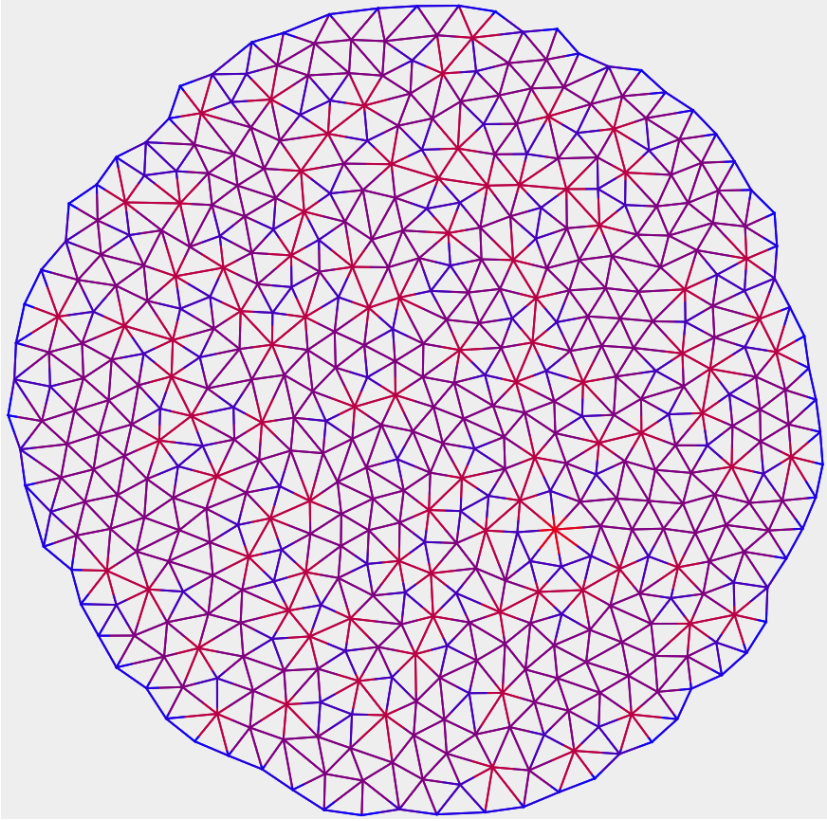
Canning for Amorphous Blob Computing

TER report

Auteur :
Lucas Labouret
M1 QDCS, Université Paris-Saclay
lucas.labouret@universite-paris-saclay.fr

Encadrant :
Frédéric Gruau
LISN
frederic.gruau@universite-paris-saclay.fr



Contents

A What is Blob Computing ?

This TER is part of a long term research project about the concept of "blob computing"[4]. To understand the exact subject of this TER, it is important to first introduce what blob computing means and how it works.

The ultimate goal of the project is to develop a new paradigm of computation that is inherently arbitrarily scalable through the use of *space* as a resource. This kind of computation is becoming increasingly useful, for example with the recent development of swarm robotic[7].

We aim to achieve this through the use of arbitrarily many processing elements (PEs) distributed through space and locally connected. These PEs, which can be severely lacking in power on their own, create a computing medium where virtual "blobs" can form and evolve. These blobs are the main primitive we use for making computations.

