CentraleSupélec | université PARIS-SACLAY

ST7 - Systèmes d'énergie embarqués

Séquence - Projet ST7

Placement optimal des selfs sur un réseau CIGRE

Auteurs :
Fernando E. B. SANTIAGO
Arthur CARSANA
Erwan DUBOURG
Ana Luiza HAAS BEZERRA
Sabrina MOCKBEL

Encadrants :
M. Philippe DESSANTE
M. Le TRUNG DUNG
Mme. Maya HAGE HASSAN

Version du
1er avril 2023

Table des matières

1 Introduction

1.1 Présentation générale

Les réseaux de distribution sont essentiels pour la transition énergétique car ils permettent de connecter les sources d'énergie renouvelable, les moyens de stockage d'énergie et les nouveaux usages électriques. Cependant, ces réseaux ne sont pas conçus pour gérer ces nouvelles installations et peuvent rencontrer des problèmes de tension dans ces nœuds en cas de forte production ou de consommation. SRD et GeePs souhaitent étudier l'utilisation d'inductances de compensation (appelée self dans ce cadre) comme solution pour contrôler le flux de réactif circulant sur le réseau et maintenir la tension dans la plage admissible en tout point du réseau, selon la réglementation en vigueur.

Ainsi, le projet vise à trouver le nombre et le placement optimal des inductances de compensation sur un réseau de distribution HTA en utilisant un algorithme d'optimisation. Les étapes comprennent une étude bibliographique, le choix d'un algorithme d'optimisation, la mise au point de l'algorithme et des tests pour analyser les résultats.

Le grand projet est réparti en trois pistes différentes : placement optimal des selfs, pilotage des selfs et pilotage des producteurs. Notre groupe est chargé de la piste 1, c'est-à-dire que nous devons choisir de placer les selfs sur le réseau pour réguler la tension, tout en minimisant les coûts (nombre et puissances installées des selfs), et en respectant les contraintes en tension en tout point du réseau.

1.2 État de l'art

Le réseau électrique français est vaste. Il est composé de lignes BT, MT, HT et THT. Les lignes THT et HT composent le réseau de transport. Les lignes MT et BT composent elles, le réseau de distribution.

Le réseau de transport sert à transporter l'énergie en quantité importante sur de longues distances. Les réseaux de distribution sont destinés à acheminer l'électricité à l'échelle locale. Ils relient donc les consommateurs de plus faible puissance au réseau électrique. En France, les réseaux de distribution électrique représentent plus d'un million de kilomètres de lignes sur tout le territoire. Cela correspond à un réseau MT de 596 200 km et un réseau BT de 669 300 km.

Le réseau de distribution a subit de nombreux changements avec la transition énergétique et l'essor des énergies renouvelables. Ainsi, la plupart des nouvelles installations de production sont raccordées au réseau de distribution (environ 80%).

La particularité de ces installations est qu'elles ont chacune leurs spécificités. Toutes les sources de production n'ont pas les mêmes puissances actives. Par exemple, pour les centrales thermiques de faible puissance, la puissance active est de 100 MW, alors que celle des centrales nucléaires peuvent monter jusqu'à des puissances autour de 1 650 MW.

Il y a donc une décentralisation de la production électrique dans tout le territoire. Si la production décentralisée ne dépasse pas la consommation locale, elle permet de produire localement l'électricité consommée sur place. Ainsi cela permet de réduire les pertes dans les lignes composant les réseaux. Dans un deuxième temps, cela permet aussi d'éliminer les grandes pointes de consommation.

Cependant, il existe aussi des points négatifs associés à l'intégration de ces producteurs sur le réseau de distribution. Les réseaux à production distribuée, comme le solaire ou l'éolienne par exemple, peuvent engendrer certaines problématiques. Ils ne permettent pas toujours un contrôle total de leur production d'électricité. Cela va amener à d'autres conséquences comme des problèmes d'augmentation du flux de puissance réactive. Cette augmentation provient, elle, de la variation de la puissance réactive de la production et de la consommation.

Un autre risque associé à ce phénomène est la création d'élévation de tension aux noeuds du réseau. Cela peut amener à des surtensions et donc aux dépassements des limites de tension autorisées. Il y a donc des contraintes de tensions qui doivent être respectés et l'objectif du projet est de modifier le réseau électrique pour améliorer l'intégration des producteurs avec l'installation de selfs dans noeuds stratégiques. Dans l'article [2] nous avons pu identifier les contraintes qui sont utilisés dans la littérature liées aux caractéristiques du réseau de distribution qui sont exposées dans la figure 1.

Caractéristiques	Valeurs admises
Fréquence	BT : 50 Hz ± 1 % pendant 99.5 % du temps sur une année. HTA : 50 Hz + 4 %/ − 6 % pendant 100 % du temps.
Tension	BT : 230 V ± 10 % pendant 95 % du temps sur une semaine. HTA : 20 kV ± 10 % pendant 95 % du temps sur une semaine.
Harmoniques	Taux global de distorsion harmonique $\leq 8\,\% \cdot U_{nominale}$.
Déséquilibre de la tension	$U_{inverse} \leq 2\,\% \cdot U_{directe}$ pendant 95 % du temps sur une semaine.

FIGURE 1 – Table de contraintes pour les réseaux de distribution HTA et BT (modifié de [2])

Il faut donc trouver un moyen d'assurer le respect des contraintes de tension dans le réseau tout en y ajoutant des GEDs. Parmi les différentes solutions de la littérature, ce travail va se focaliser sur l'étude de l'utilisation d'inductances

(selfs) pour réduire la tension localement dans un noeud du réseau afin de respecter les contraintes de tension.

Dans la figure 2, il est possible d'observer une ligne modelée par une impédance Z où il transite de la puissance P entre la source (noeud 1) et la charge (noeud 2). Il est possible d'observer qu'avec la présence d'un producteur GED, qui produit de l'énergie, donc $P_G > 0$. À partir de l'équation (1) qui montre la relation entre la chute de tension dans une ligne et la puissance qui y transite (représentée par la flèche vert), il est possible de conclure que l'introduction du producteur va conduire à une augmentation de la tension dans le noeud car il injecte puissance active sur le noeud et il va réduire la puissance qui transite dans la ligne, donc $P < 0$.

