Operation Resources
IMT Atlantique

District Heating Energy Network Optimization

Professors:
Mehrdad MOHAMMADI
Patrick MEYER

Students:
Johan MEJIA
Diego CARREÑO
Tatiana MORENO

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Contents

1 Introduction

In many countries around the world, the ability to heat and supply hot water to buildings is essential, currently, several studies are being carried out in order to determine the most efficient way to do it, one of these is through a Distric heating (DH). A DH is a system for distributing heat generated in a centralized location through a system of insulated pipes for residential and commercial heating requirements such as space heating and water heating in cold areas.

District heating can increase the efficiency of fuel use, because heat generation in a large-scale cogeneration process can achieve higher electricity and heat output efficiency than local generation due to scale effects. Another side benefit of centralized heat generation is the increased flexibility, with which one can change the energy carrier for heat generation. With central heat generation, this decision requires only intervention in a small number of large facilities, instead of a roll-out of new units to every consumer [1].

District heating networks, together with water and gas networks, belong to the class of pipe networks. Pipe networks themselves are part of general infrastructure networks that range from transportation, telecommunication to electricity. Figure 1.1 illustrates a sample district heating network with one heating source. A set of buildings form a district and they are served through the pipe passing through them (or their vicinity). The heating network is branched using intersection points.

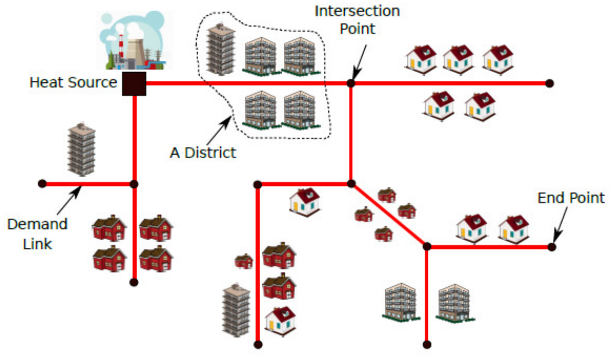


Figure 1.1: Heating district network

This paper seeks to buy the performance of PULP as an optimizer for a mixed integer linear programming (MIP) problem and an Integrated local serch (ILS) designed for a DH. The document is divided as follows, in chapter two the problem statement is found, in chapter 3 the mathematical model of the problem in general is presented, in chapter 4 the theory of design of the ILS is presented, in chapter 5 describes the implementation carried out in python of the Pulp and the ILS, in chapter 6 the results obtained from the previous implementations are presented, in chapter 7 an analysis of the results obtained is presented, and finally the conclusions obtained from the work are presented accomplished

2 Problem Statement

This problem is considered a small city in which the mayor has decided to build a centralized district heating network. As it is impossible to connect all the buildings in the city to the heating network, the mayor seeks a network that serves as much as possible the demand for heating while respecting resource constraints (heating production capacity of the heating source, capacity budget, geographical and structural limitations, etc.), the network of Figure 1.1 can be represented as a graph version as we can observe in Figure 2.1

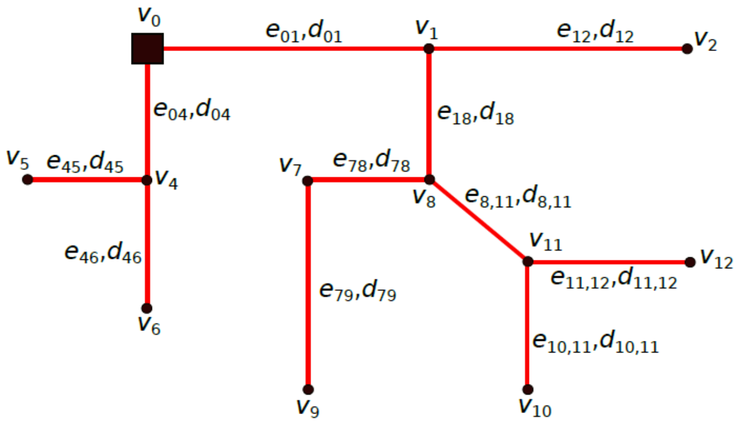


Figure 2.1: Graph representation of the heating district network

Due to technical issues in the thermal distribution network, no cycle is allowed in the network.The figure 2.2 is also called a tree. A tree is a graph in which any two vertices are connected by exactly one path.

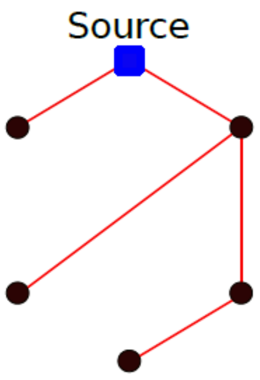


Figure 2.2: The structure where all vertices are connected to the network.

The following assumptions are taken into account:

- The location of the heating source is fixed
- There is a fixed cost for generating each unit of heat in the source
- The source power plant works for a limited hours during a year
- There is revenue for delivering each unit of heat. The revenue depends on the edge and the district
- Each edge is characterized by three attributes: peak demand (kW), annual demand (kW a) and length (m). Peak demand determines the required pipe size, while annual demand determines revenue for supplying heat
- Constructing each edge of the network consists of a fixed investment cost
- Constructing each edge of the network consists also of a variable investment cost that depends on the length of the edge
- Each constructed edge requires variable operation and maintenance costs that depend on the length of the edge
- There is an annuity factor for the investment cost
- The pipes are not ideally isolated and there are always fixed and variable thermal losses. The variable thermal losses depend on the length of the pipe
- Each edge has a capacity of heat transfer
- Unmet demands are penalized for each edge

In the chapter 3 the mathematical model is described.

3 Model

This paper presents a mixed-integer optimization program for finding the cost-optimal structure and size of a district heating network for a given city represented as a graph of its streets with to segments. Heat demand is modeled per building by peak demand (kW) and annual demand (kWh/a), which are estimated on building-level and then grouped by their nearest street.

The decision whether to build or not to build a pipeline in a given street is modeled as a binary variable, while the costs for construction scale linearly with the thermal input power into the pipe

3.1 Sets

In this model the district is represented as a graph of vertices and edges. Let V be the set of vertices v_i, corresponding to street intersections or endpoints. Set E of edges then comprises ordered tuples of vertices $e_{i,j} = (v_i, v_j)$ with $i \neq j$. In the figure 3.1 is shown an example of a district and in the table 3.1 is shown the summarize of sets

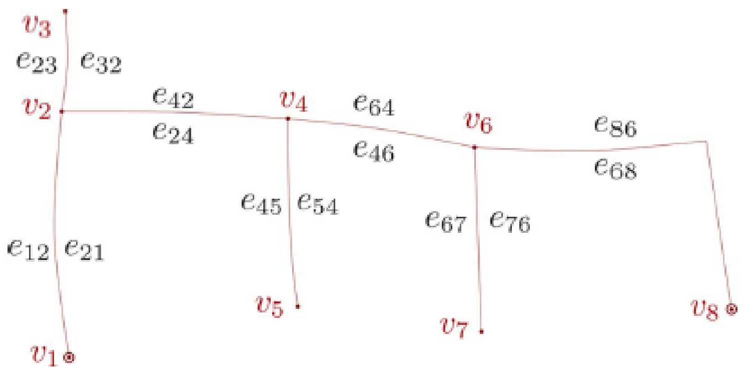


Figure 3.1: Street graph with 8 vertices, 14 edges, and 2 source vertices [1].

Table 3.1: Problem's sets

Sets	
V	Set of vertices
E	Set of edges
i,j e V	Index of vertices ($i, j \in 1, 2, ..., \|V\|$)
v_0	Index of the source node ($V_0 \in V$).Only one source node exists.
e_{ij}	Index of edges ($e_{ij} \in E$)

3.2 Parameters

The challenge of the problem is to achieve the minimum total production cost which is affected by the heat production, the supply temperature, the thermal efficiency of each heat production site combined with the thermal losses of pipes (among other issues given by the capacity limits) for the given district characterized by end user demand, etc [2]. The parameter summarise is shown in table 3.2.

Table 3.2: Model parameters

Name	**Unit**	**Description**
C_{ij}^{fix}	€/ m	Fixed investment cost at edge e_{ij}
C_{ij}^{var}	€/ (m kW)	Variable investment cost at edge e_{ij}
C_{ij}^{om}	€/ m a	Operation & Maintenance cost at edge e_{ij}
C_{i}^{heat}	€/ kWh	Heat generation cost of source $i = v_0$
C_{ij}^{rev}	€/ kWh	Revenue for delivered heat at edge e_{ij}
p_{ij}^{umd}	€/ kWh	Penalty of unmet demand at edge e_{ij}
α	$1/a$	Annuity factor for investment costs
θ_{ij}^{fix}	kW/m	Fixed thermal loss at edge e_{ij}
θ_{ij}^{var}	kW/(kW m)	Variable thermal loss at edge e_{ij}
T_{i}^{flh}	h/a	Full load hours of the source $i = v_0$
β	1	Concurrence effect
λ	1	Connection quota
l_{ij}	m	Length of edge e_{ij}
d_{ij}	kW	Potential peak demand at edge e_{ij}
D_{ij}	kWh/a	Potential peak demand at edge e_{ij}
C_{ij}^{max}	kW	Maximum pipe capacity at edge e_{ij}
Q_{i}^{max}	kW	Source's maximum capacity of heat generation

Technical parameters concern thermal losses in the pipe network θ^{fix} and θ^{var}, and the full load hours T^{flh} (operation time) of the source vertices.The dimensionless parameters β and λ quantify the effects due to the aggregation of demand on street level. Parameter β is the concurrence effect, caused by the probabilistic demand for heat per consumer; it reduces the required peak demand for the heat supply network in comparison to the sum of peak demands. Parameter λ is the connection quota, reflecting the fact that not all buildings in a given street will be connected by the district heating. Consequently, λ reduces the investment costs for heat infrastructure by lowering peak demand, while λ reduces the revenue for heat by lowering the annual demand.

3.3 Decision Variables

The main decision in this problem is to select a set of edges among the set of all possible edges in the network. Therefore, a binary decision variable is required to indicate whether an edge e_{ij} is selected or nor, in other words finding values for the binary decision variable X_{ij}.

The decision variable X_{ij} helps to design the network. When $X_{ij} = 1$, it means that a pipeline is constructed from i to j and the demand of edge e_{ij} is satisfied. It should be noted that Xij determines the direction of the corresponding edge. In contrary, when $X_{ij} = 0$, it means that the demand of edge e_{ij} is not satisfied and the unmet demand is penalized. On the other hand, when X_{ij} is equal to one, there must be a power flow P_{ij}^{in} in the direction i to j into the pipe. The power flow variable at the other end of the pipe is called P_{ij}^{out} and is reduced by thermal losses and heat demand of consumers along the edge. In the Table 3.3 are shown the model variables:

Table 3.3: Model variables

Name	**Unit**	**Description**
z	€/ a	Total expenses
X_{ij}	-	Binary decision variable: 1 = construted pipe
P_{ij}^{in}	kW	Thermal power flow from vertex v_i into edge e_{ij}
P_{ij}^{out}	kW	Thermal power flow out of edge e_{ij} into vertex v_j

3.4 Objective function development

The main objective of the problem is to **minimize the total expense**. The Total Expenses is the difference between the Total Cost and the Revenue.So it is obtained:

Total Expenses (€/a) = Total cost (€/a) - Revenue (€/a)

Where the **Total Cost** is expressed as:

Total cost (€/a) = Total Heat Generation (€/a) + Total Investment cost (€/a) + Total Maintenance Cost (€/a) + Unment Demand Penalty (€/a)

And the **Total Investment Cost** is equal to:

Total Investment Cost (€/a) = Total Fixed Investment Cost (€/a) + Total Variable Investment Cost (€/a)

Below are the equations for each part of the objective function:

Total Revenue (€/ a)

The total Revenue is obtained based on the annual demand (D_{ij}) at a constructed edge ($x_{ij} = 1$), multiplied by the costumer price (C_{ij}^{rev}) and is scaled with the connect quota λ. Parameter $\lambda$is reflecting the fact that not all potential heating demand on a particular edge will be connected to the district heating network. The equation is shown below:

$$\sum_{e_{ij} \subset E} (C_{ij}^{rev} D_{ij} \lambda) X_{ij} \tag{3.1}$$

Total Heat Generation Cost (€/ a)

