IMT Atlantique
Projet PRONTO – S6 2024-2025
Campus de Brest

Projet PRONTO
Chaud pour décider ?

Livrable rapport du projet PRONTO

Mélina WANG

Mohamed RHARRASSI

Mohamed Amine JANATI

Rayan BOUAKAR

Encadrés par : Patrick Meyer et Célia Benmansour

Date d'édition : 20 mai 2025
Version : 1.1

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

Sommaire

Liste des figures

Liste des tableaux

1. Résumé

Face à l'expansion des réseaux de chaleur dans un contexte de ressources énergétiques et budgétaires limitées, il devient crucial de dimensionner ces réseaux de manière optimale. Le but est de maximiser l'efficacité de l'argent public dépensé tout en répondant aux besoins réels des usagers. Le projet PRONTO : Chaud pour décider ? a donc pour finalité la conception d'un algorithme de dimensionnement d'un réseau de chaleur qui doit respecter les contraintes physiques du système (lois de conservation, capacités des infrastructures) et adopter une structure arborescente du réseau. L'objectif de ce projet est donc de garantir la solution optimale pour des données de petite taille (Small Data), et proposer une solution approchée dans les cas plus complexes (Big Data). Ces algorithmes sont, dans un second temps, modifiés pour être adaptés à différents scénarios.

2. Introduction

Ce document, qui constitue le rapport final du projet, retrace les différentes étapes de notre travail ainsi que les principales difficultés rencontrées. Il débute par la présentation des ensembles et paramètres manipulés, suivie d'un rappel de certaines sections du cahier des charges, notamment les fonctionnalités attendues et les scénarios envisagés. La modélisation mathématique, la mise en œuvre algorithmique et les résultats obtenus sur différents jeux de données sont ensuite détaillés. Le document se termine par un aperçu de l'organisation adoptée et une conclusion.

3. Liste des ensembles et paramètres

Cette section présente les ensembles et paramètres nécessaires à la modélisation de notre problème. On réutilisera ces variables dans la suite du rapport.

Table 1 – Liste des ensembles et paramètres utilisés

Ensembles	
V	Ensemble des sommets
E	Ensemble des arêtes $(e_{i,j} \in E)$
V_0	Ensemble des sources, dans notre cas il n'y en a qu'un seul $(v_0 \in V)$
$i, j \in V$	Indice des sommets $(i, j \in \{1, 2, \ldots, \lvert V \rvert\})$

Suite à la page suivante

Table 1 – suite

Paramètres	
$c_{i,j}^{fix}$	Coût d'investissement fixe sur l'arête (ϵ/m)
$c_{i,j}^{var}$	Coût d'investissement variable sur l'arête $(\epsilon/(m \cdot kW))$
$c_{i,j}^{im}$	Coût d'exploitation et de maintenance sur l'arête $(\epsilon/(m \cdot an))$
c_{i}^{heat}	Coût de production de chaleur à la source (ϵ/kWh)
$c_{i,j}^{\text{rev}}$	Revenu pour la chaleur livrée à l'arête (ϵ/kWh)
$p_{ij}^{\text{un d}}$	Pénalité pour la demande non satisfaite à l'arête (ϵ/kWh)
α	Facteur d'annuité pour les coûts d'investissement (1/an)
θ_{ij}^{fix}	Pertes thermiques fixes sur l'arête (kW/m)
θ_{ij}^{var}	Pertes thermiques variables sur l'arête (kW/(kW·m))
T_{i}^{fih}	Nombre d'heures de fonctionnement à pleine charge de la source (h/an)
β	Effet de concurrence
λ	Quota de connexions
l_{ij}	Longueur de l'arête (m)
d_{ij}	Demande de pointe de puissance potentielle à l'arête (kW)
D_{ij}	Demande annuelle à l'arête (kWh/an)
$C_{ij}^{\max}$	Capacité maximale du tuyau sur l'arête (kW)
$Q_{v_0}^{\max}$	Capacité maximale de production de chaleur à la source (kW)

4. Rappel du cahier des charges

4.1. Les fonctionnalités

Dans cette section, nous rappelons les principales fonctionnalités décrites dans le cahier des charges, qui ont guidé le développement de notre algorithme. Ces fonctionnalités définissent ce que le système doit accomplir.

TABLE 2 – Tableau des fonctions et des critères associés pour un réseau de chaleur

Fonction	Critère(s)
FP1 – Fournir un réseau de chaleur aux usagers	
FP1.1 – Minimiser les dépenses	Revenu Coût de génération de chaleur Coût d'investissement fixe Coût d'investissement variable Coût de maintenance
FP1.2 – Maximiser la couverture en réseaux de chaleur	Coût de pénalités liées aux demandes non satisfaites
FC1 – Imposer une structure sur le réseau de chaleur	
FC1 – Imposer une structure d'arbre	Le nombre d'arêtes sélectionnées doit être égal au nombre de nœuds moins un. Chaque arête autorise un seul sens pour la construction, le flux d'énergie et les revenus. Aucun cycle ne se produit dans le réseau. À cette fin, nous pouvons imposer que, pour les sommets non sources, il y ait au moins une arête entrante.
FC2 – Respecter la capacité des éléments du réseau de chaleur	
FC2.1 – Contrôler le flux de chaleur	Assurer que le flux traversant une arête ne dépasse pas sa capacité maximale
FC2.2 – Limiter la génération de chaleur	La quantité de chaleur générée par un sommet ne doit pas dépasser sa capacité maximale

Suite à la page suivante

Table 2 – suite

Fonction	Critère(s)
FC3 – Respecter les lois de conservation	
FC3.1 – Respecter la conservation du flux sur les arêtes du graphe	Assurer que le flux entrant en chaque arête soit égal au flux sortant plus les pertes
FC3.2 – Respecter la conservation du flux sur les sommets	Assurer que la somme des flux entrants en chaque sommet est égale à la somme des flux sortants
FS1 – Distribuer par ordre de priorité	
FS1 – Assurer une distribution prioritaire	Définir des niveaux de priorité pour les consommateurs et s'y baser pour construire le réseau Donner la priorité aux arêtes les plus sollicitées
FS2 – Renforcer la résilience du réseau	
FS2 – Renforcer la résilience du réseau	Proposer des moyens d'approvisionnement secondaires Utiliser des arêtes de secours pour répondre aux demandes des usagers en cas de pannes au niveau des tuyaux Introduire des réservoirs de secours dans le réseau afin de résoudre des pannes au niveau de source de chaleur
FS3 – Amélioration de l'efficacité énergétique du réseau	
FS3 – Amélioration de l'efficacité énergétique du réseau	Réduire les pertes thermiques Réduire les pertes thermiques en minimisant la somme des longueurs des arêtes du réseau de chaleur

4.2. Les scénarios

Au début du projet, nous avions défini trois scénarios à explorer : l'équité, la résilience du réseau de chaleur ainsi que la minimisation des pertes thermiques :

- Le scénario d'équité vise à répartir la chaleur de manière juste, en intégrant des critères sociaux, comme la priorité donnée aux familles nombreuses ou aux zones de logements sociaux.
- Le scénario de résilience, quant à lui, cherche à rendre le réseau plus flexible, en s'assurant qu'il reste fonctionnel même en cas de panne d'une arête.
- Le scénario de minimisation de pertes thermiques, lui, a pour objectif de réduire la perte d'énergie en minimisant la longueur totale des conduites du réseau de chaleur.

5. Modélisation

5.1. Les contraintes

Les contraintes définissent les limites et conditions que le modèle doit respecter, garantissant des solutions réalisables. Elles incluent des règles comme la conservation de l'énergie, les capacités des infrastructures, mais aussi la structure arborescente du réseau. Ces contraintes assurent que les solutions sont non seulement faisables, mais aussi conformes aux objectifs du projet.

- **Contrainte 1 :** Garantit la structure d'arbre au réseau

$$\sum_{i,j \in V} X_{i,j} = |V| - 1 \tag{1}$$

- **Contrainte 2 :** Traduit l'unidirectionnalité du réseau

$$\forall i, j \in V \text{ tels que } e_{i,j} \in E \text{ et } i < j : \quad X_{i,j} + X_{j,i} \leq 1 \tag{2}$$

- **Contrainte 3 :** Modélise la satisfaction de la demande

$$\eta_{i,j} \times P^{IN}_{i,j} - P^{OUT}_{i,j} = \delta_{i,j} \times X_{i,j} \tag{3}$$

Avec :

$$\begin{cases} \eta_{i,j} = 1 - \theta^{var}_{i,j}\, l_{i,j} \\ \delta_{i,j} = d_{i,j}\, \beta\, \lambda + l_{i,j}\, \theta^{fix}_{i,j} \end{cases} \tag{3.1}$$

- **Contrainte 4 :** Assure l'équilibre du flux (le flux entrant doit être égal au flux sortant)

$$\forall j \in V \backslash V_0 : \quad \sum_{i \in V, i \neq j} P^{OUT}_{i,j} = \sum_{i \in V, i \neq j} P^{IN}_{j,i} \tag{4}$$

- **Contrainte 5 :** Assure que le flux entrant dans une arête ne dépasse pas sa capacité maximale

$$\forall i, j \in V \text{ tels que } e_{i,j} \in E : \quad P_{i,j}^{IN} \leq X_{i,j} C_{i,j}^{\max} \tag{5}$$

- **Contrainte 6 :** Assure qu'il n'y a aucun flux qui entre dans la source (seulement des flux sortant)

$$\forall j \in V_0, \quad \begin{cases} \sum_{i \in V, i \neq j} P_{i,j}^{IN} = 0 \\ \sum_{i \in V, i \neq j} X_{i,j} = 0 \end{cases} \tag{6}$$

- **Contrainte 7 :** Fixe la capacité de production maximum de la source

$$\forall j \in V_0 : \quad \sum_{i \in V, i \neq j} P_{j,i}^{IN} \leq Q_j^{\max} \tag{7}$$

- **Contrainte 8 :** Couplée à la contrainte 1, elle rend le réseau acyclique

$$\forall i \in V \backslash V_0 : \quad \sum_{j \neq i} X_{j,i} \geq 1 \tag{8}$$

- **Contrainte 9 :** Spécifie le domaine de définition des variables

$$\begin{cases} X_{i,j} \in \{0, 1\} \\ P_{i,j}^{OUT} \in \mathbb{R}_+ \\ P_{i,j}^{IN} \in \mathbb{R}_+ \end{cases} \tag{9}$$

5.2. La fonction objective

La fonction objective a pour but de guider le dimensionnement du réseau de chaleur en prenant en compte toutes les dépenses et revenus liés à la construction et à l'utilisation du réseau. Concrètement, c'est cette fonction que nous allons chercher à maximiser (ou minimiser selon notre définition et notre souhait) : c'est elle qui traduit la volonté de couvrir un maximum de terrain tout en minimisant l'argent investi dans le respect des contraintes du système. La section qui suit présente donc les différents termes apparaissant dans la fonction objective.