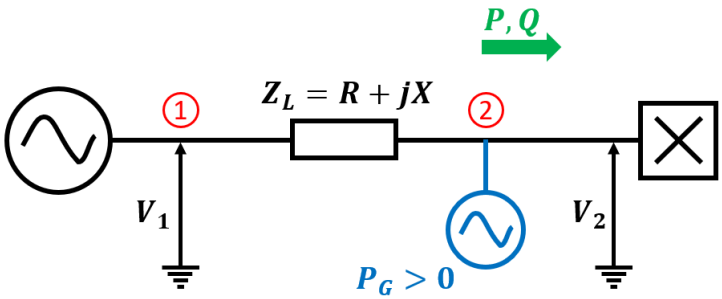

FIGURE 2 – Ligne d'exemple modelé par une impédance

$$\Delta V = V_1 - V_2 = \frac{PR + QX}{V_2} \tag{1}$$

Dans cette même logique, si on rajoute une inductance (self) au noeud 2, elle va consommer de la puissance réactive et donc va augmenter la puissance qui transite dans la ligne. L'effet est alors inverse, la tension va être réduite tout en maintenant la puissance active, ce qui justifie son application pour la régulation de surtension sur des réseaux de distribution électriques.

Dans le cadre de notre projet, nous allons étudier le placement des selfs dans le réseau pour respecter les contraintes de tension. Cependant, rajouter des selfs peut avoir un coût d'installation en dépendant de la puissance installée. Il faut donc essayer de limiter le coût des installations à mettre. Pour cela, il faut essayer de mettre le minimum de selfs, et de plus faible valeur.

1.3 Outils utilisés dans ce projet et réseau étudié

Nous avons choisi Python comme langage pour réaliser notre optimisation afin de pouvoir utiliser la bibliothèque `pandapower` qui nous permet d'avoir un modèle de réseau électrique plus fiable et développé. De plus, Python possède une vaste bibliothèque d'optimisation avec "scipy" et l'utilisation de notebooks (Jupyter-notebook) fait que la visualisation des résultats soit plus facile et le code soit plus dynamique.

Tout d'abord, nous avons commencé par un réseau de base de `pandapower`, le réseau HTA CIGRE illustré dans la figure 3. Comme il est possible d'observer dans le profil de tension du réseau à la figure 4, ce réseau n'a initialement pas d'éléments intéressants pour le projet dès lors que des surtensions de plus de 5% ne sont pas présentes.

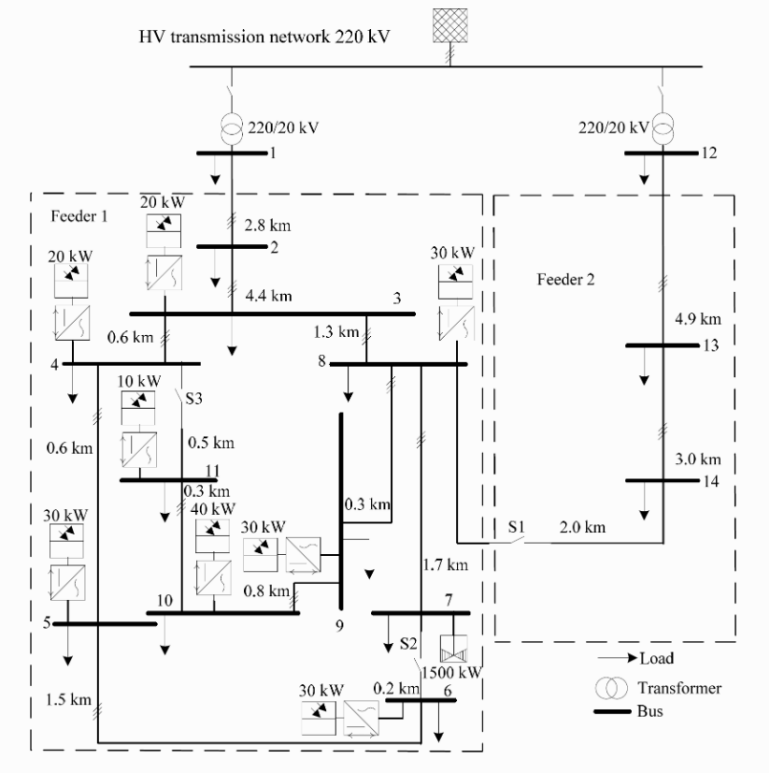

FIGURE 3 – Réseau CIGRE originale [5]

Cependant, avec la prise en main des fonctions de modification du réseau, nous avons réalisé des changements sur le réseau afin d'atteindre les résultats de tension de la figure 4.

Nous avons modifié le réseau Cigre mis à notre disposition, afin de pouvoir

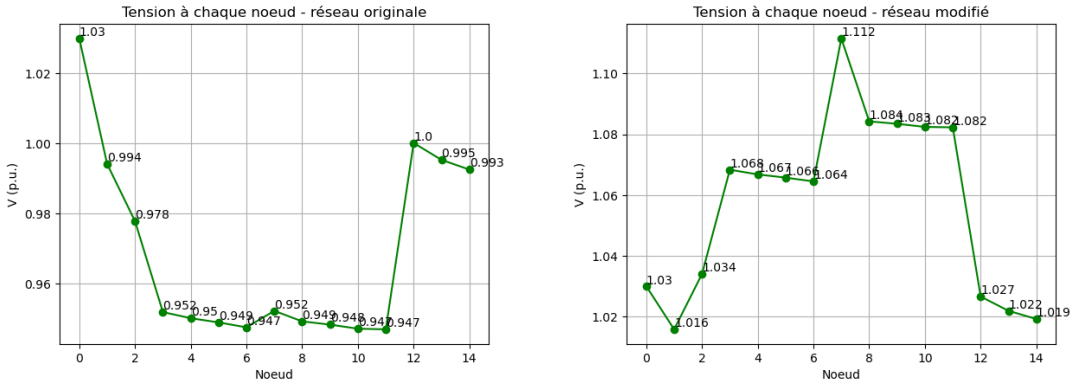

FIGURE 4 – Tension au réseau CIGRE originale (à gauche) et au réseau modifié (à droite)

travailler sur un réseau en surtension. En effet, pour dérégler la tension dans le réseau, nous avons diminué la puissance dans les noeuds qui sont les plus proches de la source, c'est à dire les noeuds 1 et 12, artificiellement en rajoutant des générateurs statiques. Les puissances dans les noeuds 1 et 12 étaient initialement d'environ 20 MW et nous avons décidé de baisser ces valeurs de 5 MW, soit des valeurs de l'ordre de 15 MW. Cette diminution est importante car elle correspond à une augmentation de tension de 25%. Par ailleurs, nous avons également augmenté la puissance du producteur éolien du réseau au noeud 7, connecté au noeud 8, nous avons modifié cette valeur de 0.3 MW à 15 MW.

Le réseau équivalent après les modifications est explicitement expliqué dans la figure 5 et va être le réseau de référence pour le reste de notre étude. Il est possible de vérifier que ce réseau possède un noeud où la surtension est la principale, qui correspond au producteur qui injecte plus de puissance sur le réseau.

