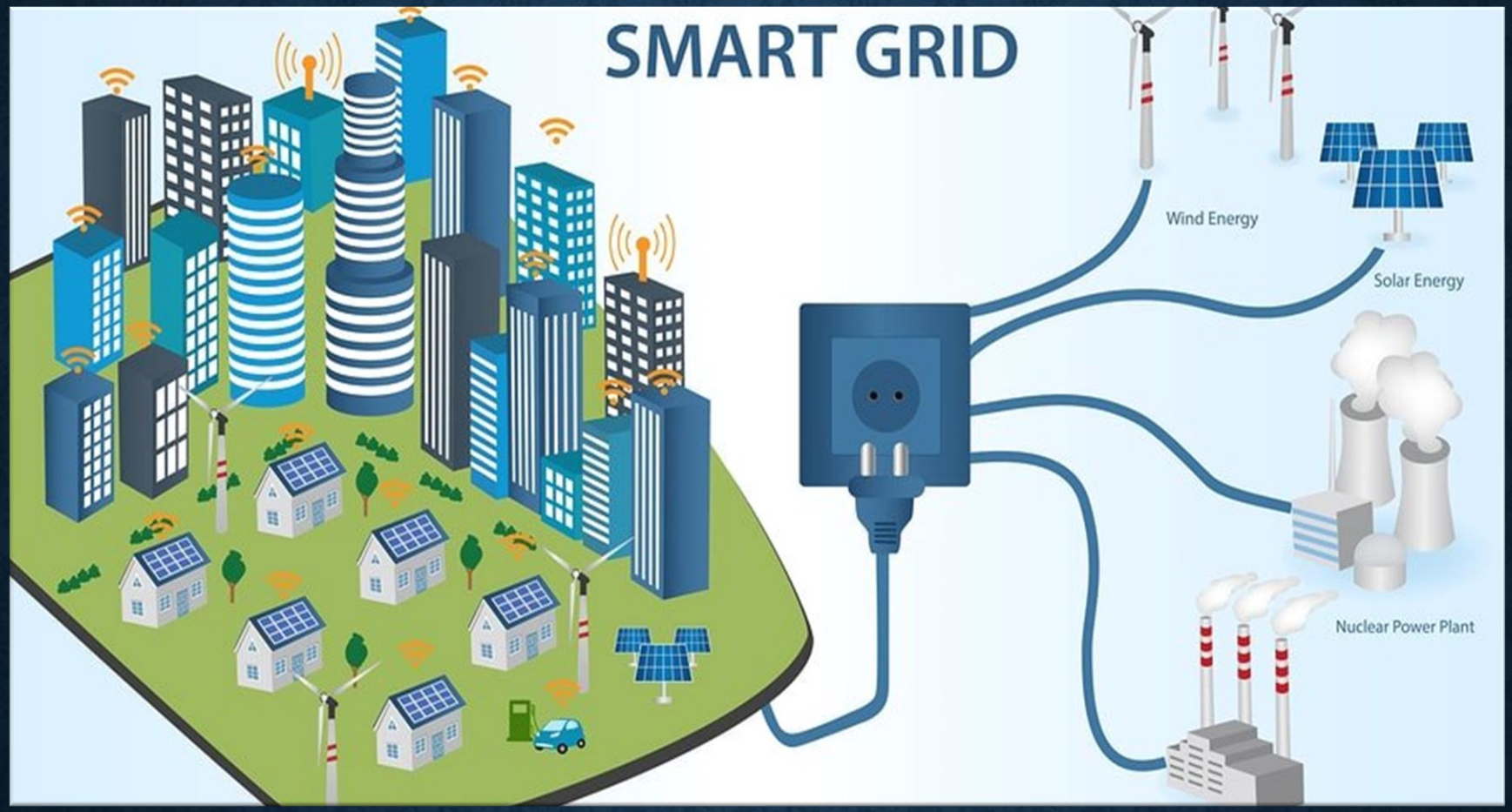


MODÉLISATION D'UN RÉSEAU ÉLECTRIQUE INTELLIGENT

Numéro d'inscription : 33112

PLAN :