- **Revenu :**

$$\sum_{i,j \in V, e_{i,j} \in E} D_{i,j} X_{i,j} \lambda c_{i,j}^{rev} \tag{10}$$

- **Coût de production de chaleur :**

$$\sum_{v \in V_0} \sum_{j \in V} p_{v,j}^{IN} \frac{T_v^{IN} c_v^{heat}}{\beta} \tag{11}$$

- **Coût de maintenance :**

$$\sum_{i,j \in V} c_{i,j}^{om} l_{i,j} X_{i,j} \tag{12}$$

- **Coût total d'investissement fixe :**

$$\sum_{i,j \in V} c_{i,j}^{fix} l_{i,j} \alpha X_{i,j} \tag{13}$$

- **Coût total d'investissement variable :**

$$\sum_{i,j \in V} \alpha c_{i,j}^{var} P_{i,j}^{IN} l_{i,j} \tag{14}$$

- **Coût de pénalité pour la demande non satisfaite :**

$$\sum_{i,j \in V, i<j} \left(1 - X_{i,j} - X_{j,i}\right) P_{i,j}^{umd} D_{i,j} \tag{15}$$

On obtient alors la fonction objective Z définie par :

$$\begin{aligned} \mathbf{Z} = &\text{ Revenu } (10) - \text{ Coût de production de chaleur } (11) - \text{ Coût de maintenance } (12) \\ &- \text{ Coût d'investissement fixe } (13) - \text{ Coût total d'investissement variable } (14) \\ &- \text{ Coût de pénalité pour la demande non satisfaite } (15) \end{aligned} \tag{16}$$

6. Les résultats

6.1. Small Data

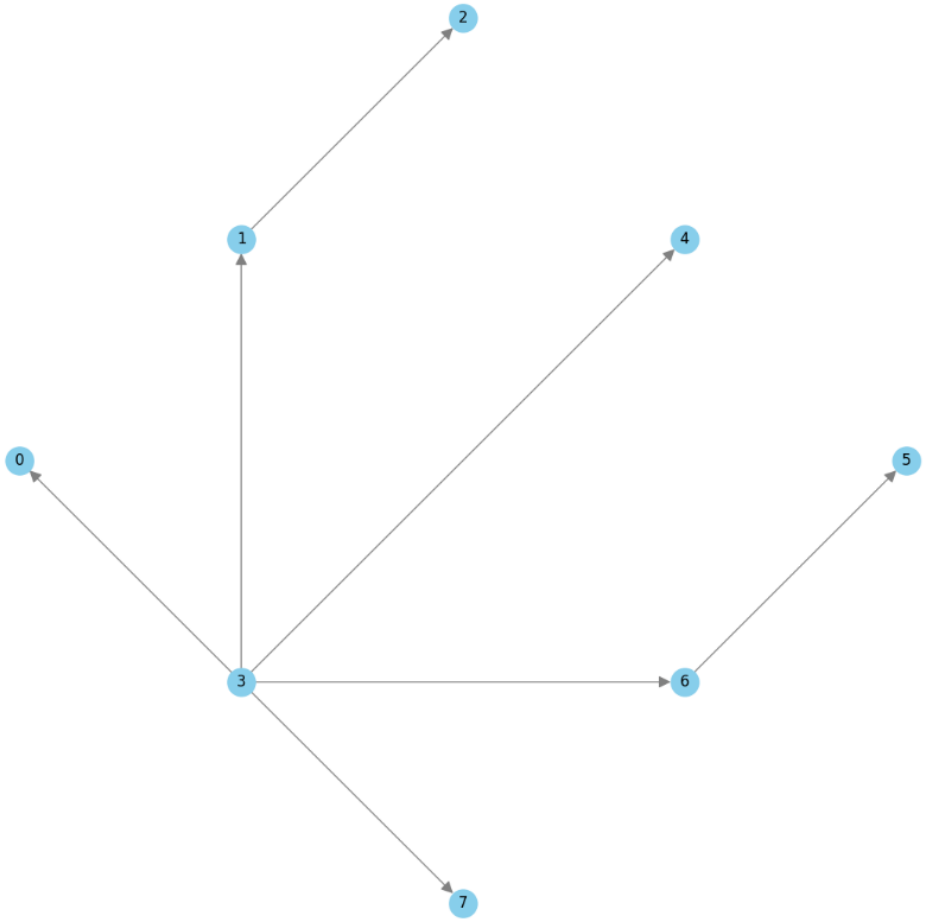


FIGURE 1 – Graphe obtenu à l'aide de PULP pour le jeu de Small Data

Une fois que la modélisation a été réalisée, nous avons commencé à implémenter nos algorithmes. Dans un premier temps, nous avons réalisé l'algorithme d'optimisation sur la Small Data afin d'obtenir une solution optimale. Pour cela, nous avons utilisé la bibliothèque PULP. Nous obtenons alors le graphe de la Figure 1. Concernant la fonction objective, nous obtenons une valeur de $25469948,48 \approx 2,55.10^7$. Cette valeur correspond exactement à celle indiquée sur le site, ce qui valide la justesse de notre modélisation ainsi que de notre implémentation.

6.2. Big Data

6.2.1. Implémentation via ILS

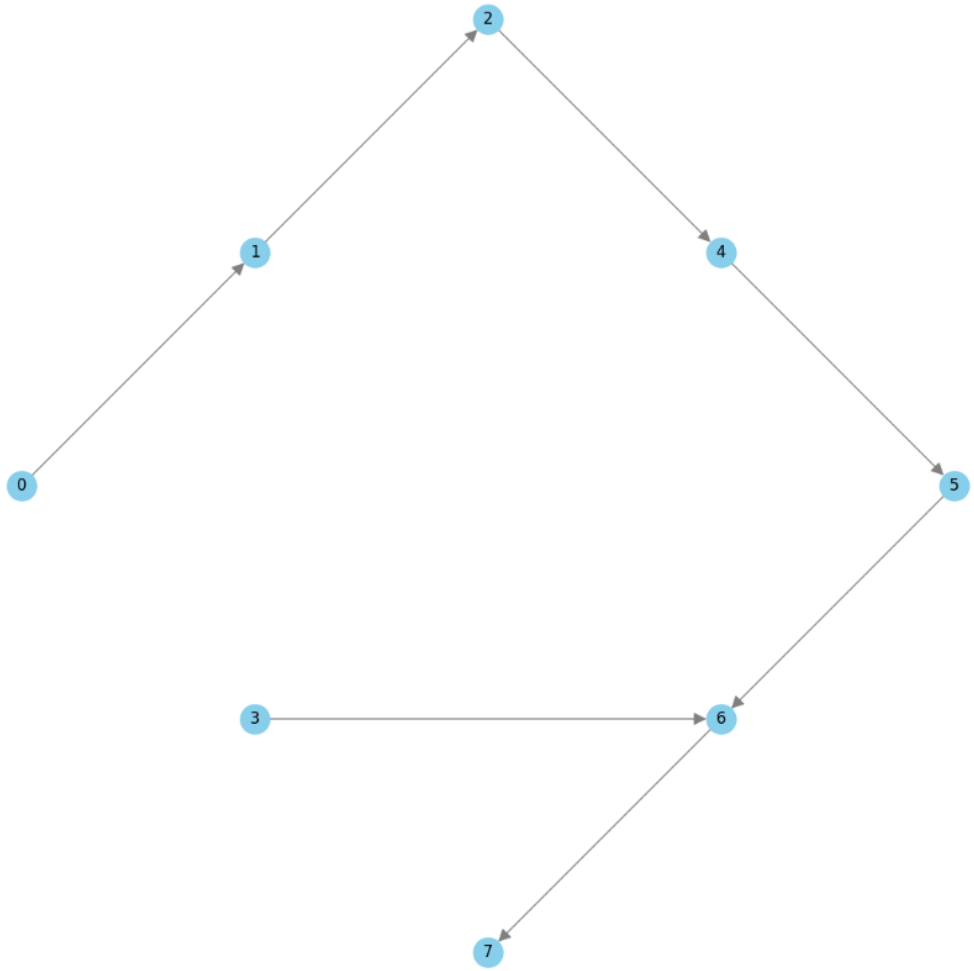


FIGURE 2 – Exemple de graphe obtenu à l'aide de l'algorithme ILS pour le jeu de Small Data

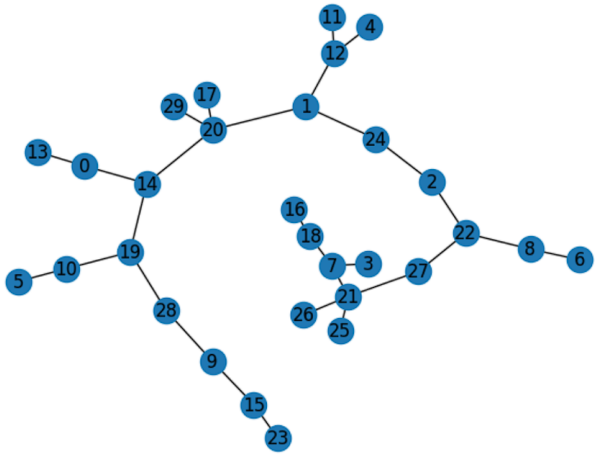
```
After 14  ILS iterations, the best solution is:
[3, 5, 4, 2, 1, 3]
with total cost: [26195180.72677859]
```

```
After 5  ILS iterations, the best solution is:
[0, 3, 4, 5, 6, 3]
with total cost: [27139969.56760589]
```

```
After 2  ILS iterations, the best solution is:
[5, 4, 2, 1, 0, 3]
with total cost: [25765736.92563972]
```

FIGURE 3 – Résultats obtenus en appliquant l'algorithme ILS sur la Small Data

Une fois que l'algorithme exact, via la librairie PULP, a été implémenté, nous avons commencé à coder l'algorithme ILS, dans l'objectif de l'appliquer sur la Big Data. Pour valider notre code, nous avons d'abord testé notre algorithme ILS sur la Small Data. Les premiers résultats obtenus étaient globalement satisfaisants, avec une certaine stabilité d'un essai à l'autre (cf. Figure 2 et Figure 3).




```
After 4  ILS iterations, the best solution is:
[27, 11, 17, 20, 4, 8, 28, 9, 28, 16, 22, 14, 7, 18, 3, 19, 8, 1, 27, 10, 27, 28, 23, 28, 5, 14, 27, 17]
with total cost: [6.44654305e+08]
```

FIGURE 4 – Résultat obtenu à l'aide de l'algorithme ILS pour le jeu de Big Data

Nous avons ensuite appliqué l'ILS sur la Big Data (cf. Figure 4) et comparé la valeur de la fonction objective obtenue à la valeur théorique connue : $2,484863919811.10^7$. La valeur obtenue nous semble bien éloignée de celle que nous devrions avoir, mais au vu de nos résultats sur la Small Data, nous en avons conclu que notre modélisation était correcte.

Malgré tout, au moment de passer à l'adaptation des différents algorithmes à nos différents scénarios, nous avons rencontré plusieurs difficultés avec l'ILS. Ainsi, nous avons poursuivi notre projet avec la librairie PULP puisque cette dernière nous donnait les résultats attendus pour la Big Data.