2 Résolution du problème d'optimisation

Nous devons à présent réguler la tension sur ce réseau en plaçant judicieusement des selfs qui auront pour rôle de consommer de la puissance réactive du réseau, ce qui résulte dans une chute de tension négative qui va réduire la surtension jusqu'au niveau accepté. Cette dynamique est résultat de l'équation (1) comme expliqué précédemment.

Une des contraintes de notre réseau est que la tension doit être dans un intervalle de confiance de plus ou moins 5 % autour de la tension nominale

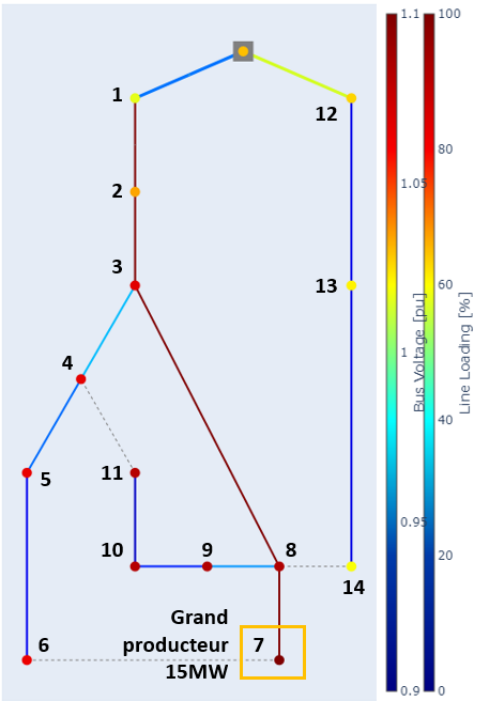

FIGURE 5 – Réseau CIGRE modifié

(contrainte classique en énergie électrique qui assure à la durée de vie des matériaux et la sécurité sur les lignes). La tension nominale du réseau utilisé dans ce travail est $V_n = 20kV$. L'autre contrainte est qu'une self doit avoir la valeur minimum de 200 kVAr comme prévu par le client.

Tout d'abord, nous avons fait une formulation mathématique du problème d'optimisation avec un nombre de selfs donné, puis nous avons essayé plusieurs types d'algorithmes d'optimisation pour trouver la puissance ainsi que la position optimale des selfs. Les méthodes que nous avons testé dans cette étude sont : méthode déterministe et méthodes stochastiques en utilisant algorithme génétique, évolution différentielle et particles swarm.

2.1 Formulation formelle du problème

Le problème a été formulé comme un problème d'optimisation mono-objective pour un nombre de selfs fixé S, avec la tension nominale du réseau et un nombre de noeuds donnés. Il fallait alors résoudre le problème d'optimisation suivant.

Problème d'optimisation :

— Paramètres :
  — Nombre de selfs fixé = s
  — Tension nominale du réseau = V_n
  — Nombre de noeuds du réseau = N
— Variables : $\forall i \in 1, ..., N$ (q_i c'est la puissance de la self installée dans le noeud i)
— Fonction objectif à minimiser : $f(q_1, ..., q_N) = \sum_{i=1}^{N} q_i$
— Contraintes : $\begin{cases} n = \sum_{i=1}^{N} \mathbb{1}_{]0,+\infty[}(q_i) = S \\ \forall i \in \{1, ..., N\}, 0,95V_n \leq V_i \leq 1,05V_n \\ \forall i \in \{1, ..., N\}, q_{act,i} \geq 0,2 \end{cases}$

Et ce problème peut être encore écrit comme dans la forme mathématique :

$$\begin{aligned} \min_{q} \quad \sum_{i=1}^{N} q_i \qquad \text{s.t.} \quad & V_i \leq V_{max} \\ & V_i \geq V_{min} \\ & q_{act,i} \geq 0,2 \\ & \sum_{i=1}^{N} \mathbb{1}_{]0,\text{inf}]}(q_i) = s \end{aligned} \tag{2}$$

Cette formulation permet de réduire la taille des variables parce que q inclut, en même temps, l'information du dimensionnement des selfs et du placement des selfs dans les noeuds du réseau en considérant que $q_i = 0$ si le self est inactif.

2.2 Spécifications sur les fonctions modèle du réseau

Les fonctions utilisées en tant que modèle du réseau pour calculer les résultats de fonction de coût et contraintes à chaque itération sont faites séparément. Comme la fonction de coût et la contrainte d'égalité du nombre de selfs dépendent seulement de la variable, elles sont réalisées en dehors de la fonction qui calcule la contrainte de tension.

La fonction qui calcule la contrainte de tension est appelée *fct_model_reseau*. Elle prend en entrée le vecteur de variables q qui correspond aux puissances réactives installées comme selfs à chaque noeud du réseau. La fonction utilise ces données pour créer des selfs à l'aide de la bibliothèque `pandapower`, puis elle exécute une simulation de réseau électrique pour calculer les tensions dans chaque

noeud. La fonction retourne un vecteur d'outputs qui contient les tensions calculées dans chaque noeud qui seront utilisées pour le calcul des contraintes de tension par l'algorithme.

Un problème qui a été rencontré plusieurs fois sur cet algorithme est que la surface d'optimisation était si restrictive à cause de la contrainte d'égalité que l'optimisation n'était pas capable de trouver de solution dans le domaine de faisabilité. La solution pour ce problème a été la création d'une fonction `shigher` qui est utilisée pour sélectionner les s plus grandes valeurs dans un tableau de données x. Cette fonction est appelée pour sélectionner les s plus grandes puissances réactives à installer comme selfs à chaque noeud et est appelée au début de chaque fonction qui calcule le coût et contraintes. Cela permet que le vecteur de variables puisse se développer plus librement en relaxant la contrainte d'égalité.

2.3 Optimisation deterministe

Pour résoudre ce problème mono-objectif, nous avons d'abord programmé un algorithme d'optimisation avec contraintes, grâce à la méthode SLSQP (Sequential Least Squares Programming, c'est-à-dire Programmation séquentielle des moindres carrés).

C'est une méthode d'optimisation numérique qui vise à résoudre des problèmes d'optimisation non linéaires avec des contraintes d'égalité et/ou d'inégalité. Cette méthode repose sur la minimisation de la fonction objectif non linéaire, tout en satisfaisant les contraintes.

Le terme "least squares" fait référence à l'utilisation de la méthode des moindres carrés pour ajuster les paramètres d'un modèle mathématique à des données expérimentales, tandis que "sequential" fait référence à l'approche itérative utilisée par la méthode pour trouver la solution optimale.

La méthode SLSQP fonctionne en combinant la méthode de gradient projeté et la méthode des moindres carrés. En effet, elle utilise une approche itérative pour trouver la solution optimale en effectuant des étapes de descente de gradient en projetant les résultats sur l'ensemble des contraintes.