I- Présentation des sous-composante

II- Estimation de la consommation l'énergie

III- Problème de l'offre et de la demande

IV- Transport de l'énergie

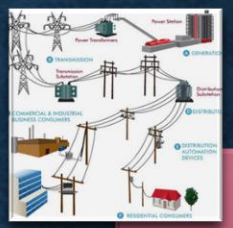
I-PRÉSENTATION DES SOUS COMPOSANTE

Niveau local

- Structure isolé
- Gestion de la consommation locale
  - ➢ Domotique
  - ➢ EnR
  - ➢ V2G
- Répartition de l'énergie locale

microgrid

- Structure en arbre
- Consensus
  - ➢ Production
  - ➢ Consommation
  - ➢ Distribution
- Equilibrage de l'offre et de la demande



T&D

- Structure maillé
- Gestion de la production globale
  - ➢ Production
  - ➢ Planification
- Distribution de l'énergie dans le réseau

II- ESTIMATION DE LA CONSOMMATION ÉNERGÉTIQUE

1) PROCESSUS D'ESTIMATION

Les données entrantes sont les appareils (consommation et priorité) et les pronostics. Le but est de répartir au mieux l'énergie en fonction des priorités de consommation sans dépasser l'énergie reçue.

Le problème du sac à doc consiste à construire une fonction optimale, donc ici nous devons sélectionner les éléments que nous pouvons alimenter en étant conforme au pronostic (la solution de l'itération précédente)

2) GESTION DE LA RÉPARTITION DE L'ÉNERGIE DANS UN NIVEAU LOCAL :

Dans la suite, nous prendrons n_i comme le nombre d'appareils d'un niveau locale i, conso_max(i) la consommation maximale et C l'énergie reçu ou pronostic