6.2.2. Implémentation via PULP

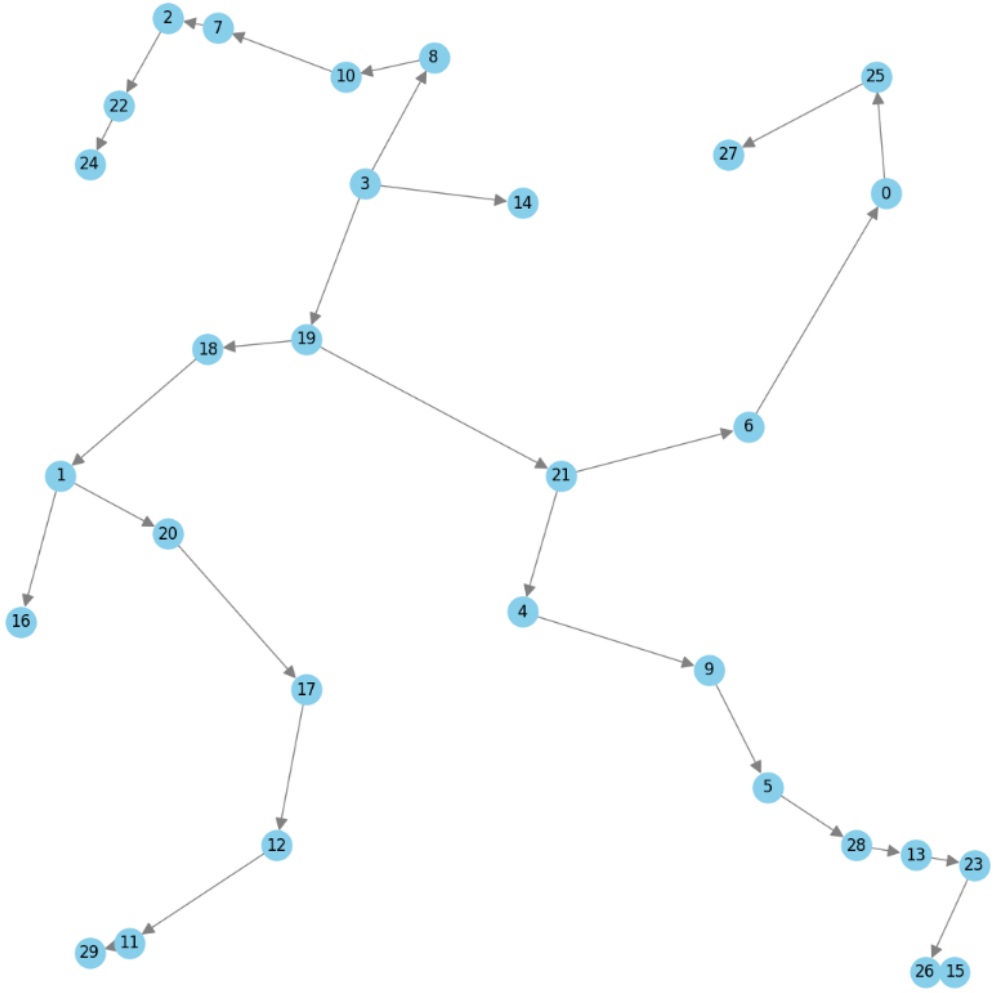


FIGURE 5 – Graphe obtenu à l'aide de PULP pour le jeu de Big Data

Après avoir essayé de résoudre le problème d'optimisation du jeu de la Big Data, à l'aide de l'ILS (Iterative local search) qui est un algorithme approché, nous cherchons maintenant à résoudre le même problème, mais avec l'aide de la bibliothèque PULP. Nous obtenons alors le graphe de la Figure 5 et pour la fonction objective, nous obtenons une valeur de $29858017,09 \simeq 2,99.10^7$ tandis qu'avec l'algorithme ILS, nous obtenons une valeur de $6,48.10^8$.

On constate ainsi que la valeur donnée par PULP est bien plus proche de celle attendue que la valeur donnée par l'algorithme ILS. De plus, alors que l'algorithme ILS prend une bonne vingtaine de minutes à s'exécuter, PULP nous donne les résultats en quelques secondes seulement.

Puisque notre problème d'optimisation est un problème de minimisation des charges, le résultat obtenu avec PULP est nettement meilleur, ce qui est cohérent avec le fait qu'il s'agit d'un solveur exact, contrairement à l'ILS, qui reste un algorithme approché pour la résolution de problèmes d'optimisation.

6.3. Scénario 1 : Equity

	A	B	C	D	E	F	G	H
1	0	0.6064265725	0.184295933	0.342054454	0.006379748232	0.8319474663	0.4898669865	0.7975489937
2	0.6064265725	0	0.591476748	0.02456517183	0.004388879881	0.3455100828	0.5795079582	0.3290890553
3	0.184295933	0.591476748	0	0.1016465302	0.8389053573	0.3463036712	0.02619258909	0.3146882488
4	0.342054454	0.02456517183	0.1016465302	0	0.7228169647	0.05084068004	0.2246224178	0.482360691
5	0.006379748232	0.004388879881	0.8389053573	0.7228169647	0	0.09158144024	0.7616193466	0.3116714888
6	0.8319474663	0.3455100828	0.3463036712	0.05084068004	0.09158144024	0	0.08265822186	0.5287137718
7	0.4898669865	0.5795079582	0.02619258909	0.2246224178	0.7616193466	0.08265822186	0	0.8155627204
8	0.7975489937	0.3290890553	0.3146882488	0.482360691	0.3116714888	0.5287137718	0.8155627204	0

FIGURE 6 – Modélisation de la priorité avant les modifications du site

Avant les nouveaux documents sur le site de PRONTO, nous avions tenté de créer un tableau (cf. Figure 6) de valeurs comprises entre 0 et 1 qui modélise la priorité de chaque consommateur : 0 étant très faible en priorité et 1 étant très haute en priorité.
Cependant, nous avons constaté que des modifications ont eu lieu sur le site du projet [1] et avons arrêté notre modélisation. À la place, nous avons opté pour une modélisation qui sera plus cohérente avec les nouvelles consignes, mais aussi avec les nouvelles données disponibles sur le fichier Excel intitulé "InputDataEnergyEquity". Ceci a impliqué certaines modifications :

- Alors que dans notre première modélisation nous servions tous les consommateurs, dans la nouvelle, nous cherchons seulement à maximiser nos gains ce qui implique que certaines demandes ne seront pas satisfaites.
- Désormais, le nouveau jeu de données associe chaque nœud i du graphe à un paramètre ConsumerType (i) allant de 1 à 3 ce qui traduit les "priorités" évoquées précédemment.
- Un autre paramètre a aussi fait son apparition : EquityThreshold. Ce paramètre représente, pour chaque type de consommateur, la proportion minimale de service à garantir.
- Certains paramètres ont disparu de la modélisation, ou du moins ne sont plus donnés directement, tels que EdgesDemandPeak(dij) et EdgesDemandAnnual(Dij). À présent, leurs valeurs doivent être calculées à partir des nouveaux paramètres EdgesDemandPeak(i), SurfaceConsumption (i) et Surface(i).

6.3.1. Les contraintes

Afin de prendre en compte ces changements, il est nécessaire de modifier nos contraintes. Ainsi, nous avons les nouvelles contraintes 1 et 8 :

- **Contrainte 1 :** Alors que notre précédente contrainte imposait certes une structure d'arbre, elle imposait aussi au graphe d'être connexe ce qui oblige le réseau à répondre à l'ensemble des demandes. Désormais, cette contrainte impose qu'un nœud i soit desservi par au maximum une seule arête entrante. Nous conservons alors la structure d'arbre sans pour autant répondre à toutes les demandes.

$$\forall i \in V \backslash V_0 : \quad \sum_{j \neq i} X_{j,i} \leq 1 \tag{17}$$

- **Contrainte 8 :** Assure que la proportion des arêtes desservant les bâtiments de type t est supérieure ou égale à la limite définie par EquityThreshold $(t) = \mu_t$.

$$\forall t : \quad \sum_{i \in V \backslash V} \sum_{j \in V, i \neq j} X_{j,i} \times N_{i,t} \geq \mu_t \times \sum_{l,k \in V, l \neq k} X_{l,k} \tag{18}$$

  Avec :

  — $N_{i,t}$ qui vaut 1 quand i est de type t et 0 sinon

  — μ_t est compris entre 0 et 1 : représente la proportion minimale des arêtes desservant les bâtiments de type t par rapport au nombre total d'arêtes construites.

6.3.2. La fonction objective

Concernant le calcul de la fonction objective, elle se calcule de la même manière que dans la modélisation précédente, à la différence qu'il faut calculer de nous-mêmes les paramètres $d_{i,j}$ et $D_{i,j}$.

- **Calcul de $d_{i,j}$:**

  Le paramètre EdgesDemandPeak(i) traduit la demande maximale en énergie du nœud i. Étant donné que la source ne consomme pas, et que chaque nœud transmet uniquement l'énergie nécessaire au nœud qu'il dessert directement, la demande maximale sur l'arête $e_{i,j}$ est EdgesDemandPeak (j).

  $\forall i, j \in V$ tels que $e_{i,j} \in E$ et $i \neq j$:

$$d_{i,j} = \text{EdgesDemandPeak}(j) \tag{19}$$

- **Calcul de $D_{i,j}$:**

  Le paramètre SurfaceConsumption(i) traduit la consommation d'énergie par unité de surface (kW/m^2). Ainsi, de manière analogue au calcul de $d_{i,j}$:

  $\forall i, j \in V$ tels que $i \neq j$:

$$D_{i,j} = \text{SurfaceConsumption}(j) \times \text{Surface}(j) \tag{20}$$

6.3.3. Les résultats

L'intégration des nouvelles données issues de **InputDataEnergyEquity** nous a d'abord posé plusieurs défis. En effet, nous avons rencontré des difficultés à déterminer les bonnes unités des différents paramètres dans le fichier Excel, ce qui a conduit notre algorithme à ne construire aucune arête (aucun réseau de chaleur). Cela nous a semblé étrange au départ. Pour y remédier temporairement, nous avons ajouté une contrainte 0, qui correspondait en fait à l'ancienne contrainte 1, afin de forcer l'algorithme à construire des arêtes.

Nous avons alors constaté que la fonction objective était négative (nous subissions uniquement des pertes). Or, comme notre objectif est de maximiser le gain, l'algorithme préférait ne pas construire d'arêtes pour rester à zéro.

Après avoir correctement déterminé les unités des paramètres, l'algorithme a commencé à fournir des valeurs positives pour la fonction objective.

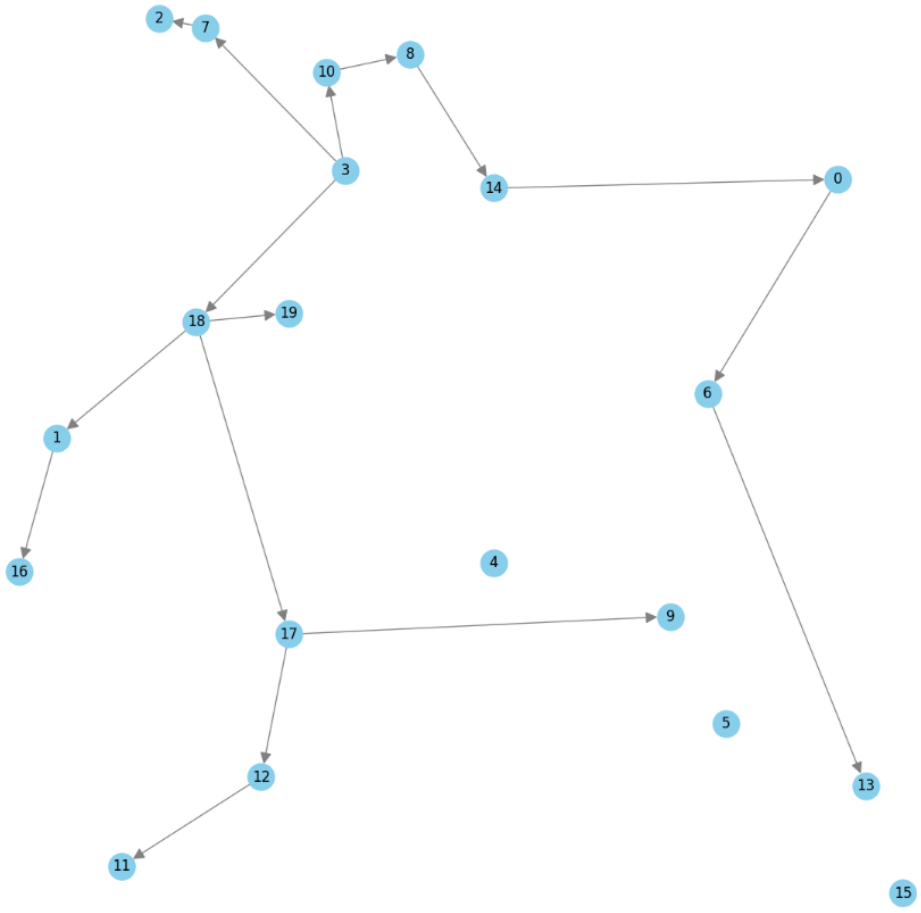


FIGURE 7 – Graphe obtenu pour le scénario 1 de Equity

Nous obtenons finalement une valeur positive de la fonction objective (931 591, 54), ce qui indique un gain. Comme on peut le voir dans le graphe, certains nœuds ne sont pas desservis (notamment les nœuds 4, 5 et 15), ce qui confirme que l'algorithme cherche à maximiser le gain plutôt qu'à desservir tous les bâtiments.

