INVERSE SPACE FILLING CURVE PARTITIONING APPLIED TO WIDE AREA GRAPHS

Cyprien Gottstein[1], Philippe Raipin Parvedy[1], Michel Hurfin[2], Thomas Hassan[1] and Thierry Coupaye[1]

[1]TGI-OLS-DIESE-LCP-DDSD, Orange Labs, Cesson-Sevigné, France
[2]Univ Rennes, INRIA, CNRS, IRISA, 35000RENNES, France

ABSTRACT

The most recent developments in graph partitioning research often consider scale-free graphs. Instead we focus on partitioning geometric graphs using a less usual strategy: Inverse Space-filling Partitioning (ISP). ISP relies on a space filling curve to partition a graph and was previously applied to graphs essentially generated from Meshes. We extend ISP to apply it to a new context where the targets are now Wide Area Graphs. We provide an extended comparison with two state-of-the-art graph partitioning streaming strategies, namely LDG and FENNEL. We also propose customized metrics to better understand and identify the use cases for which the ISP partitioning solution is best suited. Experimentations show that in favourable contexts, edge-cuts can be drastically reduced, going from more 34% using FENNEL to less than 1% using ISP.

KEYWORDS

Graph, Partitioning, Graph partitioning, Geometric partitioning, Spatial, Geography, Geometric, Space Filling Curve, SFC, ISP.

1. INTRODUCTION

Nowadays, graphs are extensively used in the IT industry. As one of the most expressive and widely used data structures, graphs have reached a scale never seen before. Along with the explosion of social networks, new marketing opportunities arose as graph knowledge about people became a very profitable resource. Many features of important services such as Google, Facebook or Amazon rely on graph analysis to provide their users with the content or experience that best satisfies their needs. As performing such analyses requires heavy computation, it is essential for those graphs to be widely distributed across a large set of machines, i.e. partitioned. As K. Andreev and H.Räcke shown, graph partitioning is a NP-Complete problem [1].

Today, a wealth of graph partitioning solutions exists with a large focus on the OnLine Analytic Processing (**OLAP**) context. Choosing the appropriate graph partitioning solution for a given graph has become a difficult task due to the wide variety of solutions. Most recent researches on graph partitioning [2] heavily focuses on scale-free graphs. Scale-free graphs, including social graphs, are graphs whose degree distribution follows a power law. **However, data spatiality tend to be neglected in these works.** This is justified because very few, if not none, scale-free graphs hold precise spatial information onto their nodes. Consequently, geometric graphs, graphs

David C. Wyld et al. (Eds): NLP, JSE, MLTEC, DMS, NeTIOT, ITCS, SIP, CST, ARIA - 2020
pp. 223-241, 2020. CS & IT - CSCP 2020 DOI: 10.5121/csit.2020.101417

whose nodes and edges are associated to spatial elements placed in a plane, and geometric based partitioning strategies have received little interest in the past decade.

Nonetheless, we believe geometric graphs and geometric partitioning will quickly regrow scientific interests with the uprising of the Internet of Things and the appeal of graph technology. Because of new services such as Microsoft Azure Digital Twins and new digital uses developed by companies working on smart-cities or smart-industry, we expect a new family of graphs will rise: geometric graphs based on real-world infrastructures with scales similar to that of social graphs. As these graphs do not exists yet, it is difficult to define them with precise characteristics. Being precise is difficult mainly because these graphs will probably be the resulting aggregate of different types of infrastructures: roads, power or water supply chain, buildings and equipment, each transformed into graphs with their associated properties. In the rest of the paper, we will refer to this family of graphs as Wide Area Graphs (**WAG**).

Some major characteristics will be common to all WAGs. WAG is a sub family of geometric graphs. Recall that geometric graphs are graphs for which vertices or edges are placed on a plane. Therefore, each node of a WAG has an associated location. Also, for a graph to be considered as a WAG it must be composed of millions of nodes and edges and cover an area as large as a country's surface or even wider. At last, a WAG should be infrastructure related e.g. mixing power grid networks, road networks, buildings, equipment's or even supply chains. Furthermore, it should have layers associating local objects interacting with each other and connected to a larger network. We do not include anything regarding connectivity and density properties into this high level characterisation of a WAG because these aspects may vary significantly from one WAG to another. However we envision a WAG as a highly polarized version of graphs issued from Finite Element Meshes (**FEM**).

FEM are used to simulate real world experiences as wind or water movement (see Figure 1). FEM graphs are planar graphs with varying density gradually peaking usually within a single continuous area. A planar graph induces a low connectivity ratio, compared to scale-free graphs, and a low edge Euclidean distance as edges only exists between nodes that are close to each other on the plane. Connectivity will not be limited to being planar in a WAG; it may contain nodes with high connectivity leading to crossing edges with higher Euclidean distance. In WAGs, we expect both smooth and harsh density gradation with several peaks scattered randomly across the plane, this would correspond to not-so-well shaped meshes [3]. At last, a WAG may have several nodes located exactly at the same place on the multidimensional plan.

This work addresses the problem of partitioning a WAG. Since WAGs are heavily related to geometry, we believe that geometric partitioning which has already been proposed and applied on graphs (See Section 2) is an appropriate approach for this new challenge. While those solutions are known to be computationally expensive, Pilkington and Baden came up with Inverse Space-Filling curve Partitioning (**ISP**) [4] a much cheaper algorithm, which produces partitions based on a Space-Filling Curve. A space-filling curve is a mapping from a multidimensional plane to a single dimension line. The curve is used with a depth, or zoom, to define its granularity. A space-filling curve is based on recursion and is self-similar through the levels. Those curves are known and used because of their ability to preserve spatial proximity into the single dimension space. ISP is at the core of our work as we propose an extension of the original solution to cope with the new context of WAGs. Our main goal is to determine in which cases such a solution is relevant.

This work has been done in the context of Orange research project's Thing' In, a platform dedicated to map connected and unconnected objects of the real world into a single graph representing their interactions. Thus, Thing' In is a spatial graph highly bound to geography and it could be seen as a prototype of a WAG. Furthermore, Thing' In is a use-case agnostic platform.

Each of its use case will need to store, retrieve and query data in a customised manner and will most likely require optimizations to perform those actions in an acceptable speed. As our objective is to study the WAG family and not only the single specific Thing' In graph, we will also consider synthetic graphs during our experiments.

This paper brings several contributions. First, we provide an update of ISP to show how it behaves when applied to WAGs which are in essence a harder version of FEM graphs. Then, we experimentally demonstrate that edge spatial distance is the decisive factor regarding the performance of ISP. We propose a fine analysis metric and decision matrix that indicates whether to choose ISP or not as a partitioning strategy. At last, we evaluate its benefits and drawbacks through a set of metrics, mainly comparing our results against FENNEL algorithm [5].

