CENTRALESUPELEC
2nd year of cursus centralien
Department of Mechanics

file=LogoCS.png,width=3.5cm

Computer aided design of a water supp
network

AI & R Lab
Laboratorio del dioporco facile del Poliminchia

Relatore: Filippo Gatti
Correlatore: Andrea Barbarulo

Auth
calvet-litzell E
Cognetti Giova
Marzorati Edoa
Vaccarino Marc

Academic year 2017-2018

In remembrance of Jean-Bapti

Abstract

Water supply remains a major issue in several countries. When desig
a water supply network optimality is a priority. The aim of this proje
to find optimal network structures using automation and machine lear
The development process is divided into several stages. During the
stage, network topology has been studied. A network has been desi
using our software on real-world data.

Next stages will involve adding further parameters such as water vel
or pressure to the existing model. The project has a multidisciplinary
ture. Using geographical data requires a certain level of acquaintance
different formats and software such as QGis. On the other hand, maste
a programming language like Python is required to implement the diffe
algorithms and libraries.

Acknowledgements

Ringrazio ....

Chapter 1

Introduction

L'introduzione deve essere atomica, quindi non deve contenere nè sottose
nè paragrafi nè altro. Il titolo, il sommario e l'introduzione devono semb
delle scatole cinesi, nel senso che lette in quest'ordine devono progres
mente svelare informazioni sul contenuto per incatenare l'attenzione de
tore e indurlo a leggere l'opera fino in fondo. L'introduzione deve e
tripartita, non graficamente ma logicamente:

1.1 General context

La prima parte contiene una frase che spiega l'area generale dove si sv
il lavoro; una che spiega la sottoarea più specifica dove si svolge il la
e la terza, che dovrebbe cominciare con le seguenti parole Ïo scopo
tesi è . . .; illustra l'obbiettivo del lavoro. Poi vi devono essere una o
frasi che contengano una breve spiegazione di cosa e come è stato f
delle attività sperimentali, dei risultati ottenuti con una valutazione e
sviluppi futuri. La prima parte deve essere circa una facciata e mezza o

1.2 Motivation and Objectives

1.3 Brief description of the approach

La seconda parte deve essere una esplosione della prima e deve q
mostrare in maniera più esplicita l'area dove si svolge il lavoro, le
bibliografiche più importanti su cui si fonda il lavoro in maniera sint

future di ricerca, quali sono i problemi aperti e quali quelli affrontati ripete lo scopo della tesi. Questa parte deve essere piena (ma non grond come la sezione due) di citazioni bibliografiche e deve essere lunga cir facciate.

1.4 Structure of the reporto

La terza parte contiene la descrizione della struttura della tesi ed è orga zata nel modo seguente. “La tesi è strutturata nel modo seguente.

Nella sezione due si mostra ...

Nella sez. tre si illustra ...

Nella sez. quattro si descrive ...

Nelle conclusioni si riassumono gli scopi, le valutazioni di questi prospettive future ...

Nell’appendice A si riporta ... (Dopo ogni sezione o appendice ci v un punto).”

I titoli delle sezioni da 2 a M-1 sono indicativi, ma bisogna cerca mantenere un significato equipollente nel caso si vogliano cambiare. Qu sezioni possono contenere eventuali sottosezioni.

Chapter 2

State of the art

Nella seconda sezione si riporta lo stato dell'arte del settore, un inqua
mento dell'area di ricerca orientato a portare il lettore all'interno della p
lematica affrontata. Bisogna dimostrare di conoscere le cose fatte fin
ora in questo campo e il perché si sia reso necessario lo svolgimento di qu
lavoro. Questa sezione deve essere grondante di citazioni bibliografiche

Chapter 3

Approach

3.1 Root architecture

Biologically inspired models can provide interesting insights. Organ
that have gone through several rounds of evolutionary selection seem t
able to deliver efficient and nearly-optimal solutions. The use of such mo
seems to have produced satisfactory results for transport networks.

Reading Chloé Arson's presentation on bio-inspired geomechanics, we
covered the potential advantage of using root system architecture to de
water lines. Prof. Arson conducted an experiment to compare the pr
tions of a root growth model with real water line networks. Root gro
is a gene-controlled phenomenon. Therefore, different species may pre
different growth patterns. In addition, soil structure has also an influ
on root structures. For example the presence of physical obstacles,
as boulders, alters geotropic growth. Prof. Arson also pointed out th
rocky soil would require a different model. Other characteristics like w
and nutrient gradients or bacteria play a key role in root growth.
Arson's experiment consisted in growing roots on a scale plastic mod
the Georgia Tech campus. The results would allow to validate the a
racy of the mathematical model. Afterwards they could be compare
the existing water network and thus assess its efficiency. Prof. Arson
introduced leaf venation systems which bear certain resemblance to w
line networks. Indeed, the growth of a leaf is governed by the presenc
auxin (plant hormones) sources which can be seen as the nutrient sourc
the root model.

plex relationship between soil structure and soil biological activity.
is a habitat for many organisms and is also responsible for the mover
and transport of resources which are necessary for their survival. Thr
their roots, plants play a key role in many soil processes. Soil prope
affect root growth which in turn affects resource acquisition and ther
the plant's impact on its environment (soil). Interest for root system
chitecture comes from the necessity in agriculture of increasing product
and minimizing water and nutrient losses. A good understanding of
processes seems necessary to achieve this end. Moreover, Pierret p
out that whereas soil biological and chemical processes have been care
studied, physical processes need more attention. The article examines
biological factors that influence soil processes. It underlines the com
interactions between physical and chemical-biological processes and the
possibility to treat them separately. According to Pierret, roots are esse
to study this complexity. In the second part of the article, the huge d
sity of root classes is examined. This implies the necessity of using spe
models for each species. The last part of the article discusses how mode
can provide clearer insights on the interactions between roots and soil.

