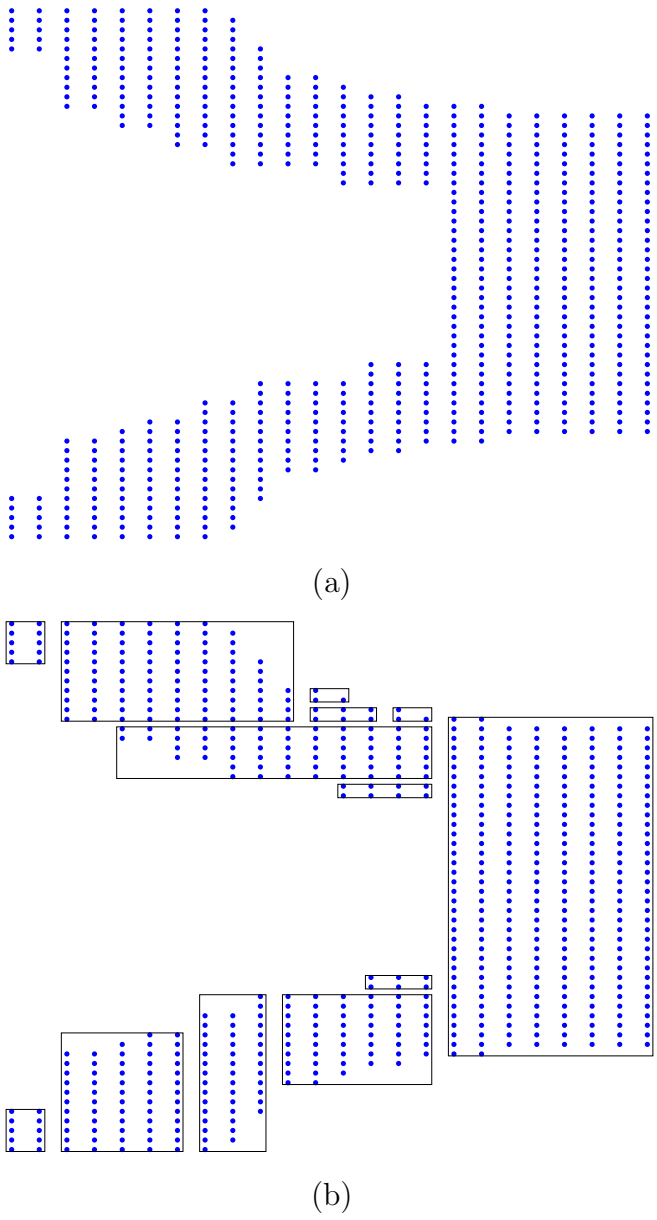


(a)



(b)

Figure 1: Test Case G1 results. Upper figure: the original cells. Lower figure: the cells with the box covering.



(a)

(b)

Figure 2: Test Case G2 results. Upper figure: the original cells. Lower figure: the cells with the box covering.