Problème du sac à dos ou Knapsack Problem

- Résolution par programmation dynamique
- complexité : $O(n_i \times C)$

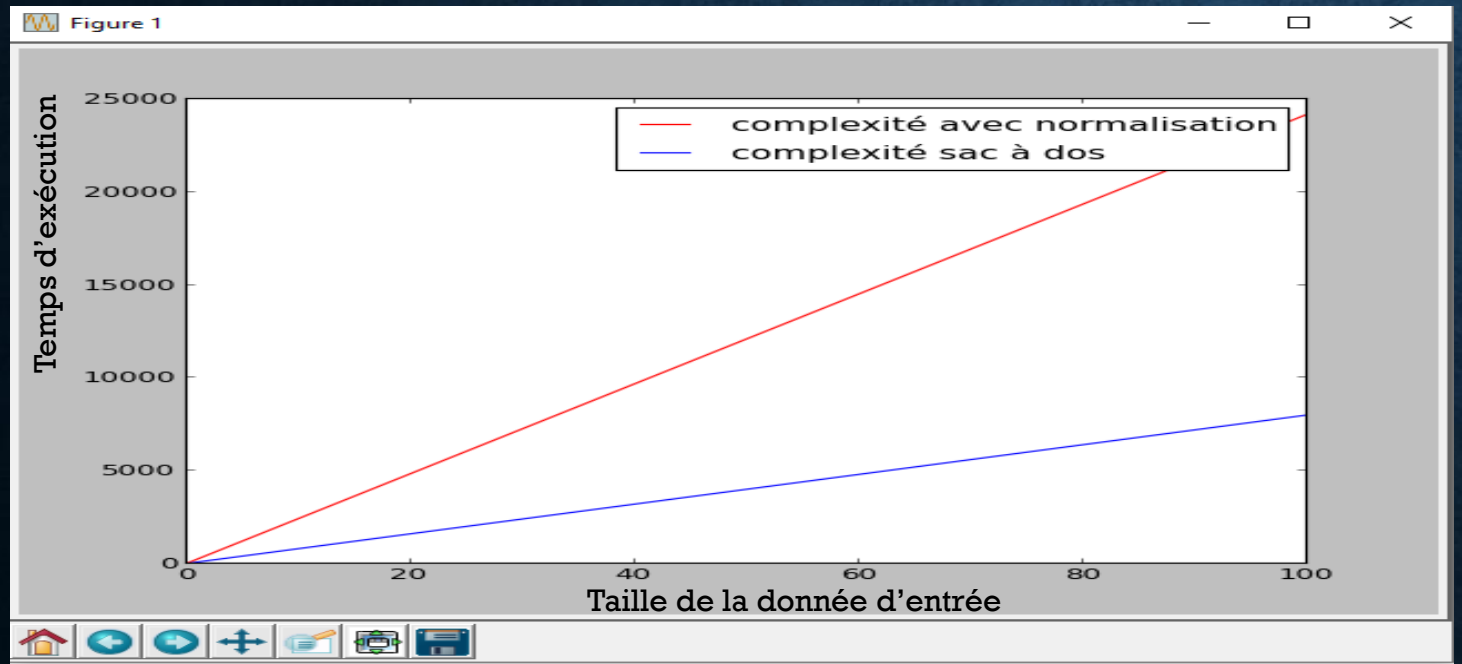
Normalisation

- Divise au plus les consommations et l'énergie reçu C par le pgcd
- Complexité : $O(n_i \times \log(\text{conso\_max}(i)))$

3) COMPLEXITÉ :

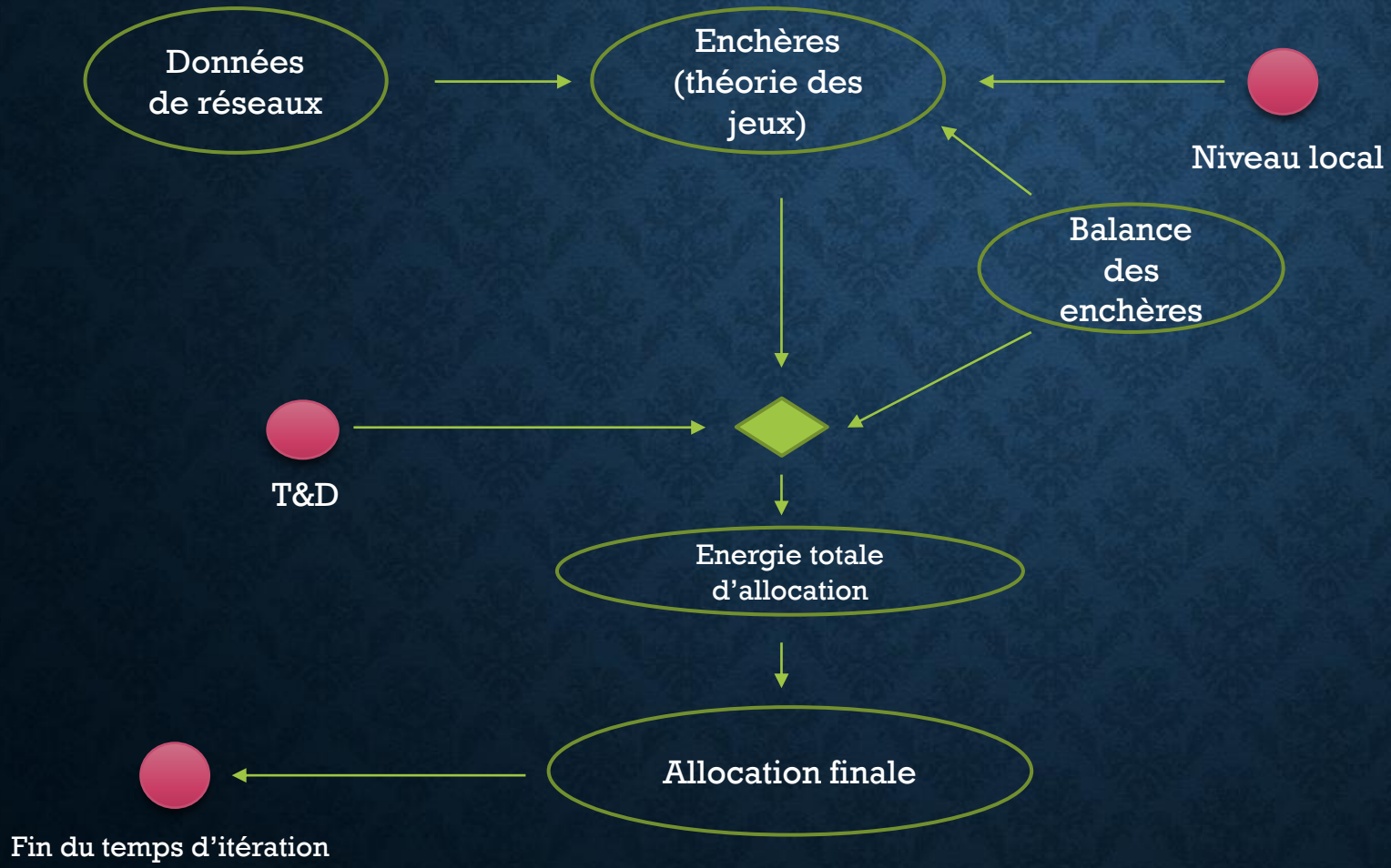
Au niveau local on retrouve une suite successive d'algorithme, d'où l'importance de calculer la complexité global du niveau local.
La normalisation a une complexité de O(n×log(conso_max(i))) tandis que la sac à dos est en O(n×C). La normalisation est donc bénéfique si, en considérant C° le poids d'origine sans normalisation (C = $\frac{C°}{min}$), C+ log(conso_max) < C°