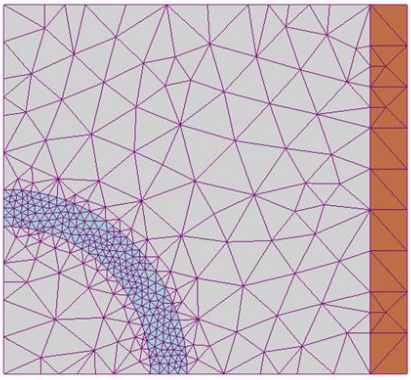

Figure 1. Illustration of a graph issued from a mesh.

2. RELATED WORK

Graph partitioning is a humongous field of research. The graph partitioning, or the balanced k-partitioning, problem is described as such: given a graph G and a number k corresponding to the number of partitions, we seek to cut G into k balanced pieces while minimizing the number of edges cut. **Edge-cut** is the default performance measure to any graph partitioning strategy; it represents the ratio of edges for which both ends are stored on separate partitions. It is important to have a minimum amount of edge-cuts as it dramatically increases edge traversal time: a basic operation for a graph algorithm. Obviously, **Load-Balancing** is also mandatory; a fairly distributed load is needed to provide horizontal scalability.

In this section, we depict some of the state-of-the-art strategies in the field of graph partitioning. We do not aim to cover the whole field of research, but instead we will limit ourselves to the most relevant research related to our target, e.g. partitioning strategies able to partition a WAG. For example, a strategy such as METIS [6], the most popular example of multi-level strategy, is considered to be extremely costly if it is intended to run on WAG scale. While METIS can be applied on any type of graphs, some strategies are specialized for geometric graphs.

A well-known approach for geometric partitioning is Recursive Coordinate Bisection [7] (RCB).The RCB method sorts the graph nodes based on the most expanded dimensional axis and then recursively bisects that axis to distribute the load evenly. This approach then evolved with random sphere [8] and partitioning based on space filling curve: Inverse Space-Filling Partitioning [4] (ISP). Random sphere strategy performs geometric sampling over the mesh to find an efficient circle center point to bisect the mesh with an even load; the algorithm is then

executed recursively to reach the given number of partitions. ISP also relies on geometry but with a much cheaper approach based on SFC. Nodes which are close on the multi-dimensional space are mapped close on the single dimension, then they are grouped together and form a partition. We describe how ISP works in Section 3.

Both RCB and Random sphere are geometric strategies which could have been used to partition WAG. But given that RCB is known to yield partitioning of poor quality [9] and random sphere works best on "well-shaped" graphs, which is not what should be expected from WAGs, we will not consider those strategies for our comparisons.

More recent works aim to partition graphs based on geometry [10]–[12]. Akdogan proposes to partition geometric graphs using Voronoi tessellation whereas Volley places data on a 3D sphere representing the earth and groups data spatially close to reduce edge-cuts. These solutions work relatively well but require a lot of computations. Delling et al. strategy is based on the presence of natural cuts within a graph that minimize edge-cuts. Their solution detects such cuts to build partitions. Unfortunately the natural cut hypothesis seems unlikely to hold within a WAG.

Today, graph partitioning is considered, at least from the analytic perspective, to be divided into two categories: offline which requires storing the whole graph in memory to perform global partitioning decision and online for cheaper strategies based on local decision making. The latter is in general based on streaming, well-suited for the heavy scaling that characterizes scale-free graphs or WAGs. Single pass streaming applied to balanced graph partitioning consists of streaming every node of the graph only once and to build the partitions on the fly. For each node streamed, scores are computed assuming the node can be associated with each partition; the node is then attached to the partition with the highest score and may never leave it. Once the streaming is done, so is the partitioning as a whole. When the graph increases, this process can be resumed without the need to restart from scratch.

To the extent of our knowledge, the first study of this strategy has been proposed by Kliot and Stanton [13] in which they analysed multiple score heuristics to handle the placement of nodes inthe partitions. They determined Linear Deterministic Greedy (LDG) as the best heuristic for graph streaming partitioning. Their work have been improved with the upcoming of FENNEL [5] as it offers a generalization framework able to perform just as LDG but with extended possibilities. Even though streaming algorithms are relatively simple in their application they still offer a lot of possibilities by adjusting the formulas used to measure the load of each partition and the possible imbalance factor.

3. ISP EXTENSION

In this section we discuss how we extend ISP previous work. As said before, geometric partitioning within a multi-dimensional space is very expensive. Alternatively, ISP [4] proposes to map the multi-dimensional space on a single dimension thanks to a space filling curve leading to a simplified 1D partitioning of the curve. The curve is divided into cells and their order is defined by the curve algorithm. Each cell is a point on the curve mapped to a bounding box on the multi-dimensional space. A cell is to be assigned to a single partition and has an associated weight defined by the number of nodes it contains. Thus, the weight of a partition is equal to the sum of the weights of its cells.

When ISP was first proposed, graph partitioning streaming strategies have not yet been identified as a particular class of strategies. Therefore, ISP is classified as a geometric partitioning strategy. We believe it is fully streaming compliant and should therefore be considered as such, the only

requirement is to stream the nodes in the order defined by the cells of the curve used to perform the partitioning.

In our implementation of ISP, based on the total number of nodes (denoted n) and the number of partitions (denoted k), we assign an expected capacity to each partition. Usually the capacity is the same for all the partitions and is more or less equal to n/k. This is not a mandatory requirement. We define the load of a partition as its weight divided by its capacity. During the partitioning process, a partition may be assigned fewer or more nodes than expected. In any case, the quality of the load balancing can and should be evaluated according to the load ratio of the partitions.

A partition corresponds to a set of contiguous cells of the SFC. Partitions are created sequentially. To create the first partition, we start from the first cell of the SFC and we consume the cells in the order defined by the curve. We greedily assemble cells until the partition is fully loaded (i.e., its load is no longer lower than its expected capacity). Then we move on to the next partition and the next cell. As a result, the curve is segmented into k intervals and each interval is corresponding to a partition. Partitions are continuous segment over the curve which corresponds to continuous area over the multi-dimensional space thanks to the space filling curve properties.

We have respected the original design choice of ISP: even if measuring the load of a partition has seen some evolution like using the number of edges [13] or other custom metric[5], the load metric of a partition is still based on the number of nodes it actually stores. We view the number of nodes as the most suited metric because we are targeting the OnLine Transaction Processing (**OLTP**) context. We also kept the Hilbert curve[14] as it has been proved to be the optimal space filling curve for spatial proximity preservation by Knoll and Weis [15]. At this point, whenever we mention a Space Filling Curve (SFC), we will implicitly assume it is the Hilbert SFC.