Lionel Dupuy's article describes the evolution of root growth models.
first models appeared in the early 1970s and focused mainly on root le
However since the 1990s new complex models have emerged thanks to
use of more powerful computers. phenomenon has been fostered by the
for predictive technologiesät different scales. Dupuy suggests a new t
retical framework which takes into account individual root developm
parameters. He introduces "equations in discretized domains that de
as a result of growth". Simulations conducted by Dupuy have reve
some patterns in what seemed a complex and heterogeneous problem.
precisely, it seems that roots develop following travelling wave patter
meristems.

V. M. Dunbabin also mentions the progress accomplished in the area of
growth modelling. The early models did not take into account the
growth in response to a heterogeneous soil environment. Nowadays, m
must include soil properties and accurate descriptions of plant function.
aim of these simulations is again to provide a better understanding o
efficient acquisition of water and nutrients by plants. Resource availab

to soil properties. This characteristic allows the plant to forage with r
precision and reduce metabolic cost. Three-dimensional models are ab
seize the complexity of the problem. Previous models were rather si
and relied upon one-dimensional functions of rooting depth vs. time.

One of the most interesting articles is Atsushi Tero's "Rules for Bio
cally Inspired Adaptive Network Design". In order to solve the proble
transport networks efficiency, Tero created a mathematical model base
organisms that build biological networks. He explains that these biolo
networks have been honed by many rounds of evolutionary selection
that they can provide inspiration to design new networks. He praises
good balance between cost, transport efficiency and, above all, fault t
ance. One of such organisms is physarum polycephalum, a type of s
mold. Tero let physarum grow on a map of the Tokyo area where m
cities were marked by food sources. A first network was obtained. In o
to improve the results, the experiment was carried out a second time. I
ever, illumination was used to introduce the real geographical constr
such as coastlines or mountains (illumination reduces physarum's grov
The results were very satisfactory and the biological network was very s
lar to the existing Tokyo transport network. Tero developed a mathema
model that tried to reproduce Physarum's behavior. The principle o
model is that tube thickness depends on the internal flow of nutrients.
a high rate tends thickens a tube and a low rate leads to its decay. As sh
by Prof. Arsons' paper "Bio-inspired fluid extraction model for rese
rocks", slime mold growth can also be used to study the flow in a po
medium. The use of Root Architecture Models was abandoned in ord
investigate the use of Machine Learning, more specifically, Artificial N
Networks.

3.2 Artificial neural networks

The growth of network usage and their increasing complexity, in partic
for communication technologies application, drives towards the impr
ment of routing technique. One track of this research is the developme
s̈martẗechniques for network design and management.

For our project we chose to follow this direction, combine sub-optima
algorithms to develop a possibly innovative solution. Lead by example

AI is applied to many complex routing problems: one example is very l
scale integration (VLSI). The process of designing integrated circuits is
due to the large number of often conflicting factors that affect the ro
quality such as minimum area, wire length. Rostam Joobbani, a knowle
based routing expert from Carnegie-Mellon University (1986), proved
an AI approach to the subject could dramatically improve performanc

A more recent example is the use of AI in Wireless Sensor Networks. W
are spatially distributed autonomous sensors to monitor physical or
ronmental conditions, such as temperature, sound, pressure, etc. an
cooperatively pass their data through the network to other locations.
agement of those networks is particularly challenging because of the dyn
environmental conditions. J. Barbancho and al. (2007) wrote a review a
the use of artificial intelligence techniques for WSNs for path discovery
other purposes. The study shows the potential of Artificial Neural Netw

Artificial Neural Networks learn to do tasks by considering examples, g
ally without task-specific programming. An ANN is based on a collecti
connected units called artificial neurons. Each connection (synapse) bet
neurons can transmit a signal to another neuron. The receiving (postsy
tic) neuron can process the signal(s) and then signal downstream ne
connected to it.

N. Ahad, J. Quadir, N. Ahsan (2016) published a review focused on
niques and applications of artificial neural networks for wireless netw
The advantage of using ANN is that can make the network adaptive
able to predict user demand.

Concerning shortest path problems Michael Turcanik (2012) used a Ho
neural network as a content-addressable memory for routing table
up. A routing table is a database that keeps track of paths in a netw
Whenever a node needs to send data to another node on a network, it
first know where to send it. If the node cannot directly connect to
destination node, it has to send it via other nodes along a proper rou
the destination node. Most nodes do not try to figure out which rou
might work; instead, a node will send the message to a gateway in the
area network, which then decides how to route the p̈ackageöf data to
correct destination. Each gateway will need to keep track of which

This excursus gives an idea of the incredibly various applications of A
in routing problems. We would like now to focus on the use of Hop
Neural Network, which is the most classical solution for routing prob
with ANN's.

Hopfield (1984) proposed the use of his algorithm to give heuristic solu
to the travel salesman problem. TSP is a well known NP-hard minin
tion problem. As defined by Karl Menger the TSP is "the task to
for finitely many points whose pairwise distances are known, the sho
route connecting the points". So having n cities, our travel salesman h
associate to each city X a position k in the tour so that:

$$\sum_{X}\sum_{Y\neq X}\sum_{j} d_{XY}y_{Xj}(y_{Y,j+1}+y_{Y,j-1})$$

is minimal, where dXY is the distance between city X and Y.

E. Wacholder and al. (1989) developed a more efficient implement
of the Hopfield NN for the travel sales-man problem. The algorithm
successfully tested on many problems with up to 30 cities and five sales
while a non-optimized brute-force approach would take billions of billio
years to return. In all test cases, the algorithm always converged to
solutions.

Mustafa K. Mehmet Ali and Faouzi Kamoun (1993) considered mod
shortest path problem with Hopfield Neural Network for the first time.
researchers asserted that HNN can find shortest path effectively and s
times it would be better to use such a network instead of classic algori
such as Dijkstra.