IMPORTANT $\longrightarrow$ blobs persist through time

A.1 Definition and Structure of a Computing Medium

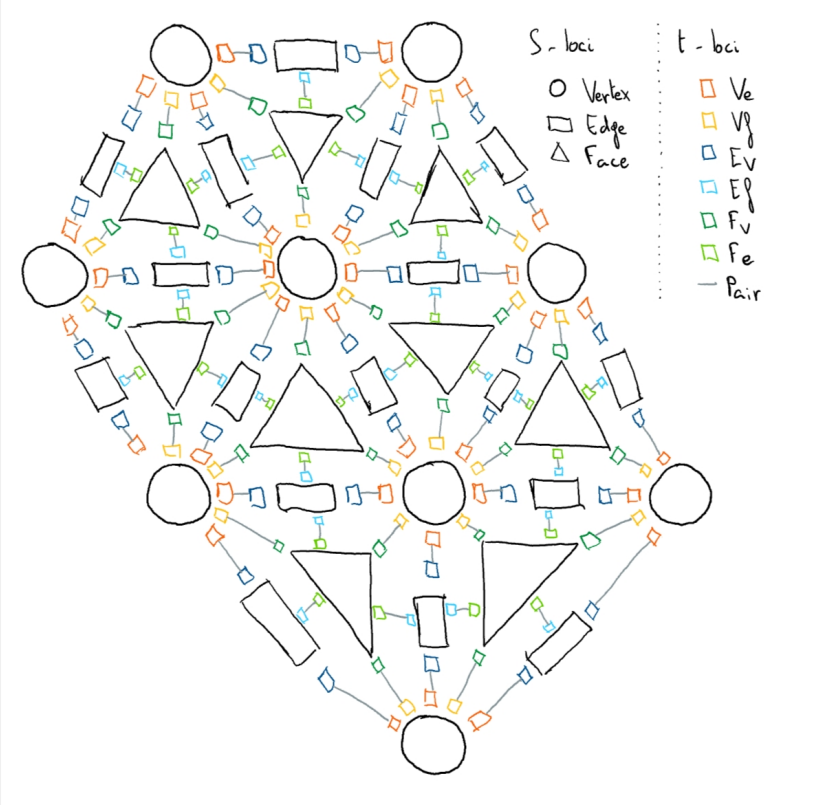


Fig. 1: Example of triangulated medium.

In its most general form, a computing medium is simply a set of processing elements distrubuted in some space which are locally connected. Here, we focus exclusively on a specific kind of meia called "triangulated computin media", hence the term "computing medium" in the rest of this report will refer to those unless specified otherwise. A triangulated medium is represented as a weakly Delaunay-triangulated graph where each vertex, edge, and face is associated with a processing element capable of executing minimal computation, storing information, and communicating with its neighbors.

Vertices, edges and faces are called "simplicial loci" (or "S-loci" for short). They determine the overall structure of their medium. Each S-locus controls a set of "transfer loci" (or "t-loci"). T-loci come in pairs that handle the communication between two S-loci. For example, if a vertex was connected to an edge, there would be a pair of t-loci (one belonging to the vertex and the other belonging to the edge) between them.

Obviously, there are 3 types of S-loci : Vertex, Edge, Face. T-loci are divided

into 6 types :

- Ve linking a Vertex to an Edge
- Vf linking a Vertex to a Face
- Ev linking an Edge to a Vertex
- Ef linking an Edge to a Face
- Fv linking a Face to a Vertex
- Fe linking a Face to an Edge

Figure 1 show an example of how all these loci are put together to form a medium.

A medium is said to be crystalline if a repeating structure emerges in the placement if its vertices. It is amorphous when no such structure emerges. Additionally, it is homogeneous if the average point density is approximately constant and there are no "holes" or "clusters", and it is isotropic if the directions of the edges of the medium follow a uniform distribution over $[0, \pi[$. This report mainly focuses on amorphous homogeneous isotropic (AHI) media.

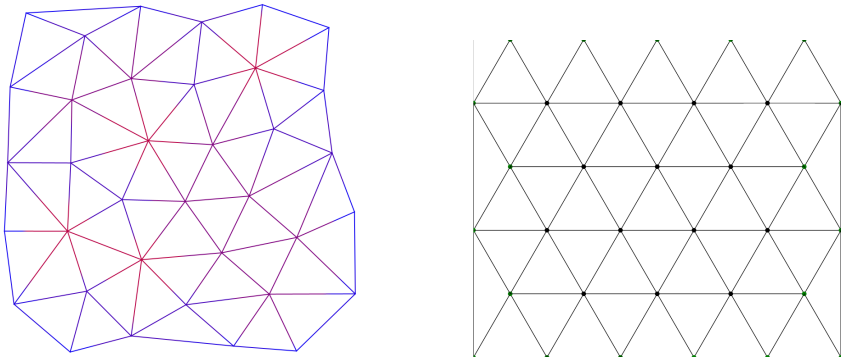


Fig. 2: An AHI medium (left) VS a crystalline hexagonal medium (right).

For now, we consider PEs to be immobile and synchronous, although lifting these two constraints will probably be the subject of future research.

Blobs form the computational basis of blob computing. A blob of a given medium M is a connected sub-graph M' of M such that all the vertices of M' share a common property (e.g.: all their values are even, or all their values are 1, etc.). Formally, it is a "True" connected component of a boolean field over the vertices (see [5]).