6.4. Scénario 2 : Résilience du réseau de chaleur

6.4.1. Quelques notions importantes

Avant de s'intéresser, à proprement parler, à l'algorithme du scénario, nous avions pensé à créer un graphe qui resterait connexe (en partant du principe que notre graphe initial est lui-même connexe) même en retirant une arête. Pour cela, il faut définir quelques notions :

- **k-connexité ou k-connectivité :**

  Un graphe est dit k-connexe s'il possède au moins deux sommets et que, pour chaque paire de sommets, il existe k chaînes (ou chemins) indépendantes reliant ces sommets. En particulier, un graphe bi-connexe est un graphe dans lequel, pour chaque couple de sommets, il existe au moins deux chemins disjoints les reliant.

- **Théorème d'Eswaran et Tarjan :**

  Par souci de complexité de ce théorème applicable pour tout graphe non orienté, nous nous contenterons de la version pour un arbre, et qui convient à notre situation. Soit G_0 un arbre non orienté, alors il est possible de le rendre biconnexe (2-connexe) en ajoutant un nombre minimal d'arêtes donné par (p étant le nombre de feuilles) :

$$\left\lceil \frac{p}{2} \right\rceil$$

- **Breadth-First Search (BFS) :**

  L'algorithme de parcours en largeur (BFS) explore un graphe à partir d'un nœud source en visitant d'abord tous ses voisins immédiats, puis leurs voisins, et ainsi de suite, couche par couche. Il utilise une file FIFO (First In First Out) pour gérer l'ordre de visite et marque chaque sommet dès sa découverte afin de ne pas le revisiter. En temps, il s'exécute en $O(|V| + |E|)$ et permet notamment de trouver le plus court chemin, non pondéré, entre la source et tous les autres nœuds.

- **Dijkstra :**

  L'algorithme de Dijkstra permet de déterminer, dans un graphe orienté ou non orienté à poids non négatifs, le plus court chemin entre un sommet source et tous les autres sommets. Il initialise la distance de la source à zéro et celles des autres nœuds à l'infini, puis, à chaque itération, sélectionne le nœud non traité au coût minimal, met à jour les distances de ses voisins en comparant le chemin passant par ce nœud aux distances déjà connues, et marque finalement le nœud comme traité. Le processus se répète jusqu'à ce que tous les sommets aient été fixés, garantissant des distances optimales en $O(|E| + |V| \log(|V|))$ avec une file de priorité.

- **Arborescence :**

  Une arborescence est un arbre orienté acyclique où un nœud spécial, appelé racine, dispose d'un unique chemin dirigé vers chacun des autres sommets. Elle contient exactement $|V| - 1$ arêtes pour $|V|$ nœuds, garantit l'accessibilité de tout nœud depuis la racine et interdit tout cycle, formant ainsi une structure hiérarchique idéale pour modéliser des flux, des hiérarchies ou des schémas de distribution.

- **Algorithme de Chu-Liu/Edmonds :**

  L'algorithme de Chu-Liu/Edmonds permet, dans un graphe orienté à poids, de construire une

arborescence couvrante de poids minimale (minimum spanning arborescence). Ce dernier peut ensuite être modifié afin d'être enraciné en un nœud donné.

6.4.2. Construction des arêtes

La première étape de notre algorithme consiste à construire les arêtes secondaires, destinées à être utilisées en cas de panne. Dans notre modélisation, nous supposons que les pannes sont peu fréquentes et ne peuvent survenir simultanément en deux endroits différents. Il s'agissait donc de rendre notre graphe 2-connexe, et plusieurs approches ont été envisagées pour atteindre cet objectif.

- **Première tentative :**

  L'idée de cette première tentative est relativement intuitive : si nous voulons qu'un graphe reste connexe même en supprimant une arête, alors il suffit que chaque sommet ait au moins trois arêtes. Pour cela, nous ajoutons une contrainte au problème qui impose qu'à chaque sommet, il existe au moins trois arêtes (entrantes et sortantes). Par conséquent, nous supprimons la première contrainte, puisque le graphe ne constitue plus un arbre. Cette approche repose sur le fait que, si une panne survient sur n'importe quelle arête, il est possible de modifier l'orientation des deux autres arêtes restantes afin de conserver au moins une arête entrante et une arête sortante (sauf dans le cas des feuilles de l'arbre). La contrainte ajoutée est la suivante :

$$\forall j \in V, \sum_{i \in V, i \neq j} X_{i,j} + \sum_{k \in V, k \neq j} X_{j,k} \geq 3 \tag{21}$$

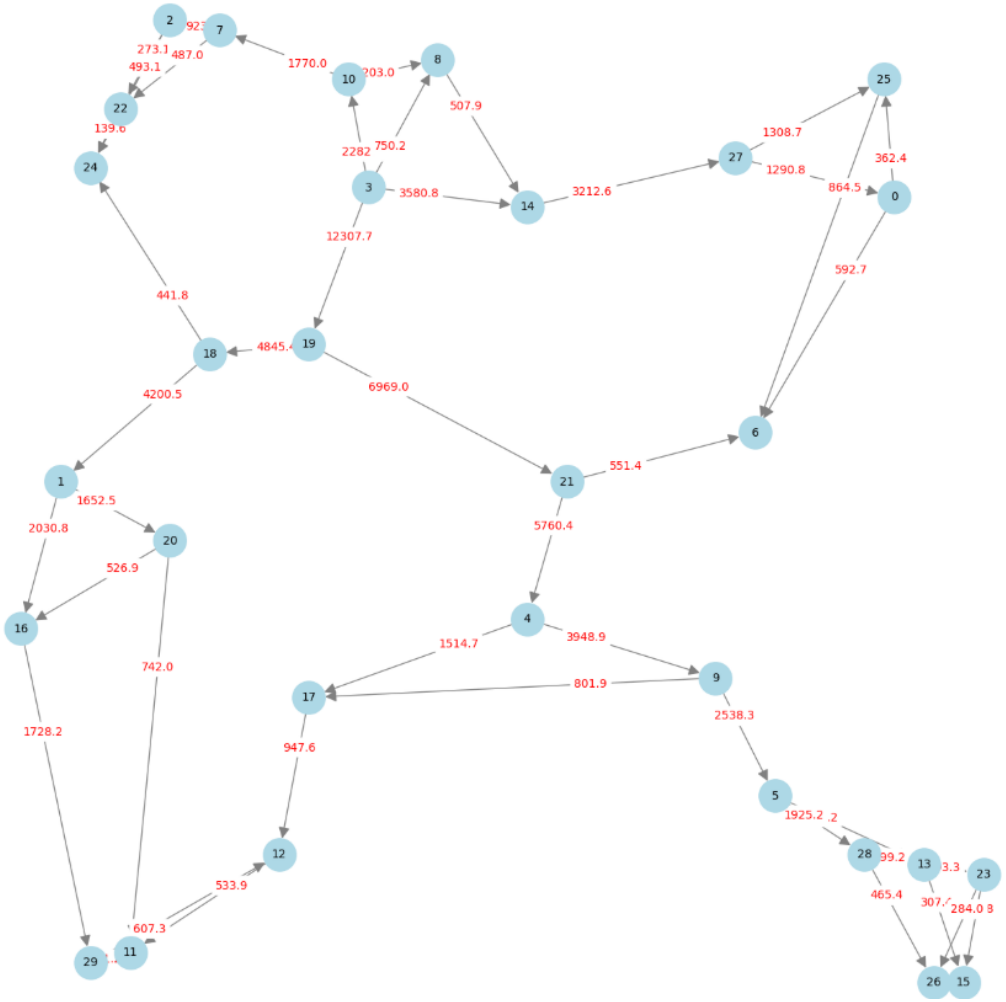


FIGURE 8 – Graphe de la tentative 1 de construction d'arêtes

Malheureusement, en appliquant l'algorithme `is_k_edge_connected` de la bibliothèque Python Networkx qui détermine si un graphe donné est k-connexe, nous trouvons que le graphe obtenu (cf. Figure 8) n'est pas bi-connexe. Il est donc possible qu'un sommet ne soit pas connecté à la source, rendant l'algorithme inutilisable dans notre contexte. Et visuellement, sur la Figure 8, nous remarquons que s'il y a une coupure sur l'arête $(9, 5)$, les nœuds $5, 28, 13, 23, 26$ et 15 seront déconnectés de la source.

- **Deuxième tentative :**

  L'idée de notre deuxième tentative est elle aussi assez intuitive. Le principe est simple : nous itérons sur l'ensemble des arêtes du graphe et nous la supprimons temporairement. Pendant ce temps, nous appliquons la fonction `has_path` de la bibliothèque Python Networkx, qui détermine s'il y a un chemin entre un couple (u, v) de sommets. S'il n'existe pas de chemin, nous générons toutes les paires possibles d'arêtes de secours à partir de l'ensemble des voisins de u et aussi ceux de v ,et nous choisissons aléatoirement une arête dans cet ensemble. Dans le cas contraire, nous remettons l'arête que nous avions supprimée et nous sortons de la boucle. Pour pouvoir utiliser les fonctions disponibles dans la bibliothèque Networkx, il faut faire attention à convertir notre graphe orienté en un graphe non orienté.