L'algorithme SLSQP est implémenté dans la fonction "minimize" de la bibliothèque scipy.optimize. Cette fonction prend en entrée la fonction objectif, ainsi que des contraintes pour les variables de la fonction objectif. Ces contraintes sont spécifiées à l'aide de la classe "Bounds", qui définit des bornes pour les valeurs des variables, et des dictionnaires qui définissent les contraintes non linéaires pour les valeurs des variables.

En premier lieu, nous avons réalisé l'optimisation pour une self avec cet

algorithme. Comme c'est une méthode déterministe, elle dépend d'un point initial x_0 et ce point peut faire que le résultat obtenu ne sera pas un optimum global.

D'où le besoin d'implémenter une stratégie de multistart où, pour améliorer les résultats obtenus, il est possible de partir de plusieurs points de départ et de les comparer entre eux avec l'objectif de trouver l'optimum global parmi les résultats obtenus. Les points de départ ont été sélectionnés au hasard dans un intervalle entre 0 et 1.

2.3.1 Résultats SLSQP avec réseau modifié pour s variant entre 1 et 4

En appliquant la méthode d'optimisation déterministe avec 50 itérations de multistart, nous avons trouvé les résultats suivants dans l'image de la figure 6 pour s = 1 (une self). Dans la figure, les selfs sont représentées par des triangles rouges. La puissance totale installée optimale est de 2,238 MVAr.

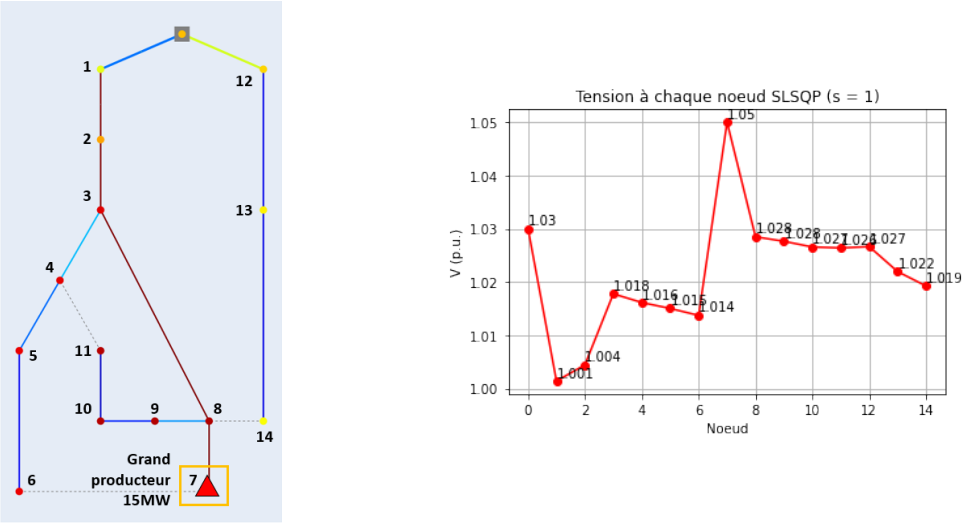

FIGURE 6 – Solution obtenue par l'algorithme pour s=1 (à gauche) et graphique de la tension résultante (à droite)

Ce résultat est cohérent avec le réseau de départ vu en figure 5 parce que le grand producteur est localisé dans le noeud 7, justement où l'algorithme va choisir de placer le self. Le graphique de la figure 6 montre que les tensions dans l'optimum sont vraiment restées entre les limites imposées par les contraintes.

Ensuite, nous avons obtenu les résultats pour 2, 3 et 4 selfs qui sont exposés dans les figures 7, 8 et 9, respectivement. Ces résultats sont moins prévisibles

que pour une self mais ils sont cohérents parce qu'à chaque fois la self active la plus grande est celle placée au noeud 7.

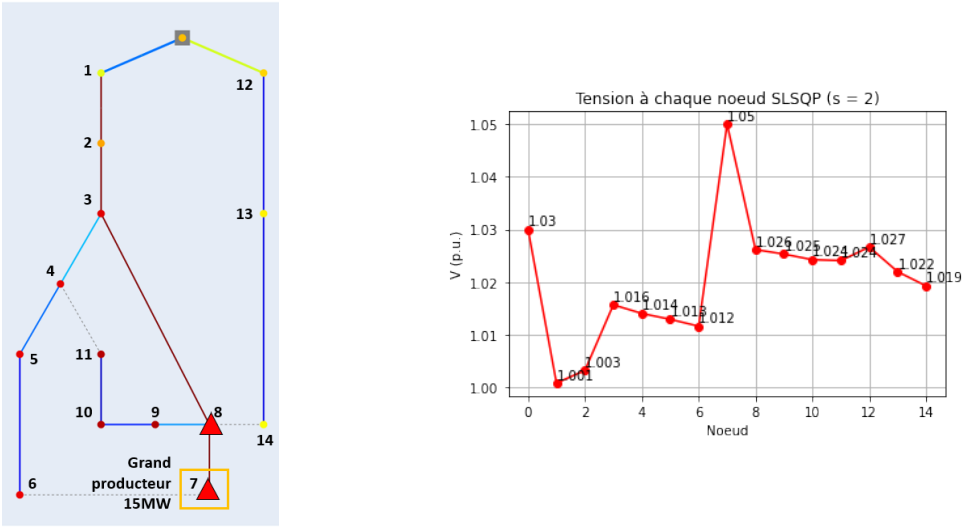

FIGURE 7 – Solution obtenue par l'algorithme pour s=2 (à gauche) et graphique de la tension résultante (à droite)

Pour le cas avec 2 selfs, nous obtenons une fonction de coût de 2.377 MVAr qui est supérieure à la fonction de coût pour 1 self. Les selfs ont été positionnées aux noeud 7, où $q_7 = 1,517$ MVAr, et noeud 8, où $q_8 = 0,86$ MVAr.