Importantly, blobs are persistent through time, meaning that two blobs at two different points in time can be considered one and the same under the right conditions, even though blobs are able to change their shape over time. This is possible for two reasons :

1. Blobs maintain their connexity, they tipycally don't merge or divide unless instructed otherwise.
2. Blobs only change shape at their border : at each time step they can grow

to unocupied vertices adjecent to their border, or they can lose vertices that are part of their border.

Therefore, two blobs at two consecutive timestep are "the same blob" if they only differ at their border. We can then extend the notion to any number of timestep through its transitive closure.

A.2 An Example of Computation : the Voronoï Diagram

[2] develops tools to program on computing media. As an example, it shows how to build a (discrete) Voronoï diagram from a set of "seeds" in a medium.

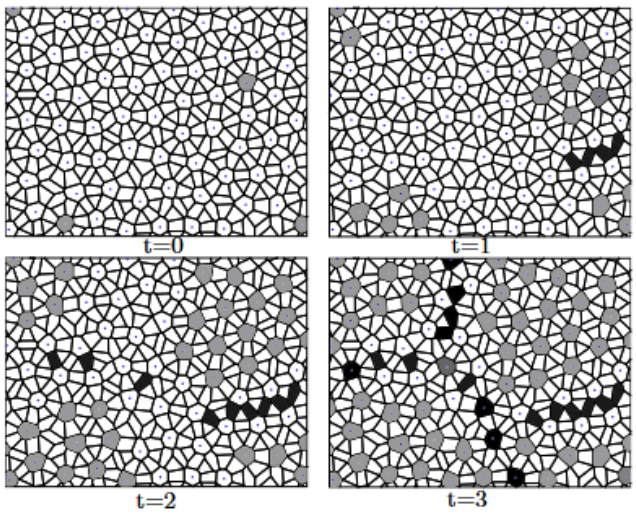


Fig. 3: Computation of a discrete Voronoï diagram.
Cells marked with a dot are the vertices of the medium. Gray cells form blobs. Black cells mark the frontiers between blobs. At t=0, blobs only cover the seeds. At t=3, blobs cover the Voronoï regions of the medium.
Source : [2]

The basic idea is the following : we start with a set of "seeds" distributed over the vertices of the network. Each seed is now the center of a blob. Then, at each synchronous step, the blobs will spread to their adjacent vertices if this wouldn't cause them to merge with another blob, until no blob can spread further. It can be implemented with very few types of local operations that are applied simultaneously to every S-locus in the network :

- Multi-cast
  The S-locus copies its value into its t-loci of a given type.

- Rotation
  Since t-loci are spatially arranged around their S-loci, we can define a

clockwise and counter-clockwise neighbor. T-loci can perform logical operations on their and their neighbors' values.

- Reduction
  The S-locus stores the result of a commutative and associative operation applied to the values of its t-loci of a given type.

- Transfer
  Each of the S-locus' t-loci exchanges its value with the value of its pair.

In fact, clever association of these primitives allows to program everything a computing medium is capable of, such as sorting arrays of integer or multiplying matrices[5].

A.3 Objective : Cannings

Up until now, Gruau mainly focused on hexagonal crystalline media. He built a plateform allowing efficient simulation of blob computation on classical hardware using SIMD[2,3].

The point of this TER is to allow the use of SIMD to optimize the simulation of AHI media as well. We do this through the development of the notion of "total canning", or simply "canning". Given a medium M, a canning is a set of 9 injective maps which assign some integer coordinates to each locus of M :

1. $\{\text{Vertex}_M\} \mapsto \mathbb{N}^{a_1}$
2. $\{\text{Edge}_M\} \mapsto \mathbb{N}^{a_2}$
3. $\{\text{Face}_M\} \mapsto \mathbb{N}^{a_3}$
4. $\{\text{Ve}_M\} \mapsto \mathbb{N}^{a_4}$
5. $\{\text{Vf}_M\} \mapsto \mathbb{N}^{a_5}$
6. $\{\text{Ev}_M\} \mapsto \mathbb{N}^{a_6}$
7. $\{\text{Ef}_M\} \mapsto \mathbb{N}^{a_7}$
8. $\{\text{Fv}_M\} \mapsto \mathbb{N}^{a_8}$
9. $\{\text{Fe}_M\} \mapsto \mathbb{N}^{a_9}$

   where $a_1, ..., a_9 \geq 1$ are specific to each canning method and designate the number of coordinates assigned to each type of locus.