The Total Heat Generation Cost occurs only in the source node.The total output power from the source (summation over $P_{v_0 j}^{In}$ for all j) is multiplied by the annual full-load hours T^{flh} to estimate the annual thermal energy output. Multiplication with the specific heat generation cost of the source $C_{v_0^{heat}}$ yields the heat generation costs. The division by β is then required to remove the down-scaling of the accumulated peak power due to concurrence effects.

$$\frac{T_{v_0}^{fth} C_{v_0}^{heat}}{\beta} \sum_{e_{v_0 j} \subset E} P_{v_0 j}^{in} \tag{3.2}$$

Total Maintenance Cost (€/ a)

Operation & maintenance have to be paid for existing pipes ($X_{ij} = 1$), and it depends on the cost of the operation (C_{ij}^{om}) and length (l_{ij}) of the given edge. The final result is the summation over all constructed links.

$$\sum_{e_{ij} \subset E} (C_{ij}^{om} l_{ij}) X_{ij} \tag{3.3}$$

Total Fixed Investment Cost (€/ a)

The fixed investment cost is paid for existing pipes ($X_{ij} = 1$), and it depends on the fixed cost of each edge (C_{ij}^{fix}) and length (l_{ij}) of the given edge. It is necessary to use the factor α to have the same units €/a as the other terms. The final result is the summation over all constructed links.

$$\sum_{e_{ij} \subset E} (C_{ij}^{fix} l_{ij} \alpha) X_{ij} \tag{3.4}$$

Total variable investment Cost (€/ a)

The total variable investment cost is equial to the summation over the whole amount of flows distributed in the network (P_{ij}^{In}) by involving the variable investment cost of each edge (c_{ij}^{var}), and the length of constructed edges (l_{ij}). On the other hand, it is necessary to use the factor α to have the same units €/a as the other terms.

$$\sum_{e_{ij} \subset E} (C_{ij}^{var} l_{ij} \alpha) P_{ij}^{in} \tag{3.5}$$

Unmet Demand Penalty Cost (€/ a)

This term of the objective function focuses on the UNCONSTRUCTED edges.As shown before, each edge has a potential annual demand. If a link is constructed, the demand of that link is met; otherwise, a penalty cost depending on the potential annual demand of that link should be paid. The totak unmet demand penalty cost is equal to the summation over all the UNCONSTRUCTED edges, expressed by the term $(1 - X_{ij} - Xji)$ that has to be included having in mind that the network is directed, multiply by the Potential peak demand (D_{ij}) and the Penalty of unmet demands (P_{ij}^{umd}) at the link.

$$0.5 \sum_{e_{ij} \subset E} (D_{ij} P_{ij}^{umd})(1 - X_{ij} - X_{ji}) \tag{3.6}$$

Finally, the objective function Z is:

$$\begin{aligned} Z = \mathbf{min} \sum_{e_{ij} \subset E} -(C_{ij}^{rev} D_{ij} \lambda) X_{ij} + \frac{T_{v_0}^{fth} C_{v_0}^{heat}}{\beta} \sum_{e_{v_0 j} \subset E} P_{v_0 j}^{in} + \\ \sum_{e_{ij} \subset E} (C_{ij}^{om} l_{ij}) X_{ij} + \sum_{e_{ij} \subset E} (C_{ij}^{fix} l_{ij} \alpha) X_{ij} \\ + \sum_{e_{ij} \subset E} (C_{ij}^{var} l_{ij} \alpha) P_{ij}^{in} + 0.5 \sum_{e_{ij} \subset E} (D_{ij} P_{ij}^{umd})(1 - X_{ij} - X_{ji}) \end{aligned}$$

3.5 Constraints

In this section the different constraints of the model are presented:

Tree structure

This constrain guarantees the tree structure of the network, for that aim, the number of selected edges must be equal to the number of nodes minus one.

$$\sum_{e_{ij}} X_{ij} = |V| - 1 \tag{3.7}$$

Unidirectionality

Unidirectionality ensures that only one direction for construction, power flow, and revenue is allowed per edge:

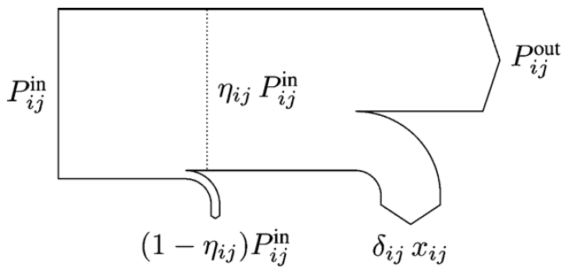
$$\forall e_{ij} \in E, i \neq j : x_{ij} + x_{ji} \leq 1 \tag{3.8}$$

Demand satisfaction

This constraint is the main edge equation linking pipe building decision to a reduction of the power ow out of the pipe by the peak demand d_{ij}. It also includes a power- and length-dependent loss term parametrized by parameters θ_{ij}^{fix} and θ_{ij}^{var}. Figure 3.2 illustrates this equation.

$$\forall e_{ij} \in E, i \neq j : \eta_{ij} P_{ij}^{\text{in}} - P_{ij}^{\text{out}} = \delta_{ij} x_{ij} \tag{3.9}$$

with $\eta_{ij} = 1 - l_{ij}\theta_{ij}^{var}$ and $\delta_{ij} = d_{ij}\beta\gamma + l_{ij}\theta_{ij}^{fix}$



P_{ij}^{in}
$\eta_{ij}\, P_{ij}^{\text{in}}$
P_{ij}^{out}
$(1-\eta_{ij})P_{ij}^{\text{in}}$
$\delta_{ij}\, x_{ij}$

Figure 3.2: Demand satisfaction equation

Flow equilibrium at each vertex

It is considered that no loss occurs when flow passes through each vertex. Therefore, the entering flow into vertex j MUST be equal to exiting flow from vertex j. This constraint is held for each node except the source node.

$$\forall j \in V, j \neq v_0 : \sum_{i \in V, i \neq j} P_{ji}^{\text{in}} = \sum_{i \in V, i \neq j} P_{ij}^{\text{out}} \tag{3.10}$$

Edge capacity constraint

This constraint ensures that the flow entering the edge will not exceed the capacity of that edge

$$\forall e_{ij} \in E, i \neq j : P^{\mathrm{In}}_{ij} \leq C^{\max}_{ij} X_{ij} \tag{3.11}$$

Source structural constraint

This constraint ensures that no flow enters the source and the source has only exiting edges.

$$\sum_{i \in V, i \neq v_0} X_{i v_0} = 0 \tag{3.12}$$

Source's heat generation capacity

This constraint limits the level of heat generation in the source node. Therefore, the total amount of generated heat exiting the source node v0 must be less than the source's maximum capacity.

$$\sum_{j \in V, j \neq v_0} P^{\mathrm{In}}_{v_0 j} \leq Q^{max}_{v_0} \tag{3.13}$$

Tour elimination

This constraint, together with Constraint 1, guarantee that no cycle occurs in the network. For this aim,it is necessary to ensure that for non-source vertices, there is at least one entering edge.

$$\forall i \in V, i \neq v_0 : \sum_{j \subset V, i \neq v_0} X_{ji} \geq 1 \tag{3.14}$$

Domain Variables

These constraints provide the domain of the decision variables

$$\begin{aligned} z &\in \mathbb{R} \\ X_{ij} &\in 0, 1 \\ P^{in}_{ij} &\in \mathbb{R}^+ \\ P^{out}_{ij} &\in \mathbb{R}^+ \\ \forall i, j &\in V \end{aligned}$$

4 Design of the ILS algorithm

Iterative native search (ILS) is a metaheuristic; it iteratively constructs a sequence of solutions generated by an embedded heuristic, resulting in much better solutions than if one were to use random testing of that heuristic. There are two stages that constitute the ILS: (i) there is a chain that is followed; (ii) the search for better solutions occurs in a space that is defined by the output of an embedded heuristic. The local search has been the most used in the embedded heuristics.

To apply the ILS algorithm, there are four elements that must be taken into account:

1. GenerateInitialSolution
2. LocalSearch
3. Perturbation
4. Acceptance Criterion

So in 1) we will start by finding a random solution; 2) for many types of problems the local search algorithm is available to arrive at a better solution 3) for the perturbation, a random movement in the node network will be effective to get out of a local minimum and iv) a first affordable assumption for the acceptance criterion is to find the lowest cost objective function, we can also choose a percentage of acceptance which will accept solutions with a slightly higher cost to some previously found in order to diversify the algorithm and although not the total minimum cost this can help us leave a local minimum in order to find other possible configurations that lead to the global minimum.

4.1 Initial solution

The applied local search defines the starting point.The first thing that is done is to arrive at a valid solution that fulfills all the constraints defined in the previous section. For this, an algorithm will be designed to design a network in a random way, defining hubs and spokes at random until a configuration is found that meets the requested constraints. If you want to reach very high quality solutions as soon as possible in the search, then starting in the best possible way becomes an important issue. This is often achieved by using greedy initial solutions.However, starting a local search from the best possible greedy solution does not necessarily result in the best possible solution, since many times when implementing a greedy algorithm can lead to difficulties with exploring possible better solutions. So, different methods to generate the initial solution were tried, these will be explained in the results section.

4.2 Perturbation

The objective here is to escape the local solution by applying current minimum cost disturbances, a crucial feature is to identify how aggressive the disturbances should be. If they are too small, the same local solution will be returned and few new solutions will be explored. If the disturbances are too aggressive, the response will be almost random, there will be no bias within the sampling, and the solution will be randomly restarted. Ideally, the type of disturbance introduced in a response should not be directly impossible by the local search algorithm and it should complement it in some way that finds a better solution

4.3 Acceptance Criterion

The disturbance mechanism, together with the local search, defines the accepted transitions from the constraints of a current solution to a neighboring solution. The acceptance criteria process consists of determining whether the neighboring solution is accepted or not. Acceptance criteria would be based on finding solutions that are lower in cost or close to it. The acceptance criteria include a strong influence on the character and effectiveness of the search, and are generally used, along with the disturbance, to regulate the balance between intensification and diversification of that search.

4.4 Local Search

Considering the behavior and performance of the ILS algorithm, we can say that it is somewhat sensitive to the selection of the built-in heuristics, we should optimize this alternative whenever possible. There are a large number of meta heuristic algorithms that can be used for the algorithm. We could assume that the better the local search is optimized, the better the results of the ILS algorithm will be. In our case the better the initial solution and the better it is optimized in theory the ILS algorithm should be more effective, however this can be variable which will be explained in the results section.

4.5 Global optimization of ILS

The overall optimisation of the ILS algorithm refers to the adjustment of the interactions between the individual components. The guidelines and recommendations that were taken into account for a good result were:

1. The perturbation must be properly implemented in order to get out of a local minimum
2. The perturbation and the acceptance criteria were the key elements to obtain a better result, since the algorithm had to decide when a less or more aggressive perturbation was necessary, also when the algorithm stayed in a loop and did not find a better solution the acceptance criteria became more flexible to diversify the algorithm and also to get out of a local minimum
3. Make good use of the local search operators and implement them in such a way that each one of them is used at the right time in order to have a wider search field for a better solution.

5 Implementation of Pulp and ILS design

In this section we present the results obtained when optimizing the network using the python pulp library and implementing the ILS design.

5.1 Pulp implementation

The PuLP package, is a free open source software written in Python, that is intended for solving linear programming (LP) and mixed integer linear programming (MIP) problems. Using the problem statement made in the section 3, these equations are entered into python in order to obtain the optimal solution to the problem.

The data provided for the district heating network has 30 nodes and 1 source. Below are the different parts of the implemented code.