Our main contribution is about using ISP [4] to partition WAG which are harder to handle than graphs issued from meshes. Besides the properties we described for WAG, we also mentioned that nodes may overlap in a WAG.

SFCs are frequently used with adaptive refinement, a method which allows the SFC to zoom in and out depending on current needs e.g. local density for graph partitioning. With graphs where nodes never overlap e.g. graph issued from meshes, adaptive refinement guarantees that each cell will store at most one node. In our context, nodes may end up at the exact same position e.g. a building with multiple-storeys stored in a 2D plane in which case adaptive refinement will not be able to split the cell's weight. This is an important issue as instead of assembling cells which have a weight of exactly one node, we may have to cope with cells with high weight which may induce load imbalance.

Figure 2 illustrates this risk. If the cells have different weights (1, 2, 3 or 5 in the example), the consumption of the cells in the order defined by the SFC sometimes leads to achieve exact load balance (at the bottom of the figure) or, on the contrary, leads to exceed this threshold by adding a last cell with too many nodes (at the top of the figure). As such, the existence of super cells (SFC cells with extreme load) may or may not be problematic.

4. EDGE DISTANCE AND DENSITY, ISP DECIDING FACTOR

ISP can potentially be applied to any kind of graph as long as each of its nodes has position on a Euclidean space. We argue in this section that there are precise conditions for ISP to perform well. The decisive factor is the distance covered on average by an edge in proportion to the

surface covered on average by a partition, we defined it as: **EDTPS** (Edge Distance To Partition Square Size).

The idea is the following: the more we increase the number of partitions, the more the space covered by each partition decreases as the space is finite and the likelier we are to produce edge-cuts using a geometric partitioning strategy such as ISP. The higher the EDTPS, the closer is an edge from being as large as the average square surface of a partition. Geometric partitioning with low EDTPS should therefore behave well.

Let G(V,E) be a directed geometric graph placed on a 2D plan. The graph G has a surface englobing polygon P composed of points $((x_i, y_j))$ so that P= $((x_0, y_0), (x_1, y_1), \ldots (x_{n-1}, y_{n-1}))$. The points in P are all sorted in order. Let D(e) be the Euclidean distance between the source and destination for each edge e of E. We first define the function used to convert a polygon to a surface area.

$$Area(P) = \frac{1}{2}\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1} y_i) \qquad \text{Where } x_n = x_0 \text{ and } y_n = y_0$$

Equation 1. Area formula

Then, the EDTPS formula is as follows:

$$EDTPS(G,k) = \frac{\sum_{e \in E} D(e)}{|E|} * \frac{1}{\sqrt{\frac{Area(P)}{k}}}$$

Equation 2. EDTPS Formula

A graph partitioning solution is evaluated over its ability to preserve edge from being cut and how well it maintains load balance. ISP, as mentioned in Section 3, may have trouble handling graph with high local density peak, especially without adaptive refinement. To measure those potential troubles we introduce our second metrics: **CDTPC** (Cell Density To Partition Capacity).

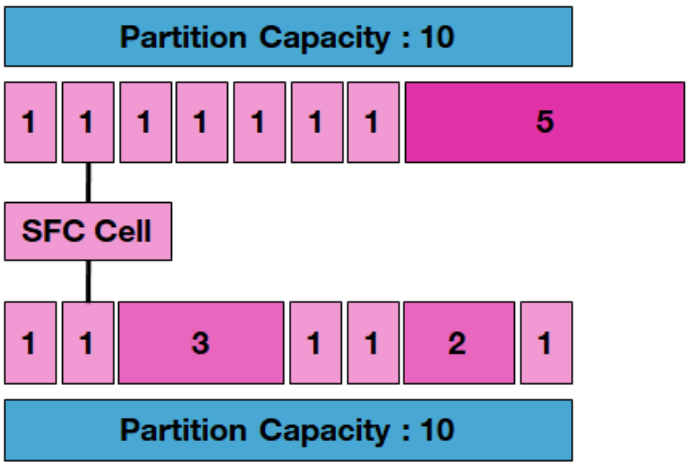

Figure 2. ISP Imbalance scenario

As shown in the second example of Figure 2, a hyper dense SFC cell may not automatically trigger imbalance, it needs to be consumed within a specific interval. This interval is the ratio between the weight of the cell and the space left in the partition. The heavier the cell, the larger the interval is to build imbalance partition. CDTPC is designed to evaluate such interval and detect in advance potential load balance troubles. The higher it is, the larger the interval will be. While a high CDTPC does not necessarily imply an imbalance problem, a small ratio most likely guarantees a very sane balance.

In order to define CDTPC we need first to compute maximum cell density which we will refer as **MCD**. To do so we map every nodes position to the SFC and retain the SFC cell which includes the most nodes. The CDTPC formula is then the following:

$$CDTPC(G,k) = \frac{MCD}{\frac{|N|}{k}}$$

Equation 3. CDTPC Formula

Note that in both formulas we need k as EDTPS and CDTPC depends on the number of partitions. It is normal since the performances of the ISP partitioning algorithm vary depending on the number of partitions.

5. GEOMETRIC GRAPH GENERATOR

Faced with the lack of appropriate datasets that may be close to the future WAGs (a geometric graph, covering a large area and with a high number of nodes and edges and due to the difficulty to crawl an equivalent in an open platform, we resorted to use a graph generator. There already exists geometric graph generators [16], [17]. Yet, none can provide a WAG like graph (with the desired density or connectivity characteristics). They also fail to enable multiple edge distances and they do not impose a minimum distance to build an edge between two nodes. To be able to produce synthetic WAGs with varying EDTPS and CDTPC, we need a generator that accepts parameters specifying density and edge Euclidean distance characteristics as an input. For all these reasons we have designed our own geometric graph generator.

Our generator is not suited to produce scale-free graph, it does not include the preferential attachment model [18]. Its goal is to produce graph similar to infrastructure or IoT graphs. The generator is written in Java and integrates its own R-Tree index [19]. The generation process is performed in three steps: process population density through a SFC then create and assign the nodes on the plane and finally build the edges.

It requires the following input parameters: the number of nodes and edges, a surface plane, indications on how the density of nodes may vary in the plane and on the length of the edges. The surface plane needs to be a square composed of GeoJSON coordinates. Density is configured through a set of categories, each holding a surface unit and density factor. The surface unit is used to define how much of the surface of the plane is covered with this category. The density factor defines relatively how dense is a category compared to the others. At last, edge distances are also defined through categories, each with a ratio, and a minimum and maximum possible distance. Each node on the plane is assigned the same number of starting edges, the edge ratio is used to determine how many of those edges belong to which distance categories. At the end, nodes will have the same number of outgoing edges but a different number of incoming edges.

