Internship report

Pierre-Gabriel Berlureau
Under the supervision of Claire Lagesse and Julien Randon-Furling

August 20, 2024

Acknowledgments

I would like to extend my sincere thanks to Claire Lagesse and Julien Randon-Furling for giving me this opportunity, and for their guidance, time, and support throughout the internship. A huge thank you to Nicolas Lepy for his invaluable advice and the time he dedicated to helping me during the first part of the intership. I also appreciate François Martin for always being nice to me, making me comfortable at my workspace. Finally, thanks to everyone I met at the lab for being so welcoming and pleasant.

1 Introduction

This report focuses on the application of formal graph theoretical methods in two different contexts. First, we look at a question stemming from an interdisciplinary problem: how to efficiently obtain graphs representing the network of roads, paths and pathways in a city, from data such as crow-sourced geographical information, from OpenStreetMap. This question lies at the interface between computer science, mathematics and geography. Building on previous work, we have proposed a new algorithm and coded its implementation in Python. The resulting package will be used in future works by geographers from the research team where this internship took place, as well as by other researchers. In the second part of this report, we present work of a more theoretical nature, still in connection with graphs. Indeed we examine how a well-known phenomenon in complex systems, named self-organized criticality (SOC), may actually be put to use in order to optimize graph coloring. In particular, we investigate here how a SOC-based method compares with a simple, random scheme. It is hoped that some of these SOC-based optimization methods could prove useful for certain tasks pertaining to urban graph analysis – even though it was not possible to explore such connections within the time-frame of the internship reported here.

2 Ways' Hypergraph

2.1 Introduction

This part of the intership focuses on urban networks analysis using graph theory. In her thesis [7], Claire Lagesse developed a model based on spatial hypergraphs to improve urban networks analysis. This model is already implemented in *QGIS*[1] via the *Morpheo*[2] plugin based on her code. However, due to huge developpements and improvements on python's libraries in the field such as the momepy python package[3], providing robust tools for urban morphology and network analysis with traditional

[1]QGIS (Quantum Geographic Information System) is a free, open-source software that allows users to create, edit, visualize, analyze, and publish geospatial information.

[2]The Morpheo plugin for QGIS is a tool for analyzing urban morphology, including spatial relationships and network structures.

graph-based methods, implementing it in python opens up new opportunities. It makes the model accessible to a broader audience, including researchers who may not have expertise in geomatics or Geographic Information System (GIS), and allows for easier integration with the latest advancements in Python's libraries. In a first subsection we present Claire Lagesse's model and our new implementation of it and in a second one we focus on an OpenStreetMap data preprocessing issue.

2.2 Model

In this subsection, an urban graph is a spatial graph[3] the nodes of which are intersections or road ends and the edges of which are road segments in an urban network. As shown in Claire Lagesse's thesis [7], global indicators measures such as closeness or betweenness on urban graphs depend on the delimitation of study zone and this choice of representation does not reflect well urban structure. She then developed a hypergraph model to represent and analyse urban networks, called ways' hypergraph. The ways' hypergraph can be obtained from a spatial graph using algorithm 1.

Algorithm 1 Ways' hypergraph algorithm

```
Require: $G = (S, A)$ an unoriented spatial graph
  for all $u \in S$ do
      while $\exists v \neq w$ neighbors of $u$ such that $\theta_{uv,uw}$ is in $[\theta_{max} - \frac{\pi}{3}, \theta_{max} + \frac{\pi}{3}]$ do
          Pair edges realizing the closest angle from $\pi$, on $u$
      end while
  end for
  $H \leftarrow \{\{u_1, \dots, u_p\}$ such that it exists $\{u_1, u_2\}, \dots, \{u_{p-1}, u_p\}$ in $A$
   and for all $i$ in $\{1, \dots, p-2\}$ $\{u_i, u_{i+1}\}$ is paired to $\{u_{i+1}, u_{i+2}\}\}$
  return $\mathcal{H} = (S, H)$
```

Less formally, ways' hypergraph is obtained by "pairing" edges incident to a same node and realizing the closest angle from π (among all angles of edges pairs incident to the considered node) being closer than a fixed threshold (Lagesse chose $\frac{\pi}{3}$ so angles have to be in $[\pi - \frac{\pi}{3}, \pi + \frac{\pi}{3}]$). Ways' hypergraph's nodes are the same as in the graph and it features exactly one hyperedge per equivalence class of the pairing relation which is the union of its unoriented edges.