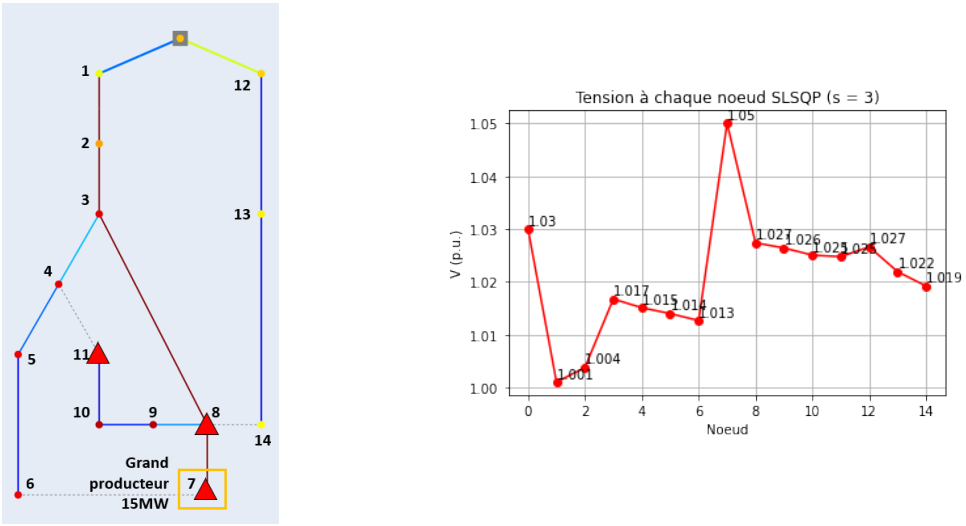

FIGURE 8 – Solution obtenue par l'algorithme pour s=3 (à gauche) et graphique de la tension résultante (à droite)

Pour le cas avec 3 selfs, nous obtenons une fonction de coût de 2.303 MVAr qui est supérieure à la fonction de coût pour 1 self. Les selfs ont été positionnées aux noeud 7, où $q_7 = 1,903$ MVAr, noeud 8, où $q_8 = 0,2$ MVAr et, finalement, noeud 11 où $q_{11} = 0,2$ MVAr. Ce résultat met plus en relief que le résultat pour deux selfs, en effet, le placement d'une seule self est plus adapté parce que l'algorithme place des selfs aux noeuds 8 et 11 dans la limite inférieure de puissance. Cela signifie que ces installations sont inutiles pour réduire l'effet de surtension et une seule self serait capable de le faire en minimisant la puissance totale installée. Ce fait est cohérent avec la présence d'un seul fort producteur qui provoque la surtension.

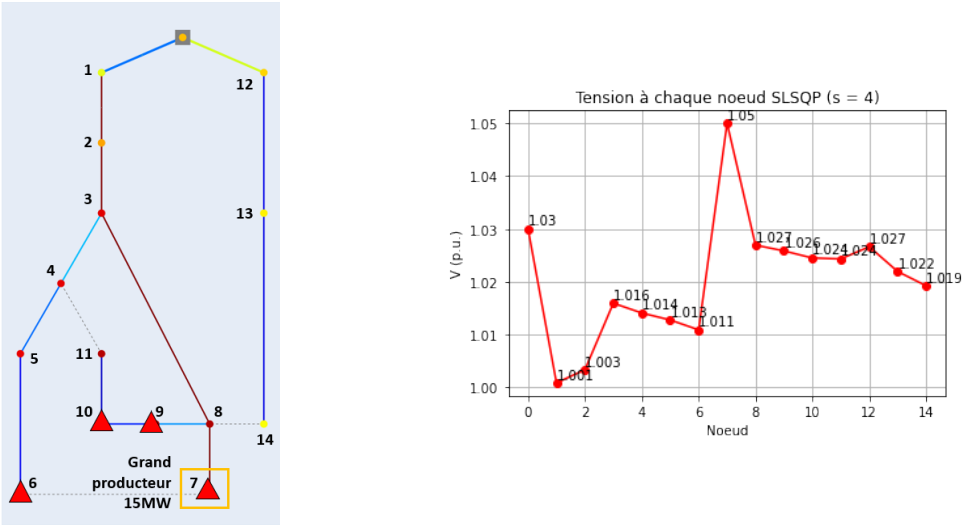

FIGURE 9 – Solution obtenue par l'algorithme pour s=4 (à gauche) et graphique de la tension résultante (à droite)

C'est analogue pour le cas avec 4 selfs, nous obtenons une fonction de coût de 2.359 MVAr qui est supérieure à la fonction de coût pour 1 self. Les selfs ont été positionnées aux noeud 7, où $q_7 = 1,759$ MVAr, noeud 6, 9 et 10 où $q_i = 0,2$ MVAr. Les résultats pour plus d'une self sont moins prévisibles que pour une self mais ceci est cohérent parce qu'à chaque fois la self active plus grande est celle placée au noeud 7. Les résultats sont mis ensemble dans le tableur 1 pour meilleur visualisation.

s	fonction objectif	selfs actives
1	2,238 MVAr	$q_7 = 2,238$ MVAr
2	2,377 MVAr	$q_7 = 1,517$ MVAr $q_8 = 0,86$ MVAr
3	2,303 MVAr	$q_7 = 1,903$ MVAr $q_8 = 0,2$ MVAr $q_{11} = 0,2$ MVAr
4	2,359 MVAr	$q_7 = 1,759$ MVAr $q_6 = 0,2$ MVAr $q_9 = 0,2$ MVAr $q_{10} = 0,2$ MVAr

TABLE 1 – Résultats d'optimisation pour le réseau 1 selon le nombre de selfs

De ces résultats, il est possible de conclure que pour ce réseau, il est plus recommandé d'installer une seule self parce que la fonction de coût est inférieur pour s = 1. En termes de temps de calcul, cet algorithme n'est pas optimisé parce qu'il prend 50 minutes pour réaliser l'optimisation pour des selfs supérieurs à 1.

2.3.2 Résultats SLSQP avec un autre réseau pour s variant entre 1 et 4

Afin d'analyser cet algorithme dans un cas plus intéressant où il y a deux producteurs principaux, le réseau CIGRE a été modifié différemment. La puissance du producteur au noeud 7 a été changée pour 8 MW et un nouveau grand producteur a été rajouté au noeud 11 avec 6 MW de production. Cette fois, les noeuds 1 et 2 ont eu leur puissance élevée à 10 MW. Le résultat est donc celui de la figure 10 où il est possible d'observer la position des grands producteurs et les tensions aux noeuds de ce réseau.