Au niveau locale avons une complexité globale de O(n×(C°+log(conso_max))). Donc dans le pire des cas la complexité est de O(n×C)



Par cette courbe nous pouvons donc bien voir que la complexité globale au niveau local est linéaire comme montré dans le pire des cas.

III- PROBLÈME DE L'OFFRE ET LA DEMANDE:



AU NIVEAU DU MICROGRID

1) ETUDE DES ENCHÈRES DE TYPE ALÉATOIRE CENTRÉE

Pour effectuer des enchères proches de la consommation moyenne nous devons utiliser la loi normale N (consomoy, σ^2)

Définition: S la variable aléatoire X suit la loi normale N (μ, σ2) et si a, b sont deux réels tels que a ≤ b alors la probabilité de se trouver entre a et b est noté $P(a \leq X \leq b) = F(b) - F(a) = \Phi(\ b - \mu\ \sigma\ ) - \Phi(a - \mu\ \sigma\ ) = \alpha$

Φ est la fonction de répartition de la loi normale centrée réduite N (0, 1),

$$\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}}\, \partial t$$

La variable aléatoire $X' = \frac{X-\mu}{\sigma}$

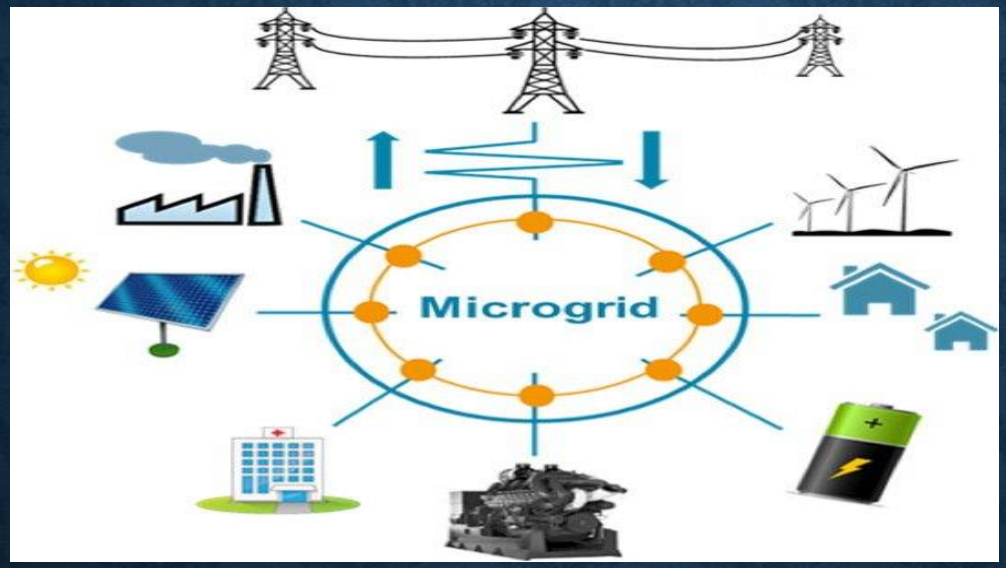
suit la loi normale centrée réduite. L'intervalle [a, b] = [μ−tσ, μ+tσ] est appelé plage de normalité au niveau de confiance α. C'est à dire que α% de la population se situe dans l'intervalle [a, b]. Dans notre cas, nous cherchons à paramétrer la loi normale en fonction des capacités de consommation de chaque niveau local i, alors les caractéristiques de la loi normale sont:

1. μ = conso_moy(i) étant la consommation moyenne d'un niveau local i
2. [μ − 3σ, μ + 3σ] est la plage de normalité au niveau de confiance 99,7%. Soit une marge de ε autour de consomoyi, alors 3σ = ε.

Après avoir calculé une enchère pour chaque niveau locale, le microgrid somme les résultats et ne renvoie donc qu'une seule valeur au niveau du T&D.

2)Calcul des enchères :

Ce calcul est fait à l'aide d'un algorithme simple dont la complexité est $O(nl \times w)$ avec n_l le nombre de niveau locaux sous la responsabilité du microgrid et w le nombre d'enchère.



Les niveaux locaux sont récompensés ou punis selon leurs enchères. La récompense ou la punition doit permettre de diminuer ou d'augmenter leurs demandes énergétiques. Le niveau local est récompensé si son enchère est inférieur à sa consommation moyenne, sinon il est puni. Dans notre modèle, les récompenses ou punitions se font en appliquant un coefficient (par exemple +3% ou −3%) à la valeur de μ de la loi normale.

IV- TRANSPORT DE L'ÉNERGIE:

1) PROBLÈME DE ROUTAGE :

Définition : Soit G = (V, A) un graphe orienté. Soit s la source et t le puits de G. A chaque arête (u, v) de A est associée une capacité C(u,v) ≥ 0 qui représente le flux maximum pouvant passer par cette arête. Soit c le vecteur dans $R^{|A|}+$ contenant les valeurs de toutes les capacités. On associe à chaque arête (u, v) un flux f (u, v) représentant la quantité de flux vérifiant :

1. Contrainte de capacité : f (u,v) ≤ C(u,v) pour toute arête (u, v) ∈ A

2. Conservation du flux : $\sum_{v \in V} f(u,v) = \sum_{w \in V} f(w,u)$ pour tout u ∈ V \ {s, t}.

3. Antisymétrie : f(u,v)= - f(u,v)

La valeur du flux est |f| = $\sum_{v \in V} f(s,v)$

V est partitionné en {S, I, P} avec S un ensemble de sources, I des nœuds intermédiaires et P un ensemble de puits.

Le problème est donc de trouver une fonction $f : A \rightarrow N$ vérifiant :