After reading the differences parameters from the excel file and assigning to variables according to section 3 it is first created the decision variables for the problem and the variable for the problem data:

```python
# Create the decision variables
self.Xij = pulp.LpVariable.dicts('x',(self.set_V,self.set_V),0,1,cat="Binary")
self.Pij_in = pulp.LpVariable.dicts('Pin',(self.set_V,self.set_V),0)
self.Pij_out = pulp.LpVariable.dicts('Pout',(self.set_V,self.set_V),0)

# Create variable for the problem data
self.heat_network = pulp.LpProblem("District-Heating-Network", pulp.LpMinimize)
```

After the Objective function is defined:

```python
# Objective function

def objective_function(self):

1. self.heat_network += (self.Ti_flh*self.ci_heat[self.v0]/self.beta)*
    pulp.lpSum(self.Pij_in[self.v0][j] for j in self.set_V if j != self.v0) +\
2. pulp.lpSum(self.cij_om[(i,j)]*self.lij[(i,j)]*self.Xij[i][j]
    for i in self.set_V for j in self.set_V if i != j) +\
3. pulp.lpSum(self.cij_fix[(i,j)]*self.lij[(i,j)]*self.alpha*self.Xij[i][j]
    for i in self.set_V for j in self.set_V if i != j) +\
4. pulp.lpSum(self.cij_var[(i,j)]*self.lij[(i,j)]*self.alpha*self.Pij_in[i][j]
    for i in self.set_V for j in self.set_V if i != j) +\
5. 0.5*pulp.lpSum(self.Pij_umd[(i,j)]*self.Dij[(i,j)]*(1-self.Xij[i][j]-self.Xij[j][i])
    for i in self.set_V for j in self.set_V if i != j) -\
6. pulp.lpSum(self.ci_rev[(i,j)]*self.Dij[(i,j)]*self.lam*self.Xij[i][j]
    for i in self.set_V for j in self.set_V if i != j)
```

A continuation the different constrains are defined:

```python
# Constraints
def constraints(self):
    # Tree structure
    self.heat_network += pulp.lpSum(self.Xij[i][j]
    for i in self.set_V for j in self.set_V if i != j) == abs(self.nodes_num) - 1

    # Unidirectionality
    for i in self.set_V:
        for j in self.set_V:
                if i != j: self.heat_network += self.Xij[i][j] + self.Xij[j][i] <= 1

        # Demand satisfaction
        for i in self.set_V:
            for j in self.set_V:
                if i is not j or i is not self.v0:
                    self.heat_network += self.eta[i,j]*self.Pij_in[i][j]
                    - self.Pij_out[i][j] == self.delta[i,j]*self.Xij[i][j]

        # Flow equilibrium at each vertex
        for j in self.set_V:
            if j != self.v0:
                self.heat_network += pulp.lpSum(self.Pij_in[j][i] for i in
                self.set_V if i != j) ==\
                pulp.lpSum(self.Pij_out[i][j] for i in self.set_V if (i != j))

        # Edge capacity
        for i in self.set_V:
            for j in self.set_V:
                if i != j: self.heat_network += self.Pij_in[i][j] <=
                self.Cij_max[(i,j)]*self.Xij[i][j]

        # Source structural
        self.heat_network += pulp.lpSum(self.Xij[i][self.v0] for i in
        self.set_V if i != self.v0) == 0

        # Source's heat generation capacity
        if len(self.Qi_max) == 1:
            self.heat_network += pulp.lpSum(self.Pij_in[self.v0][j] for j in
            self.set_V if j != self.v0) <= self.Qi_max[0]
        else:
            self.heat_network += pulp.lpSum(self.Pij_in[self.v0][j] for j in
            self.set_V if j != self.v0) <= self.Qi_max[self.v0]

        # Tour eliminitation
        for i in self.set_V:
            if i != self.v0:
                self.heat_network += pulp.lpSum(self.Xij[j][i] for j in
                self.set_V if j != i) >= 1
```

Finally it is shown the solution obtained by the pulp

```
def system_solution(self):
    new_line = 4
    self.heat_network.solve()
    print("Status: {}".format(pulp.LpStatus[self.heat_network.status]))

    count = 0
    for v in self.heat_network.variables():
        if v.varValue > 0:
            count += 1
            if count % new_line == 0: print("{} = {}".format(v.name,
            v.varValue))
            elif 'x' in v.name: print("{} = {}".format(v.name, v.varValue),
            end ="\t\t")
            else: print("{} = {}".format(v.name, v.varValue), end ="\t")

    print("\nObjective value problem =
    {}".format(pulp.value(self.heat_network.objective)))

    return pulp.LpStatus[self.heat_network.status]
```

5.2 ILS Implementation

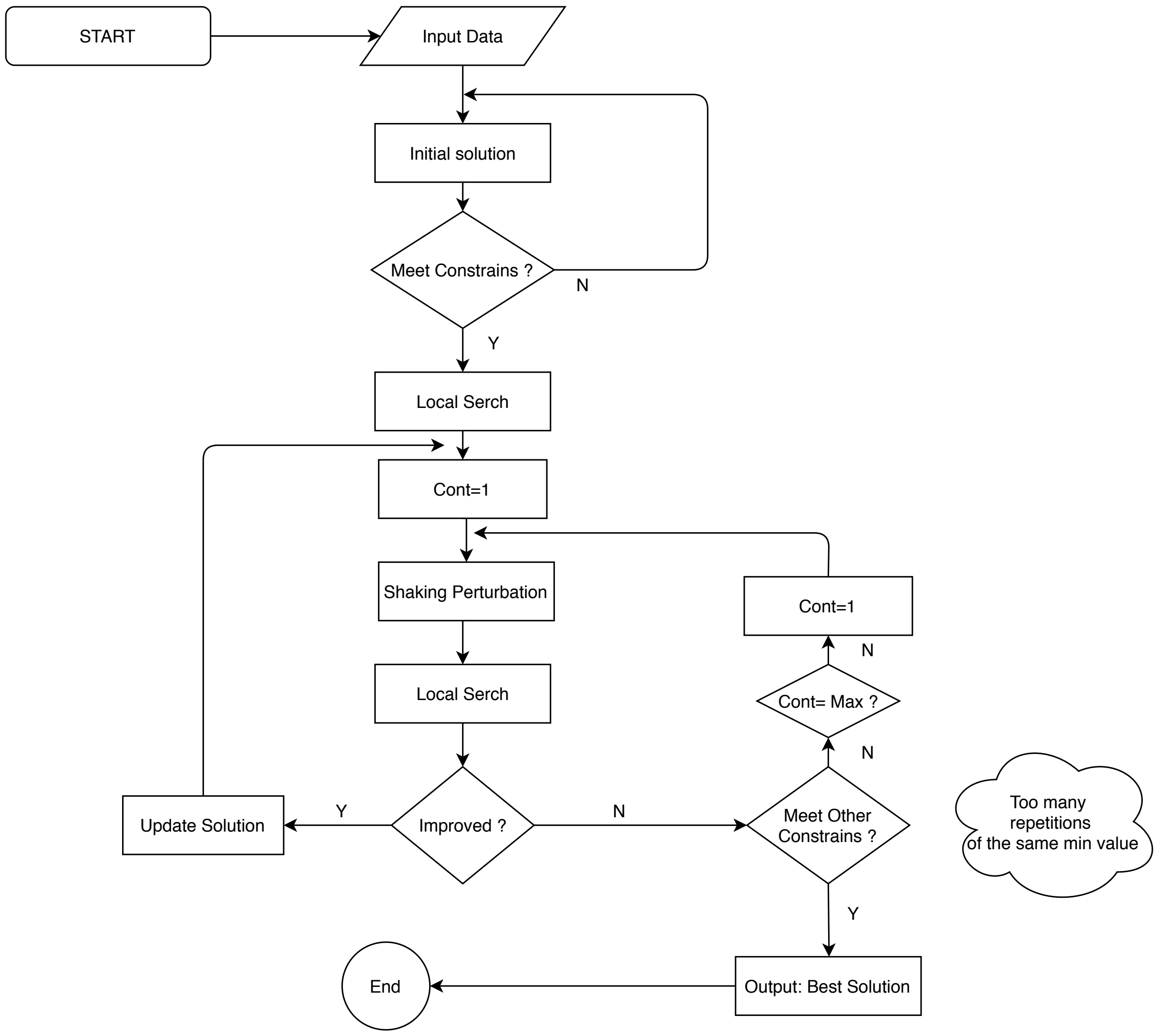


Figure 5.1: ILS Algorithm

Different types of tests were performed with the ILS algorithm in order to reach approximately the optimal result, this algorithm also has to find solutions according to the constraints explained above in the section of the Pulp results. As explained in the section section 5.1 the equations and problems explained in chapter chapter 3, were applied in the ILS algorithm to find the optimal solution. All the necessary parameters for the solution of the equations are provided by an excel file in which the problem and the data to be optimized are shown.

In Code 5.1 we can observe the method to first of all find the objective function that we are going to optimize:

Code 5.1: Program section in python to find the objective cost function of the heating district network

```
    # Evaluate the object function #
```

```python
    def objectFunction(self):
        revenue, heat_gen_cost, maintenance_cost = 0,0,0
        fixed_inv_cost, variable_inv_cost, unmet_demand_pen_cost = 0,0,0

        for j in self.set_V:
            for i in self.set_V:
                revenue += self.ci_rev[i,j]*self.Dij[i,j]*self.X[i,j]
                maintenance_cost += self.cij_om[i,j]*self.lij[i,j]*self.X[i,j]
                fixed_inv_cost += self.cij_fix[i,j]*self.lij[i,j]*self.X[i,j]
                variable_inv_cost += self.cij_var[i,j]*self.lij[i,j]*self.Pin[i,j]
                unmet_demand_pen_cost += self.Pij_umd[i,j]*self.Dij[i,j]*(1 - self.X[i,j] - self.X[j,i])
            heat_gen_cost += self.Pin[self.v0,j]

        revenue *= self.lam
        heat_gen_cost *= (self.Ti_flh * self.ci_heat[self.v0]) / self.beta
        fixed_inv_cost *= self.alpha
        variable_inv_cost *= self.alpha
        unmet_demand_pen_cost *= 0.5
        Z = heat_gen_cost + maintenance_cost + fixed_inv_cost + variable_inv_cost + unmet_demand_pen_cost -
            revenue
        return Z
```

Then, in Code 5.2 we describe the constraints that the network must fulfill to confirm that it is a possible solution.

Code 5.2: Program section in python to describe the constraints of the network

```python
    # Evaluate all constraints and show if anybody is unsatisfied #
    def constrainsEval(self, warnings_off = False):
        tree_struct_sum = 0.0
        for j in self.set_V:
            flow_eq_sum_Pin, flow_eq_sum_Pout, tour_elim_sum = 0.0, 0.0, 0.0
            for i in self.set_V:
                if self.X[i,j] != 0 and self.X[i,j] != 1: # Constraint 9: Domain of variables
                    if not warnings_off: print(Fore.YELLOW + "["+time.strftime("%x %I:%M:%S %p")+ "][WARNING:]
                        Constraint 9 unsatisfied. X[{},{}] != 0 or 1.".format(i,j) + Fore.RESET)
                    return False
                if self.Pin[i,j] < 0.0: # Constraint 9: Domain of variables
                    if not warnings_off: print(Fore.YELLOW + "["+time.strftime("%x %I:%M:%S %p")+ "][WARNING:]
                        Constraint 9 unsatisfied. Pin[{},{}] < 0.".format(i,j) + Fore.RESET)
                    return False
                if self.Pout[i,j] < 0.0: # Constraint 9: Domain of variables
                    if not warnings_off: print(Fore.YELLOW + "["+time.strftime("%x %I:%M:%S %p")+ "][WARNING:]
                        Constraint 9 unsatisfied. Pout[{},{}] < 0.".format(i,j) + Fore.RESET)
                    return False

                tree_struct_sum += self.X[i,j] # Constraint 1: Tree structure

                if i != j:
                    if self.X[i,j] + self.X[j,i] > 1: #  Constraint 2: Unidirectionality
                        if not warnings_off: print(Fore.YELLOW + "["+time.strftime("%x %I:%M:%S %p")+ "][WARNING:]
                            Constraint 2 unsatisfied in ({},{}).".format(i,j) + Fore.RESET)
                        return False

                    if np.abs(self.eta[i,j]*self.Pin[i,j] - self.Pout[i,j] - self.delta[i,j]*self.X[i,j]) > 1e-4 :
                         # Constraint 3: Demand satisfaction
                        if not warnings_off: print(Fore.YELLOW + "["+time.strftime("%x %I:%M:%S %p")+ "][WARNING:]
                            Constraint 3 unsatisfied in ({},{}).".format(i,j) + Fore.RESET)
                        return False

                    if j != self.v0: # Constraint 4 - 8
                        flow_eq_sum_Pin += self.Pin[j,i]
                        flow_eq_sum_Pout += self.Pout[i,j]
                        tour_elim_sum += self.X[i,j]

                    if self.Pin[j,i] > self.Cij_max[j,i]*self.X[j,i]: # Constraint 5: Edge capacity constraint
                        if not warnings_off:
                            print(Fore.YELLOW + "["+time.strftime("%x %I:%M:%S %p")+ "][WARNING:] Constraint 5
                                unsatisfied in ({},{}).".format(j,i) + Fore.RESET)
                            print(Fore.YELLOW + "["+time.strftime("%x %I:%M:%S %p")+ "][WARNING:] Pin = {}, Cmax =
                                 {}, X = {}.".format(self.Pin[j,i],self.Cij_max[j,i],self.X[j,i]) + Fore.RESET)
                        return False

            if np.abs(flow_eq_sum_Pin - flow_eq_sum_Pout) > 1e-4: # Constraint 4: Flow equilibrium at each vertex
```

```python
                if not warnings_off: print(Fore.YELLOW + "["+time.strftime("%x %I:%M:%S %p")+ "][WARNING:] Constraint 4 unsatisfied with j = {}.".format(j) + Fore.RESET)
                return False

            if tour_elim_sum < 1 and j != self.v0: # Constraint 8: Tour elimination
                if not warnings_off: print(Fore.YELLOW + "["+time.strftime("%x %I:%M:%S %p")+ "][WARNING:] Constraint 8 unsatisfied with i = {}.".format(j) + Fore.RESET)
                return False

        if tree_struct_sum != self.nodes_num - 1: # Constraint 1: Tree structure
            if not warnings_off: print(Fore.YELLOW + "["+time.strftime("%x %I:%M:%S %p")+ "][WARNING:] Constraint 1 unsatisfied." + Fore.RESET)
            return False

        source_struct_sum, source_heat_gen_sum = 0.0, 0.0
        for m in self.set_V:
            if m != self.v0:
                source_struct_sum += self.X[m,self.v0] # Constraint 6: Source structural constraint
                source_heat_gen_sum += self.Pin[self.v0,m] # Constraint 7: Source's heat generation capacity

        if source_struct_sum != 0: # Constraint 6: Source structural constraint
            if not warnings_off: print(Fore.YELLOW + "["+time.strftime("%x %I:%M:%S %p")+ "][WARNING:] Constraint 6 unsatisfied." + Fore.RESET)
            return False

        if len(self.Qi_max) == 1: # Constraint 7: Source's heat generation capacity
            if source_heat_gen_sum > self.Qi_max[0]:
                if not warnings_off: print(Fore.YELLOW + "["+time.strftime("%x %I:%M:%S %p")+ "][WARNING:] Constraint 7 unsatisfied." + Fore.RESET)
                return False
        else:
            if source_heat_gen_sum > self.Qi_max[self.v0]:
                if not warnings_off: print(Fore.YELLOW + "["+time.strftime("%x %I:%M:%S %p")+ "][WARNING:] Constraint 7 unsatisfied." + Fore.RESET)
                return False
        return True
```