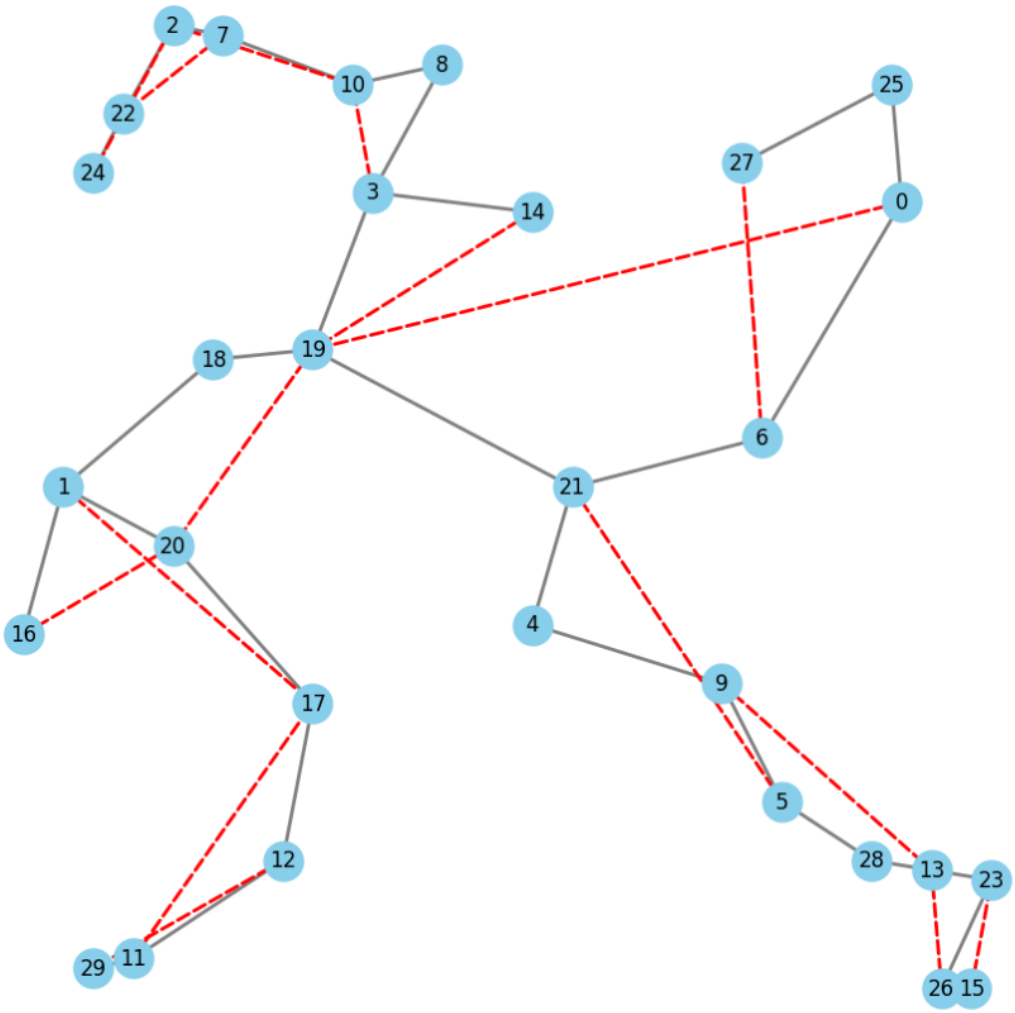


FIGURE 9 – Graphe de la tentative 2 de construction d'arêtes

Avec ce procédé, nous obtenons bien un graphe 2-connexe capable d'assurer le service en cas de panne, cependant, en respectant les résultats obtenus, nous construirions une trop grande quantité de canaux (plus que nécessaire) ce qui n'est pas raisonnable si nous souhaitons économiser.

- **Troisième tentative :**

  La troisième et dernière tentative utilise essentiellement une fonction de la bibliothèque Networkx, appelée `k_edge_augmentation`. Cette fonction implémente un algorithme optimal qui ajoute le nombre minimal d'arêtes pour rendre un graphe k-connexe. Selon le théorème d'Eswaran et Tarjan (cf. 6.4.1), nous nous attendons à ce que l'algorithme crée trois arêtes de secours pour six feuilles (cf. Figure 5 pour voir le graphe initial).

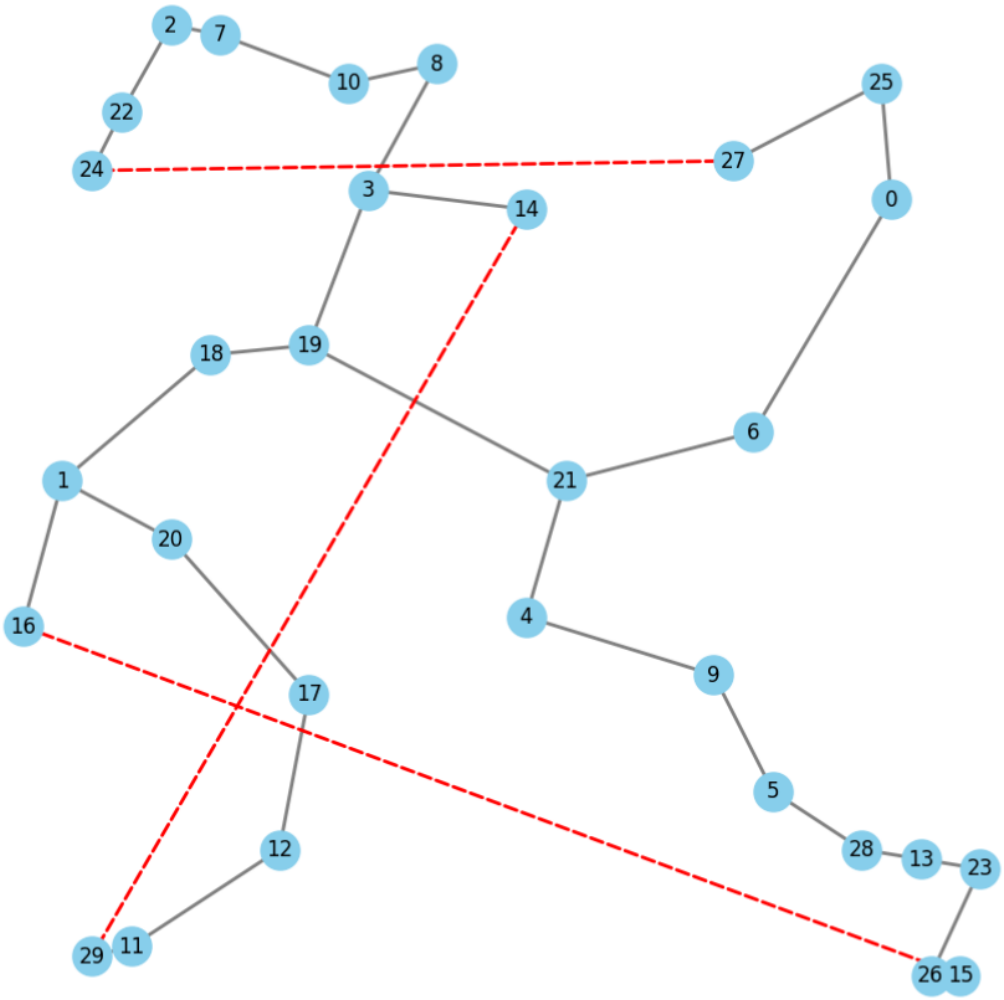


FIGURE 10 – Graphe de la tentative 3 de construction d'arêtes

Nous constatons sur la Figure 10 que nous obtenons bien ce à quoi nous nous attendions. De plus, nous remarquons qu'en réalité, l'algorithme relie les différentes feuilles du graphe. Cette tentative est beaucoup plus performante en termes de coût, puisque nous minimisons le nombre d'arêtes à construire tout en garantissant la connexité du graphe. La vérification par la fonction `is_k_edge_connected` confirme le résultat voulu : le nouveau graphe obtenu après l'ajout des arêtes de secours est bien bi-connexe.

6.4.3. Réorientation du graphe

Afin de construire les arêtes de secours, nous avons dû utiliser un graphe non orienté. Avec l'ajout de ces arêtes, un nouveau problème se pose donc : comment peut-on rediriger le flux à partir de la source vers tous les nœuds ?

La deuxième étape de notre algorithme consiste alors à réorienter notre graphe en cas de panne. Pour cela, nous avons implémenté une fonction 'panne' qui, à partir d'une arête défaillante, recalcule la configuration du réseau pour pouvoir réalimenter l'ensemble des nœuds depuis la source.

L'algorithme utilisé dans ce processus de réorientation repose sur un graphe pondéré et suit les étapes suivantes : nous supprimons d'abord l'arête défaillante, puis nous ajoutons les arêtes de secours issues de l'algorithme de résilience (cf. 6.4.2). Les poids utilisés correspondent aux distances $l_{i,j}$.

Afin de minimiser les pertes et la distance — et, par conséquent, d'optimiser la distribution —, nous calculons l'arbre couvrant minimal de ce graphe pondéré à l'aide de la fonction `minimum_spanning_tree` de la bibliothèque `NetworkX`. Cela garantit non seulement que tous les nœuds restent connectés avec un coût total minimal, mais aussi que l'utilisation des arêtes de secours soit limitée au strict nécessaire.

Vient ensuite l'étape la plus importante : la réorientation. Pour cela, nous avons opté pour un algorithme de parcours en largeur, directement implémenté à l'aide de la bibliothèque `NetworkX`. Cet algorithme exact assure une redirection optimale du flux, de la source vers les autres nœuds.

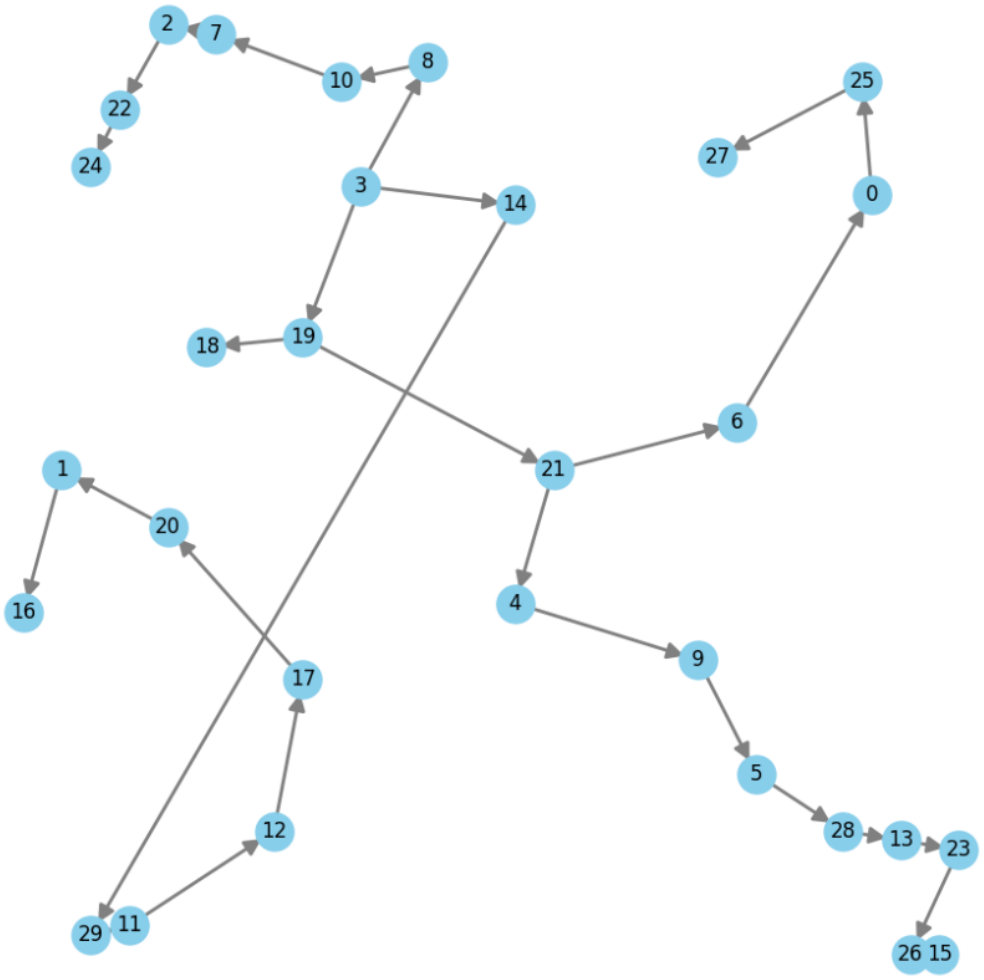


FIGURE 11 – Simulation de la panne (18, 1) en utilisant l'arête de secours (14, 29)

Après avoir implémenté ce scénario, nous avons constaté qu'en cas de suppression d'une arête, cela entraînait de nombreux changements de sens dans les arêtes restantes. Ce changement de sens de flux dans un contexte de réseau de chaleur, un réseau physique, nécessite des efforts et des coûts élevés, comme l'utilisation de pompes pour contrer la différence d'altitude ou les pertes de pression.