To generate the density distribution of the node population, we first slice the plane into a grid and map each cell to an SFC. The number of columns and rows of the grid is determined through the definition of the SFC level of granularity given by the user. The surface units of each density categories are converted into relative ratio to determine how much of the SFC (and consequently of the underlying plane) will be assigned to a given category. The density factor weighted by the surface units enables us to process how many of the total nodes will fit in a given category and determine its local cell density (the number of nodes will be populated inside a SFC cell mapped to the plane).

In order to produce a representative density of the real world, the whole surface of a given density is segmented into several blocks spread across the SFC. We proceed from highest to lowest density. For each new segment, we start at a free random point on the curve and apply a random Pareto function to determine its length. The denser a category is, the harder it will be to build long continuous segments. At the end, the whole SFC is corresponding to a sequence of multiple segments assigned to a density category. It remains only to populate those segments with nodes and proceed to the edge binding. We provide a visual example at Figure 3.

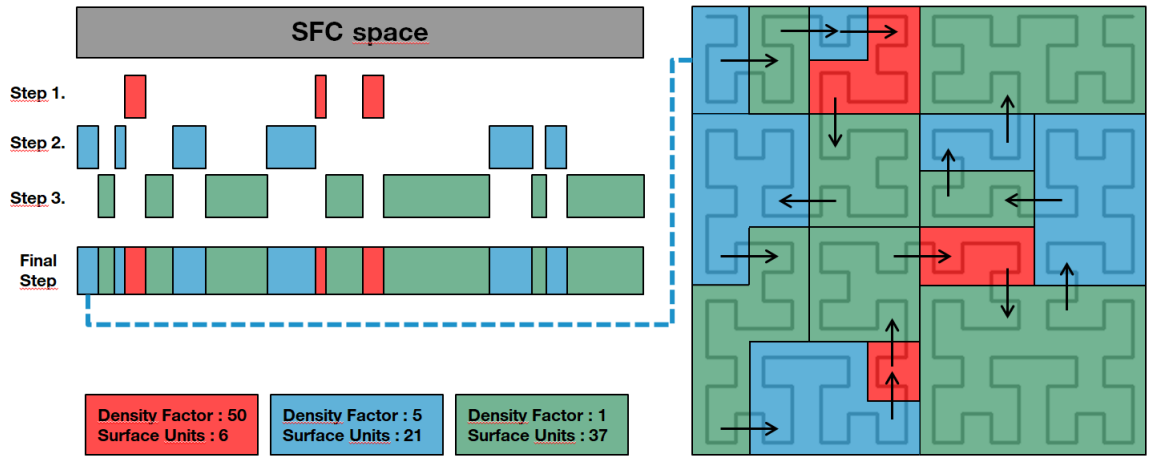

Figure 3. Density distribution of the node population

Populating the nodes only consists in taking a cell from the SFC, checking its density, and producing the corresponding number of nodes. Each node is assigned to GPS coordinates of a random point located within the cell bounding box. Edge binding requires more work. We rely on the embedded R-Tree to query the nodes using geometric squares. For each edge distance category, we form squares with a diameter twice the size of the maximum distance allowed for this edge distance category, thus large enough to contain the longest edges, and slice the whole plane in a grid manner. It enables us to perform edge binding within a limited part of the graph while still satisfying the distance requirements. To prevent any form of advantage which would lead to anomalies, randomized selection is performed each time a target node is needed to build an edge. Also, we perform several passes with offsets into the geometric square query to make sure no connected component is formed because of our approach.

6. EXPERIMENTAL SETUP

6.1. Setup

We evaluated our solution on a high processing computing machine with 92 GB of RAM and 48 CPU cores. The spatial graph generator and the metric processor were written in Java 8, everything related to visualization was done using Python 3.6. We choose to rely on the GraphX library of Spark (https://spark.apache.org/) to perform parts of our metrics; Spark is an expensive overhead in both development and computation but brings high scaling potential. Because Spark is built to work over Hadoop (See https://hadoop.apache.org/) and HDFS, we deployed a standalone HDFS service to handle the data read and produced by the metrics jobs. To evaluate the different partitioning solutions, we measure the amount of edge-cuts produced and the imbalance of the partitions. Regarding load imbalance, we use the **normalized maximum load** metric [5] (defined Section 4.1). We also include the custom heuristics EDTPS and CDTPC defined in section 4.

6.2. Dataset

We used two kinds of graphs in our experimentation: the graph of Thing' In whose characteristics are presented in the following Section 6.2.1 and synthetic WAGs produced using the graph generator already described in Section 5. We consider different WAGs with distinct characteristics as it is impossible to reduce WAGs to a single set of properties. Precisely, we aimed to first produce WAGs close to Thing'In properties and then gradually deviate to obtain a wide range of results for the application of ISP. We also wanted to find an inflexion point where ISP becomes better or worse than concurrent strategies.

6.2.1.Thing'In Graph

The Thing'In graph is an aggregate of multiple open data sources holding representation of physical objects from the real world. It is a generic graph; it holds potentially any kind of data as long as there is the appropriate definition from the semantic web to describe its content. As it is a research project not every node is associated to a GPS coordinates, we removed those to only keep a fully compliant geometric graph. As nodes were excluded, the remaining data, representing about half of the graph (~ 27.2 millions of nodes and ~27.5 millions of edges), are mainly composed by transportation network like trains, buses, roads and subways.

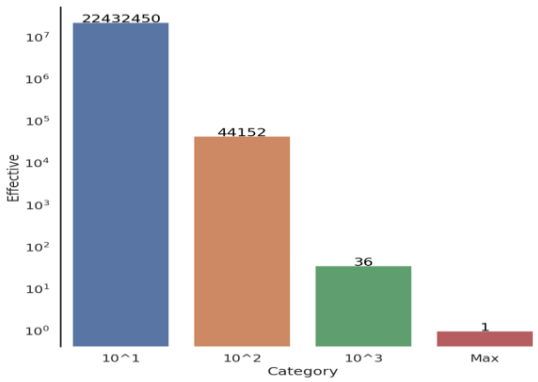

Figure 4. Thing' In Degree Distribution, blue column is for nodes with degree less than 10.

