# Applications de la théorie des graphes à l’analyse de réseaux urbains

Pierre-Gabriel Berlureau

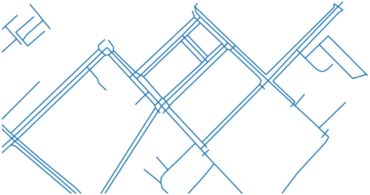
Lundi 9 septembre 2024

# L'équipe du projet


- Matthieu Latapy
  *LIP6 Sorbonne Université*
- Claire Lagesse
  *ThéMA Univ. Franche-Comté*
- Julien Randon-Furling
  *Centre Borelli ENS P-S*

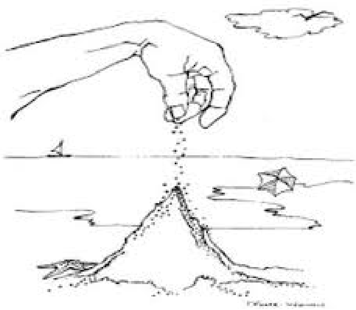
```python
def preprocess(gdf, buffsize):
    """
    A function to get rid of multiple linestrings on the "same" street by
    computing the skeleton of the union of all buffers around LineStrings using Voronoi polygons.

    Parameters
    ----------
    gdf: gpd.GeoDataFrame
        Input shapefile

    buffsize:
        Size of buffers used to merge linestrings

    Returns
    -------
    gpd.GeoDataFrame
        Preprocessed GeoDataFrame Geometry column
    """
    def construct_centerline(input_geometry, interpolation_distance=buffsize/10):
       #find the voronoi verticies (equivalent to Centerline._get_voronoi_vertices_and_ridges())
       borders = input_geometry.segmentize(interpolation_distance) #To have smaler verticies (equivalent to Centerline._get_densified_borders())
       voronoied = shapely.voronoi_polygons(borders,only_edges=True) #equivalent to the scipy.spatial.Voronoi

       #to select only the linestring within the input geometry (equivalent to Centerline._linestring_is_within_input_geometry)
       centerlines = gpd.GeoDataFrame(geometry=gpd.GeoSeries(voronoied.geoms)).sjoin(gpd.GeoDataFrame(geometry=gpd.GeoSeries(input_geometry)),predicate="within")

       return centerlines.unary_union

    buffer = gdf.buffer(buffsize) #Creates the buffer
    union = gpd.GeoSeries(list(shapely.union_all(buffer).geoms), crs=buffer.crs).buffer(buffsize*(1/10-1)) #Computes the union of all buffers and shrink it
    return buffer, union, gpd.GeoDataFrame(geometry=np.hstack(np.vectorize(lambda x: construct_centerline(x).geoms)(union)), crs=gdf.crs)
```