We will now explain what Hopfield Neural Networks are, with parti
attention to the TSP application, although the definition we will gi
general. It is a recurrent ANN, as opposed to feed forward NN, w
means neurons interconnections forms a directed cycle, so neurons are
input and output. Hopfield nets are sets n2 nodes where X [1, n] and
n] and the state is characterized by the binary activation values y = (
of the nodes. A TSP problem with n cities can be modeled as an Hop
net of dimension n2, where yXj is 1 if the city X is in the k-position o

$$s_k\left(t+1\right) = \sum_{j\neq k} y_i\left(t\right) w_{jk} + \theta_k$$

where wjk is the weight of the connection between j and k and is the bias forward function is applied to the node input to obtain the new activa value at time t+1:

$$y_k\left(t\right) = sgn(s_k(t-1))$$

The energy function is as follow so that the optimal solution will mini it:

$$E = \frac{A}{2}\sum_X\sum_j\sum_{k\neq j} y_{Xj}y_{Xk} + \frac{B}{2}\sum_j\sum_X\sum_{X\neq Y} y_{Xj}y_{Yj} + \frac{C}{2}(\sum_X\sum_j y_{Xj} -$$
$$+\frac{D}{2}\sum_X\sum_Y\sum_j d_{XY}y_{Xj}(y_{Y,j+1}+y_{Y,j-1})$$

The first two terms are null if and only if the there is a maximum o active neuron for each row and column respectively. The third term is if and only if there are n active neurons. The last term takes in accoun distance of the path, that should be minimized as well.

The Hebbian rule to update the weights is deduced from the energy func

$$w_{Xj,Yk} = \ -A\delta_{XY}\left(1-\delta_{jK}\right) - B\delta_{jk}\left(1-\delta_{XY}\right) - C - Dd_{XY}(\delta_{k,j+1}+\delta_{k}$$

where kj = 1 if j = k and zero otherwise. As in the energy functio first term inhibits connection within each row, the second within colu the third is the global inhibition and the last term takes into accoun distance between the cities.

Under the hypothesis $w_{Xj,Yk} = w_{Yk,Xj}$ the method can be proved to stable points. At each iteration the net updates his parameters accordi the Hebbian rule and the evolution of the state can be proved to be m tonically nonincreasing with respect of the energy function. Performing a gradient descent, after a certain number of repetition the state con to a stable point that is a minima of the energy function.

the ANN will deliver a solution that will converge and whether this solution is optimal. Thus ANNs as a poorly efficient way to solve the problem. Nonetheless, machine learning can still be useful. Indeed, the interpretation of geographical information files can be done through clustering algorit

Chapter 4

Progetto logico della soluzione del problema

4.1 Problem staterment and modelisation

The overall idea of our software solution is to connect water sources consumers to a network of pipes in the most efficent way possible. In the definition of the best net is a key problem. Factors that make aqueduct the optimal one are not easily modeled. We can suppose variables such length, height, water speed and pressure, viscosity ecc sh be taken into account.

For our first approach, we decided to consider only the pipeline length so we simplify acqueduct design to a classical routing problem. On this we will then be able to add complexity. We can now more formaly d our poblem. Being a topography a graph representing the meshed su of a region, the problem of designing an acqueduct as the one of finding recovering-graph on the topograpy graph connecting water consumers sources. We will use the euclidian metric on a space with tree dimensi

4.2 Data reading

In **??** is shown to the flow chart of our software. The input is comp of three geographical files: mesh represent the topology of the studie gion, source-sinks contains the locations for each water source and cons

fer to 5.1 Reading those input two graphs can be initialised: the topogr
and the aqueduct describing graphs. The first is a graph where nodes r
sent positions and edges the possible transitions between them. Transi
on roads will be preferable. The aqueduct graph is initialised with so
and sinks as nodes and no edges. An insight of the data structure we
to represent graph is given in section 5.2

4.3 Clustering

Running classical algorithms such a brute-force TSP or a minimum s
ning tree to link the nodes in the sink-source graph would not be fea
for computational reasons. Thanks to the particular nature of our pro
a simplification is possible: we divide the aqueduct system in two la
adduction and distribution nets. The adduction layer brings water from
source to the inhabited areas whereas the distribution segment is in ch
of the "last kilometre" distribution. This two layer solution is comm
used in aqueduct design and network design in general: internet is an e
ple. To achieve this need to recognise group of buildings such as villag
neighbourhoods. Those sets are called sink clusters. This approach c
multiple advantages. From the computational point of view reduce th
mension of the sets on which we run the routing algorithms. On the o
side, once the two layers are identified we can use different strategi
connect the nodes, as we will se in the next paragraph. Efficient imple
tations of clusterings algorithms are provided in scikit-learn. Scikit-lear
well-known machine learning library for Python and it features various
sification, regression and clustering algorithms. After this operation s
are labeled as part of they respective cluster. A more detailed explan
can be found at 5.3.

4.4 Routing

We now can design the water systems connecting the sinks. Let's first
sider the distribution layer, which is to say the problem of connecting
sinks of a cluster. This operation is broken down in two tasks. First,
all the paths connecting sinks, than chose the smallest recovering grap
the smallest aqueduct satisfying the specifics. To find the path conne

to the fact that edges on the aqueduct graph are paths on the topogr graph. This operation creates a complete graph for each cluster. Note the set of those graph is a partition of the aqueduct graph. The second s consists in eliminating the redundant edges: to do this we run the Kr algorithm and calculate the minimum spanning tree. For more inform read section 5.3. An other approach, favoured in classical aqueducts d would be to calculate a partially connected graph where certain node connected to exactly one other node whereas others are connected to t more other nodes. This makes possible to have some redundancy wi the expense and complexity required for a connection between every no the network. At this first stage of the project this part is left for furth vestigation. Considering now the adduction system a very similar app can be used: the clusters should be interlinked and connected to a so Each distribution network, as is a tree, has a root node. We can init the adduction graph with all the cluster's roots nodes and the water so Finally this graph is connected with the technique previously used.