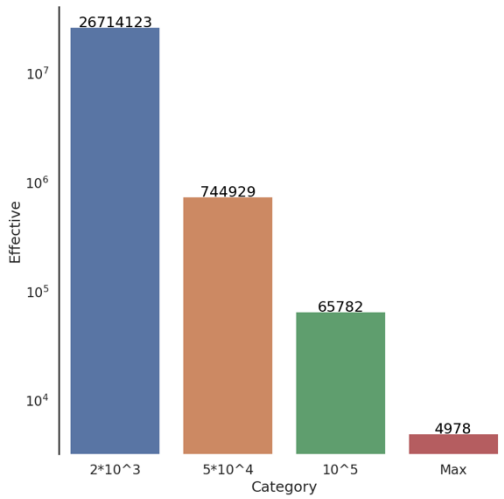

Figure 5. Thing'In edge euclidean distance distribution

As shown in Figure 3 we see that indeed, Thing'In is a spatial graph. There is no super nodes (nodes with hyper connectivity) and the most connected node represent the city of Rennes (It is the city where the Thing'In main developer team is located. They performed many experimentations related to this city and, consequently most edges of the graph are related to this city). The cumulative degree distribution function shows that Thing'In does not follow a power-law distribution. In average, the Euclidean distance covered by the edges (see Figure 4) is very short: 97,03% of the edges are shorter than 2km.

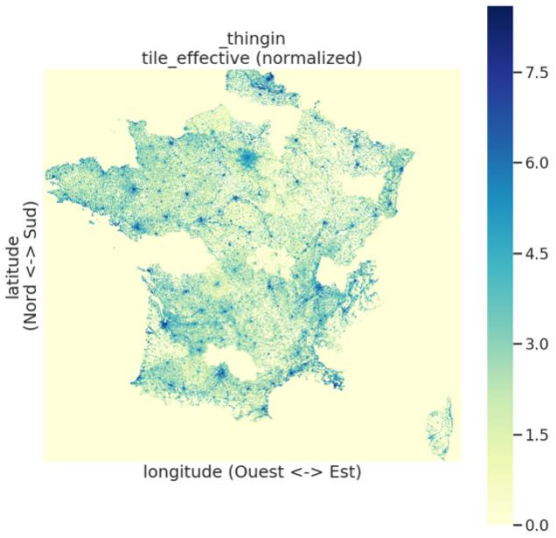

Figure 6. Thing'In node population distribution

Nodes are distributed in a uniform manner across the territory of France (See Figure 6). There are still some regions where no data has been injected, thus with a density of 0, whereas all large

cities show progressive density towards the town center. Based on its properties, we consider Thing'In as a WAG. Consequently, it represents an interesting graph to be partitioned with ISP.

6.2.2.Synthetic datasets

We built the synthetic datasets under two assumptions. First, we assume that population density in a WAG should behave the same as human density. Second, we think ISP performances (and in fact any geometric partitioning strategy), applied to a geometric graph, should be directly related to the EDTPS and CDTPC metrics (see Section 4). Regarding density, there will be deserts, a zone with low to no density, and metropolis, a zone with extremely high density.

Table 1. Density configuration

Category	**0**	**0.5**	**5.0**	**20.0**	**50.0**	**500.0**
HD	1000	1000	100	0	10	1
MD	1000	1000	100	0	10	0
LD	1000	1000	100	10	0	0

The density settings can be found in Table 1; each column represents a density category, the header represents the density units and the values are the surface units. The HD configuration is based on a roughly simplified Zipf's (https://en.wikipedia.org/wiki/Zipf\%27s_law) distribution of human population across the Earth. It is composed by 47.37% empty surface (ocean and desert), 47.37% scarce density surface (rural area), 4.73% low density (urban area), 0.47% of medium to high density (large city) and 0.05% of extreme density (metropolis). We declined it with medium (MD) and low (LD) densities to obtain CDTPC and load balance variation. If a row as 0 surface units for a given column, it means it does not use this density category.

The same density and volumetry configuration will behave differently with a larger or tinier plane. A large plane will smooth out the density whereas it will spike even more with a tiny one. We consider graphs with varying density and plane size to see how it impacts the CDTPC metric and ISP's load balance. In particular, hyper dense hot spot may be difficult to manage.

Table 2. Edge distance (km) configuration, each interval respect the following pattern [min_distance : max_distance](%total_edges)

Category	**interval**
TINY-	[0:2[(97%), [2:30](3%)
TINY	[0:30](100%)
SHORT	[0:2[(30%), [2:50[(50%), [50:100](20%)
MEDIUM	[0:20[(25%), [20:50[(25%), [50:100](50%)
MEDIUM+	[0:10[(40%), [10:100[(40%), [100:200](20%)
LONG	[0:160](100%)
LONG+	[100:250](100%)
HUGE	[300:500](100%)

Table 3. Synthetic plane configuration

Type	**Latitude**	**longitude**
FR (Country)	[41.15:50.33]	[-5:9.85]
EU (Continent)	[41.15:65.33]	[-5:40.85]

Concerning ISP performances correlated to EDTPS and CDTPC, various configurations are tested to obtain a wide range of EDTPS values and evaluate its accuracy. We established 8 edge distance configurations described in Table 2. Note that the first distance category together with the medium density specifically aims at producing graphs similar to Thing'In. We limited ourselves to two plane sizes (See Table 3). We seek with this variation to prove that absolute plane size has no impact on ISP performance unlike EDTPS which is a ratio relative to plane size. By default the density is set to HD, the most penalizing load-balance wise. TINY- has been generated only with MD density whereas LONG has been tested with all 3 densities.

Table 4. Average edge distance and EDTPS of synthetics WAGs

Category	**AVG_Dist**	**EDTPS**	
		FR	**EU**
TINY-_MD	1041.49	0.00192	0.00076
TINY	13521.19	0.02123	0.00967
SHORT	25860.80	0.04773	0.01765
MEDIUM	48902.51	0.09026	0.03395
MEDIUM+	55766.50	0.10293	0.03562
LONG	84392.80	0.15608	0.0562
LONG_MD	86869.69	0.16034	0.05508
LONG_LD	89290.85	0.14142	0.06008
LONG+	176261.23	0.32547	0.12684
HUGE	391988.63	0.7235	0.2865

We have performed some early metrics on the generated graphs to check our requirements were respected. We measured edge distance with EDTPS (See Table 4) and cumulative connectivity distribution. As expected, we could not generate graph with scale-free properties: the connectivity of the generated graphs remained very low. Moreover we could not exactly reproduce the properties of Thing'In with our generator (See the line TINY-), the EDTPS should be even lower. On the other hand we managed to approach closely its value in absolute terms and obtained a great set of EDTPS values ranging from 0.1% to 72.35% with 4 partitions. We also included graphs with the same configuration on plane, edge distance and density but with varying volumetry to ensure volumetry is not a performance factor of ISP. Due to space limitations, these additional experiments are not reported here.

7. RESULTS

We seek in this experiment to compare existing state of the art streaming graph partitioning method to ISP. We applied ISP, FENNEL and LDG on each dataset with the same streaming approach. The only difference aside the decision algorithm is the streaming order. In ISP nodes were streamed ordered by the cells of the SFC whereas nodes were streamed randomly for FENNEL and LDG because, given our scale, DFS (Depth First Search) and BFS (Breadth First Search) were too expensive to apply. We first look at how strategies behave in terms of edge-cuts and secondly how they handle balance across their partitions. Note that LDG results are not showed in the tables: LDG is always worse than FENNEL regarding edge-cuts and equivalent in load-balance.