Implementation Despite the need of the team and several doctorants/researchers, there was no existing python package to build ways' hypergraph. We coded one based on the networkx[1], geopandas[6] and xgi[8] packages. It implements:

- A class to represent a WayGraph. It is built from a shapefile[4] and is mostly a wrapper around a networkx MultiGraph representing the urban graph of the shapefile. It also contains a geopandas GeoDataFrame which represents the shapefile.
- A class to represent a ways' hypergraph. It is built from a WayGraph using Claire Lagesse's algorithm and is a wrapper around an xgi HyperGraph additionally to some data from the original WayGraph.
- Some shapefiles preprocessing functions 2.3.

As it appears that the algorithm consists of computing an equivalence relation which is equivalent to compute equivalence classes, our implementation of Claire Lagesse's algorithm relies on DisjointSet structure. At the beginning every edge is alone in its equivalence class and "pairing" two edges is then

[3]A spatial graph is a graph the nodes and edges of which have coordinates

[4]A shapefile is a simple, nontopological format for storing the geometric location and attribute information of geographic features such as what interests us: the roads.

realized by merging the equivalence classes of those edges. Despite the fact that the package is written in Python, it only requires a minute to run on the whole *Franche-Comté* (The internship took place in *Besançon*) on a standard laptop.

Contribution to the xgi package Indicators are computed using the linegraph of the ways' hypergraph. The original hypergraph to linegraph function of xgi used casts between sets and tuples. As hypergraph's nodes' labels are only required to be hashable values, we directly used the coordinates of each nodes as labels instead of mapping them to integers. This lead to a bug we spotted and fixed. Our solution was approved by the authors, see details here .

2.3 Preprocessing

2.3.1 Introduction

Street network datasets exhibit considerable variation in both their level of detail and data quality. The suitability of a particular dataset largely depends on the intended application. A common challenge in processing these datasets involves reducing their level of detail to fit different use cases while ensuring that no essential information is lost and no new inaccuracies are introduced.

2.3.2 Face artifacts

Face artifacts are a common issue in the representation of geospatial street networks, particularly those with a high level of detail where individual traffic lanes and intersections are digitized separately. These artifacts manifest as faces[5] within the network that do not correspond to actual urban blocks[6] Figure 1. Such artifacts can significantly distort analyses in urban morphology and street network studies by introducing superfluous network edges. This, in turn, affects the accuracy of metrics related to node degree, shortest path computations, and the overall topology of the network. The identification and removal of face artifacts are critical for ensuring the validity of these analyses; however, the process remains challenging due to the lack of fully automated methods, necessitating manual intervention that is both time-consuming and prone to inconsistencies. In a recent article, Martin Fleischmann and Anastassia Vybornova show how they tackle the problem of face artifacts detection[2].They identify face artifacts by calculating the area and compactness of each face in a given area, using five different compactness metrics. These metrics are then combined with the face's area to create a face artifact index, which helps capture the relationship between compactness and size. To manage outliers, they apply a logarithmic scale that smooths the distribution. By examining the distribution of these indices, they often find two clear peaks—one representing face artifacts and the other urban blocks. The threshold between these peaks helps distinguish face artifacts from regular urban blocks. The problem we tried to solve can be described as getting rid of those face artifacts while keeping the network as close from the original as possible. We developed a method to clean the network without having to detect face artifacts. In this subsection we explain it and discuss its results.

2.3.3 Method

The method consists in the four following steps Figure 2:

1. Compute a buffer geometry around each line in the shapefile, large enough so it covers eventuals artifacts but not too thick so it is still possible to distinguish each street, looking at the buffers.