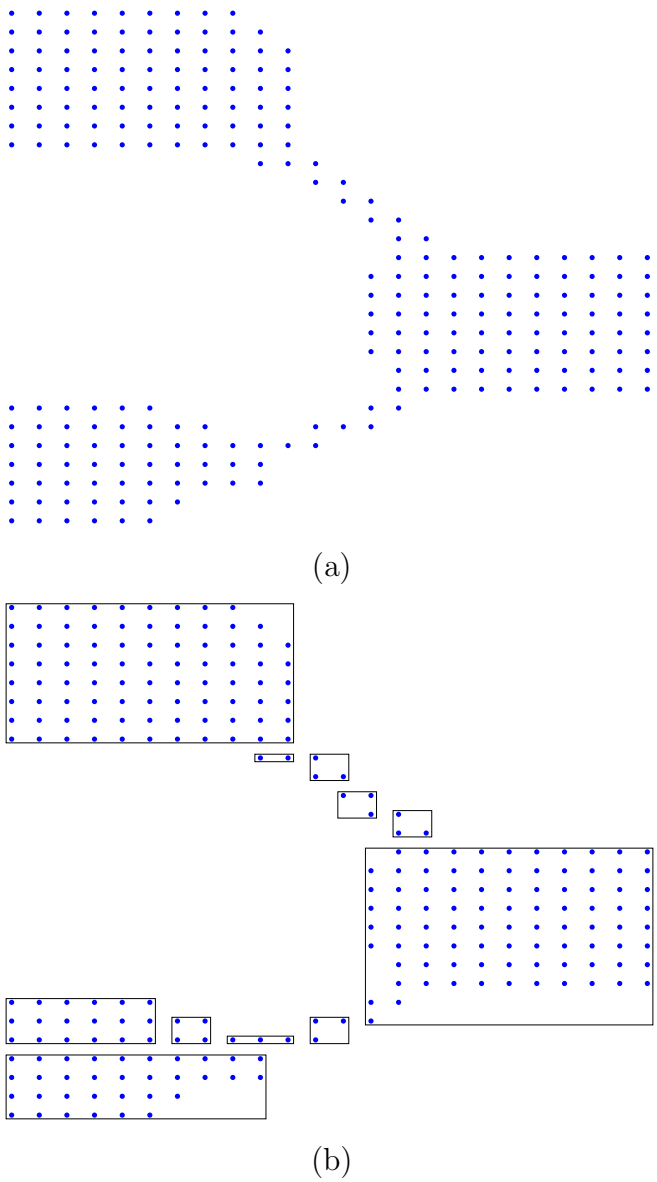
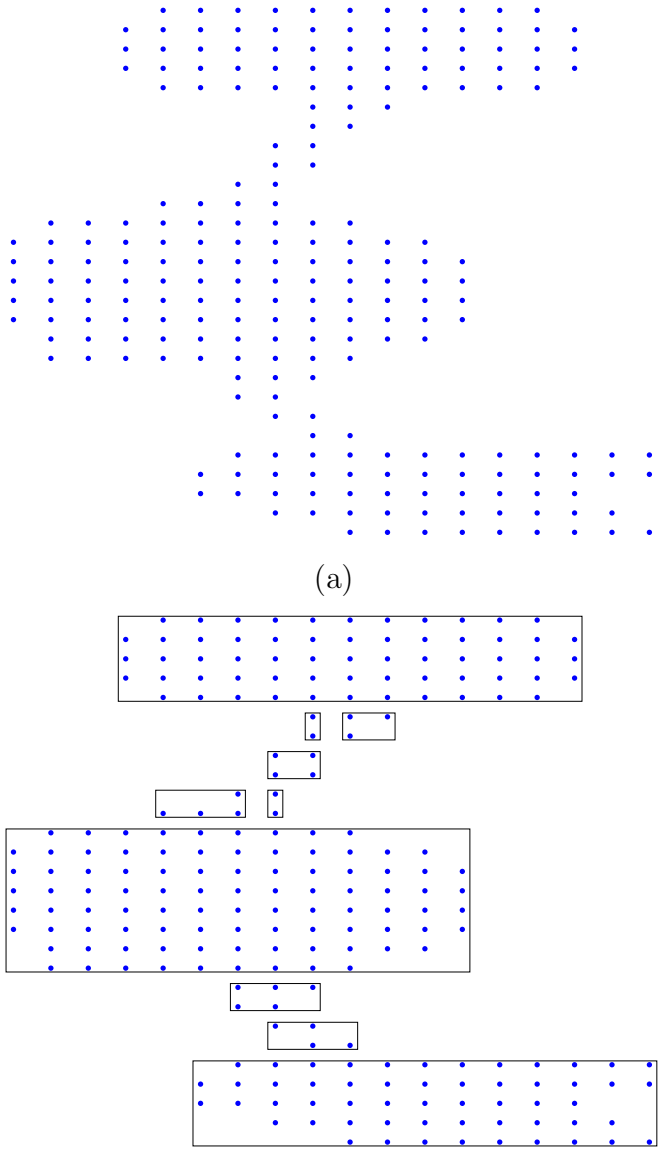


Figure 3: Test Case G4 results. Upper figure: the original cells. Lower figure: the cells with the box covering.



(a)

(b)

Figure 4: Test Case G5 results. Upper figure: the original cells. Lower figure: the cells with the box covering.