7.1. Edge-cuts

As it was our objective, we managed to reach a breaking point where the performance of FENNEL finally exceeds ISP. It is shown in Table 5 (See LONG+) that on a plane with the size

of a country like France, with edge Euclidean distance past 175 kilometres on average and 16 partitions, it becomes better to apply FENNEL rather than ISP.

We expected while building our datasets that the performances of ISP would heavily correlates to EDTPS, it has been confirmed through the results. Each time EDTPS increases so does the number of edge-cuts produced by ISP without exceptions. We also managed to reach a scale where ISP is bound to perform extremely poorly (HUGE), in that case up to **95%** of the edges are cut. On the contrary, when edges are extremely short (e.g. TINY-), ISP outperforms **112** times the results of FENNEL in synthetic graphs and up to **125** times when applied to Thing'In (See Table 5 and 7).

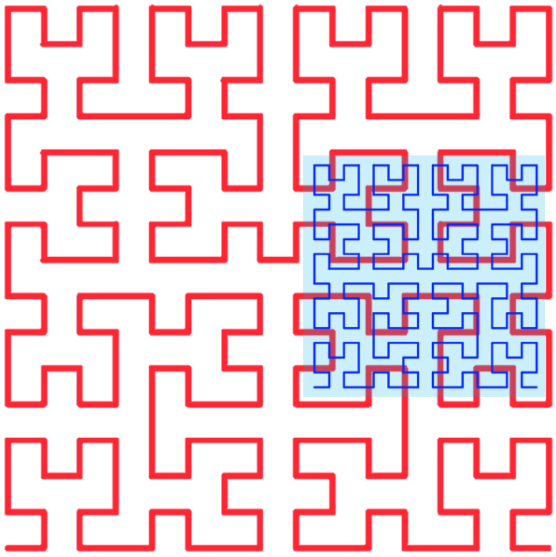

Figure 7. Visual example of generator (Blue SFC) and ISP (Red SFC) mismatch

We used the same space-filling curve based strategy for graph generation and ISP. Yet this has no impact on the results for two reasons. First, both SFCs are not matching. For two curves to match, those needs to use the same algorithm, granularity and map the same multidimensional plan. Although algorithm and granularity used are identical, the multidimensional plan are different. ISP covers the whole world map whereas the generator cover only the bounding boxes given in Table 3, changing both SFC cell bounding box surface and SFC cell ordering (See Figure 7 for a visual example). Second, the generator edge binding step ignores the curve presence, edges are created randomly around a node with respect to edge Euclidean distance configuration.

Interestingly, we see multiple EDTPS and ISP results correlation, it must however be weighted with the amount of borders induced by the partitions which is currently ignored by our metric. Each new partitions produces additional border which may cut edges. As EDTPS does not include those borders in its estimation, we use the following interpretation rule: two graphs with identical EDTPS but different number of partitions should have different behaviour with ISP. The one with less partition should yield better results. We joined a part of the results obtained for the EU plane size in Table 6 and this interpretation rule does hold true.

Table 5. EDTPS and performances of ISP and FENNEL over the generated graphs for FR plane

Category	K	EDTPS	FENNEL	ISP
TINY-_MD	4	0.00192	38.232±0.002	0.34±0.0
	8	0.00272	45.811±0.002	0.55±0.0
	16	0.00384	50.194±0.002	0.83±0.0
TINY	4	0.02123	47.733±0.0	2.61±0.003
	8	0.03002	56.241±0.0	5.1±0.007
	16	0.04245	61.074±0.0	8.05±0.006
SHORT	4	0.04773	47.872±0.0	5.89±0.008
	8	0.0675	56.337±0.0	9.86±0.017
	16	0.09546	61.129±0.0	15.21±0.007
MEDIUM	4	0.09026	48.014±0.001	10.9±0.018
	8	0.12764	56.545±0.001	17.9±0.032
	16	0.18052	61.345±0.001	27.59±0.021
MEDIUM+	4	0.10293	47.908±0.001	11.35±0.013
	8	0.14556	56.372±0.001	19.45±0.014
	16	0.20586	61.16±0.001	27.37±0.01
LONG	4	0.15608	47.914±0.0	16.77±0.013
	8	0.22074	56.379±0.0	29.39±0.017
	16	0.31217	61.175±0.0	41.18±0.006
LONG_MD	4	0.16034	47.917±0.0	18.12±0.026
	8	0.22675	56.38±0.0	28.21±0.031
	16	0.32067	61.171±0.0	41.06±0.04
LONG_LD	4	0.14142	47.897±0.0	15.5±0.05
	8	0.2	56.361±0.0	25.99±0.082
	16	0.28285	61.159±0.0	38.21±0.116
LONG+	4	0.32547	48.07±0.0	30.06±0.017
	8	0.46028	56.564±0.001	47.71±0.013
	16	0.65093	61.364±0.001	67.48±0.004
HUGE	4	0.7235	48.056±0.001	63.88±0.017
	8	1.02318	56.557±0.002	86.02±0.007
	16	1.44699	61.359±0.002	95.49±0.004

If we compare the results of EU LONG+ and FR MEDIUM with k = 8 and 16 (see Table 5 and 6), they have similar EDTPS but EU LONG+ has consistently better edge-cuts results as it uses fewer partitions to reach the same EDTPS. The same pattern can be detected comparing EU LONG_LD to FR MEDIUM k = 8.It can also be applied to graphs within the same plane: take FR TINY with k = 16 and FR MEDIUM+ k = 4 or EU HUGE k = 4 and EU LONG_LD k = 16. At last, ISP performances are similar when both EDTPS and number of partitions are matching(See FR TINY and EU SHORT). The EDTPS values are interleaving and so do the edge-cuts results with ISP. Finally, it is easy to see EDTPS and edge-cuts correlation (See Figure 8). The only unmatched curve corresponds to HUGE which is perfectly acceptable as there is a ceiling effect for the edge-cuts. EDTPS may go further than 100% but edge-cuts are ultimately limited to 100%.