2. Compute the union of all buffers (so two buffers sharing a border merge into a single geometry) and shrink back the obtained geometry by computing a negative width buffer around it.

[5]As in a graph, a face is an area enclosed by the lines or edges of a network.

[6]An urban block is an area of land surrounded by streets, often containing buildings, parks, or other types of land use.

3. Compute the skeleton of the obtained geometries using Voronoï polygons, by taking points on the border of the union as centroids and keeping only the polygon's edges which are included in the union.

4. Compute the ways' hypergraph of the obtained skeleton.

2.3.4 Results

Our method effectively removes artifacts with whidths not exceeding twice the buffer width; however, it has the unintended effect of reducing some roundabouts of small and medium sizes to mere point intersections Figure 3, which is not always desirable. Additionally, the method introduces unwanted curvatures in small road segments Figure 4. A potential approach to address the roundabout issue could involve a preliminary pass to preserve and reintegrate these features. Meanwhile, the curvature problem might be mitigated by generating lines that represent the averaged geometries—such as through linear regression—instead of relying on skeleton extraction via Voronoi polygons. However, this approach also risks distorting the original network, and its effectiveness depends on the specific use of the data. In comparison, the approach presented in [2] leaves the challenge of managing face artifacts to a separate model possibly allowing for more flexibility. This remaining tasks is very challenging as artifacts can have various shapes and are often adjacents, making them hard to process. While our method successfully removes most face artifacts, its drawbacks make it unusable for some use cases such as traffic-routing or tracking temporal evolution of urban networks. Indeed, the latter may require that the lines representing the same streets at different times overlap perfectly, which our method cannot guarentee. To sum up, our approach provides a more direct and streamlined process but it may require additional steps to refine the network for specific applications, especially when preserving the original network's structural integrity is crucial. Moving forward, future research could involve developing an approach that reconstructs an urban network while minimizing face artifacts and preserving the morphological integrity of the original network. This could involve optimizing a function that balances the reduction of artifacts with the preservation of the network's structural properties. The subsequent section explores a method for finding global minima in a non-convex function, where the variables are mapped onto the nodes of a graph, offering a potential solution for this optimization.

3 Self Optimized Criticality applied to graph coloring

3.1 Introduction

Self-organized criticality (SOC) is a concept that has become fundamental in the study of complex systems. Introduced by Bak, Tang, and Wiesenfeld in 1987, SOC describes the intrinsic tendency of certain dynamical systems to naturally evolve towards a critical state, where no external tuning of parameters is required. At this critical state, the system exhibits scale-invariant behavior, with fluctuations ranging widely in magnitude, typically following a power-law distribution. SOC is observed in various natural phenomena, such as earthquakes, forest fires, and neural activity, suggesting that many systems in nature inherently self-organize into a critical state.

A prominent model that exemplifies SOC is the Abelian sandpile model. Originally defined on a square lattice, this model provides a simple yet insightful way to understand SOC dynamics. The model is defined on a graph with N nodes, where each node i has an associated integer variable x_i, representing the number of grains of sand at that node. The dynamics of the system are governed by two processes: the slow addition of grains and the fast redistribution of grains when a threshold is exceeded.

Grains are added one at a time to randomly chosen nodes, incrementing the sand count at node i by 1:

$$x_i \to x_i + 1.$$

If the number of grains at a node exceeds its threshold, $x_i > d_i$ (where d_i is the degree of node i), the node becomes unstable and topples, redistributing grains to its neighboring nodes. The toppling rule is defined as:

$$x_i \to x_i - d_i, \quad x_j \to x_j + 1 \text{ for all } j \in \mathcal{N}(i),$$

where $\mathcal{N}(i)$ denotes the set of neighboring nodes of i. This process can trigger further topplings in neighboring nodes, leading to a cascade of events, or an avalanche. The system eventually reaches a critical state where the distribution of avalanche sizes follows a power law, indicating SOC.