Chapter 5

Technical aspects

5.1 Geographical Data

The overall idea is to take maps and automatically trace an aqueduct c
in order to do that, we start from the map's shapefile. Shapefile is a pop
geospatial vector data format for geographic information systems softw
It spatially describes geometries: points, polylines and polygons. These
example, could represent water wells, roads or buildings. As those prim
geometrical data types come without any attributes to specify what
represent, a table of records to store attributes is provided. Websites
osm2shp or Geofabrik provide an immense database of shapefiles avai
for download. Moreover desktop software like Qgis provides shapefile
ing tools. This way we can both download real-world maps and create
own. Then through Qgis' meshing plug-in Gmsh we can mesh the sur
of the map and export the result in vtk format as seen in Fig. 3.1 How
shapefiles seldom have information on the elevation (that is the Z co
nate) of the objects they represent. It is therefore necessary to use and
format: the Digital Elevation Model (DEM). Digital Elevation Models
vide this missing piece of information that can subsequently be adde
the shapefile's attribute table. DEMs can be converted into meshes th
to software such as SAGA. Meshes saved as vtk files can easily be use
Python. Vtk files are a simple and efficient way to describe mesh-like
structures. The vtk file boils down to those two elements: points and c
Points have 3D coordinates while cells are surfaces, expressed by the p
delimiting them. Point and cell data (scalar or vector) can also be assig

to use, widespread programming language, good for rapid prototyping
rich in package and libraries. The problem is divided in two main t
modelling the data structure that represents the graph and the algorit
part, the aqueduct design.

5.2 Data Structure

To implement the data-structure we chose to use NetworkX. NetworkX
Python package for the creation, manipulation, and study of complex
works. The package provides classes for graph objects, generators to c
standard graphs, IO routines for reading in existing datasets, algorithm
analyze the resulting networks and some drawing tools. The software t
as input two shape files: the first describes the topology, the second
source and sinks. The topology is either a mesh, representing the geogr
of the region or a polyline with just the road network of the region.
roads are particularly important because aqueducts are built along road
logistical reasons. The second file is a polygon file containing the build
that should be served by the aqueduct and the water sources. From t
data, a first graph is obtained. The graph has as nodes the points descr
in topology file plus the buildings. The coordinates of binding-represe
nodes are the average of the coordinates that also have the metadat
sociated. The edges are the edges described in the topology file plus
edges connecting the building to the nearest node of the network in o
to obtain a connected graph.

5.3 Clustering

In this section we will explain the different routing techniques used.
Dijstrak
TSP

Chapter 6

Presentation and validation of experimental results

Si mostra il progetto dal punto di vista sperimentale, le cose materialm realizzate. In questa sezione si mostrano le attività sperimentali svolt illustra il funzionamento del sistema (a grandi linee) e si spiegano i r tati ottenuti con la loro valutazione critica. Bisogna introdurre dati complessità degli algoritmi e valutare l'efficienza del sistema.

Chapter 7

Conclusions

7.1 Summary of Thesis Achievements

Si mostrano le prospettive future di ricerca nell'area dove si è svolto il lav
Talvolta questa sezione può essere l'ultima sottosezione della preced
Nelle conclusioni si deve richiamare l'area, lo scopo della tesi, cosa è s
fatto,come si valuta quello che si è fatto e si enfatizzano le prospettive fu
per mostrare come andare avanti nell'area di studio.

7.2 Applications

7.3 Future wok

Appendix A

Documentazione del progett
logico

Documentazione del progetto logico dove si documenta il progetto logic
sistema e se è il caso si mostra la progettazione in grande del SW e dell'
Quest'appendice mostra l'architettura logica implementativa (nella Se
4 c'era la descrizione, qui ci vanno gli schemi a blocchi e i diagrammi).

Appendix B

Documentazione della programmazione

Documentazione della programmazione in piccolo dove si mostra la strut
ed eventualmente l'albero di Jackson.