Table 6. Partial EDTPS and performances of ISP and FENNEL over the generated graphs for EU plane

Category	K	EDTPS	FENNEL	ISP
SHORT	4	0.01765	47.279±0.001	2.49±0.003
	8	0.02495	55.79±0.001	4.18±0.004
	16	0.03529	60.601±0.001	7.24±0.004
LONG	16	0.1124	61.167±0.001	18.36±0.014
LONG_MD	16	0.11017	61.163±0.0	17.81±0.01
LONG_LD	16	0.12016	61.159±0.0	18.65±0.002
LONG+	4	0.12684	48.365±0.001	14.37±0.022
	8	0.17937	56.899±0.001	25.49±0.018
	16	0.25367	61.709±0.001	36.19±0.007

Table 7. EDTPS and performances of ISP and FENNEL over Thing'In

	K	EDTPS	FENNEL	ISP
Thing'In	4	0.00006	6.349	0.132
	8	0.00008	34.636	0.277
	16	0.00011	37.883	0.435

While EDTPS seem appropriate to evaluate ISP performance, it ignores other potential factors like connectivity and volumetry, we argue that it does not affect its veracity. We did run tests to check the volumetry impact and we couldn't observe anything significant, the problem is more about connectivity. Connectivity is ignored in both ISP and EDTPS and our dataset does not present much connectivity variation. Still, in essence, as connectivity degree rise for a given node, it has to reach further and further to connect to new nodes as there is a finite number of nodes in its neighbourhood. Consequently, edge distance tends to be longer. But as edge distance is already used in EDTPS, ignoring connectivity should not be a problem.

We expected ISP results to be a bit dissonant; nonetheless we are surprised by FENNEL extreme stability. Again, our most serious guess is based on the fact that all the synthetics graphs share the same low connectivity with absence of super nodes which FENNEL relies on to build its partitions.

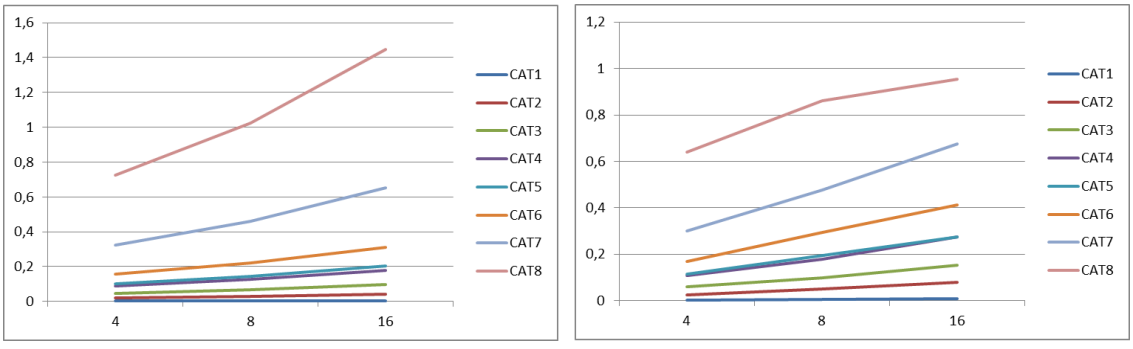

Figure 8. (Left) EDTPS, (Right) Edge-Cuts percentage of graph partitioned with ISP for each category. X-axis is the partition number, bounding box is FR, density information excluded as impact less on EDTPS and edge-cuts.

Contrary to our synthetics graphs, Thing'In possess less edges but its connectivity degree distribution has a better scaling, it could explain why FENNEL yields better results when applied on Thing'In. There remains a wide disparity between 4 and 8 partitions with FENNEL on Thing'In for which we have no appropriate explanation. Overall, we cannot be sure that the results similarities for FENNEL are really due to the connectivity. It is however safe to assume that edge distance has no direct impact on this solution.

7.2. Partition balance

We discuss now about how the different strategies handle balance across the partitions. FENNEL and LDG are pretty much perfect at preserving load balance across the partition, the same cannot be said for ISP looking at Table 8. We didn't put the other graph categories but they all use the same density except for TINY-. The partition imbalance ranges from 2.26% to 11.18% with 16 partitions on FR plane. Obviously, the results regarding imbalance are better for the EU plane. Its worst imbalance case has been recorded at 2.31% which is normal: there is the same number of nodes and density but for a larger plane.

Table 8. CDTPC and load imbalance of ISP and FENNEL over the generated graphs for LONG+

Category	**K**	**CDTPC**	**FENNEL**	**ISP**
LONG+	4	0.04916	0.0±0.0	0.22±0.002
	8	0.09831	0.0±0.0	2.13±0.013
	16	0.19662	0.0±0.0	7.09±0.022
LONG+_MD	4	0.00836	0.0±0.0	0.16±0.002
	8	0.01673	0.0±0.0	0.6±0.003
	16	0.03345	0.0±0.0	1.96±0.003
LONG+_LD	4	0.00325	0.0±0.0	0.1±0.001
	8	0.0065	0.0±0.0	0.19±0.001
	16	0.01299	0.0±0.0	0.61±0.004

The balance provided using the ISP strategy is very fragile. It hugely depends on the graphs if it is "well behaved" or not that is, if the density does not peak too much relatively to the precision of the SFC. Even though a huge CDTPC ratio will not automatically trigger imbalance problem, it is a good risk indicator as a graph with a very low CDTPC will never encounter balance trouble. With the graphs generated based on the density HD given in Table 1 over a plane of a size similar to France and the Hilbert SFC of zoom 12, a single cell might contain more than 500k of nodes all by itself. This does pose a problem because in some circumstances with a high number of partitions and extremely peaked density a single cell has a load high enough to provoke single-cell partitions.

This problem can be solved using SFC with higher zoom as it splits the load of a single cell into multiple cells. There is no complete guarantee that zoom splits the load but in practice it works. A cell which stores 500k nodes at zoom 12 may be reduced to 64 sub cells at zoom 15 with a load of at most 10k nodes by cell, dropping effectively the CDTPC from 17% to 1.5%. With such a small CDTPC, the risk of imbalance becomes almost zero, and in all fairness, zoom 15 is a rather standard precision level. Unfortunately, we could not afford zoom 15 as our metric system wouldn't have been able to hold the additional load memory wise.

8. CONCLUSION

In this paper we went back through an unusual graph partitioning solution reserved to FEM issued graph and applied it, after little extension, to a new class of graph we defined and called WAG. We proposed additional metrics, along with the solution, to analyse why ISP performs well or not and provided our customized geometric graph generator designed to produce customized WAG. We evaluated our partitioning solution through an extended synthetic dataset and compared it to state of the art graph streaming partitioning solutions: LDG and FENNEL. Overall, we showed that when ISP is applied to WAG, performances mainly depend on the graph edge distance and the distribution of its density across the plane. Although ISP is unsuited for long range distance edges, e.g. social graphs, and loses to streaming strategies, it obtains great result for short to medium edge distance typical of spatial networks and outperforms other solutions. In a future with large scale WAG composed of tiny objects with edge distance under one hundred meters as physical interaction between objects is characterized to such range, ISP could prove to be the best solution for such graphs.