Two different methods were used to make the first random network that would act as the initial solution and then be optimized with the Local Search algorithm that is part of the ILS.

Code 5.3: First method in python to create a random network

```python
    # First method to generate the random network #
    def generateGreedyNetwork(self):
        self.edge_based = [] # Restart edge based representation
        self.hubs = random.sample(self.set_V, k = random.randint(min(self.set_V),max(self.set_V)))
        self.spokes = [x for x in self.set_V if x not in self.hubs]

        # Hubs created randomly and conect them with Greedy Heuristic
        hubs = [random.choice(self.hubs)] # Initial value
        hubs_dist = {} # Save the edge with its distance
        while len(hubs) < len(self.hubs): # Greedy Heuristic
            hub_min, dist = -1, 0
            for hub in [x for x in self.hubs if x not in hubs]: # Remove hubs organice previously
                if self.lij[hubs[-1], hub] < dist or hub_min == -1:
                    dist, hub_min = self.lij[hubs[-1], hub], hub
            hubs_dist[(hubs[-1], hub_min)] = dist
            hubs.append(hub_min)

        # Join start hub with end hub and Take len(self.hubs) - 1 conexions, at close distance.
        hubs_dist[(hubs[-1], hubs[0])] = self.lij[hubs[-1], hubs[0]]
        self.edge_based = [edge[0] for edge in sorted(hubs_dist.items(), key = lambda x: x[1])[:len(self.hubs) - 1]]

        # Spokes conect to the nearlest hub
        for spoke in self.spokes:
            hub_min, dist = -1, 0.0
            for hub in self.hubs:
                if self.lij[spoke, hub] < dist or hub_min == -1:
                    dist, hub_min = self.lij[spoke, hub], hub
            self.edge_based.append((hub_min, spoke))
```

Code 5.3 describes the first method to create a random network that consists in selecting from the list of hubs n random hubs and connect between them through the greedy algorithm (is to join the nearest node), also the network spokes are made up of other nodes that were not selected from the list and these are connected to the hubs also using the greedy algorithm.

Code 5.4: Second method in python to create a random network

```python
    # Second method to generate the random network #
    def generateRandomNetwork(self, start_node = None):
        self.edge_based = [] # Restart edge based representation
        if start_node is None or start_node not in self.set_V: start_node = random.choice(self.set_V)
        hubs = [start_node]
        spokes = [x for x in self.set_V if x != start_node]
        while len(spokes) > 0:
            spk, hb = random.choice(spokes), random.choice(hubs)
            self.edge_based.append((hb,spk))
            spokes.remove(spk)
            hubs.append(spk)
```

Code 5.4 is the second method to create a random network, which consists in having two empty lists L1 and L2, the objective is to have all the nodes in a list, in this case L2, and randomly choose a node that will go to the other list L1, then in the L2 list is selected at random one of the nodes with which it is filling, the same process is done in L2 and these two nodes are connected so that the node that was in L2 will go to L1, so on until leaving the initial list empty L1, was done in this way for several reasons: The first is to keep the randomness of the network as intact as possible, also to give another method with which you can solve the problem and which can be contrasted with Code 5.3 for a more complete analysis.

Code 5.5: Local Search Algorithm

```python
    # Find a minimal solution with initial value and Local Search algorithm #
    def localSearch(self, method_choose = 1, epochs = 100):
        # Input update
        method_save = method_choose
        # self.variablesUpdate()
        if self.constrainsEval():
            min_z = self.objectFunction() # Initial value of object function value
            best_edge = self.edge_based.copy()
        else:
            print(Fore.YELLOW + "["+time.strftime("%x %I:%M:%S %p")+ "][WARNING:] Initial solution invalid.
                 Imposible to apply local search algorithm." + Fore.RESET)
            return self.edge_based.copy(), []

        z_history, spoke_i = [min_z], None
        # print(Fore.RED + "Initial z = {}".format(min_z) + Fore.RESET)
        for i in range(epochs): # Number of executions
            if method_save not in [1,2,3,4]: method_choose = random.choice([1,2,3]) # Random method
            self.variablesUpdate() # Update list spokes and hubs
            edge_based = self.edge_based.copy()
            if method_choose == 1 or method_choose == 4: # Local Search 1
                index_list = [x for x in range(len(edge_based)) if edge_based[x][1] in self.spokes]
                spoke_list = [edge_based[x][1] for x in index_list]
                start_sp, final_sp = tuple(random.sample(range(len(index_list)), k = 2)) # Choose position
                     randomly
                if final_sp < start_sp: start_sp, final_sp = final_sp, start_sp # final_sp must be bigger than
                     start_sp
                spoke_list = spoke_list[:start_sp] + spoke_list[start_sp:final_sp + 1][::-1] + spoke_list[final_sp
                     + 1:]
                for i in range(len(index_list)):
                    self.edge_based[index_list[i]] = (edge_based[index_list[i]][0], spoke_list[i]) # Swap from
                         start_sp to final_sp
                self.variablesUpdate() # Update values of decision variables
                if self.constrainsEval(warnings_off=True):
                    z = self.objectFunction() # Evaluate object function with new network
                    if z < min_z:
                        min_z, best_edge = z, self.edge_based.copy()
                        # print(Fore.CYAN + "Nueva minima red1 = {} with z = {}".format(best_edge, z) + Fore.RESET
                             )
            if method_choose > 1:
                spoke_pv = spoke_i # For method 2 and 3
                while spoke_i == spoke_pv or spoke_i is None: spoke_i = random.choice([x for x in self.spokes if x != self.v0])
                index_spoke_i = [sp[1] for sp in self.edge_based].index(spoke_i)
                hub_spoke = self.edge_based[index_spoke_i][0]

                # print("Choose spoke {} in pos {}".format(spoke_i,index_spoke_i))
                node_list = []
                if method_choose == 2 or method_choose == 4: node_list += [x for x in self.hubs if hub_spoke != x] # Local Search 2
                if method_choose > 2: node_list = [x for x in self.spokes if x != spoke_i] # Local Search 3

                # print(Fore.RED + "Random spoke = {}".format(spoke_i) + Fore.RESET)
                # print(Fore.RED + "\nHUBS = {}. SPOKES = {}".format(self.hubs, self.spokes) + Fore.RESET)
                # print(Fore.RED + "({},{}), node_list = {}".format(hub_spoke, spoke_i, node_list) + Fore.RESET)
                for new_hub in node_list: # Local Search 2-3
                    index_spoke_i = [sp[1] for sp in self.edge_based].index(spoke_i)
                    self.edge_based[index_spoke_i] = (new_hub, spoke_i)
                    self.variablesUpdate() # Update values of decision variables
                    if self.constrainsEval(warnings_off=True):
                        z = self.objectFunction() # Evaluate object function with new network
                        # print(" Intento de z = {}".format(z))
                        if z < min_z:
                            min_z, best_edge = z, self.edge_based.copy()
                            # print(Fore.CYAN + "Nueva minima red2-3 = {} with z = {}".format(best_edge, z) + Fore.RESET)
                    self.edge_based = edge_based.copy()
            self.edge_based = best_edge.copy() # Best solution to next iteration
            z_history.append(min_z)

        self.variablesUpdate()
        return self.edge_based.copy(), z_history # Solution and history return
```

In Code 5.5, we can see the implementation of the Local search algorithm that as explained above is based on optimizing a network that is entered and through different operators can modify the network nodes, such as changing hubs to spokes, or exchange position of the spokes, these operators are made to find a configuration that provides a lower cost to the entered but always complying with the constraints necessary that assure the network work.

It also occurs the case that this algorithm is stuck because it finds a local minimum, to get out of this, the algorithm has an operator that performs a disturbance, this is responsible for making a random change in the network as a random position of a node this can cause it to increase the total cost but gives a new approach to the network so the algorithm can leave this loop, also this disturbance can be effective depending on the use and situation that is configured, because if this is too aggressive can lead to the network entered is a completely different and new configuration, which is not the objective of the algorithm, however if the disturbance is very small may not be able to leave this local minimum and get stuck. The intensification operators that were used to apply the local search meta-heuristic were:

- Swap on the spokes : In this method we use a reallocation of the spokes towards new hubs selecting randomly a start and end index of the spokes list and inverting it completely so that now the spoke that belongs to the final index is in the initial with a different hub and in the same way with the other selected spokes, this is, the list will now be located from the end to the initial, although they will only be modified within the randomly chosen range.
- Re-allocation of singles spokes to the existing hubs : this operator selects a speaker from its list and this is reallocated to an existing hub chosen from the list of hubs, this way the spokes are relocated step by step to reach new configurations.
- Re-allocation of single spokes to a new hub : A spoke is selected from its list and it is relocated with a non existing hub, that is to say a spoke becomes a hub and the selected spoke is added to this new hub, this is done taking into account the distances between the nodes and if the algorithm decides that this operator is the most suitable to continue.

The objective of the operators is to make changes in the network in order to modify it little by little and to find out if this modification leads to a better solution. The three operators described above were made by reallocating nodes in the network and changing their positions according to criteria described in the algorithm.

For the implementation of the local search meta-heuristics it was decided to do it in two different ways to increase the options of success.

The first way is to apply the three operators sequentially in each of the nodes, this implies more processing time and many more operations, however this can give us more variety of results.

The other way is to apply a single operator to each node by choosing randomly from the 3 possible methods, this maintains the randomness of the algorithm and can reach different configurations with respect to the previous method.

The results will be presented in the next section, and an analysis will be made on which was the closest to the best solution.