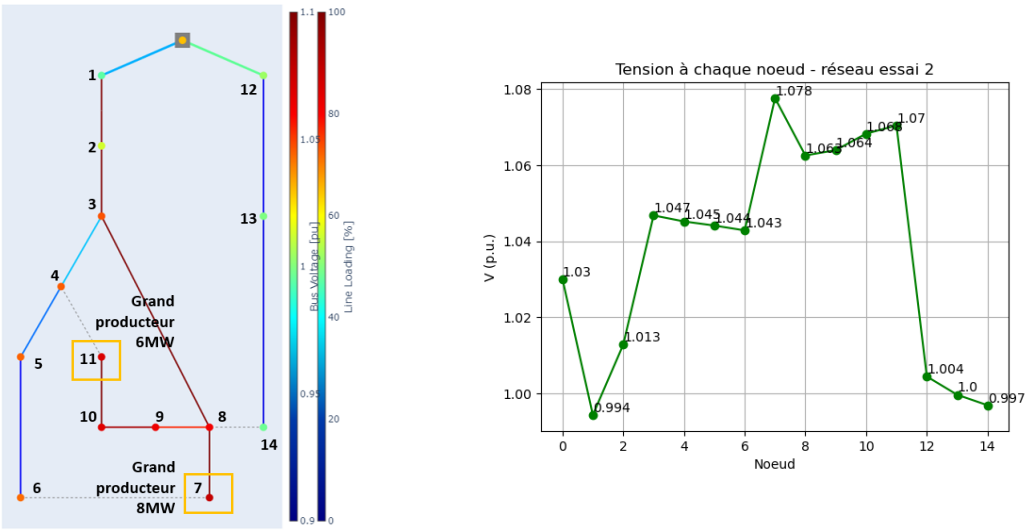

FIGURE 10 – Nouveau réseau d'essai plus complexe

En appliquant la méthode d'optimisation déterministe avec 50 itérations de multistart, nous avons trouvé les résultats suivants dans l'image de la figure 11 pour s = 1 (une self). Dans la figure les selfs sont représentées par des triangles bleus. La puissance totale installée optimale est de 1,03 MVAr.

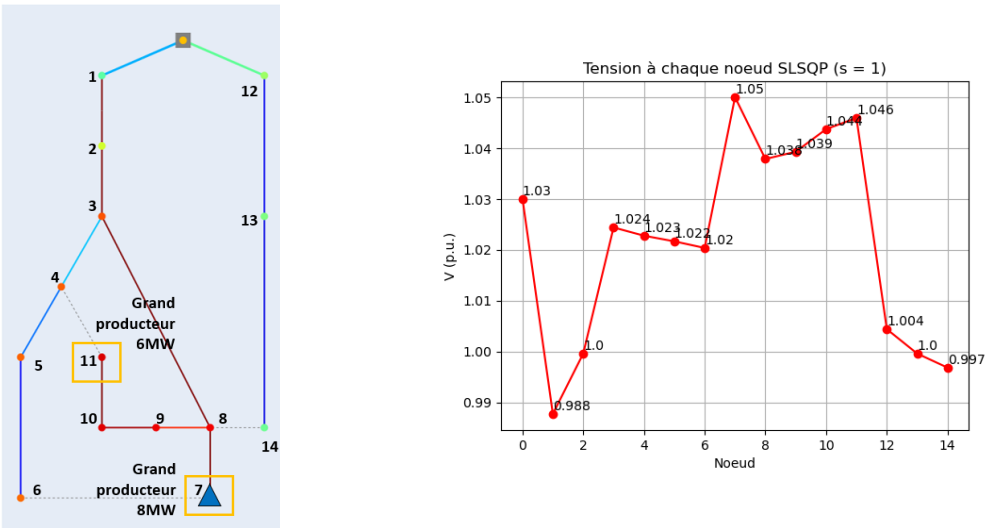

FIGURE 11 – Solution obtenue par l'algorithme pour s=1 (à gauche) et graphique de la tension résultante (à droite) - Résultats réseau d'essai

Ce résultat est cohérent avec le réseau de départ vu en figure 10 parce que le plus grand producteur est localisé dans le noeud 7, justement où l'algorithme va

choisir de placer la self. Le graphique de la figure 11 montre que les tensions dans l'optimum sont vraiment restées entre les limites imposées par les contraintes.

Ensuite, nous avons obtenu les résultats pour 2, 3 et 4 selfs qui sont exposés dans les figures 12, 13 et 14, respectivement. Ces résultats sont moins prévisibles que pour une self vu que ce réseau est plus complexe que l'ancien, mais ils réussissent aussi à avoir les tensions dans les contraintes.

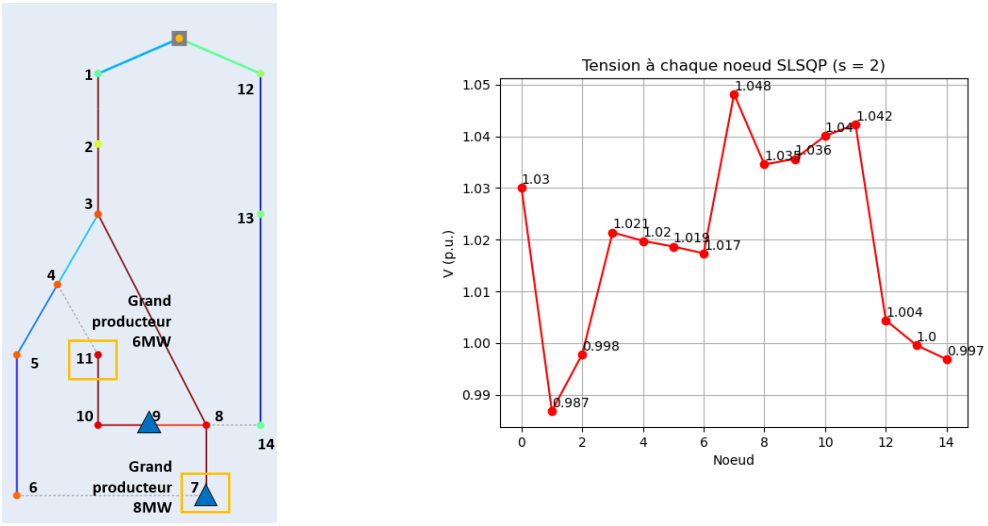

FIGURE 12 – Solution obtenue par l'algorithme pour s=2 (à gauche) et graphique de la tension résultante (à droite) - Résultats réseau d'essai

Pour le cas avec 2 selfs, nous obtenons une fonction de coût de 1.189 MVAr qui est supérieure à la fonction de coût pour 1 self. Les selfs ont été positionnées aux noeud 7, où $q_7 = 0.589$ MVAr, et noeud 9, où $q_9 = 0,6$ MVAr.

Pour le cas avec 3 selfs, nous obtenons une fonction de coût de 1.479 MVAr qui est supérieure à la fonction de coût pour 1 self. Les selfs ont été positionnées aux noeud 2, où $q_2 = 0,506$ MVAr, noeud 8, où $q_8 = 0,446$ MVAr et, finalement, noeud 11 où $q_{11} = 0,527$ MVAr. Ce résultat est distribué d'une manière qui "favorise" le producteur du noeud 11 par rapport au du noeud 7 en rajoutant des selfs plus proches du 11. Cette solution est sûrement plus coûteuse que les deux autres et ne semble pas être un optimum global. Cette configuration considérée peut être un optimum local donc la méthode n'a pas réussi à s'en sortir juste avec la génération des points initiaux aléatoires.