For the upcoming work, we would like to further extend ISP. Rather than the analytical context we are much more interested in the database context and we would like to combine ISP with diffusive load balancing technique to optimize both edge cuts and load balance. Replication within ISP is also an interesting subject we would like to explore. At last we have ideas to enhance our WAG generator, we believe we can mix it with the preferential attachment model to produce WAG with high, spatially logical, connectivity.

REFERENCES

[1] K. Andreev et H. Räcke, « Balanced Graph Partitioning », in Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, New York, NY, USA, 2004, p. 120–124, doi: 10.1145/1007912.1007931.

[2] Recent Advances in Graph Partitioning | SpringerLink ». https://link.springer.com/chapter/10.1007/978-3-319-49487-6_4 (consulté le oct. 07, 2020).

[3] F. Payan, C. Roudet, et B. Sauvage, « Semi-Regular Triangle Remeshing: A Comprehensive Study », Comput. Graph. Forum, vol. 34, no 1, p. 86-102, 2015, doi: 10.1111/cgf.12461.

[4] J. R. Pilkington et S. B. Baden, « Partitioning with Spacefilling Curves », 1994.

[5] C. Tsourakakis, C. Gkantsidis, B. Radunovic, et M. Vojnovic, « FENNEL: streaming graph partitioning for massive scale graphs », in Proceedings of the 7th ACM international conference on Web search and data mining - WSDM '14, New York, New York, USA, 2014, p. 333-342, doi: 10.1145/2556195.2556213.

[6] G. Karypis et V. Kumar, « Multilevelk-way Partitioning Scheme for Irregular Graphs », J. Parallel Distrib. Comput., vol. 48, no 1, p. 96-129, janv. 1998, doi: 10.1006/jpdc.1997.1404.

[7] Partitioning of unstructured problems for parallel processing - ScienceDirect ». https://www.sciencedirect.com/science/article/abs/pii/095605219190014V (consulté le janv. 20, 2020).

[8] J. R. Gilbert, G. L. Miller, et S.-Hua. Teng, « Geometric Mesh Partitioning: Implementation and Experiments », SIAM J. Sci. Comput., vol. 19, no 6, p. 2091-2110, nov. 1998, doi: 10.1137/S1064827594275339.

[9] K. Schloegel, G. Karypis, et V. Kumar, « Graph partitioning for high-performance scientific simulations », in Sourcebook of parallel computing, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, p. 491–541.

[10] A. Akdogan, « Partitioning, Indexing and Querying Spatial Data on Cloud », p. 80.

[11] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, et H. Bhogan, « Volley: Automated Data Placement for Geo-Distributed Cloud Services », p. 16.

[12] D. Delling, A. V. Goldberg, I. Razenshteyn, et R. F. Werneck, « Graph Partitioning with Natural Cuts », in 2011 IEEE International Parallel Distributed Processing Symposium, mai 2011, p. 1135-1146, doi: 10.1109/IPDPS.2011.108.

[13] I. Stanton et G. Kliot, « Streaming graph partitioning for large distributed graphs », in Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '12, Beijing, China, 2012, p. 1222, doi: 10.1145/2339530.2339722.
[14] D. Hilbert, « Über die stetige Abbildung einer Linie auf ein Flächenstück », in Dritter Band: Analysis • Grundlagen der Mathematik • Physik Verschiedenes: Nebst Einer Lebensgeschichte, Berlin, Heidelberg: Springer Berlin Heidelberg, 1935, p. 1–2.
[15] M. Knoll et T. Weis, « Optimizing Locality for Self-organizing Context-Based Systems », in Self-Organizing Systems, 2006, p. 62-73.
[16] B. M. Waxman, « Routing of multipoint connections », IEEE J. Sel. Areas Commun., vol. 6, no 9, p. 1617-1622, déc. 1988, doi: 10.1109/49.12889.
[17] M. D. Penrose, « Connectivity of soft random geometric graphs », Ann. Appl. Probab., vol. 26, no 2, p. 986-1028, avr. 2016, doi: 10.1214/15-AAP1110.
[18] R. Albert et A.-L. Barabasi, « Statistical mechanics of complex networks », Rev. Mod. Phys., vol. 74, no 1, p. 47-97, janv. 2002, doi: 10.1103/RevModPhys.74.47.
[19] A. Guttman, « R-trees: a dynamic index structure for spatial searching », in Proceedings of the 1984 ACM SIGMOD international conference on Management of data, New York, NY, USA, juin 1984, p. 47–57, doi: 10.1145/602259.602266.

AUTHORS

Cyprien Gottstein graduated from the University of Rennes 1 with a master degree in Software Engineering in 2016, Rennes, France. He worked from 2016 to 2018 as an engineer at Thales services and started his PhD in large scale graph partitioning for the Internet of Things (IoT) in 2018 at Orange Labs, Rennes, France.

Philippe Raipin Parvedy has received his PhD from the University of Rennes 1, Rennes, France, in 2004. He was a temporary lecturer and research assistant in the university of Rennes 1 from 2004 to 2006 and a post doc researcher in Orange from 2006 to 2007, since 2007, he has been a research engineer in Orange. He is currently the research program manager of the Web of Things platform in Orange. His research interests include dependability, large systems and graph databases.

Michel Hurfin holds a Ph.D. in Computer Science from the University of Rennes (1993). After one year spent at Kansas State University, he is currently conducting his scientific research activities at the INRIA Rennes Bretagne Atlantique research center. He defended his HDR (Habilitation à Diriger des Recherches) in 2004. Since 2012 he is a member of the CIDRE project that aims at addressing security problems. His research interests include distributed systems, software engineering and middleware for distributed operating systems. His recent works focus mainly on dependability and security issues in distributed systems (in particular, intrusion detection mechanisms).

Thomas Hassan is a researcher at Orange Labs, Rennes, France since 2019. He obtained a Thesis in Computer Science from the University of Burgundy, Dijon, France in 2017. His main research areas are Semantic Web, Machine Learning and Big Data. His research was applied to the domains of pervasive computing, Internet of Things, geographic information system and recommender system.

Thierry Coupaye is head of research on Internet of Things (IoT) inside Orange, Orange Expert on Future Network and Orange Open Source Referent. Prior to that, after he completed his PhD in Computer Science in 1996, he had several research and teaching positions at Grenoble University, European Bioinformatics Institute (Cambridge, U.K.) and Dassault Systems. He joined France Telecom in 2000 where he had several research expert, project manager, project and program director positions in the area of distributed systems architecture, autonomic computing, cloud/fog/edge computing and networking. He is the author of more than 75 refereed articles and has participated in multiple program and organisation committees of conferences in these areas. His current research interests include Digital Twins and Edge Intelligence (AI@Edge).