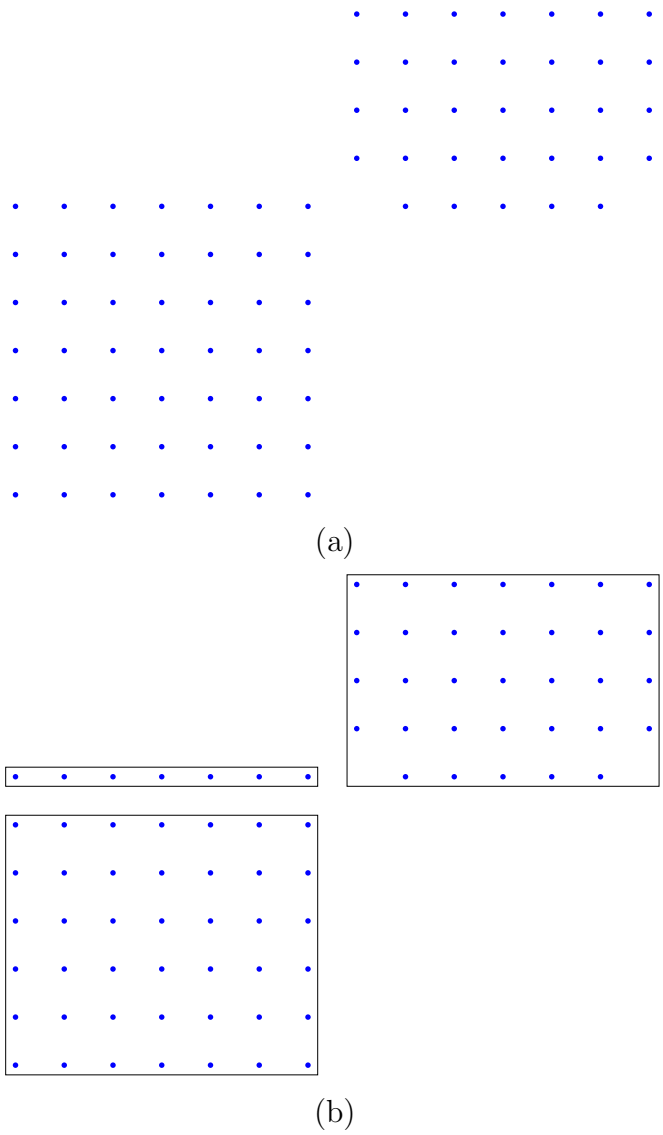


Figure 5: Test Case H1 results. Upper figure: the original cells. Lower figure: the cells with the box covering.