Code 5.6: Implementation of Perturbation in the network

```python
     # Shaking operator implementation as perturbation method #
    def perturbation(self, max_total_operations = 100, num_perturbation = 1):
        self.updateHubsSpokes() # Update list spokes and hubs
        if len(self.hubs) < 2:
            print(Fore.YELLOW + "["+time.strftime("%x %I:%M:%S %p")+ "][WARNING:] Impossible to apply shaking
                operator. At least 2 hubs required." + Fore.RESET)
            return

        cont = 0
        for _ in range(max_total_operations):
            hub1, hub2 = tuple(random.sample(self.hubs, k = 2)) # Choose two hubs randomly
            edge_based = self.edge_based.copy()
            # print("Initial red: {} with hub1 = {} and hub2 = {}".format(edge_based,hub1,hub2))
            for i in range(len(edge_based)):
                new_edge = edge_based[i]
                if new_edge[0] == hub1: new_edge = (hub2, new_edge[1]) # Exchanbe hub1 with hub2 in hubs position
                elif new_edge[0] == hub2: new_edge = (hub1, new_edge[1])
                if new_edge[1] == hub1: new_edge = (new_edge[0], hub2) # Exchanbe hub1 with hub2 in spokes
                    position
                elif new_edge[1] == hub2: new_edge = (new_edge[0], hub1)
                self.edge_based[i] = new_edge # Exchange save
            # print("Final red: {}".format(self.edge_based))
            self.variablesUpdate() # Update Pin, Pout and X (decision variables)
            if self.constrainsEval(warnings_off = True):
                cont += 1
                if cont >= num_perturbation: break # Acceptable perturbation. Return
            else: self.edge_based = edge_based.copy() # Restore edge based values

        if self.edge_based == edge_based:
            print(Fore.YELLOW + "["+time.strftime("%x %I:%M:%S %p")+ "][WARNING:] Invalid shaking operator" +
                " in {} iterations. Network doesn't change.".format(max_total_operations) + Fore.RESET)
            self.variablesUpdate() # Update to previous values
```

To perform the perturbation (diversification operator) it was selected two random hubs from the list of hubs and change their role between them (ie all nodes that were connected to one of the selected hubs are now relocated and become connected to the other selected node), this is done in order to make an aggressive change in the configuration of the network and thus increase the co-diversification that has the algorithm and find a solution that has not yet been visited in the solution space.

Code 5.7: Evaluation of the best configuration with less cost

```python
    # Evaluate and choose the best configuration #
    def acceptanceCriterion(self, new_edge_base, previous_edge_base = None, admission_error = 0.05, **kwargs):
        # Save previous result
        edge_based = self.edge_based.copy()
        if previous_edge_base is None: previous_edge_base = edge_based.copy()

        previous_cons = self.setCurrentER(previous_edge_base.copy())
        previous_z = self.objectFunction()
        new_cons = self.setCurrentER(new_edge_base.copy())
        new_z = self.objectFunction()

        if not (previous_cons or new_cons): self.edge_based = edge_based.copy()
        elif previous_cons and not new_cons: self.edge_based = previous_edge_base.copy()
        elif not previous_cons and new_cons: self.edge_based = new_edge_base.copy()
        elif (new_z - previous_z)/max([new_z, previous_z]) < admission_error: self.edge_based = new_edge_base.copy
            ()
        else: self.edge_based = previous_edge_base.copy()

        self.variablesUpdate()
        return self.edge_based.copy(), self.objectFunction()
```

Code 5.7 shows us how to compare which network is the most optimal in terms of cost and which is its configuration, this function is responsible for defining the acceptance criteria that will indicate the code when to stop running the search, which networks will be accepted and which others will be discarded.

At first it was decided that this criterion was simply met when you find a network with a lower cost than the previous one, however after several tests we saw that the network did not vary much and was stuck in the algorithm, so it was decided to add a percentage of error to accept networks that have a worse solution, however, this cost must be within a selected percentage with respect to the best solution found at the moment in the ILS algorithm, this percentage will be an acceptance criteria that will also help to diversify the algorithm more, we tested with several acceptance percentages to evidence which one helped us to get out of local minimum more effectively. In the next section will be presented the results, this percentage can not be very high or very low due to if it is very high the target function will only increase, if it is very small will not be enough for the algorithm to escape from a local minimum and will stay with the same minimum until the end of the algorithm. It was resorted to choose a criterion of acceptance with the objective of being able to escape from local minimums, this will increase when the same minimum value is found repeatedly during 20 iterations and will increase until finding a better solution or a worse one but that enters within this criterion of acceptance.

Two specific stop criteria were handled which indicated to the algorithm when to stop its search and these were:

1. A counter N=1000 which indicated that a thousand iterations of the ILS algorithm would be performed, therefore a better solution N number of times would be searched.
2. It was declared a maximum limit of repetition, this was responsible for identifying the number of times that the minimum solution had no variation and simply repeat, I handled a value of 375 and 500 iterations for some tests, this means that if the ILS algorithm did not vary its minimum solution for 375 or 500 iterations depending on the test, this parameter would stop the code.

6 Results

This chapter presents the results obtained from the implementations described in the previous section chapter 5

6.1 PULP Results

The final result obtained by the pulp is **optimal** according to the program's response, the following network connections are obtained:

$X_{1,26} = 1.0$	$X_{7,29} = 1.0$	$X_{15,28} = 1.0$	$X_{22,7} = 1.0$
$X_{2,21} = 1.0$	$X_{8,3} = 1.0$	$X_{18,13} = 1.0$	$X_{23,25} = 1.0$
$X_{2,17} = 1.0$	$X_{9,11} = 1.0$	$X_{19,2} = 1.0$	$X_{24,27} = 1.0$
$X_{3,23} = 1.0$	$X_{11,8} = 1.0$	$X_{20,22} = 1.0$	$X_{27,16} = 1.0$
$X_{4,9} = 1.0$	$X_{12,30} = 1.0$	$X_{20,19} = 1.0$	$X_{28,1} = 1.0$
$X_{4,20} = 1.0$	$X_{13,12} = 1.0$	$X_{21,18} = 1.0$	$X_{29,6} = 1.0$
$X_{4,15} = 1.0$	$X_{14,24} = 1.0$	$X_{22,5} = 1.0$	$X_{29,14} = 1.0$
$X_{6,10} = 1.0$			

Given the previous connections, the following power values are obtained:

$P^{\text{in}}_{11,8} = 926.74228$	$P^{\text{in}}_{12,30} = 121.22882$	$P^{\text{in}}_{13,12} = 655.58933$	$P^{\text{in}}_{14,24} = 504.93458$
$P^{\text{in}}_{15,28} = 1299.0392$	$P^{\text{in}}_{18,13} = 1069.7446$	$P^{\text{in}}_{19,2} = 2796.6175$	$P^{\text{in}}_{1,26} = 362.41634$
$P^{\text{in}}_{20,19} = 2994.5119$	$P^{\text{in}}_{20,22} = 3485.2951$	$P^{\text{in}}_{21,18} = 1606.3178$	$P^{\text{in}}_{22,5} = 280.09037$
$P^{\text{in}}_{22,7} = 2572.7611$	$P^{\text{in}}_{23,25} = 139.63932$	$P^{\text{in}}_{24,27} = 374.59924$	$P^{\text{in}}_{27,16} = 90.309604$
$P^{\text{in}}_{28,1} = 695.52953$	$P^{\text{in}}_{29,14} = 692.55936$	$P^{\text{in}}_{29,6} = 486.49779$	$P^{\text{in}}_{2,17} = 296.61819$
$P^{\text{in}}_{2,21} = 1990.8547$	$P^{\text{in}}_{3,23} = 413.07064$	$P^{\text{in}}_{4,15} = 1661.141$	$P^{\text{in}}_{4,20} = 6948.3753$
$P^{\text{in}}_{4,9} = 1376.2376$	$P^{\text{in}}_{6,10} = 229.2437$	$P^{\text{in}}_{7,29} = 2011.0231$	$P^{\text{in}}_{8,3} = 570.17373$
$P^{\text{in}}_{9,11} = 1132.0812$	$P^{\text{out}}_{11,8} = 570.17373$	$P^{\text{out}}_{13,12} = 121.22882$	$P^{\text{out}}_{14,24} = 374.59924$
$P^{\text{out}}_{15,28} = 695.52953$	$P^{\text{out}}_{18,13} = 655.58933$	$P^{\text{out}}_{19,2} = 2287.4729$	$P^{\text{out}}_{20,19} = 2796.6175$
$P^{\text{out}}_{20,22} = 2852.8515$	$P^{\text{out}}_{21,18} = 1069.7446$	$P^{\text{out}}_{22,7} = 2011.0231$	$P^{\text{out}}_{24,27} = 90.309604$
$P^{\text{out}}_{28,1} = 362.41634$	$P^{\text{out}}_{29,14} = 504.93458$	$P^{\text{out}}_{29,6} = 229.2437$	$P^{\text{out}}_{2,21} = 1606.3178$
$P^{\text{out}}_{3,23} = 139.63932$	$P^{\text{out}}_{4,15} = 1299.0392$	$P^{\text{out}}_{4,20} = 6479.807$	$P^{\text{out}}_{4,9} = 1132.0812$
$P^{\text{out}}_{7,29} = 1179.0571$	$P^{\text{out}}_{8,3} = 413.07064$	$P^{\text{out}}_{9,11} = 926.74228$	

The optimal objective function is: **z = 30125335.77425636 €/ year**. In the figure Figure 6.1 is presented the network configuration obtained by MILP algorithm

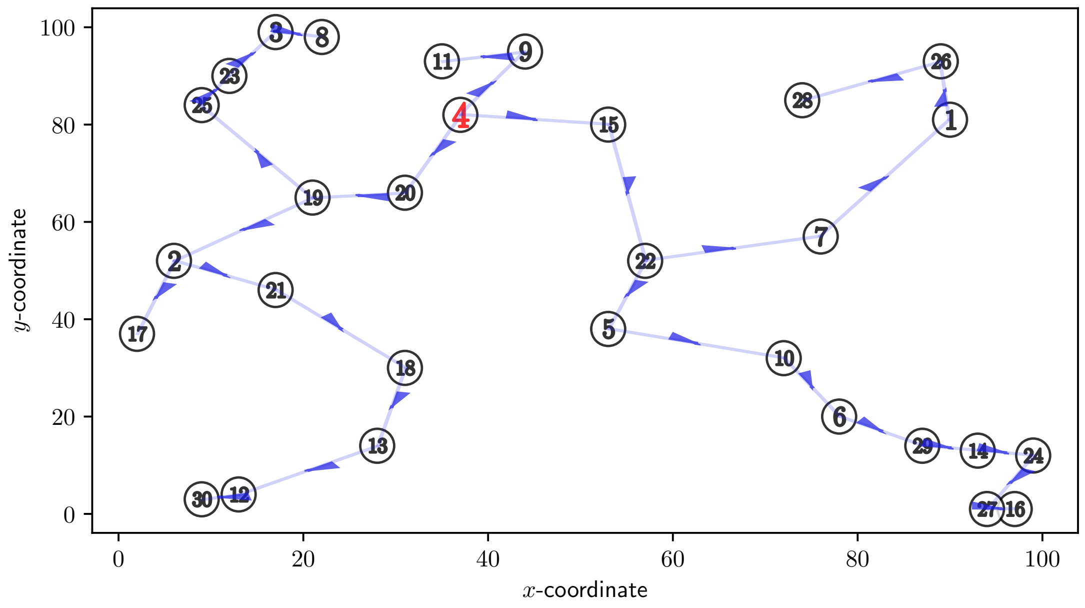


Figure 6.1: Diagram of the network obtained with PULP

6.2 ILS results

In order to analyze different configurations of the ILS design explained in the section chapter 5, various program executions are performed taking into account the parameters presented in the table Table 6.1 in order to verify which configuration generates the best solution for the ILS.