Implémentation d’un modèle

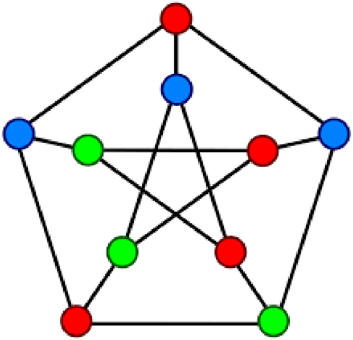


Données OpenStreetMap

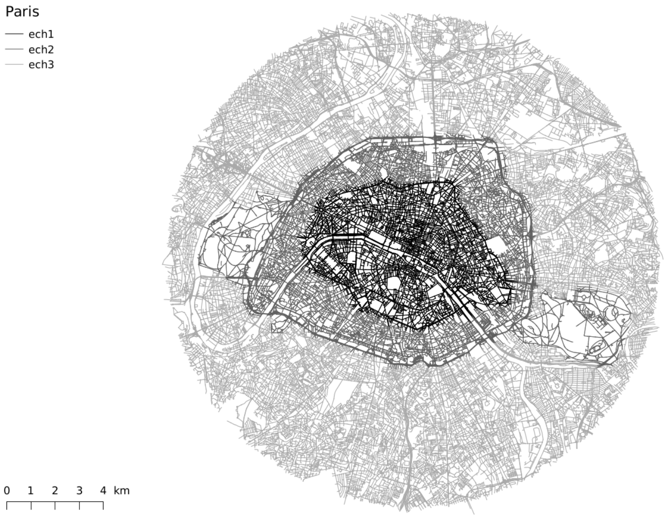
Résultat voulu



Physique statistique

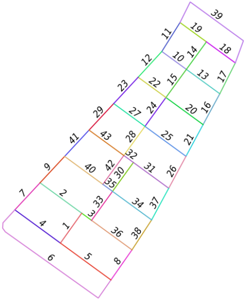


Coloration de graphe

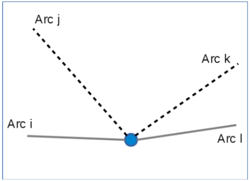



Différents découpages

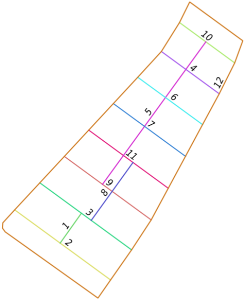
# Modèle



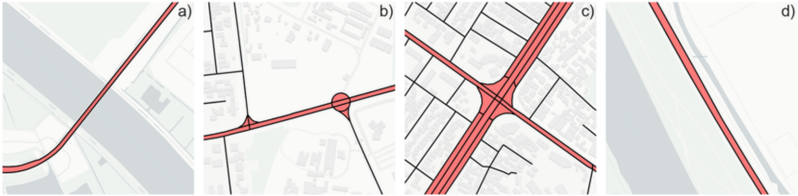
Graphe initial




“Pairage d’arêtes”

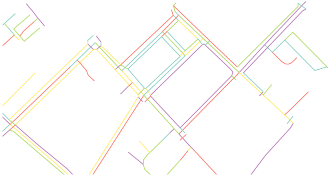


Hypergraphe des voies

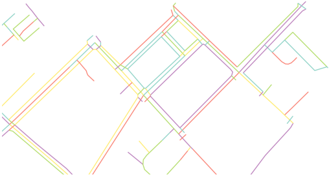
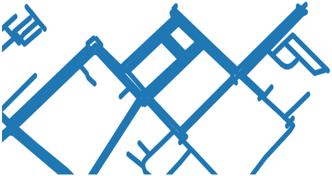