listings

Appendix C

Listato

Il listato (o solo parti rilevanti di questo, se risulta particolarmente es
con l'autodocumentazione relativa.

```
# -*- coding: utf-8 -*-

import networkx as nx
import math
import sys

# sys.setdefaultencoding('utf8')

"""
Created on Mon Dec  4 22:57:30 2017

@author: Conrad
"""


class Router(object):
    #
    ---------------------------------------------------------------------
    # -- CLASS ATTRIBUTES
    #
    ---------------------------------------------------------------------

    # Class description
    CLASS_NAME = "Router"
    CLASS_AUTHOR = "Marcello Vaccarino"
```

```
    acqueduct = nx.Graph()
    #
    ----------------------------------------------------------------
    # -- INITIALIZATION
    #
    ----------------------------------------------------------------

    def __init__(self, topo_file=None, building_file=None):

        if topo_file != None and building_file != None:
            try:
                # [TODO] this function does not read building
    but single points
                self.read_shp(topo_file, building_file)
            except Exception as e:
                raise e
        elif building_file != None:
            try:
                self.read_shp_bilding(building_file)
            except Exception as e:
                raise e
        elif topo_file != None:
            try:
                self.read_vtk(topo_file)
            except Exception as e:
                raise e

    #
    ----------------------------------------------------------------
    # -- CLASS ATTRIBUTES
    #
    ----------------------------------------------------------------

    def avg(self, node_list):
        x = 0
        y = 0
        for node in node_list:
            x += node[0]
            y += node[1]
        x /= len(node_list)
        y /= len(node_list)
        return (x, y)
```

```
            import shapefile
        except ImportError:
            raise ImportError("read_shp requires pyshp")

        sf = shapefile.Reader(file_name)
        fields = [x[0] for x in sf.fields]

        for shapeRecs in sf.iterShapeRecords():
            shape = shapeRecs.shape
            for cell in self.row_chuncker(shape.points, shape
    parts):
                center_coord = self.avg(cell)
                attributes = dict(zip(fields, shapeRecs.recor
                attributes['pos'] = center_coord
                self.graph.add_node(center_coord)

    # chuncker: see commpresed row storage
    def row_chuncker(self, array, p_array):
        p_array.append(len(array))
        return [array[p_array[i]:p_array[i+1]]
                for i in range(len(p_array)-1)]

    def read_shp(self, file_name, point_file=None):
        """Generates a networkx.DiGraph from shapefiles. Poin
    geometries are
        translated into nodes, lines into edges. Coordinate
    tuples are used as
        keys. Attributes are preserved, line geometries are
    simplified into
        start and end coordinates. Accepts a single shapefile
     directory of
        many shapefiles.

        "The Esri Shapefile or simply a shapefile is a popula
    geospatial
        vector data format for geographic information systems
    software."

        Parameters
        ----------
        path : file or string
           File, directory, or filename to read.

        simplify:  bool
            If ``True``, simplify line geometries to start an
    end coordinates
```

```
repeated for each
        edge comprising that feature.

    Returns
    -------
    G : NetworkX graph

    Examples
    --------
    >>> G=nx.read_shp('test.shp') # doctest: +SKIP

    References
    ----------
    .. [1] http://en.wikipedia.org/wiki/Shapefile
    """
    try:
        import shapefile
    except ImportError:
        raise ImportError("read_shp requires pyshp")

    sf = shapefile.Reader(file_name)
    fields = [x[0] for x in sf.fields]

    for shapeRecs in sf.iterShapeRecords():

        shape = shapeRecs.shape
        if shape.shapeType == 1:    # point
            attributes = dict(zip(fields, shapeRecs.recor
            attributes["pos"] = shape.points[0]
            self.graph.add_node(shape.points[0], attribut

        if shape.shapeType == 3:    # polylines
            attributes1 = dict(zip(fields, shapeRecs.reco
)
            attributes2 = dict(zip(fields, shapeRecs.reco
)
            for i in range(len(shape.points) - 1):
                '''
                attributes1["pos"] = shape.points[i]
                n1 = self.add_node_unique(shape.points[i]
attributes1)
                attributes2["pos"] = shape.points[i + 1]
                n2 = self.add_node_unique(shape.points[i
1], attributes2)
                attribute = {'dist': self.distance(n1, n2
```

```
                    self.graph.add_node(n1, attributes1)
                    attributes2["pos"] = shape.points[i + 1]
                    n2 = shape.points[i + 1]
                    self.graph.add_node(n2, attributes2)
                    attribute = {'dist': self.distance(n1, n2
                    self.graph.add_edge(n1, n2, attribute)

            if shape.shapeType == 5:    # polygraph
                # chuncker: see commpresed row storage
                def chuncker(array, p_array):
                    p_array.append(len(array))
                    return [array[p_array[i]:p_array[i+1]]
                            for i in range(len(p_array)-1)]

                # given a cell returns the edges (node touple
implicitely defined in it
                def pairwise(seq):
                    return [seq[i:i+2] for i in range(len(seq
-1)]

                for cell in chuncker(shape.points, shape.part
                    for n1, n2 in pairwise(cell):
                        attributes1 = dict(zip(fields, shapeR
.record))
                        attributes1["pos"] = n1
                        attributes2 = dict(zip(fields, shapeR
.record))
                        attributes2["pos"] = n2
                        # add nodes of the shape to the graph
                        n1 = self.add_node_unique(n1,
attributes1)
                        n2 = self.add_node_unique(n2,
attributes2)
                        attribute = {'dist': self.distance(n1
n2)}
                        # add edge
                        self.graph.add_edge(n1, n2, attribute

        if point_file != None:
            sf = shapefile.Reader(point_file)
            new_fields = [x[0] for x in sf.fields]
            nodes2attributes = {node: data \
                                for node, data in self.graph.