Table 6.1: ILS Parameters in python code

ILS section	Variable name	Values	Description
Initial solution Generation	choose_method	True/False	Selection of the initial solution generation algorithm (False: Random, True: Greedy)
	start_node	4	Selection of the initial node to generate solution
Local serch	method_choose	0, 1, 2, 3, 4	Selection of the intensification method to be applied in the local serch: - 1: Swap on the spokes - 2: Re-allocation of singles spokes to the existing hubs - 3: Re-allocation of single spokes to a new hub - 4: Running all three sequentially -0: Random selection among the three
	epochs	150, 175, 200, 300	Number of epocs of the local serch
Perturbation	max_total_operations	1000	Number of epocs of the ILS
	num_perturbation	1, 2	Number of perturbation to make a bigger perturbation
Acceptance Criteria	admission_error	0.1%, 0.5%, 1%, 1.2%, 1.5%, 3%	Acceptance percentage for new solution
General	lenght_data_comp	375, 500, None	Number of solutions without change to stop the algorithm
	ILS_epochs	1000, 1200, 1500	Total number of epocs of the ILS algorithm

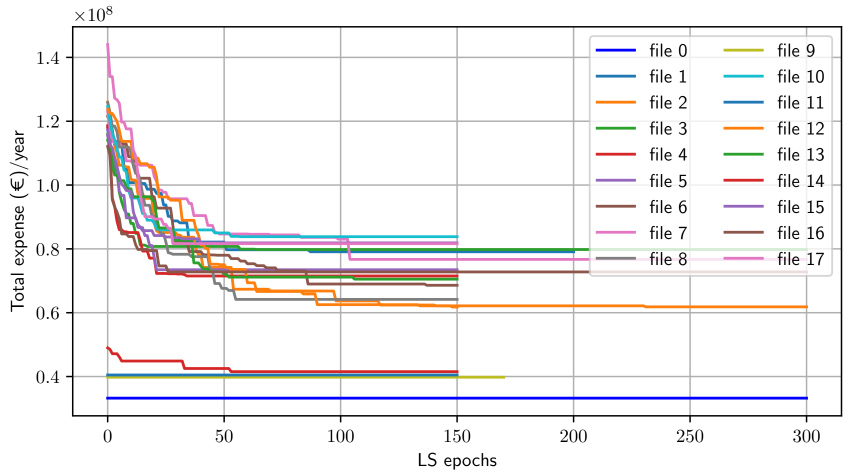


Figure 6.2: Optimization of the initial solution with the Local Search algorithm

We performed 18 tests with different input parameters as a test of the algorithm, each of which took a considerable amount of processing time, in Figure 6.2 we can observe the local search algorithm applied to the initial solution and how the solution is optimized only in the first iteration of the whole algorithm. applied to the initial solution and how the solution is optimized only in the first iteration of the whole algorithm, with different configurations as the local search intensifier operators and the way they are selected. We can see that some initial solutions start with a quite high cost objective function and when applying the meta-heuristic optimization its value decreases considerably in the first iteration, however some solutions can also be observed that they started with a quite good value so the local search cannot optimize it too much and these solutions remain constant.

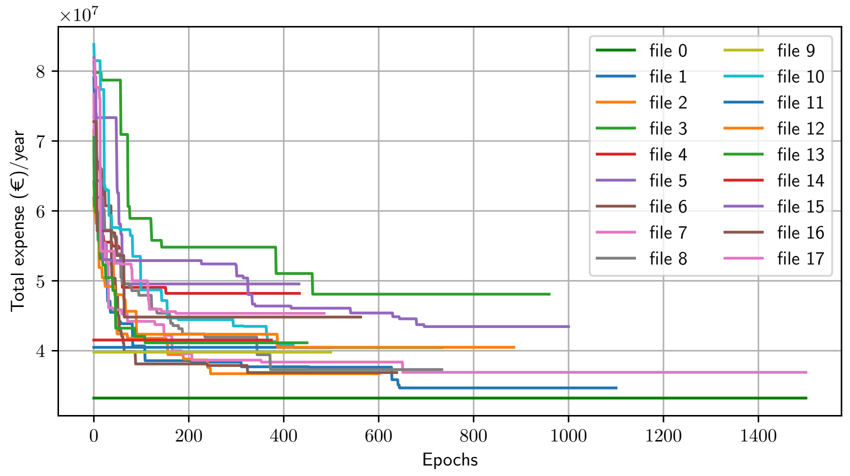


Figure 6.3: Best total expensive cost obtained per configuration

In figure Figure 6.3 the different values obtained of Z (cost function) are presented throughout the execution of the ILS given different configurations as shown in table Table 6.1, it is observed that **the best solution** obtained has a value of **Z = 33229254.223 €/year** , with local search method: 0 (random selection of intensification operators), Generation solution False (gerated Greedy) and an acceptance percentage of 0.1% and **the worst solution** obtained has a value of **Z =**

49554569.547 with local search method: 4 (random selection of intensification operators) and an acceptance percentage of 3%

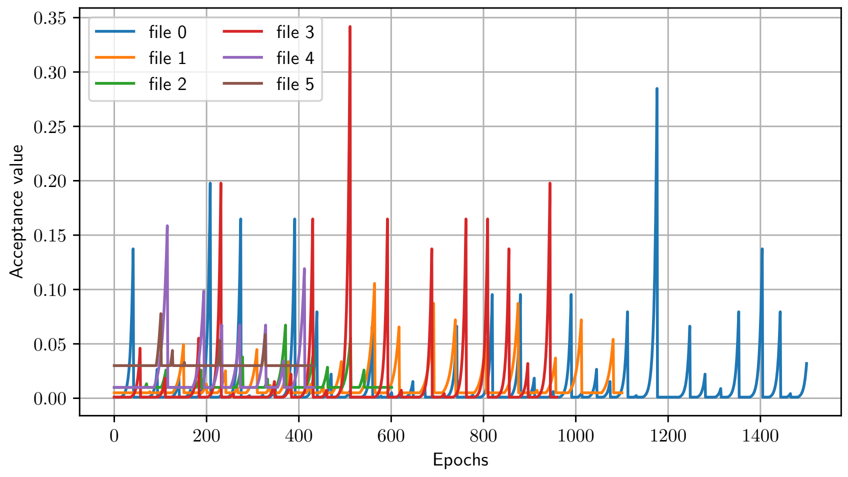


Figure 6.4: Variation of acceptance value along the ILS for the 3 best and worst files

In Figure 6.4 we can see the behavior of the acceptance criterion for a total of 6 files, compared the 3 best and worst solutions, handling a percentage of 1% which will serve to diversify the algorithm and the search for better initial solutions, even in some files varied this criterion in order to find better results which will be explained in the next section.

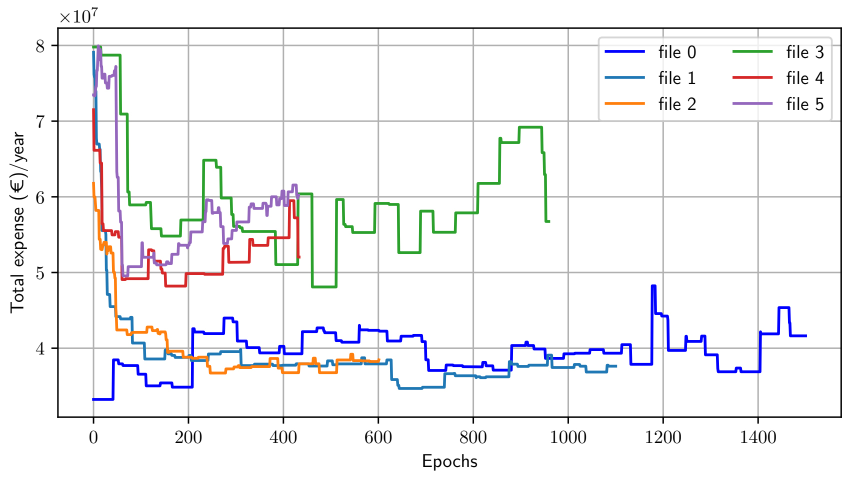


Figure 6.5: Variation of cost according to acceptance value

In the figure Figure 6.5 the variation of Z is shown after being disturbed and the acceptance percentage evaluated, it is observed that not only lower values are accepted but also some above the minimum in order to explore new solutions.

In figure Figure 6.6, is is shown the best solution with **Z = 33229254.223 €/year** , the initial generated solution has a small value, which the algorithm does not reduce any further, it is observed that given the disturbance, higher acceptance values are generated each time in order to escape the local minimum, however it is not achieved a big change from the value initially generated

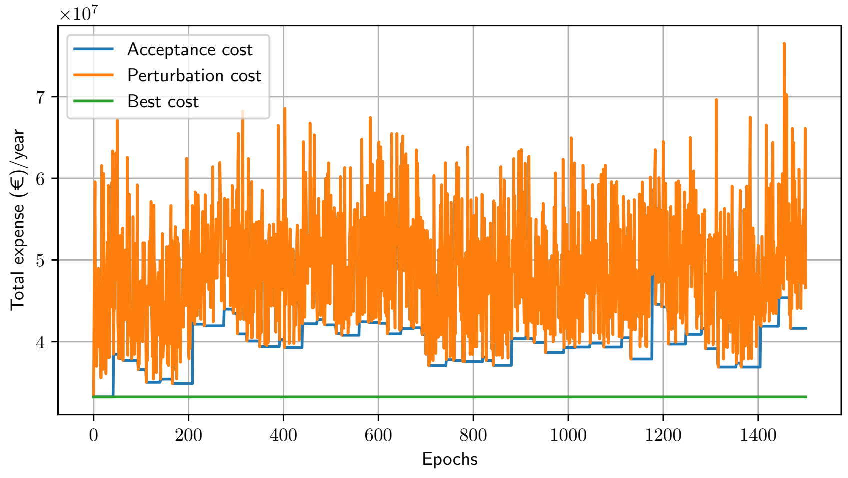


Figure 6.6: Minimum cost obtained z= 33229254 : Acceptance cost, Perturbation Cost and Best cost

In Figure 6.7 we can see the DH network obtained with the best cost result for the heating district network with a value of Z = 33229254.223 €/year.

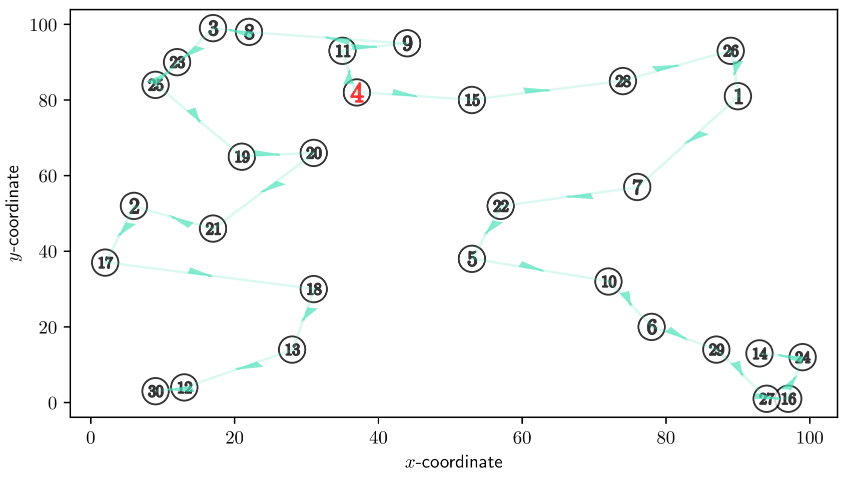


Figure 6.7: Best heat district network connection with Z = 33229254.223 €/year

Finally, the edge connections and flow powers are presented below:

$X_{2,17}= 1$	$X_{3,25}= 1$	$X_{4,11}= 1$	$X_{4,15}= 1$
$X_{5,10}= 1$	$X_{6,29}= 1$	$X_{7,22}= 1$	$X_{8,3}= 1$
$X_{9,8}= 1$	$X_{10,6}= 1$	$X_{11,9}= 1$	$X_{12,30}= 1$
$X_{13,12}= 1$	$X_{15,28}= 1$	$X_{16,24}= 1$	$X_{17,18}= 1$
$X_{18,13}= 1$	$X_{19,20}= 1$	$X_{20,21}= 1$	$X_{21,2}= 1$
$X_{22,5}= 1$	$X_{24,14}= 1$	$X_{25,19}= 1$	$X_{25,23}= 1$
$X_{26,1}= 1$	$X_{27,16}= 1$	$X_{28,26}= 1$	$X_{29,27}= 1$
$X_{1,7}= 1$			