$$0 \leq f(x, X) - f(X, x) \leq a(x) \text{ pour tout } x \in S$$

$$f(x, X) - f(X, x) = 0 \text{ pour tout } x \in I$$

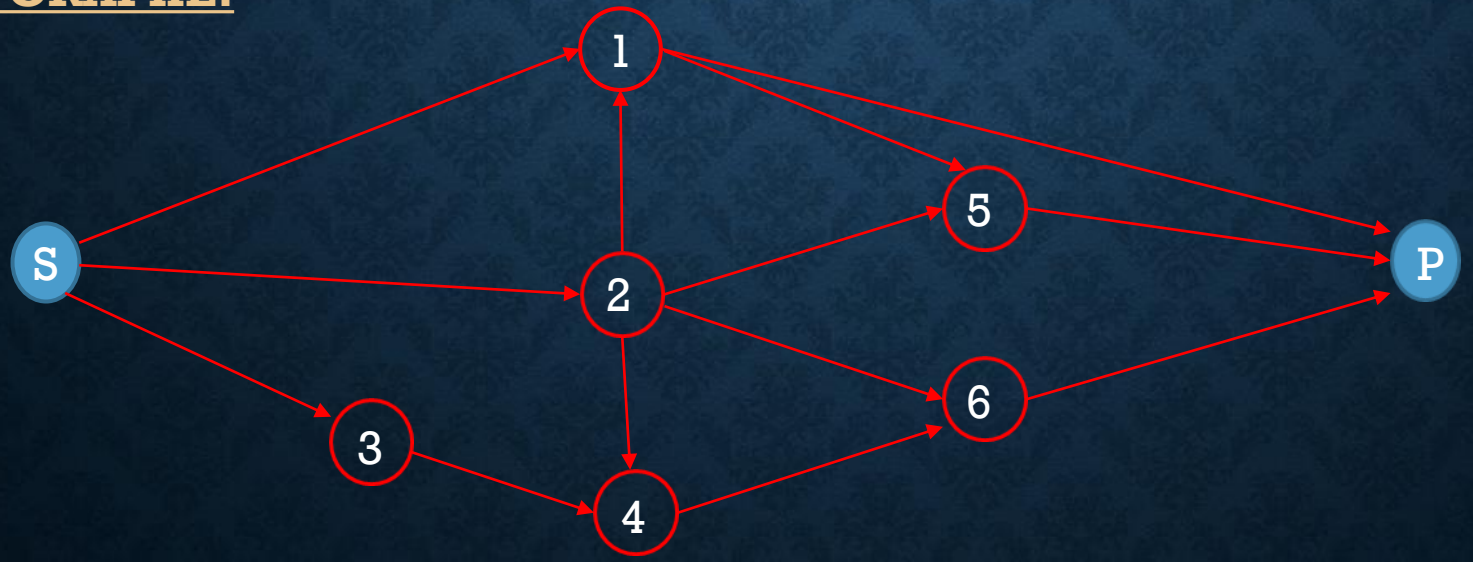
$$0 \leq f(X, x) - f(x, X) = b(x) \text{ pour tout } x \in P$$

$$0 \leq f(x, y) \leq c(x, y) \text{ pour tout } (x, y) \in X$$

L'algorithme de Ford-Fulkerson va permettre d'optimiser ces flux à l'aide d'un outil de modélisation mathématique.

Le flux maximal est atteint lorsque le programme ne trouve plus aucun chemin améliorant dans le graphe. Donc nous obtenons une solution optimale si l'algorithme se termine.
Le temps d'exécution de Ford-Fulkerson est borné par O(N×f) avec N le nombre d'arête du graphe et f le flux maximum.

EXEMPLE DE GRAPHE:



Il est donc question de trouver le chemin optimal

À partir de là il y'a recherche d'une solution réalisable, le réseau T&D met à jour son routage à l'aide du graphe résiduel. Et enfin la station prend connaissance de l'énergie possible par le routage.

ANNEXE :

Résolution dynamique du problème du sac à dos :

```
#element est une liste de tuples et chaque tuple contient dans cet ordre, nom , consommation d'energie et l'utilité
#capacite , c'est l'énergie maximale donner par l'itération précédente

def sac_optimal(capacite,elements):
    tableau=[[0 for i in range(capacité+1)]for i in range(len(elements)+1)]
    for i in range(1,len(elements)+1):
        for E in range (1,capacite+1)
        if elements[i-1][1]<= E:
            tableau[i][E]= max(elements[i-1][2]+ tableau[i-1][E-elements[i-1][1]],tableau[i-1][E])
        else:
            tableau[i][E]= tableau[i-1][E]
    #retrouver les éléments de la solution optimale en fonction de la somme
    w= capacite
    n=len(elements)
    elements_selection = []
    while w >= 0 and n>=0:
        e= elements[n-1]
        if tableau[n][w] == tableau[n-1][w-e[1]]+e[2]:
            elements_selection.append(e)
            w -= e[1]
        n -= 1
    return tableau[-1][-1],elements_selection
```