The principles underlying SOC and the Abelian sandpile model can be applied to tackle complex optimization problems, such as graph coloring. Graph coloring involves assigning colors to the vertices of a graph such that no two adjacent vertices share the same color, while minimizing the total number of colors used. This problem is NP-hard, making it difficult to solve efficiently for large graphs. However, by mapping the graph structure onto the SOC model, we can leverage the natural dynamics of SOC to explore the solution space effectively.

In this approach, each node in the graph corresponds to a site in the Abelian sandpile model, and the avalanches generated by the SOC process are used to produce test patterns for graph coloring. The SOC process allows for a broad exploration of the solution space, helping to avoid local minima that commonly trap traditional optimization methods. This section investigates the application of the Abelian sandpile model to the graph coloring problem as presented in [4].

3.2 Model

To apply SOC search, a cost function is needed. The following has been shown to work [4, 5]:

$$E = -\sum_{i=1}^{k} |c_i|^2 + \sum_{i=1}^{k} 2|c_i||e_i|$$

where $|c_i|$ is the number of nodes in color class i, $|e_i|$ the number of bad edges[7] attached to nodes of color i, and k an upper bound on the number of colors. Using this cost function, legal colorings are local minima, and the number of colors is implicitly minimized and is obtained as the number of nonzero $|c_i|$ values. SOC search for graph coloring consists in algorithm 2:

Algorithm 2 SOC search

```
Require: $G = (S, A)$ a graph, $k$ an upper bound on the number of colors and $n \in \mathbb{N}$ a number of steps
  Let $c$ be a color function which assigns a random color in $\{0, \dots, k-1\}$ to each node
  Iterate abelian sandpile model's slow and fast process until SOC state is reached.
  for all $i$ from 0 to $n-1$ do
      Iterate abelian sandpile model's slow process until an avalanche occurs and ends
      Let $\mathcal{A}$ be the set of nodes caught in the avalanche
      Let $c'$ be $c$ in which a new random color in $\{0, \dots, k-1\}$ is assigned to each node in $\mathcal{A}$
      if $E(c') < E(c)$ then
          $c \leftarrow c'$
      end if
  end for
  return $c$
```

[7] An edge $\{u, v\}$ with u and v sharing the same color

3.3 Comparison with two simple models

As this model uses the Abelian sandpile model to select a batch of nodes to recolor, it relies on an arbitrary order on colors. Indeed, for a node to be caught in the avalanche it needs to have its color variable increased to the threshold by the process. This means that nodes with a higher color value are more likely to be recolored. This remark suggested to us an other simple model to select a batch of nodes at each coloring step.

3.3.1 Model 1

In the first model, already tested in the paper, the batch of nodes consists in a single random node.

3.3.2 Model 2

The second model consists in a random choice of a node. Then for each node of the batch we add its neighbors sharing its color to the batch. We iterate this last step until it converges.

3.3.3 Results

Different graphs We tested and compared those models on three different graphs each with 1000 nodes:

- A grid graph
- An Erdős–Rényi graph with $p = \frac{1}{100}$
- A random geometric graph with an adjacency radius of 0.05.

Discussion The SOC search algorithm consistently found the lowest values for the cost function across all three types of graphs Figure 5, demonstrating its effectiveness in optimizing graph coloring. However, the performance of the other two models was also notable. Both Model 1 and Model 2 achieved nearly comparable cost function values to SOC search but with significantly reduced computational time. Specifically, both models ran in roughly half the time it took for the SOC search to complete, and required fewer steps to reach their respective solutions.

This suggests that while SOC search excels in finding highly optimized solutions, the simpler models offer a compelling trade-off between accuracy and efficiency. Their faster runtime makes them attractive alternatives in applications where computational resources are limited and near-optimal solutions are sufficient. Furthermore, the complementary strengths of these models suggest that a hybrid approach could potentially combine the thorough exploration of SOC search with the efficiency of Model 1 and Model 2, leading to even more effective graph coloring strategies.

4 Conclusion

This research internship explored the application of graph theoretical methods in urban network analysis and graph coloring optimization. In the first part, we addressed the challenge of efficiently generating graphs from OpenStreetMap data, proposing and implementing an algorithm in Python that enhances the analysis of urban networks using Claire Lagesse's ways' hypergraph model. The resulting Python package provides a promising tool, improving upon existing implementations by leveraging recent advancements in Python libraries.