$P^{\text{in}}_{1,7}$= 3437.827589531791	$P^{\text{in}}_{2,17}$= 2233.3007182773067	$P^{\text{in}}_{3,25}$= 4649.342523007933	$P^{\text{in}}_{4,11}$= 5837.420317100873
$P^{\text{in}}_{4,15}$= 5263.211950256981	$P^{\text{in}}_{5,10}$= 1980.1487288494368	$P^{\text{in}}_{6,29}$= 1145.004830091795	$P^{\text{in}}_{7,22}$= 2829.361011081717
$P^{\text{in}}_{8,3}$= 4810.986645493382	$P^{\text{in}}_{9,8}$= 5302.232739995136	$P^{\text{in}}_{10,6}$= 1378.5660692348113	$P^{\text{in}}_{11,9}$= 5518.4906810872835
$P^{\text{in}}_{12,30}$= 121.22881812829931	$P^{\text{in}}_{13,12}$= 655.5893259544345	$P^{\text{in}}_{15,28}$= 4889.493776297683	$P^{\text{in}}_{16,24}$= 419.1221586815692
$P^{\text{in}}_{17,18}$= 1930.068140634666	$P^{\text{in}}_{18,13}$= 1069.7445652260694	$P^{\text{in}}_{19,20}$= 3533.511845196642	$P^{\text{in}}_{20,21}$= 3334.100780363591
$P^{\text{in}}_{21,2}$= 2619.57076484844	$P^{\text{in}}_{22,5}$= 2266.3122658458237	$P^{\text{in}}_{24,14}$= 129.92474231944726	$P^{\text{in}}_{25,19}$= 3996.0819421599053
$P^{\text{in}}_{25,23}$= 139.63932220959828	$P^{\text{in}}_{26,1}$= 3809.375442345296	$P^{\text{in}}_{27,16}$= 509.70856633535055	$P^{\text{in}}_{28,26}$= 4268.157458616274
$P^{\text{in}}_{29,27}$= 886.1837193183088	$P^{\text{out}}_{1,7}$= 2829.361011081717	$P^{\text{out}}_{2,17}$= 1930.068140634666	$P^{\text{out}}_{3,25}$= 4135.721264369504
$P^{\text{out}}_{4,11}$= 5518.4906810872835	$P^{\text{out}}_{4,15}$= 4889.493776297683	$P^{\text{out}}_{5,10}$= 1378.5660692348113	$P^{\text{out}}_{6,29}$= 886.1837193183088
$P^{\text{out}}_{7,22}$= 2266.3122658458237	$P^{\text{out}}_{8,3}$= 4649.342523007933	$P^{\text{out}}_{9,8}$= 4810.986645493382	$P^{\text{out}}_{10,6}$= 1145.004830091795
$P^{\text{out}}_{11,9}$= 5302.232739995136	$P^{\text{out}}_{13,12}$= 121.22881812829931	$P^{\text{out}}_{15,28}$= 4268.157458616274	$P^{\text{out}}_{16,24}$= 129.92474231944726
$P^{\text{out}}_{17,18}$= 1069.7445652260694	$P^{\text{out}}_{18,13}$= 655.5893259544345	$P^{\text{out}}_{19,20}$= 3334.100780363591	$P^{\text{out}}_{20,21}$= 2619.57076484844
$P^{\text{out}}_{21,2}$= 2233.3007182773067	$P^{\text{out}}_{22,5}$= 1980.1487288494368	$P^{\text{out}}_{25,19}$= 3533.511845196642	$P^{\text{out}}_{26,1}$= 3437.827589531791
$P^{\text{out}}_{27,16}$= 419.1221586815692	$P^{\text{out}}_{28,26}$= 3809.375442345296	$P^{\text{out}}_{29,27}$= 509.70856633535055	

The following graphs show the results obtained for the second best response with **Z= 34.679.923** which was optimized by the algorithm, with an initial value of 118.164.041,632. In the Figure 6.8 the variation of the best response after the execution of ILS is presented, it is observed that not only are lower values accepted after a disturbance, however, after various modifications, a lower value is not achieved. This solution was generated with Method 0 (random selection of intensification operators), Generation solution False (gerated Greedy) and an acceptance percentage of 0.5%.

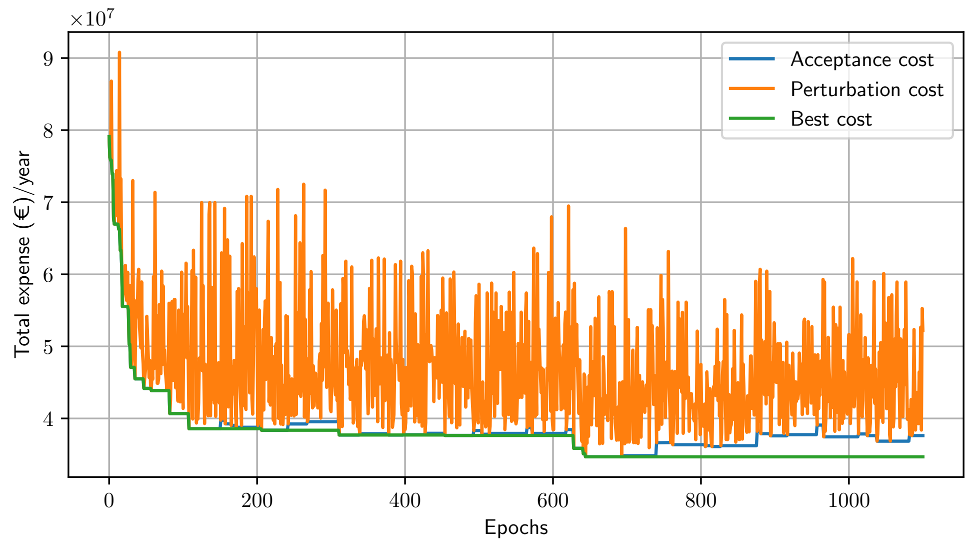


Figure 6.8: Acceptance cost, perturbation cost, best cost for the district heat network with Z = 34.679.923

The Figure 6.9 shows the behavior of the Local Search algorithm, in subfigure a) the local search applied to the initial network generated by the Greedy algorithm is observed, in b) the Local search for random epoch is observed, in c) Final local search before the algorithm stopped and in sub figure d) the local search for all iterations is presented

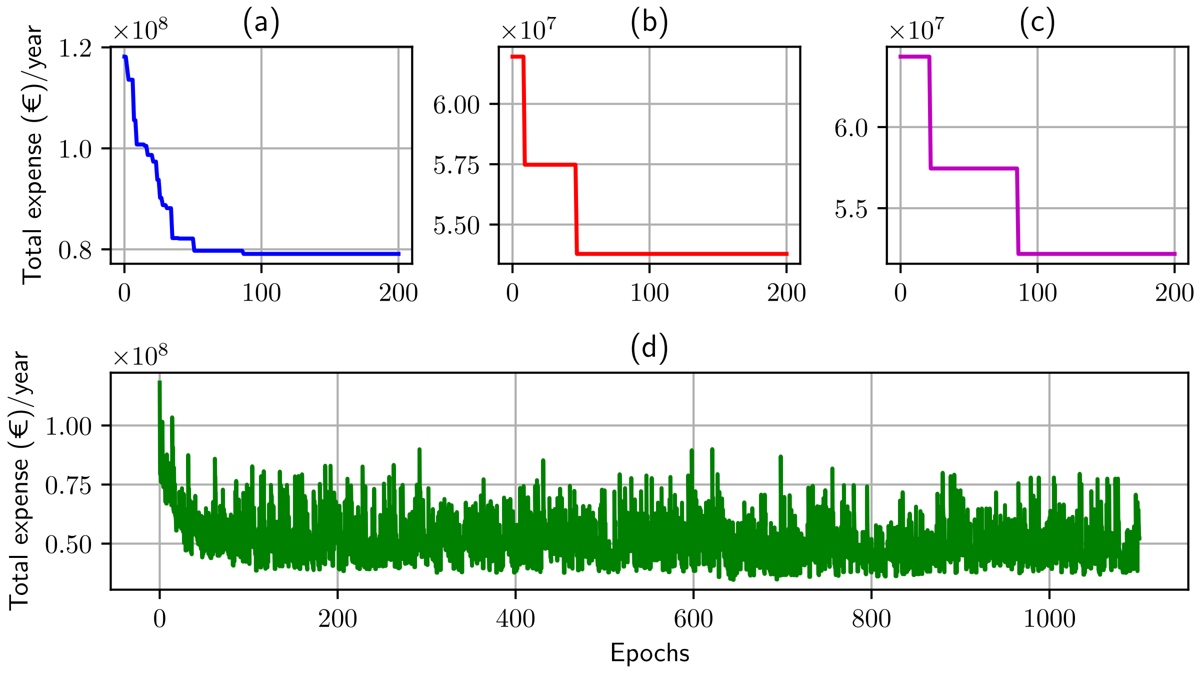


Figure 6.9: Best result with Z = 34.679.923 - Local search results: a) Local search for initial solution b) Local search for random epoch c) Final local search d) local search for all iterations

Figure 6.10 shows us the variation of the acceptance criterion for each iteration, the value of this criterion is 0.5%, also if the algorithm does not change its minimum solution during 20 iterations, this value will increase until finding a worse solution within this acceptance percentage.

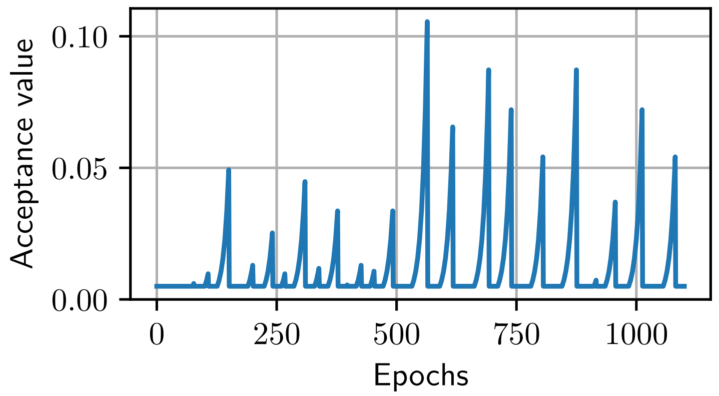


Figure 6.10: Acceptance history for the network with Z = 34.679.923

In Figure 6.11 we can see the processing time of the heat district network with Z = 34.679.923 , here we can see the average time it takes to perform a single iteration of the ILS algorithm , and how long it takes per iteration of the algorithm.

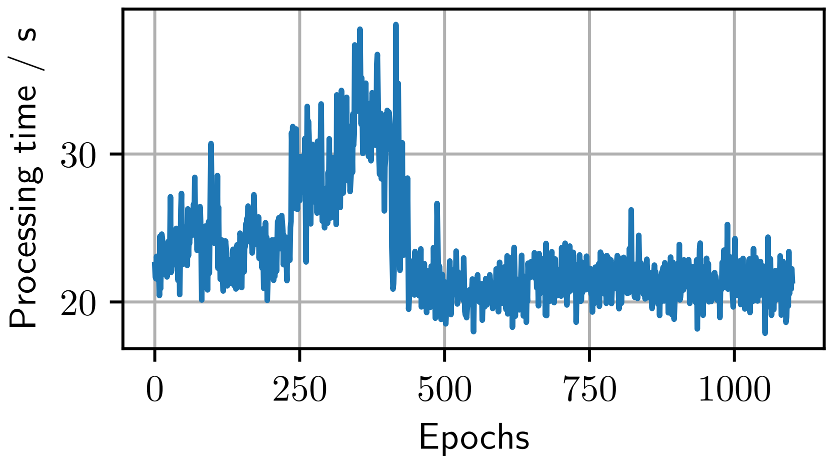


Figure 6.11: Total time for the district heat network with Z = 34.679.923

In Figure 6.12 we can observe the second best network, we consider this network relevant because, unlike the best one, it started with a cost of Z = 118,164,041.632 until reaching a Z = 34.679.923, after a large number of iterations and optimizations of the local search.