nodes(data=1)}
            for shapeRecs in sf.iterShapeRecords():
                shape = shapeRecs.shape
```

```
                new_attributes = dict(zip(new_fields, shapeRe
record))
                nodes = [tuple(point) for point in shape.poin
                for node in nodes:
                    # print node in nodes2attributes,
new_attributes
                    nodes2attributes[node].update(new_attribu
)
            for node, data in self.graph.nodes(data=1):
                data.update(nodes2attributes[node])
            # nx.set_node_attributes(self.graph,
nodes2attributes)

    def write2shp(self, G, filename):
        try:
            import shapefile
        except ImportError:
            raise ImportError("read_shp requires pyshp")

        w = shapefile.Writer(shapeType=3)
        #w.field("DC_ID", "LENGHT", "NODE1", "NODE2", "DIAMET
", "ROUGHNESS", "MINORLOSS", "STATUS", "C")

        w.fields = [("DeletionFlag", "C", 1, 0), ["DC_ID", "N
9, 0],
            ["LENGHT", "N", 18, 5], ["NODE1", "N", 9, 1], ["
NODE2", "N", 9, 0],
            ["DIAMETRE", "N", 18, 5], ["ROUGHNESS", "N", 18,
 ["MINORLOSS", "N", 18, 5],
            ["STATUS", "C", 1, 0]]

        i = 0
        lenghts = nx.get_edge_attributes(G, 'dist')
        print(lenghts)
        for edge in lenghts:
            line = [edge[0], edge[1]]
            w.line(parts=[line])
            print (edge[0], edge[1])
            w.record(i, lenghts[(edge[0], edge[1])],
                    1, 2, 100, 0.1, 0, "1")
            i+=1
        w.save(filename)

    def write2vtk(self, G):

        # import sys
```

```
        points = [list(node) for node, data in G.nodes(data=T
    )]
        line = []
        for edge in G.edges():
            for i, node in enumerate(G.nodes()):
                if node == edge[0]:
                    n1 = i
            for i, node in enumerate(G.nodes()):
                if node == edge[1]:
                    n2 = i
            line.append([n1,n2])

        vtk = pyvtk.VtkData(pyvtk.UnstructuredGrid(points, li
    line))
        vtk.tofile('example1', 'ascii')

    def add_node_unique(self, new_node, new_attributes):
        """
        grants that the node added is unique with respect to
     pos
        attribute equality relationship
        """
        for node in self.graph.nodes(True):
            if node[1]["pos"] == new_attributes["pos"]:
                return node[0]
        self.graph.add_node(new_node, new_attributes)
        return new_node

    def read_vtk(self, file_name):
        import numpy as np
        try:
            from mesh import Mesh
        except ImportError:
            raise ImportError("read_vtk requires pymesh")

        # initialize the vtk reader
        reader = Mesh()

        # read the vtk
        reader.ReadFromFileVtk(file_name)

        # add nodes to the graph
        for index, node in enumerate(reader.node_coord):
            self.graph.add_node(index, pos=reader.node_coord[
    index])
```

```
Row Storage)
    dont voici une illustration :
    Soient six n uds num rotes de 0    5 et quatre    el
 ements   form   es   par
    les n uds (0, 2, 3) pour l' el ement 0, (1, 2, 4) po
l'element,
    (0, 1, 3, 4) pour  l    el    e − ment 2 et (1, 5) pou
 l    el    ement   3.
    Deux tableaux sont utilis   es ,  l un  pour stocker d
fa  con  contigu  e
    les listes de n uds qui composent les   el    ements
table 1),   l autre
    pour indiquer la position, dans ce tableau, ou‘ comme
 chacune de
    ces listes (table 2).
    Ainsi, le chiffre 6 en position 2 dans le tableau p
elem2node indique
    que le premier n ud de  l    el    ement  2 se trouve
 position 6 du
    tableau elem2node. La derni‘ere valeur dans p elem2no
correspond au
    nombre de cellules (la taille) du tableau elem2node.

    elem2node
    0 | 2 | 3 | 1 | 2 | 4 | 0 | 1 | 3 | 4 | 1 | 5
    1   2   3   4   5   6   7   8   9   10  11  12
    ^       ^           ^               ^       ^


    p_elem2node
    0 | 3 | 6 | 10 | 12
    1   2   3    4    4
    '''

    def chuncker(array, p_array):
        return [array[p_array[i]:p_array[i+1]]
                for i in range(len(p_array)-1)]

    # given a cell returns the edges implicitely defined
it
    def pairwise(seq):
        return [seq[i:i+2] for i in range(len(seq)-2)] +
            [[seq[0], seq[len(seq) -1]]]

    datas = np.asarray([data['pos']
                        for _, data in self.graph.nodes(d
=True)])
```

```
            yi = datas[u][1]
            zi = datas[u][2]
            xj = datas[v][0]
            yj = datas[v][1]
            zj = datas[v][2]
            return math.sqrt((xi-xj)*(xi-xj) +
                             (yi-yj)*(yi-yj) + (zi-zj)*(zi-zj

        # add edges to the graph
        for cell in chuncker(reader.elem2node, reader.
    p_elem2node):
            for u, v in pairwise(cell):
                if u not in self.graph[v]:
                    self.graph.add_edge(u, v, weight=distance3
    u, v, datas))

    def distance(self, nodei, nodej):
        xi = nodei[0]
        yi = nodei[1]
        xj = nodej[0]
        yj = nodej[1]
        if len(nodei) == 3 and len(nodej) == 3:
            zi = nodei[2]
            zj = nodej[2]
            return math.sqrt((xi-xj)*(xi-xj) + (yi-yj)*(yi-yj
                             (zi-zj)*(zi-zj))
        return math.sqrt((xi-xj)*(xi-xj)+(yi-yj)*(yi-yj))

    # cartesian norme in 2D
    def distance2D(self, nodei, nodej):
        xi = self.graph.nodes(data=True)[nodei][1]['pos'][0]
        yi = self.graph.nodes(data=True)[nodei][1]['pos'][1]
        xj = self.graph.nodes(data=True)[nodej][1]['pos'][0]
        yj = self.graph.nodes(data=True)[nodej][1]['pos'][1]
        return math.sqrt((xi-xj)*(xi-xj)+(yi-yj)*(yi-yj))

    # cartesian norme in 3D
    def distance3D(self, nodei, nodej):
        xi = self.graph.nodes(data=True)[nodei][1]['pos'][0]
        yi = self.graph.nodes(data=True)[nodei][1]['pos'][1]
        zi = self.graph.nodes(data=True)[nodei][1]['pos'][2]
        xj = self.graph.nodes(data=True)[nodej][1]['pos'][0]
        yj = self.graph.nodes(data=True)[nodej][1]['pos'][1]
        zj = self.graph.nodes(data=True)[nodej][1]['pos'][2]
        return math.sqrt((xi-xj)*(xi-xj) + (yi-yj)*(yi-yj) +
    -zj)*(zi-zj))
```

```
        coord2D = {}
        for key, value in nx.get_node_attributes(G,
                                                         'pos').
    iteritems():
            coord2D[key] = [value[0], value[1]]
        return coord2D