Through misuse of language, the term "canning" will designate either this set of maps, or the algorithm used to build it.

The objective is twofold : firstly, we want to find a way to measure the efficiency of a canning on a given medium. Secondly, we want to create a canning that is efficient on most -if not all- media.

B Preliminary Work

In this section, I briefly introduce some work I did last year during another internship with Gruau to generate AHI media. While it isn't the main focus of this year's interniship, it is what made it possible in its current form.

B.1 Delaunay Triangulation

A Delaunay triangulation is a particular type of triangulation where a triplet of vertices form a face iif no other vertex can be found inside the circumcircle of the triplet. We relax this definition slightly by allowing the hull of the graph not to be convex. It presents interested properties for our purpose as the edges of the triangulation tend to remain "localized" (i.e. vertices that are connected tend to be spatially close).

I used the triangulation algorithm described in [6]. However, there is a bug in my implementation where vertically-aligned are often handled incorrectly, which I couldn't fix before the end of my internship last year. Since this problem wasn't particularly consequential this year either, I didn't take the time to fix it this year either.

Additionally, the FPO algorithm (see B.2) performs a lot of removals and insertions to the set of vertices. Currently, each time this happens, the entire set is retriangulated. This is a vastly inefficient approach to the problem. If I had the time, I would've liked to implement the removal and addition algorithms described in [6, 1].

B.2 Farthest Point Optimization

The Farthest Point Optimization algorithm[8] allows the construction of the irregular yet homogeneous sets of points that we need for blob computing. It yields better results than more common algorithm like Poisson-Disk sampling or Lloyd algorithm, and fits naturally into the project as it relies heavily on the Delaunay triangulation.

The implementation of FPO in my case wasn't straightforward. This is because the article applied the algorithm to sets of points in the unit torus, while the media use bounded subsets of the euclidean plane instead. In particular, I had to adapt the algorithm to work around the existence of borders with no predetermined shape, and that can contain fixed points that cannot be moved even though they can still influence the placement of other points.

1 Borders and Media

1.1 Choice of Border

1.2 Different Types of Medium

2 Canning and Evaluation

Before we begin our search for total cannings, let us define the notion of partial canning. A partial canning of a medium M is simply a subset of a total canning of M (or the algorithm used to build it). In particular, we focus on vertex cannings, i.e. partial cannings which only contain the map from the vertices of M to their coordinates. From there, we will exhibit a generic method for build

total cannings using vertex canning as a base. Finally, we will show how we can evaluate the efficiency of different cannings so we can compare them.

2.1 Vertex Cannings

To align with the way crystaline media are hendled, we set $a_1 = 2$, i.e. we want to assign 2D integer coordinates to each vertex of the medium. The problem now becomes equivalent to placing the vertices in a 2D grid. The vertex canning is valid if all the vertices have been placed in the grid and if each cell of the grid contains at most one vertex. The coordinates of each vertex then becomes the 2D index of its cell. Based on this idea, I have developped two diffrent vertex cannings with opposite philosophies :

1. **TopDistanceXStorted** vertex canning
   This vertex canning was developed to rely as much as possible on the graph structure of the medium, making as little use as possible of its spatial aspect. It was developed quite early in the project, before soft-bordered media appeared, so it only works on hard-bordered media. It works in the following way :
   - First, compute the minimum graph-distance of each vertex to any vertex on the topside of the border. All vertices at distance Y from the top are placed in line Y of the grid.
   - Second, place the leftmost vertex of each line in the first cell of its line, the second leftmost vertex of each line in the second cell of its line, repeating until all vertices are placed in the grid.

2. **RoundedCoordinates** vertex canning
   This vertex canning was developed to rely exclusively on the spatial embedding of the medium, completing ignoring its graph structure. It works in the following way :
   - First set a counter to 1.
   - Multiply the spatial coordinates of each vertex by the counter and round the result. Use the two integers obtained as a 2D index to put each vertex in the grid.
   - If two vertices share the same cell, increment the counter by one and go back to the previous step. Otherwise, we have a vertex canning.