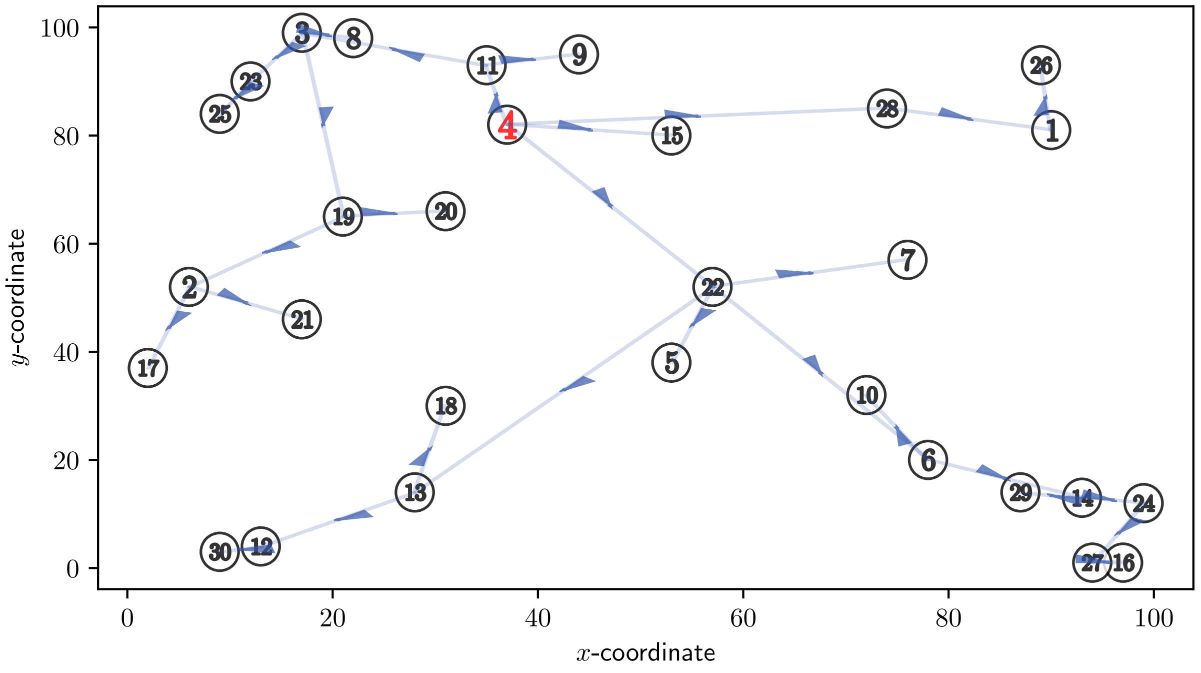


Figure 6.12: Best result for the district heat network with Z = 34.679.923

For this network configuration, the results of node-connections and input/output flow powers are shown:

$X_{2,17}= 1$	$X_{2,21}= 1$	$X_{3,8}= 1$	$X_{3,19}= 1$
$X_{3,23}= 1$	$X_{4,11}= 1$	$X_{4,15}= 1$	$X_{4,22}= 1$
$X_{4,28}= 1$	$X_{6,10}= 1$	$X_{6,14}= 1$	$X_{11,3}= 1$
$X_{11,9}= 1$	$X_{12,30}= 1$	$X_{13,12}= 1$	$X_{13,18}= 1$
$X_{14,24}= 1$	$X_{14,29}= 1$	$X_{19,2}= 1$	$X_{19,20}= 1$
$X_{22,5}= 1$	$X_{22,6}= 1$	$X_{22,7}= 1$	$X_{22,13}= 1$
$X_{23,25}= 1$	$X_{24,27}= 1$	$X_{27,16}= 1$	$X_{28,1}= 1$
$X_{1,26}= 1$			

$P^{\text{in}}_{1,26}= 362.4163383202287$	$P^{\text{in}}_{2,17}= 296.6181901049616$	$P^{\text{in}}_{2,21}= 380.096730141253$	$P^{\text{in}}_{3,8}= 156.66029730231742$
$P^{\text{in}}_{3,19}= 2047.3220872650138$	$P^{\text{in}}_{3,23}= 413.07064024107785$	$P^{\text{in}}_{4,11}= 3714.0020854865234$	$P^{\text{in}}_{4,15}= 357.8990150714865$
$P^{\text{in}}_{4,22}= 6217.026189998072$	$P^{\text{in}}_{4,28}= 1456.0677225807217$	$P^{\text{in}}_{6,10}= 229.24370139954814$	$P^{\text{in}}_{6,14}= 1069.2333036550958$
$P^{\text{in}}_{11,3}= 3197.5064953198075$	$P^{\text{in}}_{11,9}= 203.02627779026096$	$P^{\text{in}}_{12,30}= 121.22881812829931$	$P^{\text{in}}_{13,12}= 655.5893259544345$
$P^{\text{in}}_{13,18}= 411.4762814477023$	$P^{\text{in}}_{14,24}= 504.9345822846631$	$P^{\text{in}}_{14,29}= 186.79413142109186$	$P^{\text{in}}_{19,2}= 1176.53384172165$
$P^{\text{in}}_{19,20}= 190.0025461067545$	$P^{\text{in}}_{22,5}= 280.09037326654885$	$P^{\text{in}}_{22,6}= 2094.628239754256$	$P^{\text{in}}_{22,7}= 551.41255283277$
$P^{\text{in}}_{22,13}= 2356.2932012607885$	$P^{\text{in}}_{23,25}= 139.63932220959828$	$P^{\text{in}}_{24,27}= 374.5992427206698$	$P^{\text{in}}_{27,16}= 90.30960433886365$
$P^{\text{in}}_{28,1}= 695.529526908766$	$P^{\text{out}}_{3,19}= 1366.5363878284047$	$P^{\text{out}}_{3,23}= 139.63932220959828$	$P^{\text{out}}_{4,11}= 3400.5327731100683$
$P^{\text{out}}_{4,22}= 5282.424367114364$	$P^{\text{out}}_{4,28}= 695.529526908766$	$P^{\text{out}}_{6,14}= 691.728713705755$	$P^{\text{out}}_{11,3}= 2617.053024808409$
$P^{\text{out}}_{13,12}= 121.22881812829931$	$P^{\text{out}}_{14,24}= 374.5992427206698$	$P^{\text{out}}_{19,2}= 676.7149202462147$	$P^{\text{out}}_{22,6}= 1298.477005054644$
$P^{\text{out}}_{22,13}= 1067.0656074021367$	$P^{\text{out}}_{24,27}= 90.30960433886365$	$P^{\text{out}}_{28,1}= 362.4163383202287$	

7 Analysis of results

According to the graphs Figure 6.2 and Figure 6.3 it is evident that for the experiments carried out the convergence of the algorithm takes place approximately in the first 300 iterations in the ILS and 100 iterations in the local search, this suggests that the set of parameters with which the different experiments were carried out make it possible to convince a local minimum from which the algorithm is not able to escape by itself. For this reason, techniques such as dynamism in the acceptance criteria were applied, but new techniques are required that allow to get out of the local minimum in a better way.

In the figure Figure 6.3 it is observed that the best solution obtained has a value of Z =33,229,254.223, which corresponded to the configuration initially generated by the Greedy, from the figure Figure 6.4 it is observed that the percentages of for the acceptance criteria are increasingly larger in order to try to get it out of the local minimum, which after 1500 iterations was not possible.

On the other hand, from the Figure 6.5 the behavior of the file 5 the value of the cost according to the criteria of acceptance begins decreasing, that is to say that only minor values are accepted, after approximately 85 iterations it begins to accede in different proportions until finalizing the algorithm, that is to say it accepts major values; in a same order of idea, the file 2 is presented, this one decreases in different proportions until the iteration 220 approximately, from which it maintains minimum proportions of change. These changes are translated in low amplitudes of the peaks presented in the signals of the figure Figure 6.4 that is to say, the greater the proportion of change, the smaller the variation in the percentage of acceptance.

In the Figure 6.7 is observed the best network configuration obtained from the initial generation based on Greedy, where were selected initially (randomly) all the nodes as "hubs", after the selection of hubs, this algorithm connects them using the greedy method, since no node was left as "spoke" the algorithm made the connection of all the network, obtaining finally the same connection that would obtain when applying the greedy method. Since the algorithm cannot escape the local minimum obtained with this initial solution, the analysis of the ILS behavior is performed based on the second best solution obtained

In the Figure 6.8 it is observed that initially only minor values are accepted to the previous one, nevertheless after the iteration 750 it is observed that they begin to accept major values with the purpose of escaping from the local minimum until its completion, however a minor value is not obtained to Z= 34.679.923. In comparison with the Figure 6.10 it is found that starting from the 500 iteration a higher percentage of acceptance is required (worse solutions) in order to obtain a better solution.

In the Figure 6.11 you can see the average time of the execution which is approximately 23.34 s per epoch, that is a total of 7.14h to execute the whole algorithm.

In the Figure 6.12 is shown the final configuration obtained from the network, which has more spokes

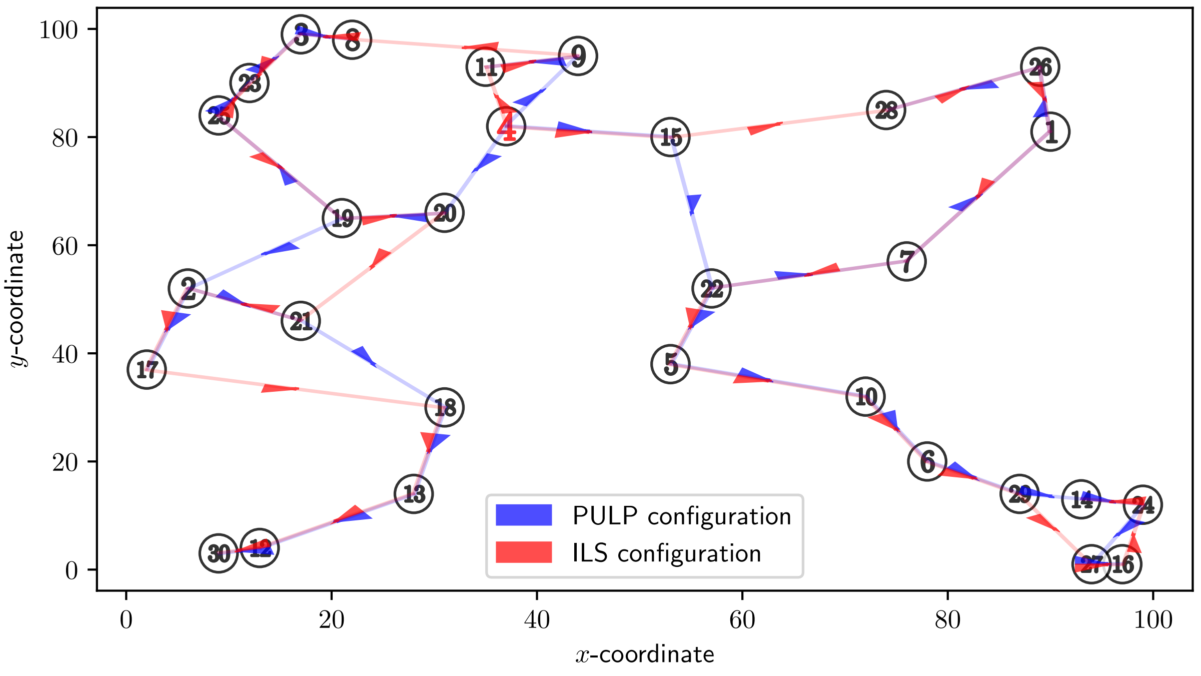


Figure 7.1: Comparison between pulp and ILS district heat network

In Figure 7.1 it can be observed the purchase of the heat district network optimal for the algorithm implemented with the pulp library and the ILS algorithm, in this it is observed as some differences between them, however, it can be evidenced that the flow between the nodes is the same in both cases, also both are based on a greedy search, reaching a final value of cost Z very similar.

8 Conclusions

- It was observed that after trying various configurations of the designed ILS algorithm, it is difficult to obtain a similar or better response to that obtained with the python PULP library, however it is obtained an error rate below 10% with algorithm that can be implemented in any other programming language
- For future applications, a new function is required that is able to adapt better as the number of iterations in the algorithm increases, for example the use of another metauristic capable of focusing greater efforts after a large number of iterations to ensure greater effectiveness when leaving a local minimum and avoid too much stabilization of the ILS algorithm
- According to the results obtained, it was observed that the ILS can be used to give a good initial solution to another metaurism in order to find an optimal value
- According to the tests carried out, it can be concluded that the solution space is composed of several local minima around the global minimum and despite all the tests carried out in none of them could achieve the optimal minimum value
- According to the theory, the better the initial solution delivered to the ILS, the better its performance, however, according to the results, it can be seen that the better the solution, the more difficult the algorithm is to get out of the local minimum to continue with the search, this type of algorithm requires a perturbation to leave the local minimum, but according to the experiments performed, this can not be too large because it could leads to a very high distance from the global minimum, we think of the use of algorithms capable of evaluating at what time and what type of disturbance to run for future experiments
- In the algorithm were implemented two different ways to optimize the network in each iteration, as explained in section 5.2, the first is based on using the three operators of intensification on each node, the other method is responsible for selecting a random operator per node, from the results we could conclude that method 2 leads to a better optimal solution and taking into account the complexity of time is much more useful, since it consumes fewer computer resources and its scale is of the order O(n).

Bibliography

[1] J. Dorfner and T. Hamacher. Large-scale district heating network optimization. *IEEE Transactions on Smart Grid*, 5(4):1884–1891, 2014.

[2] Mattias Vesterlund, Andrea Toffolo, and Jan Dahl. Optimization of multi-source complex district heating network, a case study. *Energy*, 126:53 – 63, 2017.