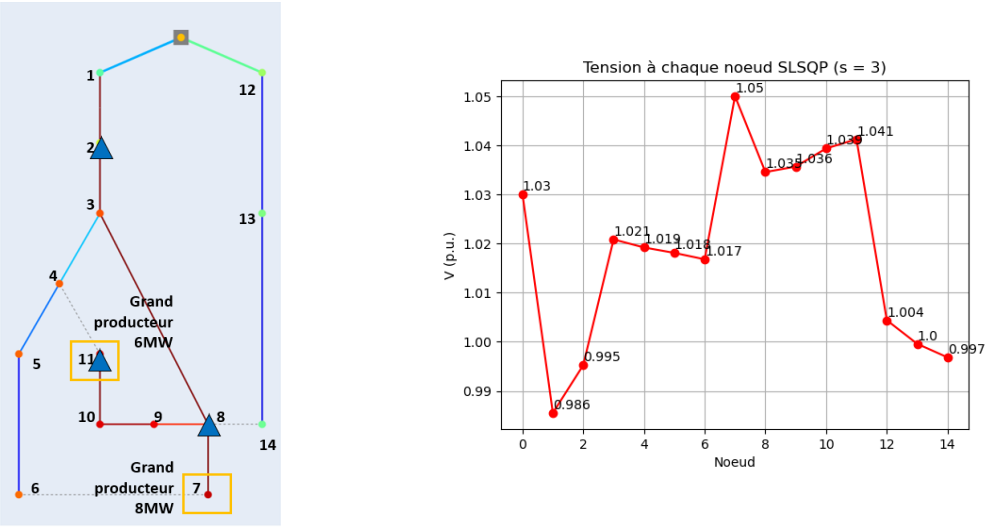

FIGURE 13 – Solution obtenue par l'algorithme pour s=3 (à gauche) et graphique de la tension résultante (à droite)

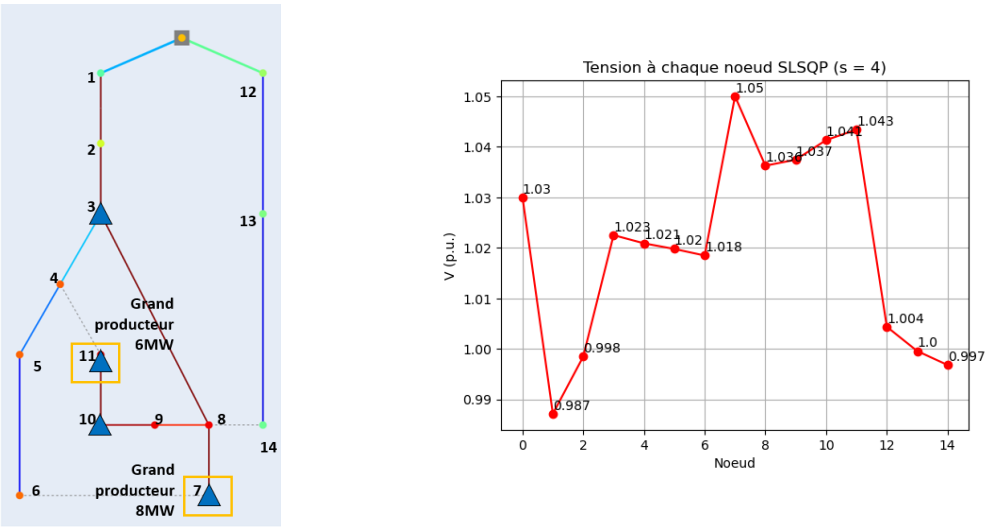

FIGURE 14 – Solution obtenue par l'algorithme pour s=4 (à gauche) et graphique de la tension résultante (à droite)

Dans le cas avec 4 selfs, nous obtenons une fonction de coût de 1.138 MVAr qui est supérieure à la fonction de coût pour 1 self. Les selfs ont été positionnées aux noeud 7, où $q_7 = 0,535$ MVAr, noeud 3, 10 et 11 où $q_i = 0,2$ MVAr. Les résultats pour plus d'une self sont moins prévisibles que pour une self mais ceci est cohérent parce qu'à chaque fois la self active plus grande est celle placée

au noeud 7 et les contraintes sont activées au niveau des autres noeuds. Les résultats pour ce deuxième réseau sont mis ensemble dans le tableur 2.

s	fonction objectif	selfs actives
1	1,103 MVAr	$q_7 = 1,103$ MVAr
2	1,189 MVAr	$q_7 = 0,589$ MVAr $q_9 = 0,6$ MVAr
3	1,479 MVAr	$q_2 = 0,506$ MVAr $q_8 = 0,446$ MVAr $q_{11} = 0,527$ MVAr
4	1,138 MVAr	$q_7 = 0,535$ MVAr $q_3 = 0,2$ MVAr $q_{10} = 0,2$ MVAr $q_{11} = 0,2$ MVAr

TABLE 2 – Résultats d'optimisation pour le réseau 2 selon le nombre de selfs

De ces résultats, il est possible de conclure que l'algorithme développé est adaptable à d'autres configurations de réseau. De plus, l'analyse de ce cas est beaucoup plus riche avec la présence de deux producteurs dominants et il est possible de conclure que, comme pour le dernier, 1 self est encore plus adaptée mais la solution avec 4 selfs a une fonction de coût bien basse également.

2.4 Optimisation avec l'algorithme génétique

De plus, pour vérifier notre solution, nous pouvons tester avec un autre algorithme. Nous utilisons donc un algorithme stochastique : l'algorithme génétique. L'Algorithme Génétique a été développé par John Henry Holland dans les années 1970. Il se base sur l'évolution des êtres vivants. Cet algorithme utilise 3 opérateurs d'évolution :

— La sélection : on choisit les individus les mieux adaptés.
— Le croisement : on mélange les particularités des individus choisis en utilisant la reproduction.
— La mutation : on altère aléatoirement les particularités d'un individu.

Ces trois opérateurs se basent sur les principes de la théorie de Darwin, respectivement : le principe d'adaptation, le principe d'hérédité et le principe de variation.

Le but est de converger vers un extremum local mais il faut éviter de l'atteindre trop vite pour essayer d'obtenir le résultat le plus optimal. Pour cela, la

mutation permet d'éviter une convergence prématurée de l'algorithme vers une solution.

L'avantage d'un algorithme de ce type est le temps de calcul. Celui-ci à un temps de calcul inférieur à celui de certaines méthodes déterministes, pour obtenir une solution approchée de la solution optimale. Cependant comme c'est un algorithme de type heuristique, il y a un risque que la solution proposée ne soit pas toujours optimale.