Face artifacts

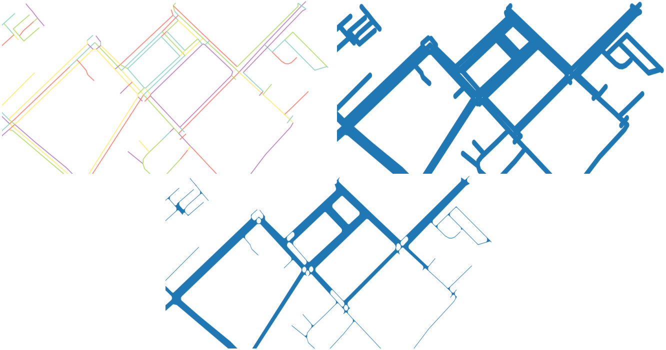
# Méthode



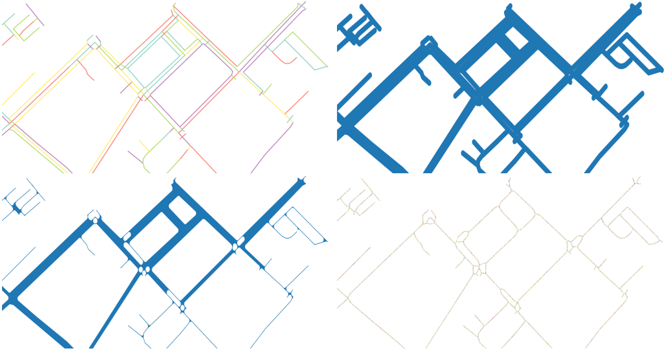
# Méthode

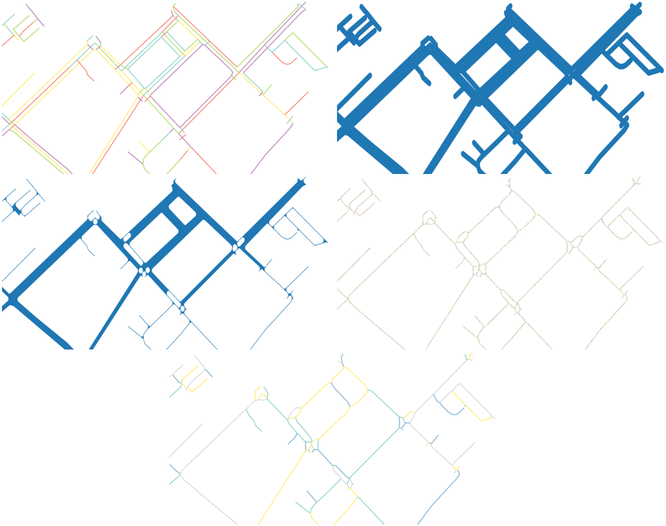
# Méthode



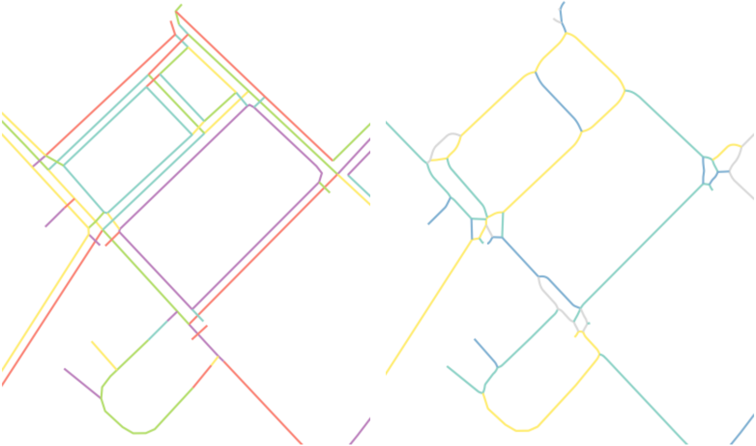
# Méthode



# Méthode



# Résultats



# Résultats

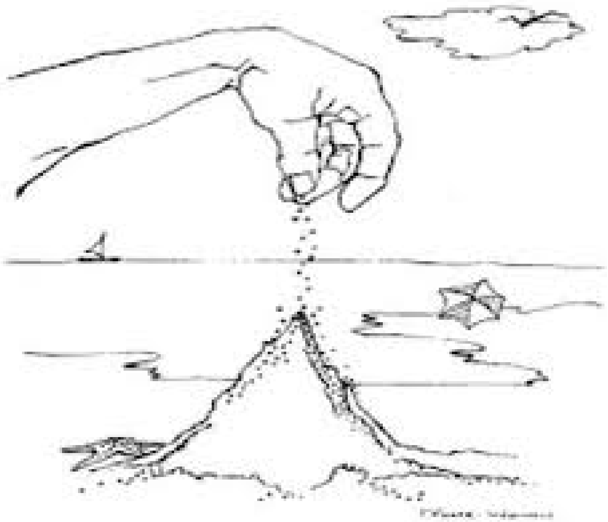
# Un autre point de vue

- **Idée** Se ramener à un problème d'optimisation

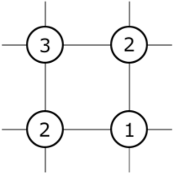
# Un autre point de vue

- **Idée** Se ramener à un problème d'optimisation
- **Exemple** Chercher un graphe qui minimise le nombre de face artifacts tout en conservant la structure du réseau