Cela nous a conduits à définir un nouveau scénario qui vise à minimiser les changements de sens des arêtes.

6.4.4. Minimisation des changements d'orientation

Accomplir l'objectif de réorienter le réseau en minimisant les inversions s'est avéré difficile. Globalement, nous avons eu trois idées, mais seule la dernière a donné des résultats pertinents.

- **Première tentative :**

  L'idée de cette tentative est de créer un graphe pondéré orienté à partir du graphe de la Figure 10 : nous définissons les poids de chaque arête dans ce graphe de sorte à ce que le sens direct de l'arête (i.e. celui qui correspond au sens dans le graphe de départ) soit pondéré par 0 , le sens inverse par une grande valeur (100 dans notre cas) et pour les nouvelles arêtes de secours non présentes dans le graphe de départ, elles sont pondérées par 1, pour les deux sens. Nous exécutons ensuite l'algorithme de Dijkstra depuis la source pour obtenir, pour chaque nœud, un chemin de coût minimal. Notre modélisation permet ainsi de prioriser le sens des arêtes initiales et si besoin, utiliser les arêtes de secours plutôt que de changer le sens des arêtes initialement présentes dans le graphe.

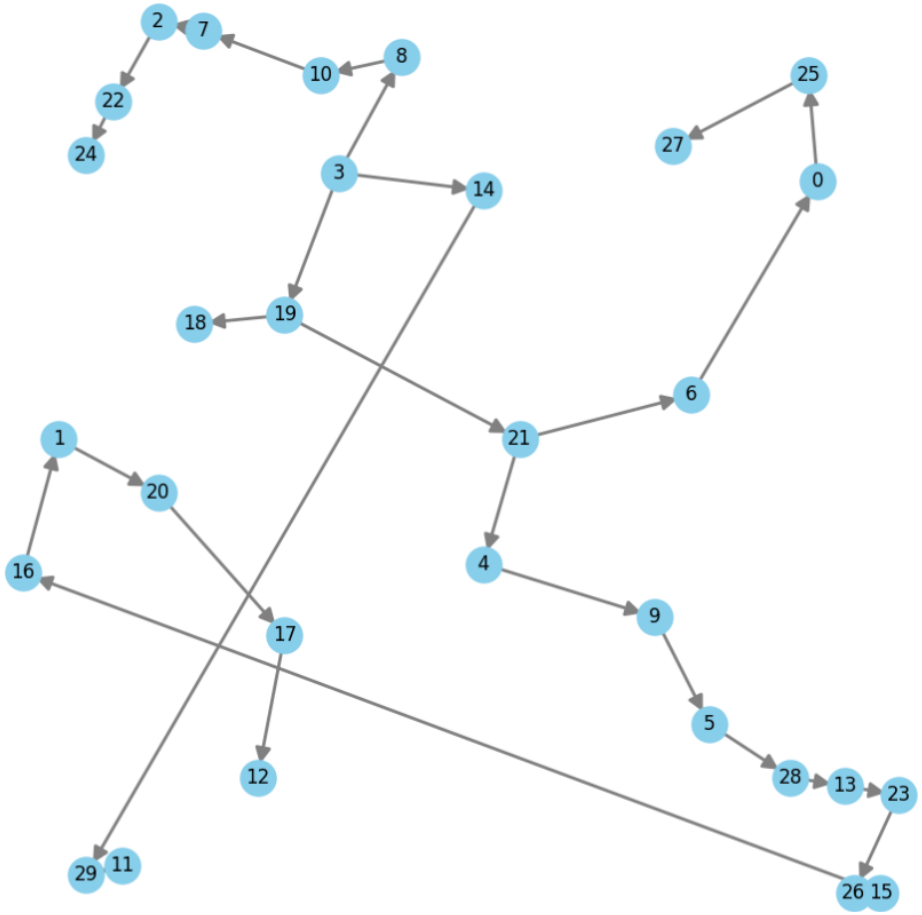


FIGURE 12 – Simulation de la panne (18, 1) pour la tentative 1 de minimisation

Cependant, cet algorithme n'est pas optimal. En effet, la solution attendue serait d'utiliser l'arête

de secours $(26, 16)$ qui modifierait seulement le sens de l'arête $(16, 1)$ or, ce n'est pas ce qui est proposé ici : nous remarquons effectivement l'utilisation de l'arête $(14, 29)$ ainsi que la suppression de l'arête $(12, 11)$. Ces modifications, notamment la suppression de l'arête $(12, 11)$ qui oblige un changement de sens de l'arête $(11, 29)$, ne sont pas nécessaires. Ce problème provient de l'utilisation de l'algorithme de Dijkstra qui garantit le plus court chemin entre la source et chaque nœud.

- **Deuxième tentative :**

  Cette deuxième tentative vise à chercher un équivalent de l'arbre couvrant minimal (MST : Minimum Spanning Tree) pour le cas d'un graphe orienté. En effet, l'algorithme de MST ne marche que pour des graphes non orientés. L'alternative que nous avons trouvée est l'algorithme de Chu-Liu/Edmonds (cf. 6.4.1). Dans la bibliothèque que nous utilisons, Networkx, cet algorithme est implémenté mais ne prend pas en considération la source, ce qui nous a poussé à chercher d'autres implémentations sur d'autres librairies ou projets disponibles sur Git. Cette démarche s'est avérée peu pratique : l'algorithme est peu exploité, et les seules sources que nous trouvons sont destinées au langage *C*. Finalement, nous avons décidé de résoudre ce problème manuellement. Le début de notre algorithme est identique à la première tentative : nous créons un graphe pondéré orienté de la même manière que précédemment. Pour contrer le problème de source, nous avons simplement supprimé l'entièreté des arêtes qui entraient dans la source puis, nous avons appliqué I'algorithme Chu-Liu/Edmonds implémenté sur Networkx. Cette petite modification nous a ainsi permis d'imposer une source.

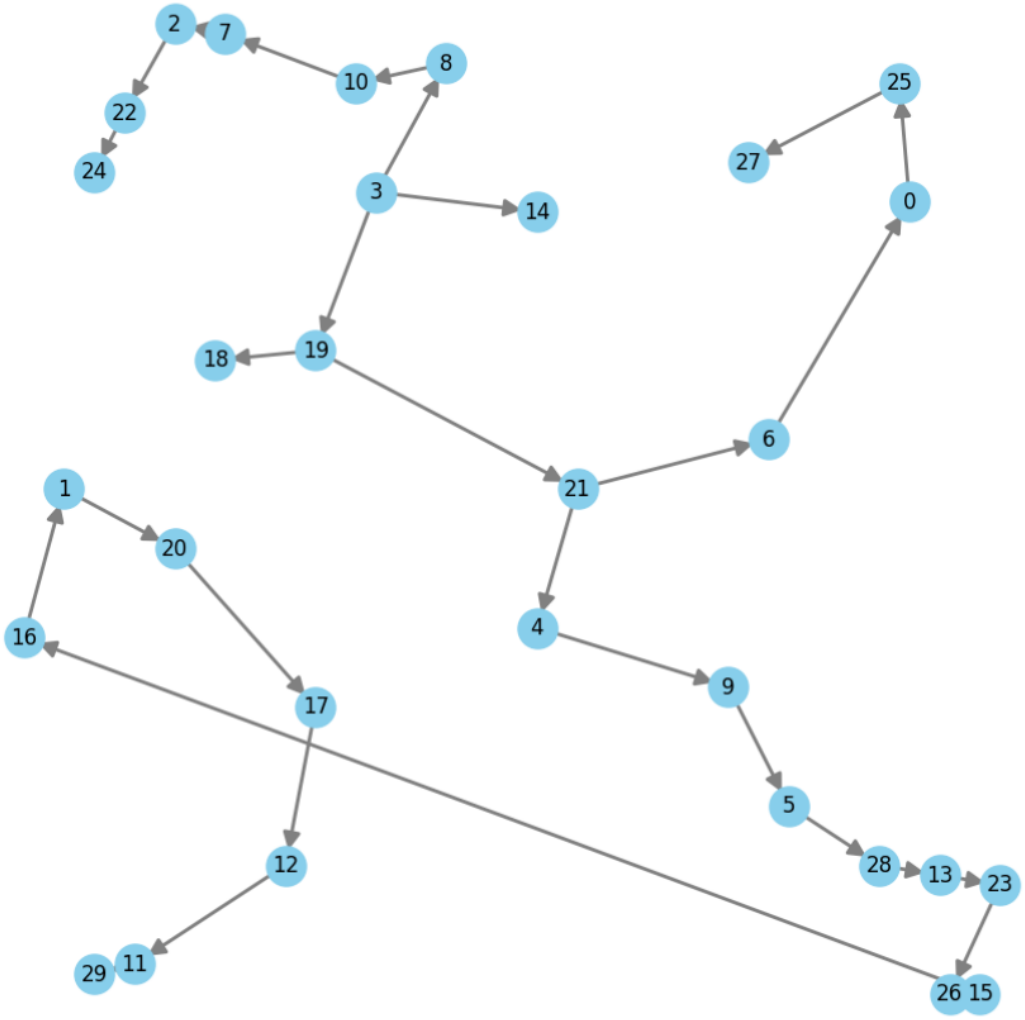


FIGURE 13 – Simulation de la panne $(18, 1)$ pour la tentative 2 de minimisation

```
plt.title("Graphe de secours")
plt.show()

Erreur lors de la construction de l'arborescence : No minimum spanning arborescence in G.
```

FIGURE 14 – Erreur dans la simulation pour la panne $(19, 21)$

On observe sur la Figure 13 que nous obtenons bien la solution attendue malheureusement, l'algorithme ne fonctionne que pour certaines pannes. Dans le cas présenté en Figure 14, la fonction retourne une erreur.

- **Troisième tentative :**

  Cette troisième et dernière tentative consiste à considérer la situation comme un problème d'optimisation afin de le résoudre avec la bibliothèque PULP. On commence de la même manière que les tentatives précédentes en créant un graphe pondéré par des poids suivant leur orientation initiale dans le graphe de départ. Nous déclarons ensuite le problème PULP, cette fois-ci en prenant des variables binaires de décision $Y_{i,j}$ qui sont uniquement dans l'ensemble des arêtes disponibles dans le graphe pondéré. Pour ce nouveau problème d'optimisation, nous n'avons besoin que de trois contraintes :

  — **La contrainte 2** qui garantit l'unidirectionnalité du flux.
  — **Les contraintes 1 et 8** qui, combinées, garantissent la structure d'arborescence.