Regarding the TopDistanceXStorted algorithm, I first tried sorting the lines using the distance to the leftside border, however it resulted in invalid vertex cannings where two vertices would share the same cell.

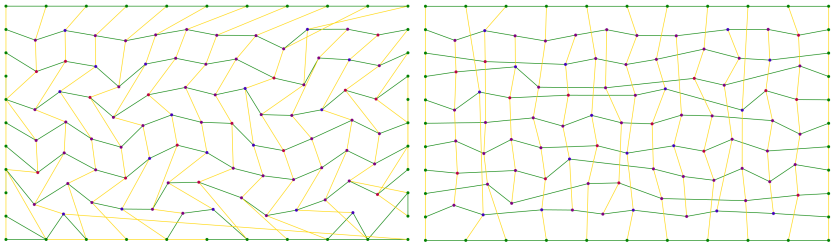


Fig. 4: TopDistanceXSorted (left) next to RoundedCoordinates (right) applied to the same medium.
Two vertices are joint by a green line if they are consecutive on a line of the grid, and a yellow line if they are consecutive in a column.

Intuitively from figure 4, we can infer that RoundedCoordinates is better as it creates a more "grid-like" canning, whereas the columns in TopDistanceXSorted seem to "drift" to the left. We will confirm this intuition later.

2.2 From Vertex to Total Cannings

2.3 Evaluation

3 GUI

A large portion of the TER was focused on the development of a software equipped with a GUI that allows the user to

1. Randomly generate new media, save them, and load them.
2. Visualize different aspects of the media.
3. Compute and visualize different cannings.

If developing a GUI for this application ended up being time-consuming, it was incredibly helpful all around while debugging everything else. Overall it was probably a net positive on development time alone, and that is still neglecting the advantage of being able to see what i am doing.

Below are some pictures of the kind of visualization the GUI provides.

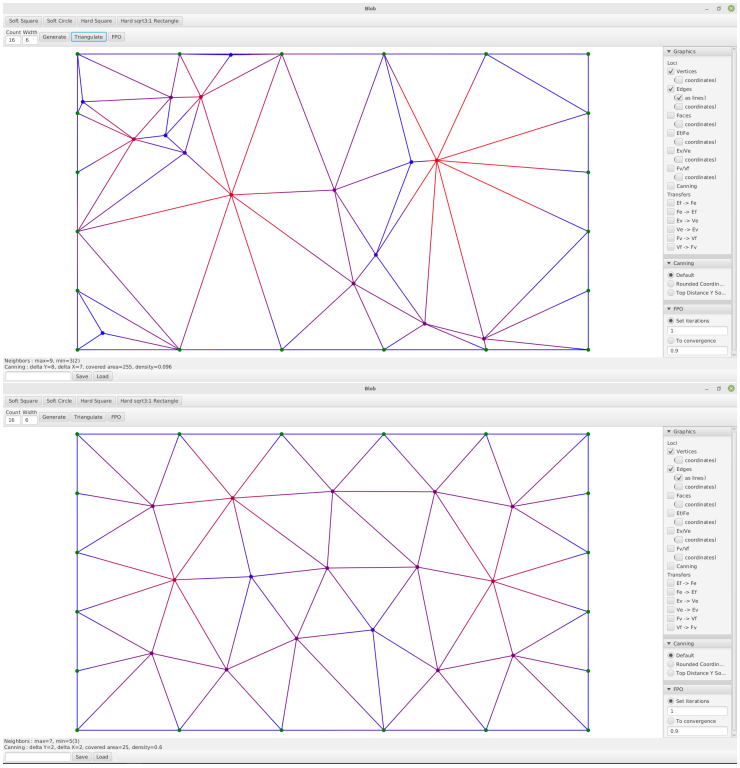


Fig. 5: "Canonical" view of a medium before (top) and after (bottom) optimization with FPO.