    # display the mesh using networkx function
    def display_mesh(self):
        nodelist = []
        node_color = []
        for node in self.graph.nodes(data=1):
            node_type = node[1]['FID']
            if not node_type == '':
                nodelist.append(node[0])
                node_color.append('r' if node_type == 'sink'
    else 'b')
        try:
            nx.draw_networkx(self.graph, pos=self.coord2D(),
    nodelist=nodelist,
                                 with_labels=0, node_color=
    node_color)
        except:
            pass

    # display the mesh with a path marked on it
    def display_path(self, path):
        nodelist = []
        node_color = []
        for node in self.graph.nodes(data=1):
            node_type = node[1]['FID']
            if not node_type == '':
                nodelist.append(node[0])
                node_color.append('r' if node_type == 'sink'
    else 'b')

        color = {edge: 'b' for edge in self.graph.edges()}
        # returns an array of pairs, the elements of seq two
    two
        def pairwise(seq):
            return [seq[i:i+2] for i in range(len(seq)-2)]
        # colors the edges
        for u, v in pairwise(path):
            if (u, v) in color:
                color[(u, v)] = 'r'
            if (v, u) in color:
```

```
importatant!
        array = []
        for edge in self.graph.edges():
            array += color[edge]
        nx.draw_networkx(self.graph, pos=self.coord2D(),
                                    nodelist=nodelist, with_labels
                                    node_color=node_color, edge_co
=array)

    def shortest_path(self, node1, node2):
        """
        Calculates the shortest path on self.graph.
        Path is a sequence of traversed nodes
        """
        try:
            path = nx.shortest_path(self.graph, source=node1,
target=node2,
                                            weight="weight")
        except:
            pass
        return path

    def path_lenght(self, path):
        """
        given a path on the graph returns the lenght of the p
 in the
        unit the coordinats are expressed
        """
        if path is None:
            return float("inf")
        lenght = 0.0
        # given the path (list of node) returns the edges
contained
        path_edges = [path[i:i+2] for i in range(len(path)-2)
        # itereate to edges and calculate the weight
        for u, v in path_edges:
            lenght += self.distance(u, v)
        return lenght

    def TSP(self, cities):
        '''
        declaring the adjacency matrix
        T = numpy.empty(shape=(len(cities),len(cities)))
        for u, i in cities:
            for v, j in itertools(cities):
                uv_path = shortest_path(u, v)
```

```
            paths = []
            for combo in itertools.permutations(range(1,len(T
    [0]))):
                lenght = 0
                prev = 0
                path = []
                path += [0]
                for elem in combo:
                    lenght += T[prev][elem]
                    prev = elem
                    path += [elem]
                lenght += T[combo[len(combo) -1]][0]
                path += [0]
                paths.append((path, lenght))
            stop = timeit.default_timer() # stop timer
            time = stop - start
            return paths, time
        '''

    def is_sourcesink(self, node):
        '''given a node as in the networkx.Graph.nodes(data=1
        returns 1 if the node is a sink or a source, 0 elsewh
    '''
        if not node[1]['FID'] == '':
            return 1
        return 0

    def compute_source_matrix(self):
        for node in self.graph.nodes(data=1):
            if self.is_sourcesink(node):
                self.sinksource_graph.add_node(node[0], node[

        for n1 in self.sinksource_graph.nodes():
            for n2 in self.sinksource_graph.nodes():
                if n1 is not n2:
                    path = self.shortest_path(n1, n2)
                    if path is not None:
                        self.sinksource_graph.add_edge(n1, n2
                                                 {'dist': self.
    path_lenght(path),
                                                  'path': path})

    def design_minimal_aqueduct(self, G):
        minimal = nx.minimum_spanning_tree(G, weight='dist')
        return minimal
```

```
            path += edge[2]['path']
        self.display_path(path)

    def complete_graph(self, G):
        for n1 in G.nodes():
            for n2 in G.nodes():
                if n1 != n2:
                    # attributes = {'path': [n1, n2], 'dist':
self.distance(n1,n2)}
                    G.add_edge(n1, n2)
                    G.edges[n1, n2]['dist'] = self.distance(n
n2)
                    G.edges[n1, n2]['path'] = [n1, n2]

    def mesh_graph(self, G, weight):
        """complexity (len(G.nodes))^3"""
        distances = nx.get_edge_attributes(G, weight)
        # condition to create the gabriel relative neighbour
graph
        def neighbors(p, q):
            for r in G.nodes:
                if r != q and r != p:
                    def dist(n1, n2):
                        if (n1, n2) in distances:
                            return distances[(n1,n2)]
                        else:
                            return distances[(n2,n1)]
                    if max(dist(p,r), dist(q,r)) < dist(p,q):
                        return False
            return True

        # connect graph
        gabriel_graph = nx.Graph()
        for n1 in G.nodes():
            for n2 in G.nodes():
                if n1 != n2:
                    if neighbors(n1, n2):
                        gabriel_graph.add_edge(n1, n2)
        return gabriel_graph

    def graphToEdgeMatrix(self, G):
        node_dict = {node: index for index, node in enumerate
}

        # Initialize Edge Matrix
        edgeMat = [[0 for x in range(len(G))] for y in range(
```

```
1)
        for i, node in enumerate(G):
            tempNeighList = G.neighbors(node)
            for neighbor in tempNeighList:
                edgeMat[i][node_dict[neighbor]] = 1
            edgeMat[i][i] = 1

        return edgeMat

    def clusters(self, G):
        '''
        Finds the clusters
        '''
        # imports from a machine learning package skit-learn
        from sklearn.cluster import MeanShift,
estimate_bandwidth

        # creates a array with the 2D coordinats for each nod
        X = [[node[0], node[1]] for node in G.nodes()]
        # extimates the dimensions of single clusters