  La nouvelle fonction objective doit cette fois-ci minimiser les redirections de flux :

$$\sum_{(i,j)\in D} Y_{i,j} w_{i,j} \tag{22}$$

  Avec :
  — D l'ensemble des arêtes du graphe pondéré construit au début de l'algorithme.
  — w_{ij} le poids de chaque arête e_{ij} (0 si elle a même sens que le graphe de départ, 100 si c'est le sens inverse et 1 si c'est une arête de secours).

  L'objectif est donc de minimiser cette fonction tout en respectant les trois contraintes citées ci-dessus.

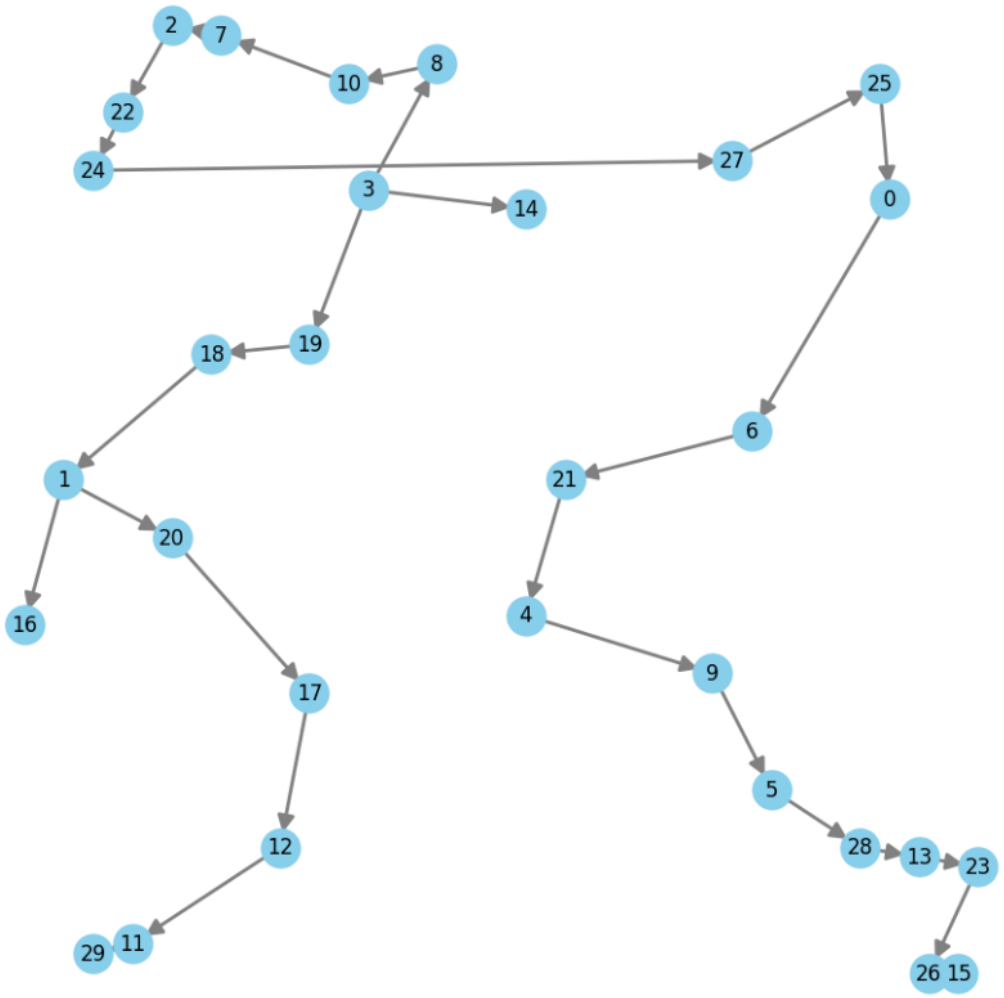


FIGURE 15 – Simulation de la panne (19, 21) pour la tentative 3 de minimisation

On réussit alors en testant avec toutes les pannes, à garantir un réseau redirigé avec le minimum de changements de sens de flux, même avec les pannes que l'algorithme appliqué dans la deuxième tentative n'a pas réussi à résoudre.

7. Notre organisation

7.1. Le travail d'équipe

Concernant l'organisation du groupe, nous nous retrouvions une à deux fois par semaine, selon nos disponibilités, afin d'avancer sur le projet. Tout au long du projet, nous avions utilisé différents outils de collaboration tels que :

- Google Drive, pour regrouper les fichiers Google Colab
- WhatsApp, afin de communiquer entre nous, fixer des dates et des lieux de rendez-vous, mais aussi pour suivre l'avancement du projet dans le cas où nous coderions depuis chez nous
- Partage IMT, pour le diagramme de Gantt
- Discord, pour la possibilité de créer différents salons à thème afin de ne pas tout mélanger

Globalement, le groupe fonctionne de manière très satisfaisante. Chaque membre participe activement au projet, en s'impliquant dans les différentes tâches. En cas de désaccord, nous privilégions toujours la communication : nous prenons le temps d'échanger collectivement afin d'écouter les points de vue de chacun et de parvenir à un compromis satisfaisant pour tous.

7.2. Les différents échelons

L'un des outils qui nous a vraiment aidés à nous organiser tout au long du projet est le diagramme de Gantt. Grâce à lui, nous avions une vision claire des tâches à accomplir à chaque séance. Dès notre arrivée, chacun savait ce qu'il avait à faire et combien de temps y était consacré. Cela nous a permis de gagner en efficacité, mais aussi de mieux nous coordonner. Cet outil a joué un rôle clé dans la dynamique positive du groupe : tout le monde avançait dans la même direction.

Dans l'ensemble, nous avons réussi à respecter le planning que nous nous étions fixé. Cela dit, nous avons traversé une période un peu plus compliquée : nous étions bloqués par un problème technique pour lequel nous n'avions pas de solution immédiate. Ce moment de doute a naturellement entraîné un peu de retard. Mais une fois l'obstacle dépassé, nous avons su nous remettre en route : en nous répartissant les tâches différemment et en travaillant de manière plus intensive, nous avons réussi à rattraper notre retard et à revenir sur le bon rythme.

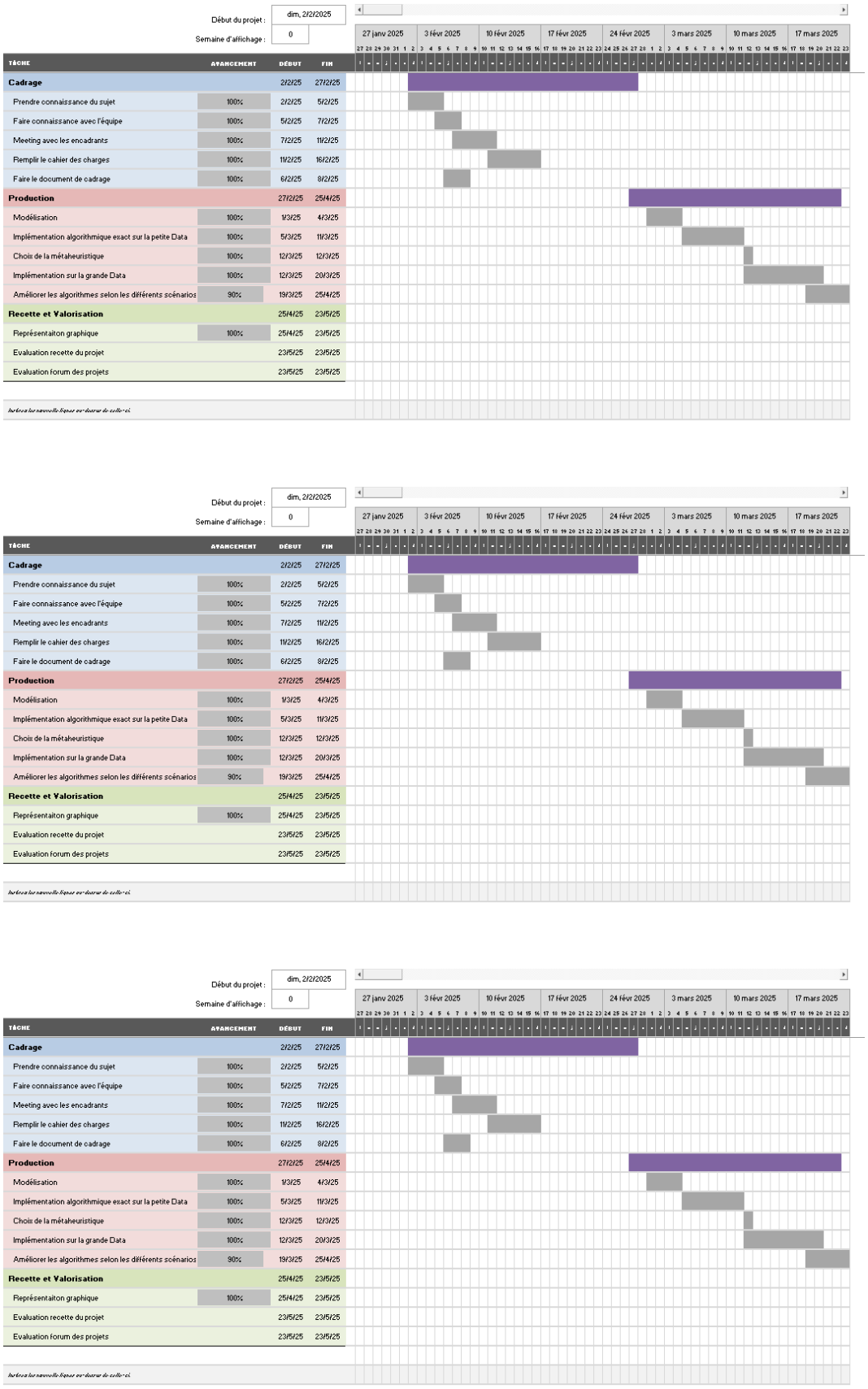


FIGURE 16 – Diagramme de Gantt du projet

8. Conclusion

Au terme de ce projet, nous avons pu évaluer la pertinence de nos scénarios à travers la modélisation et l'implémentation d'algorithmes adaptés à des problématiques concrètes. Le scénario d'équité, bien qu'initialement complexe à mettre en œuvre en raison des nouvelles données et des unités peu explicites, a fini par produire des résultats cohérents, illustrant la possibilité d'intégrer des critères sociaux tout en optimisant les performances économiques du réseau. Le scénario de résilience, quant à lui, s'est révélé particulièrement riche et pertinent d'un point de vue algorithmique. Il nous a poussés à explorer plusieurs approches (heuristiques, théoriques et exactes), jusqu'à aboutir à une solution fiable permettant de maintenir la connectivité du réseau en cas de panne, tout en maîtrisant le coût des ajouts d'arêtes.

En revanche, nous avons fait le choix de ne pas implémenter le scénario 3 sur la minimisation des pertes thermiques, pour des raisons de temps et de priorisation. En effet, ce scénario aurait nécessité une reformulation complète du modèle, notamment pour prendre en compte les longueurs cumulées des arêtes dans la fonction objective, ce qui aurait mobilisé des ressources que nous avons préféré consacrer à la consolidation du scénario 2. Ce dernier répondait à une problématique plus critique dans le contexte de réseaux physiques réels : garantir la continuité de service.

Nos résultats, bien qu'encourageants, présentent toutefois certaines limites. Les performances des algorithmes approchés (notamment ILS) se sont révélées insuffisantes face à la complexité des données Big Data, ce qui nous a conduits à privilégier l'usage de solveurs exacts comme PULP. Enfin, l'un des points d'amélioration majeurs reste la prise en compte du réalisme physique du réseau, notamment en matière d'orientation des flux et de contraintes opérationnelles, sujet que nous avons seulement effleuré dans la dernière partie du projet.