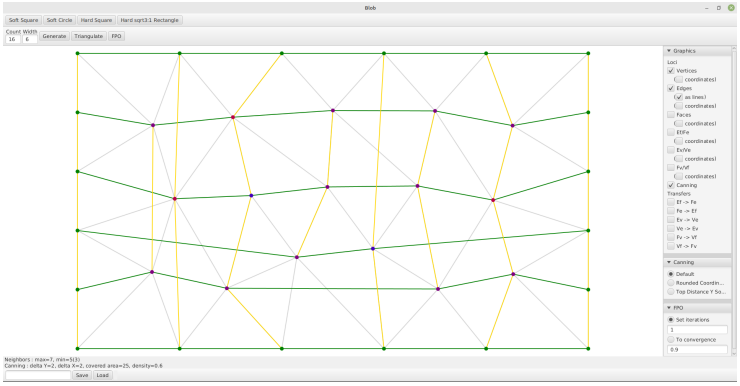


Fig. 6: View of a partial canning (see 2.1) of the vertices of the medium in figure 5

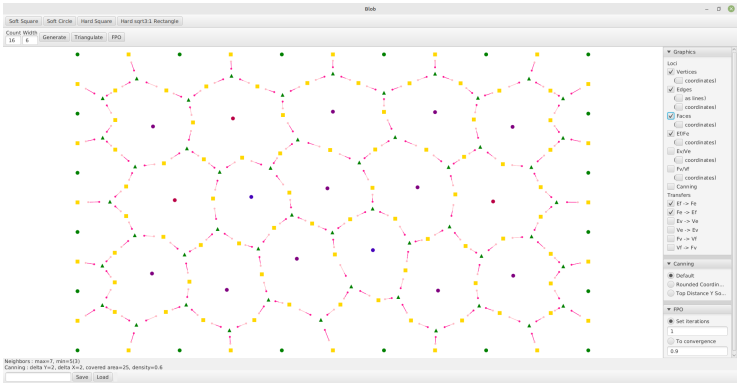


Fig. 7: View of the Ef-Fe pairs in the medium of figure 5.

4 Conclusion

References

[1] Olivier Devillers. "On Deletion in Delaunay Triangulations". In: *International Journal of Computational Geometry and Applications* 12 (2002), pp. 193–205. DOI: 10.1142/S0218195902000815. URL: https://inria.hal.science/inria-00167201.

[2] Frédéric Gruau. "A parallel data-structure for modular programming of triangulated computing media." In: *Natural Computing* 22.4 (2023), pp. 753–766. ISSN: 1572-9796. DOI: 10.1007/s11047-022-09906-1. URL: https://doi.org/10.1007/s11047-022-09906-1.

[3] Frédéric Gruau. *platform-CA2*. URL: https://github.com/fredgruau/platform-CA2.

[4] Frédéric Gruau et al. "BLOB computing". In: *Proceedings of the 1st Conference on Computing Frontiers*. CF '04. Ischia, Italy: Association for Computing Machinery, 2004, 125–139. ISBN: 1581137419. DOI: 10.1145/977091.977111. URL: https://doi.org/10.1145/977091.977111.

[5] Frédéric Gruau, Christine Eisenbeis, and Luidnel Maignan. "The foundation of self-developing blob machines for spatial computing". In: *Physica D: Nonlinear Phenomena* 237.9 (2008). Novel Computing Paradigms: Quo Vadis?, pp. 1282–1301. ISSN: 0167-2789. DOI: https://doi.org/10.1016/j.physd.2008.03.046. URL: https://www.sciencedirect.com/science/article/pii/S0167278908001255.

[6] Leonidas Guibas and Jorge Stolfi. "Primitives for the manipulation of general subdivisions and the computation of Voronoi". In: *ACM Trans. Graph.* 4.2 (Apr. 1985), 74–123. ISSN: 0730-0301. DOI: 10.1145/282918.282923. URL: https://doi.org/10.1145/282918.282923.

[7] Iñaki Navarro and Fernando Matía. "An Introduction to Swarm Robotics". In: *International Scholarly Research Notices* 2013.1 (2013), p. 608164. DOI: https://doi.org/10.5402/2013/608164. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.5402/2013/608164. URL: https://onlinelibrary.wiley.com/doi/abs/10.5402/2013/608164.

[8] Thomas Schlömer, Daniel Heck, and Oliver Deussen. "Farthest-point optimized point sets with maximized minimum distance". In: *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. HPG '11. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2011, 135–142. ISBN: 9781450308960. DOI: 10.1145/2018323.2018345. URL: https://doi.org/10.1145/2018323.2018345.