        bandwidth = estimate_bandwidth(X, quantile=0.1,
                                       random_state=0, n_jobs
        # find clustes
        ms = MeanShift(bandwidth=bandwidth)
        ms.fit(X)

        # labels is an array indicating, for each node, the
cluster number
        labels = {node: ms.labels_[i] for i, node in enumerat
.nodes())}

        # --- ADDUCTION ---
        '''
        # add cluster centers to the graph
        for node in cluster_centers:
            attribute = {'label': 'water tower', 'pos': node}
            G.add_node(node, attribute)
            attribute = {'type' : 'sink'}
            self.sinksource_graph.add_node(node, attribute)
        '''
        adduction = nx.Graph()
        cluster_centers = [(node[0], node[1]) for node in ms.
cluster_centers_]
        for node in cluster_centers:
            adduction.add_node(node)
        self.complete_graph(adduction)
```

```python
 adduction.nodes(data=True)}
        # nx.draw_networkx(adduction, pos=coord, label=False)
        self.write2shp(adduction, "adduction_network")
        self.acqueduct.add_edges_from(adduction.edges())

        # --- DISTRIBUTION ---
        # add label info to the graph
        nx.set_node_attributes(G, labels, 'label')
        # initialize distribution graphs
        distribution = [nx.Graph() for cluster in
cluster_centers]
        for node in labels:
            cluster = labels[node]
            distribution[cluster].add_node(node)
        '''
        # connect each node with his the cluster center
        node_list = []
        for index, node in enumerate(G):
            node_list.append(node)
            labels = nx.get_node_attributes(G, 'label')
            label = labels[node]
            if label is not 'water tower':
                G.add_edge(node, cluster_centers[label])
        '''
        for dist_graph in distribution:
            self.complete_graph(dist_graph)
            dist_graph = nx.minimum_spanning_tree(dist_graph,
weight='dist')
            self.acqueduct.add_edges_from(dist_graph.edges())

    def route_vesuvio(self, n1, n2):
        try:
            import shapefile
        except ImportError:
            raise ImportError("read_shp requires pyshp")
        # route
        path = self.shortest_path(n1, n2)

        # turn path into acqueduct graph
        datas = [data['pos'] for _, data in self.graph.nodes(
data=True)]
        path_coord = [tuple(datas[node]) for node in path]
        path_edges = [path_coord[i:i + 2] for i in range(len(
path_coord) - 2)]
        self.acqueduct.add_edges_from(path_edges)
```

```
            w.field("name", "C")
            line = path_edges
            w.line(parts=line)
            w.record('path')
            w.save('path')

def render_vtk(file_name):

    import vtk

    # Read the source file.
    reader = vtk.vtkUnstructuredGridReader()
    reader.SetFileName(file_name)
    reader.Update()  # Needed because of GetScalarRange
    output = reader.GetOutput()
    scalar_range = output.GetScalarRange()

    # Create the mapper that corresponds the objects of the v
    vtk file
    # into graphics elements
    mapper = vtk.vtkDataSetMapper()
    mapper.SetInputData(output)
    mapper.SetScalarRange(scalar_range)

    # Create the Actor
    actor = vtk.vtkActor()
    actor.SetMapper(mapper)

    # Create the Renderer
    renderer = vtk.vtkRenderer()
    renderer.AddActor(actor)
    renderer.SetBackground(1, 1, 1)  # Set background to whit

    # Create the RendererWindow
    renderer_window = vtk.vtkRenderWindow()
    renderer_window.AddRenderer(renderer)

    # Create the RendererWindowInteractor and display the
    vtk_file
    interactor = vtk.vtkRenderWindowInteractor()
    interactor.SetRenderWindow(renderer_window)
    interactor.Initialize()
    interactor.Start()

def tsp_example():
    return 0
```

```
def template_clustering(path_sample, eps, minpts,
    amount_clusters=None, visualize=True, ccore=False):
    sample = read_sample(path_sample);

    optics_instance = optics(sample, eps, minpts,
    amount_clusters, ccore);
    (ticks, _) = timedcall(optics_instance.process);

    print("Sample: ", path_sample, "\t\tExecution time: ", ti
    , "\n");

    if (visualize is True):
        clusters = optics_instance.get_clusters();
        noise = optics_instance.get_noise();

        visualizer = cluster_visualizer();
        visualizer.append_clusters(clusters, sample);
        visualizer.append_cluster(noise, sample, marker = 'x'
        visualizer.show();

        ordering = optics_instance.get_ordering();
        analyser = ordering_analyser(ordering);

        ordering_visualizer.show_ordering_diagram(analyser,
    amount_clusters);

def vesuvio_example():
    router = Router(topo_file="vtk/Vesuvio")
    router.route_vesuvio(32729, 31991)
    # write to vtk
    router.write2vtk(router.acqueduct)
    # render_vtk("vtk/Vesuvio")

def paesi_example():
    router = Router(building_file="geographycal_data/paesi_el
    paesi_elev")
    router.clusters(router.graph)
    router.write2shp(router.acqueduct, "acqueduct1")

def cluster_simple_example():
    import random;

    from pyclustering.cluster import cluster_visualizer;
    from pyclustering.cluster.optics import optics,
    ordering_analyser, ordering_visualizer;
```

```
    from pyclustering.samples.definitions import SIMPLE_SAMPL
    FCPS_SAMPLES;

    template_clustering(SIMPLE_SAMPLES.SAMPLE_SIMPLE1, 0.5, 3

paesi_example()
# National__Hydrography__Dataset_NHD_Points_Medium_Resolution
    National__Hydrography__Dataset_NHD_Points_Medium_Resolutio
# National__Hydrography__Dataset_NHD_Lines_Medium_Resolution/
    National__Hydrography__Dataset_NHD_Lines_Medium_Resolution
# Railroads/Railroads
# Routesnaples/routesnaples
# "shapefiles/Domain", "shapefiles/pointspoly"
# "shapeline/shapeline", "shapeline/points"
# nx.draw_networkx(router.sinksource_graph, pos=router.coord2
    , with_labels=0)
# router.design_minimal_aqueduct()
# router.display_path(path)
# nx.draw_networkx_nodes(router.graph, pos=router.coord2D())
# router.display_mesh()
# print nx.clustering(router.graph)
# print nx.floyd_warshall_numpy(router.graph, weight='dist')
# runfile('/Users/Conrad/Documents/EC/Course deuxi mme ann
    Project Inno/Projet_P5C006/router.py', wdir='/Users/Conrac
    Documents/EC/Course deuxi mme ann e/Project Inno/
    Projet_P5C006')
```

Appendix D

Il manuale utente

Manuale utente per l'utilizzo del sistema
color

Appendix E

Datasheet

Eventuali Datasheet di riferimento.