In the second part, we investigated the potential of self-organized criticality (SOC) to optimize graph coloring. By comparing SOC-based methods with simpler random schemes, we demonstrated that while SOC can achieve lower energy states, alternative methods offer competitive results with faster convergence times. Although not fully explored within the internship's timeframe, the findings suggest that SOC-based optimization may hold promise for specific applications in urban graph analysis.

Overall, this work contributes both practical tools and theoretical insights, laying some groundwork for future research in urban network analysis and optimization using graph theory.

References

[1] Daniel A. Schult Aric A. Hagberg and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In *Proceedings of the 7th Python in Science Conference*, pages 11–15, 2008.

[2] Anastassia Fleischmann, Martin & Vybornova. A shape-based heuristic for the detection of urban block artifacts in street networks. *Journal of Spatial Information Science*, 2024(28), 2023.

[3] Martin Fleischmann. momepy: Urban morphology measuring toolkit. *Journal of Open Source Software*, 4(43):1807, 2019.

[4] Heiko Hoffmann and David Payton. Optimization by self-organized criticality. *Scientific Reports*, 8, 02 2018.

[5] David Johnson, Cecilia Aragon, Lyle McGeoch, and Catherine Schevon. Optimization by simulated annealing: An experimental evaluation; part ii, graph coloring and number partitioning. *Operations Research*, 39:378–406, 06 1991.

[6] Kelsey Jordahl, Joris Van den Bossche, Martin Fleischmann, Jacob Wasserman, James McBride, Jeffrey Gerard, Jeff Tratner, Matthew Perry, Adrian Garcia Badaracco, Carson Farmer, Geir Arne Hjelle, Alan D. Snow, Micah Cochran, Sean Gillies, Lucas Culbertson, Matt Bartos, Nick Eubank, maxalbert, Aleksey Bilogur, Sergio Rey, Christopher Ren, Dani Arribas-Bel, Leah Wasser, Levi John Wolf, Martin Journois, Joshua Wilson, Adam Greenhall, Chris Holdgraf, Filipe, and François Leblanc. geopandas/geopandas: v0.8.1, July 2020.

[7] Claire Lagesse. *Lire les Lignes de la Ville : Méthodologie de caractérisation des graphes spatiaux.* PhD thesis, Universite Paris Diderot-Paris VII, 2015.

[8] Nicholas W. Landry, Maxime Lucas, Iacopo Iacopini, Giovanni Petri, Alice Schwarze, Alice Patania, and Leo Torres. Xgi: A python package for higher-order interaction networks. *Journal of Open Source Software*, 8(85):5162, 2023.