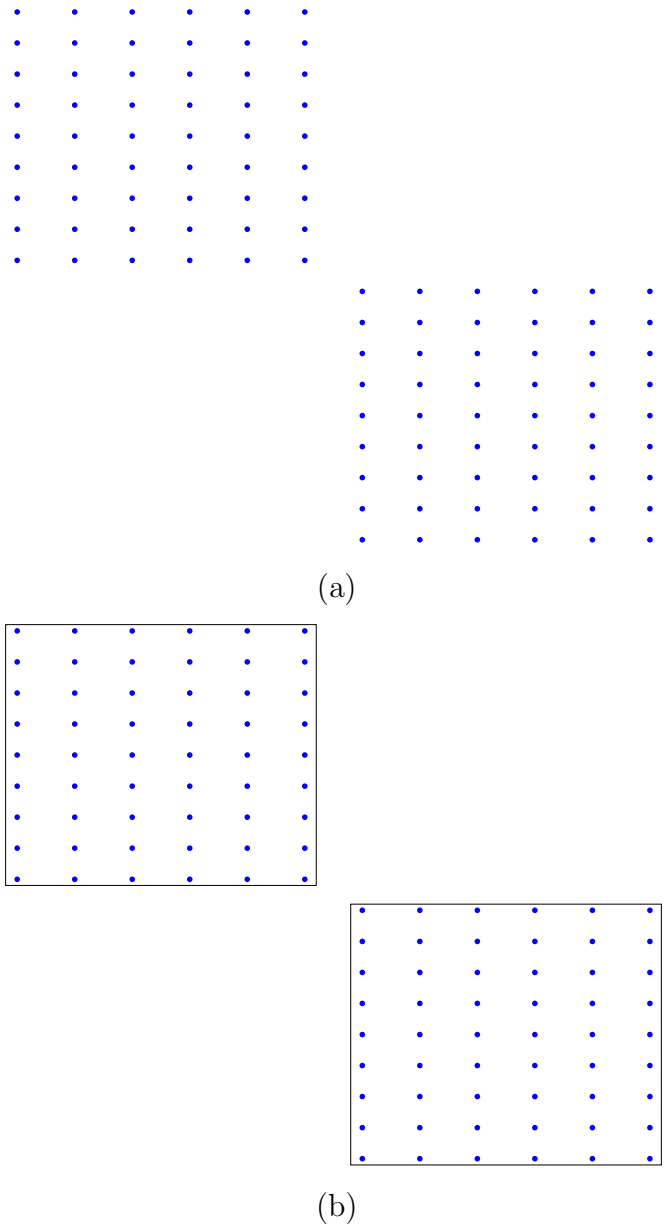


Figure 6: Test Case H2 results. Upper figure: the original cells. Lower figure: the cells with the box covering.

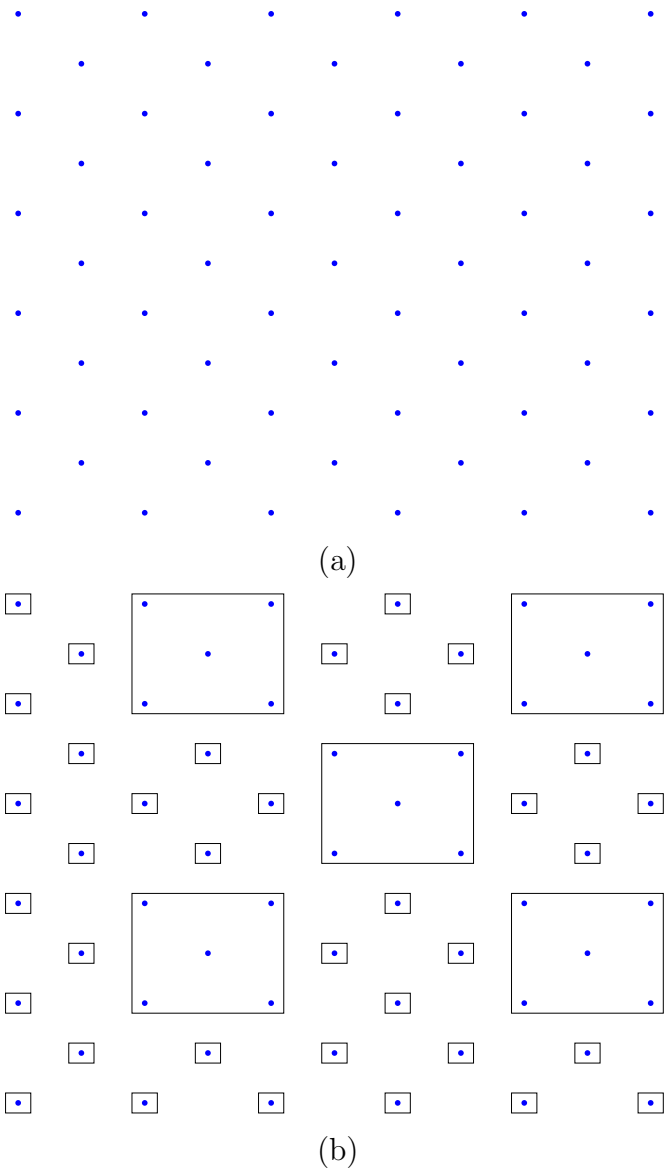
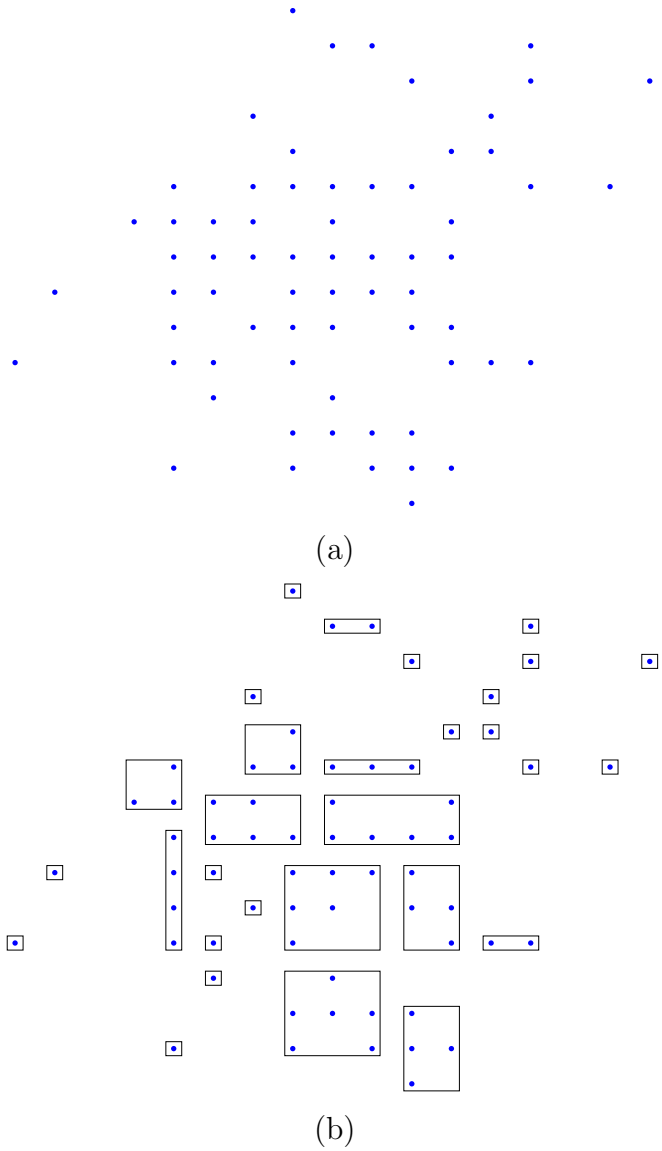