Ce projet nous a permis de mettre en œuvre une approche complète de résolution de problème : modélisation, implémentation, tests, ajustements, tout en maintenant une dynamique d'équipe solide. Il ouvre également des pistes intéressantes pour des développements futurs, notamment autour de l'optimisation multi-objectifs et de la simulation dynamique du réseau.

9. Références bibliographiques

Références

[1] PRONTO, https://formations.telecom-bretagne.eu/or/courses/PRONTO-DecisionAiding/

[2] Weisstein Eric W. k-Connected Graph. From MathWorld - A Wolfram Web Resource. https://mathworld.wolfram.com/k-ConnectedGraph.html

[3] Eswaran K. P., Tarjan R. E. «Augmentation problems ». SIAM Journal on Computing. Volume 5. Numéro 4. 1976. Pages 653-665. https://doi.org/10.1137/0205044

[4] J. Edmonds, «Optimum Branchings ». Journal of Research of the National Bureau of Standards, 1967, Vol. 71B, p.233-240, https://archive.org/details/jresv718n4p233

Annexes

Annexe 1 – Google Colab

Lien du Google Drive avec accès aux codes Python :
https://drive.google.com/drive/u/0/folders/1TdfzwTMAYT-evmhyEqpgjY7IxIqePqAu

Annexe 2 – Retours individuelles

- **Mélina WANG**

  Lors de la réalisation d'un projet en équipe, nous devions produire un rapport final synthétisant notre travail. Chaque membre contribuait activement à la rédaction, en développant les parties qui lui étaient attribuées. Cependant, au fil de l'avancement, j'ai constaté un manque de cohérence dans le document : certaines sections étaient rédigées en anglais alors que d'autres étaient en français, les figures et tableaux ne suivaient pas le même format, les polices d'écriture variaient d'une section à l'autre, et la mise en page (marges, indentations, alignements) était très irrégulière, ce qui rendait la lecture particulièrement désagréable.

  Face à cette situation, j'ai décidé de prendre l'initiative en réunissant le groupe pour partager mes observations. J'ai expliqué, de manière constructive, pourquoi une présentation cohérente était essentielle : un document désorganisé pouvait donner une impression de travail peu soigné, même si le contenu était pertinent. Malgré quelques réticences – certains pensaient que ces détails avaient peu d'importance – j'ai maintenu ma position en argumentant que ces éléments influenceraient nécessairement la perception du lecteur, notamment lors de l'évaluation. J'ai proposé des solutions concrètes : unifier la langue du document en utilisant uniquemment le français (à l'exception de Small et Big Data qui avaient été définis dans le résumé), standardiser les styles de figures et de tableaux (titres, légendes, numérotation), harmoniser la police et sa taille, corriger les marges, indentations et autres éléments de mise en page. J'ai ensuite pris en charge la relecture et la normalisation de l'ensemble du rapport, tout en intégrant les remarques et préférences de chacun.

  À la suite de ces modifications, le rapport était bien plus cohérent et agréable à lire. Cette expérience m'a permis d'apprendre que tout le monde ne porte pas forcément attention aux aspects formels ou visuels d'un document, même si ces éléments jouent un rôle important dans la qualité perçue du travail. J'ai compris que ce qui peut sembler secondaire pour certains peut, au contraire, avoir un impact significatif sur la clarté et la fluidité de lecture. Cela m'a amenée à prendre du recul sur la manière de collaborer efficacement : il ne s'agit pas seulement de diviser les tâches, mais aussi d'assurer une cohérence d'ensemble.

- **Mohamed Amine JANATI**

  Au fil de notre projet PRONTO, nous avons été confrontés à plusieurs situations où plusieurs solutions s'offraient à nous. Dans le cadre de la réorientation des flux en cas de panne, avec pour objectif de minimiser le nombre de réorientations, notre premier débriefing nous a poussés à envisager l'utilisation de l'algorithme de Dijkstra. Cet algorithme, bien connu des cours de classe préparatoire et du module PYRAT du semestre 5, nous semblait tout indiqué. Lors des premiers tests, l'algorithme donnait des résultats convaincants. Cependant, ce succès fut de courte durée : à l'occasion d'une visualisation graphique, Mélina a relevé une solution plus pertinente que celle fournie par Dijkstra. Cela nous a surpris, car nous pensions que l'algorithme garantissait une solution optimale.

  Après une analyse approfondie, nous avons compris que le cadre d'application de Dijkstra ne correspondait pas exactement à notre problème, et qu'il nous fallait envisager une autre approche. Nous avons alors pensé à l'algorithme de Chu-Liu/Edmonds, adapté aux graphes orientés pour une minimisation des traversées. Bien que théoriquement approprié, son implémentation pratique s'est avérée difficile, en raison du faible nombre de bibliothèques Python disponibles. Après plusieurs essais, Mélina a finalement réussi à contourner ces obstacles et faire fonctionner l'algorithme. Toutefois, lors du changement de scénario de panne, une nouvelle erreur est apparue, remettant en question la fiabilité de cette solution. Nous étions donc revenus au point de départ.

  Face à cette impasse, j'ai eu l'idée d'aborder le problème sous un angle différent. Puisqu'il s'agissait d'une question d'optimisation, pourquoi ne pas la traiter avec des outils d'optimisation ? J'ai donc modélisé le problème sous forme d'un programme linéaire de minimisation via la bibliothèque PULP, où l'objectif était de réduire au maximum le nombre de réorientations. Cette fois-ci, la solution a répondu à toutes nos attentes : le code fonctionnait quel que soit le scénario de panne. Nous avons ainsi réussi à résoudre le deuxième scénario du projet avec rigueur et efficacité.

- **Mohamed RHARRASSI**

  Lors de la phase « scénarios de résilience » de notre projet de réseau de chaleur — dont l'objectif était de garantir la continuité de service en cas de panne d'une arête — nous devions rendre le graphe initial, un arbre, capable de supporter la suppression de n'importe quelle arête sans perdre la connexion entre la source et chaque nœud.

  Nous avons d'abord mis en œuvre une contrainte de degré minimal en imposant qu'à chaque sommet, le degré (entrantes + sortantes) soit au moins de trois.Malheureusement le graphe donné n'est pas 2-connexe et le coût en nouvelles arêtes restait trop élevé.

  Ensuite j'avais l'idée d'utiliser l'algorithme : pour chaque arête, je la retirais temporairement et utilisais la fonction `has_path(u, v)` de NetworkX pour savoir s'il y'a un chemin alternatif entre

u et v,autre que celui que je viens de supprimer. Si il n y'a pas de chemin alternatif, l'algorithme génére toutes les arêtes de secours possibles entre les voisins à partir des voisins de u et ceux de v , puis il en sélectionne une aléatoirement. Cette méthode assurait la 2-connexité, mais produisait davantage d'arêtes qu'il n'était nécessaire, augmentant significativement la taille du réseau.

Enfin, nous avons adopté la fonction `k_edge_augmentation` de NetworkX pour calculer de manière optimale le nombre minimal d'arêtes à ajouter afin d'atteindre la 2-connexité. Testée sur plusieurs instances, cette méthode réduisait le nombre d'arêtes à construire par rapport à la tentative précédente, tout en garantissant la même robustesse.

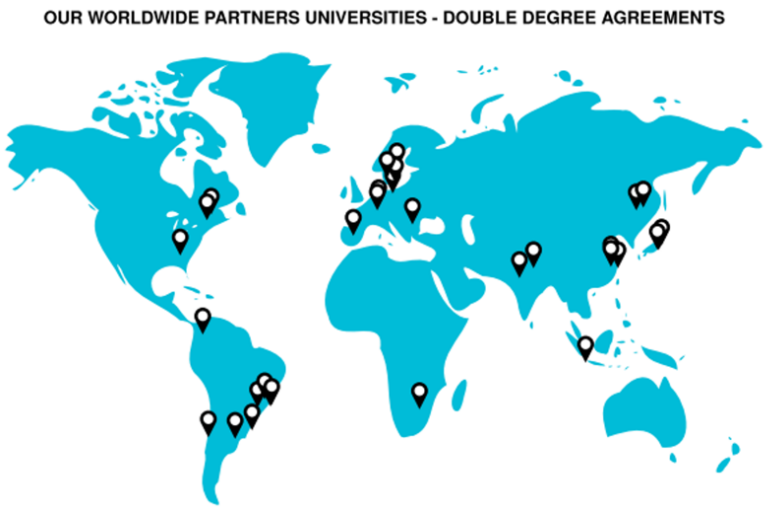
Au final, la solution retenue a permis de transformer notre réseau en graphe 2-connexe avec seulement 2 arêtes de secours. Nous avons ainsi validé qu'en cas de coupure de n'importe quelle arête, un chemin alternatif existe systématiquement, tout en maîtrisant le coût et la complexité de déploiement.

- **Rayan BOUAKAR**

  Vers le milieu du projet, nous avons rencontré des difficultés à implémenter l'un des algorithmes de la phase de modélisation. La fin de la séance approchait, et cela faisait près d'une heure que nous étions bloqués dessus. Malgré notre volonté de terminer à temps pour ne pas prendre de retard sur notre diagramme de Gantt, le découragement commençait à s'installer, moi le premier.

  Comme j'étais un peu pressé de rentrer, j'ai proposé un plan d'action simple et rapide pour résoudre le problème efficacement. Tout d'abord, j'ai suggéré que nous prenions conscience de notre baisse de moral afin de nous recentrer collectivement. Ensuite, j'ai proposé de clarifier précisément l'origine du problème. Nous savions qu'il s'agissait d'un souci lié à l'algorithme, mais la bibliothèque Python que nous utilisions ne fournissait pas de messages d'erreur explicites, ce qui complique le diagnostic. J'ai donc proposé d'ajouter des logs entre les blocs de code afin d'identifier précisément les lignes responsables. Une fois les lignes incriminées repérées, nous nous sommes appuyés sur la documentation pour comprendre la nature de l'erreur : s'agissait-il d'un problème de syntaxe ou d'une mauvaise implémentation ? Pour gagner du temps, j'ai également sollicité une intelligence artificielle afin d'obtenir un exemple clair d'utilisation des méthodes concernées, que j'ai ensuite adapté à notre cas.

  Grâce à cette méthode, nous avons pu résoudre le problème rapidement et atteindre l'objectif prévu pour la séance. Cette expérience s'est révélée très formatrice pour l'équipe, même si nous avons perdu une heure sur ce bug, nous avons appris à réagir plus efficacement. Par la suite, chaque fois que nous avons rencontré une difficulté similaire, nous avons su adopter la même démarche et gagner en réactivité.



IMT Atlantique Bretagne–Pays de la Loire – **http://www.imt-atlantique.fr/**

Campus de Brest
Technopôle Brest-Iroise
CS 83818
29238 Brest Cedex 3
France
T +33 (0)2 29 00 11 11
F +33 (0)2 29 00 10 00

Campus de Nantes
4, rue Alfred Kastler
CS 20722
44307 Nantes Cedex 3
France
T +33 (0)2 51 85 81 00
F +33 (0)2 99 12 70 08

Campus de Rennes
2, rue de la Châtaigneraie
CS 17607
35576 Cesson Sévigné Cedex
France
T +33 (0)2 99 12 70 00
F +33 (0)2 51 85 81 99

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