Figures

Preprocessing figures

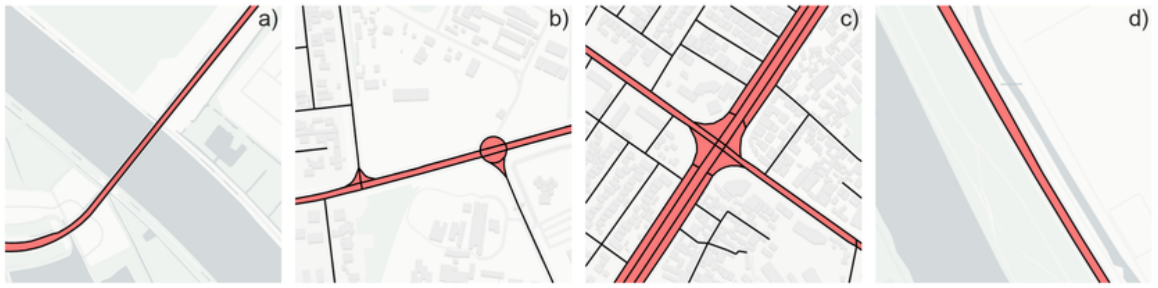
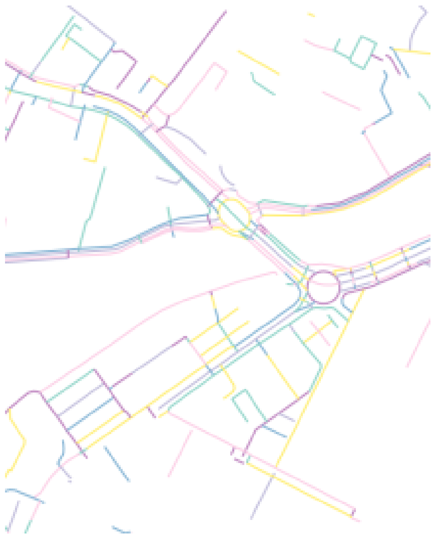
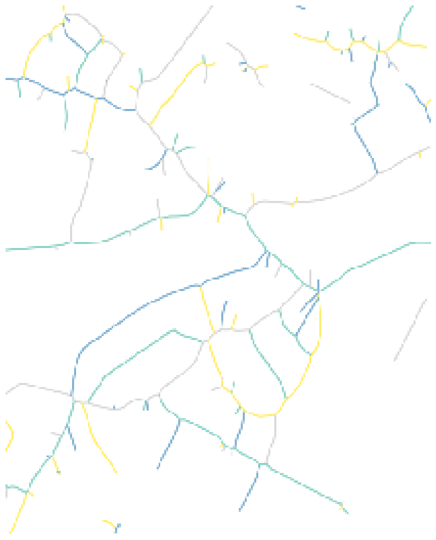


Figure 1: a) Bridge, Amsterdam; b) Roundabout, Abidjan; c) Intersection, Kabul; d) Motorway, Vienna. Faces classified as face artifacts are shown in red, and the OSM street network (without service roads) is shown in black. Map data ©OpenStreetMap contributors ©CARTO. [2]

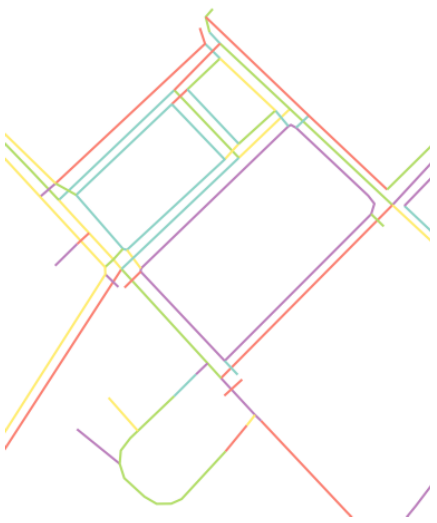


(a) Initial data



(b) Preprocessed data

Figure 2: Preprocessing method reducing roundabouts to mere intersections. a) The Initial data in a specific area of *Besançon*; b) The same area preprocessed, missing its two central roundabouts, both reduced to mere intersections.


(a) Initial data

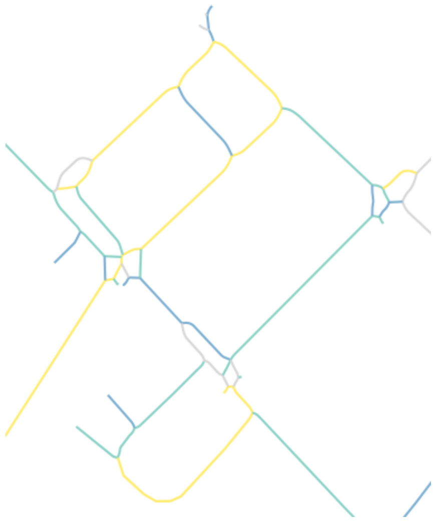
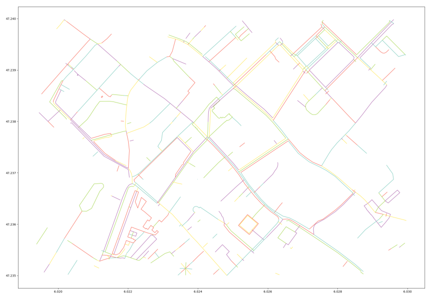
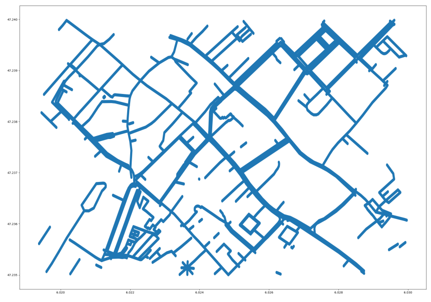