Figure 7: Test Case H3 results. Upper figure: the original cells. Lower figure: the cells with the box covering.



(a)

(b)

Figure 8: Test Case H4 results. Upper figure: the original cells. Lower figure: the cells with the box covering.

5 Conclusions

We examined the behavior of the Berger-Rigoustos clustering algorithm on a battery of test cases. The main findings are as follows.

- The results reported in [BR91],[JBW94] have been reproduced, at least qualitatively (missing details and wrong data in the papers make it hard to compare bit-by-bit, but the pictures of box covering look similar in our results and the papers' results).
- The algorithm's objective is to maximize efficiency (Objective 1). Given a minimum efficiency threshold, the algorithm attained it in all test cases except H3, and in many cases, yielded yet a much higher efficiency. Objective 5 (making the boxes as cubical as possible) has been implicitly implemented in the algorithm, making sure that box dissections are to be preferred in the "longest edge dimension". The results show that the average ratio of edge lengths is not very small (typically, around .5), which can be considered a "good value" of this measure.
- The algorithm produces very small boxes, for almost all cases (the minimal box size is 1, mostly). Also, there is no limit on the maximum box volume. The algorithm should be modified to fit our needs. Control parameters for the minimum and maximum box sizes should and can be specified (this will appear in a next report on this subject).
- The minimum distance of flagged cells from the boundary is of course zero in this algorithm, that uses tight bounding boxes whenever it can, to maximize efficiency. A pre-processing phase of "dilating" the flagged cells areas can solve this problem to obtain Objective 4.

References

[BR91] M. J. Berger and I. Rigoustos. An algorithm for point clustering and grid generation. *IEEE. Trans. Sys. Man Cyber.*, 21 (5):1278–1286, 1991.

[JBW94] J. Saltzman J. Bell, M. J. Berger and M. Welcome. Three-dimensional adaptive mesh refinement for hyperbolic conservation laws. *SIAM. J. Sci. Comput.*, 15 (1):127–138, 1994.