Programme de calcul du PGCD de la consommation des appareils d'un niveau local:

```
def PGCD(a,b):
    if a>b :
        X=a
        Y=b
    else :
        X=b
        Y=a
    while X%Y!=0 :
        R=X%Y
        X=Y
        Y=R
    return Y

#mini calcule le PGCD de la liste de la consommation des appareils
def mini(L):
    mini= PGCD(L[0],L[1])
    i=2
    if len(L)>2:
        while i<= len(L)-1:
            A=PGCD(mini,L[i])
            mini = A
            i=i+1
        return mini
```

Programmes au niveau local

```
def attribution_énergie(pronostics,elements):
    conso_moy= pronostics
    for i in range(len(elements)+1):
        conso_min=+elements[1][i]
    if conso_min<= conso_moy:
        return conso_moy
    else:
        return sac_optimal(conso_moy,elements)


#liste des priorités des appareils d'un niveau local

def Données_consommation(P,elements):
    conso_moy=0
    for i in range (len(elements)+1):
        if P[i+1]>0:
            elements[1][i]=elements[1][i]*mini
    for i in range (len(elements)+1):
        L=sac_optimal(pronostic,elements)
        if P[i+1]=0 and elements[i] in L:
            conso_moy=conso_moy+elements[1][i]
```

Programme de calcul des enchères :

```
from random import*
# L est une liste de tuplet et chaque tuple contient dans l'odre l'indice,la consommation
# minimale,laconsommation moyenne et la consommation maximale de chaque niveau locale.
# e représente lécart-type de la loi normale
def calcul_enchères(L,approximationΦ(x),e):
    stockage=[]
    conso= 0
    n= len(L)+1
    for i in range(1,n):
       Φ(x)= uniform(0,1)
       Z= approximationΦ(x)
       X=Z*e+ L[i][2]
       if X < L[i][1]:
           X= L[i][1]
       if X > L[i][3]:
           X= L[i][3]
       conso= conso+ X
       stockage.append(X)
    return stockage
```

Programme de traçage des courbes:

```
import numpy as np
import matplotlib.pyplot as plt
import math as m
def courbe_du_sac(conso_max,C):
    c=mini*C
    x=np.linspace(0,100,10)
    y=x*(c+ m.log(conso_max,10))
    Z=C*x
    plt.plot(x,y,'r',label='complexité avec normalisation')
    plt.plot(x,Z,'b', label='complexité sac à dos')
    plt.legend()
    plt.show()
```

Programmes pour le problème de routage :

```
def recherche_chaine_améliorante(V,A,C,s,t):
    Z=[]
    Z.append(s)
    while Z!=[] or t is not in Z:
        x=s
        Z.pop(X)
        for y in A and i>y:
            if and f(u,v)<C(u,v):
                Z.append()
        for y in A and i<y:
            if and f(u,v)>0:
                Z.append()
    if t is in Z:
        return "il ya une chaine améliorante"
    else :
        return "pas de chaine améliorante"

def Ford_fulkerson():
    for u in V and v in V:
        f(u,v)=0
    if recherche_chaine_améliorante(V,A,C,s,t)== "il ya une chaine améliorante":
        e=min{c(u,v)-f(u,v)for (u,v) in A}
        d=min{f(v,u)for (u,v) in A}
        E=min(e,d)
        if (u,v) in A:
            def ff(u,v):
                ff(u,v)=f(u,v)+E
            ff(u,v)
        else:
            def fg(u,v):
                fg(u,v)=f(u,v)-E
            fg(u,v)
```

Références :

Encyclopédie énergie, figure 1 : https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.echosciences-grenoble.fr%2Farticles%2Fles-microgrids-contribuent-ils-a-la-transition-energetique&psig

Smart Energy international, image credit : stock : https://www.smart-energy.com/industry-sectors/smart-grid/bangladesh-receive-smart-grids/

Smart Grid: http://www.ecobase21.net/Lesmotsduclimatsmartphone/Smartgrids.html

Guillaume Guérard : Optimisation de la diffusion de l'énergie dans les smarts-grids, Université de versaille, 2015 : https://tel.archives-ouvertes.fr/tel-01241153

LA GTB : https://www.actu-environnement.com/ae/dossiers/smart-grid/gtb.php4