(b) Preprocessed data

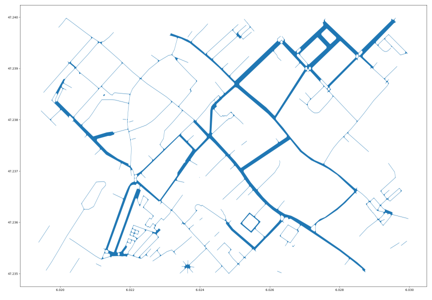
Figure 3: Preprocessing method curving small road segments. a) The initial data in a specific area of *Besançon*; b) The same area preprocessed, having some of its road segments curved by the preprocessing method.



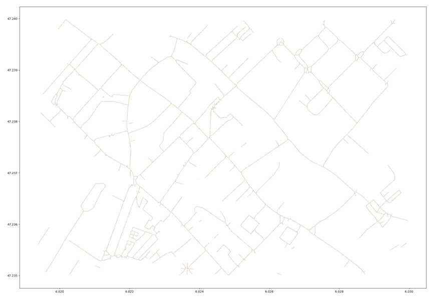
(a) Initial data



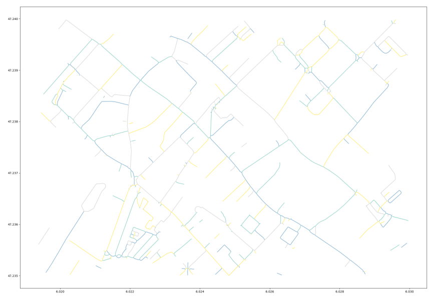
(b) Buffers



(c) Shrinked union geometry



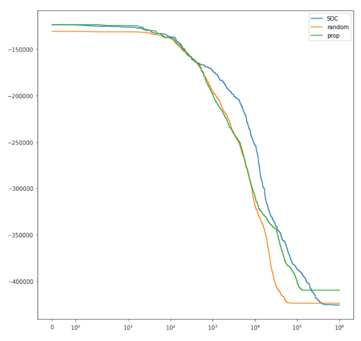
(d) Skeleton



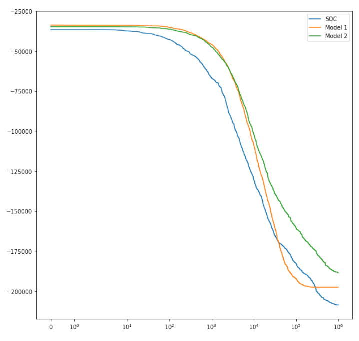
(e) Ways' hypergraph

Figure 4: Successive steps of the preprocessing method. a) Initial data in a specific area of *Besançon*; b) Buffer geometry around each line in the initial data; c) Shrinked backed buffers with negative width; d) Skeleton of the obtained geometry; e) Ways' hypergraph of the skeleton network

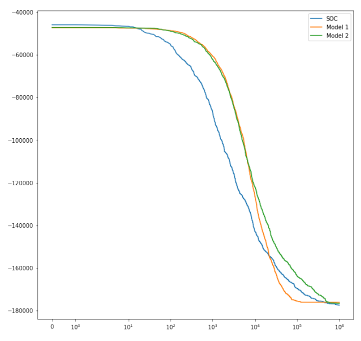
SOC applied to graph coloring figures



(a) Grid graph



(b) Erdős–Rényi graph



(c) Random geometric graph

Figure 5: Values of the cost function with respect to the number of steps number: a) On the grid graph; b) On the Erdős–Rényi graph; c) On the random geometric graph.