# Criticité auto-organisée

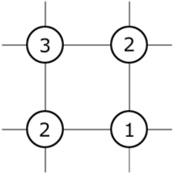


État initial



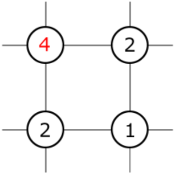
# Abelian sandpile model

État initial



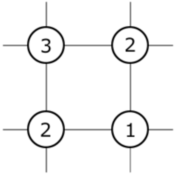

Processus lent

$x_i \to x_i + 1$



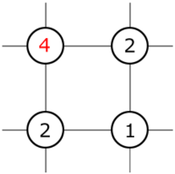
# Abelian sandpile model

État initial




Processus lent

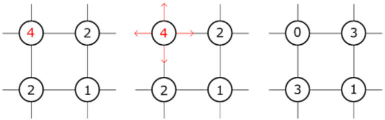
$x_i \to x_i + 1$




Avalanche

$x_i \to x_i - d_i$

$x_j \to x_j + 1$ pour $j$ voisin de $i$



# Algorithme

Energy function :

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

# Algorithme

**Algorithm** SOC search

**Entrée:** $G = (S, A)$ et $k, n \in \mathbb{N}$

$c \leftarrow$ colorage aléatoire

Itérer le abelian sandpile model jusqu'à atteindre l'état critique

**pour** $i$ from 0 to $n - 1$ **faire**

  Itérer le processus lent jusqu'à déclencher une avalanche

  $\mathcal{A} \leftarrow$ noeuds pris dans l'avalanche

  $c' \leftarrow c$ avec $\mathcal{A}$ recoloré aléatoirement

  **si** $E(c') < E(c)$ **alors**

    $c \leftarrow c'$

  **fin si**

**fin pour**

**renvoyer** $c$

# Deux autres modèles simples

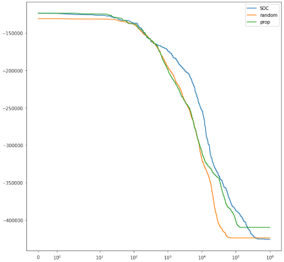
$$\mathcal{A} := \text{ensemble de noeuds choisis}$$

**Model 1**

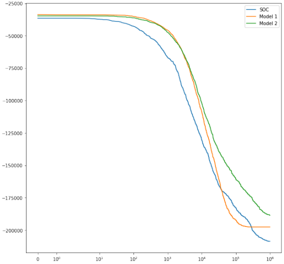
$$\mathcal{A} \leftarrow \text{unique noeud aléatoire}$$

**Model 2**

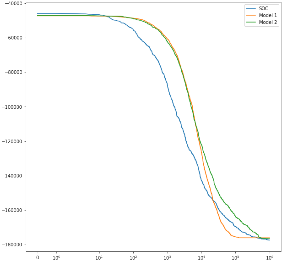
$$\mathcal{A} \leftarrow \text{unique noeud aléatoire}$$
$$\mathcal{A} \leftarrow \mathcal{A} \cup \{u \in \mathcal{V}(\mathcal{A}),\ c(u) = c(\mathcal{A})\}$$




Grille




Aléatoire $p = \frac{1}{100}$




Géométrique aléatoire
$rayon = 0.05$

# Conclusion

- Hypergraphe des voies
- Pré-traitement de face artifacts
- Optimisation basée sur la Criticité auto-organisée

# Conclusion

- Hypergraphe des voies
- Pré-traitement de face artifacts
- Optimisation basée sur la Criticité auto-organisée

- Plusieurs passes de pré-traitement ?
- Squelette linéaire sur les buffers ?
- Optimisation pour le pré-traitement de réseaux urbains ?