2.4.1 Reformulation du problème pour un problème mixte

Afin d'améliorer l'abordage de l'algorithme génétique, nous avons reformulé le problème de façon à prendre en compte une deuxième variable x qui correspond à la position des selfs actives sur le réseau. La formulation formelle est décrite ci-dessous :

Problème d'optimisation :

— Paramètres :
  — Nombre de selfs fixé = s
  — Tension nominale du réseau = V_n
  — Nombre de noeuds du réseau = N
— Variables :
  — $\forall i \in 1, ..., N$ (q_i est la puissance de la self au noeud i (si active)
  — $\forall i \in 1, ..., N$ (x_i indique s'il y a une self active au noeud i ($x_i = 1$) ou pas ($x_i = 0$))
— Fonction objectif à minimiser : $f(q_1, ..., q_N) = \sum_{i=1}^{N} q_i$
— Contraintes : $\begin{cases} n = \sum_{i=1}^{N} x_i = S \\ \forall i \in \{1, ..., N\}, 0,95V_n \leq V_i \leq 1,05V_n \\ \forall i \in \{1, ..., N\}, q_{act,i}\dot{x}_i \geq 0,2 \end{cases}$

Et ce problème peut être encore écrit comme dans la forme mathématique :

$$\min_{q,x} \quad \sum_{i=1}^{N} q_i \cdot x_i \qquad \text{s.t.} \quad \begin{aligned} V_i &\leq V_{max} \\ V_i &\geq V_{min} \\ q_{act,i} &\geq 0,2 \\ \sum_{i=1}^{N} x_i &= s \end{aligned} \tag{3}$$

D'un autre côté, il a fallu aussi définir une fonction de coût qui intègre les

contraintes du modèle sous forme de pénalités. Pour chaque point observé, on vérifie que celui-ci respecte chacune des contraintes. En cas de non respect d'une contrainte, une pénalité personnalisée au non-respect en question est accordée : on intervient avec une pénalité sous la forme alpha*Delta. Le Delta correspond à la différence entre la valeur du point observé et la valeur de la contrainte. Le alpha est un facteur qui permet de pondérer la pénalité accordée à cet écart (la pénalité sera tout de même plus élevée dans le cas d'un non-respect plus grand grâce au Delta).

Un point vigilance est aussi est le fait que nous avons limité l'espace d'optimisation pour les variables q_i entre 0 et 10 pour éviter que l'algorithme diverge en utilisant une borne supérieure élevée.

Pour l'étude paramétrique qui suit, on reprend le réseau CIGRE modifié de la figure 5.

2.4.2 Étude paramétrique - Probabilité de mutation

Le choix des paramètres d'entrée de l'algorithme génétique a une influence directe sur les résultats obtenus, notamment sur le taux de convergence des solutions vers un point de minimum global.

En conséquence de cette corrélation entre les paramètres et le résultat obtenu avec l'algorithme génétique, il est nécessaire de réaliser une étude paramétrique afin d'identifier le meilleur ensemble de paramètres pour que l'algorithme converge dans la plupart des cas où il est lancé.

Ainsi, l'algorithme génétique, tel que fourni par le package `geneticalgorithm` présent dans Python, peut recevoir comme paramètres : le nombre maximum d'itérations, la taille de la population, la probabilité de mutation, le taux d'élitisme, la probabilité de crossing-over, la proportion de la population parentale dans la population totale à chaque itération et le type de crossover utilisé. Dans cette étude, nous ne ferons varier que la probabilité de mutation (appelée m dans cette section), en gardant fixes tous les autres paramètres, à savoir un nombre maximum d'itérations de 500, une taille de population de 100 et un facteur de croisement de $0,7$.

Après avoir exécuté l'algorithme, en utilisant le réseau CIGRE modifié, pour différentes probabilités de mutation (m) et différents nombres de selfs (s), les résultats suivants ont été obtenus :

Pour les valeurs testées de la probabilité de mutation (m), la valeur de m entre 0.05 et 0.1, fournit des résultats plus proches des valeurs obtenues avec l'algorithme déterministe, ce qui est une indication de la meilleure valeur à choisir pour la probabilité de mutation. Les autres paramètres ont également

Nombre de selfs (s)	$m = 0.05$	$m = 0.1$	$m = 0.2$
1	2.25054	2.26438	2.71539
2	2.69498	2.53047	2.73522
3	2.46620	2.54038	4.19049
4	2.92201	2.72549	3.07126

TABLE 3 – Valeurs de la fonction objectif pour l'algorithme génétique pour différentes probabilités de mutation

une influence sur les résultats, mais à ce stade, ils n'ont pas été pris en compte dans cette étude paramétrique. Donc, comme étapes suivantes du projet, il serait envisageable réaliser l'étude paramétrique selon le nombre de générations ainsi que le nombre d'individus de chaque population.

2.5 Optimisation avec l'algorithme d'évolution différentielle

La motivation pour essayer une troisième approche pour l'optimisation du problème réside dans le fait que, parfois, les résultats trouvés par une méthode ne coïncident pas entre eux et, même s'ils le font, ils ne présentent pas nécessairement les mêmes temps de calcul. Compte tenu de cette possibilité et afin de ne pas limiter la technique d'optimisation stochastique à l'algorithme génétique, une approche utilisant un troisième algorithme méta-heuristique est réalisée.

Nous avons aussi essayer de résoudre notre problème avec un autre algorithme : l'algorithme d'évolution différentielle. Il fait partie des méthodes méta-heuristiques stochastiques d'optimisation. Il s'inspire de l'algorithme génétique. Pour réaliser cet algorithme, nous commençons par générer une population aléatoire de N éléments. Ensuite, c'est au tour de la boucle d'optimisation qui se déroule en deux étapes :

— La mutation
— Le croisement
— La sélection

Le croisement permet de créer une nouvelle solution. Pour cela, nous combinons le vecteur mutant et le vecteur cible. Nous choisissons au préalable un seuil de croisement CR. Cela permet de choisir le niveau d'impact du vecteur cible et du vecteur mutant sur la nouvelle solution.

Enfin, la sélection permettra de choisir la meilleure solution entre la nouvelle solution créée et celle du vecteur cible. (**Le pseudocode de l'algorithme**

d'évolution différentielle est à l'appendice)

L'avantage de ce type d'algorithme est qu'il permet de traiter des problèmes mixtes avec des contraintes non linéaires.

L'algorithme d'évolution différentielle peut être utilisé directement à partir de la bibliothèque `scipy`, qui, entre autres caractéristiques, fournit des utilitaires d'optimisation, tels que l'algorithme d'évolution différentielle lui-même, dont la documentation peut être consultée à l'adresse Scipy - Differential Evolution.

De même, comme cela a été fait avec l'algorithme génétique, la fonction objectif nécessite quelques modifications par rapport à la fonction définie à l'origine `fct_model_reseau`, où, à la valeur de la fonction objectif originale qui reflète la somme des puissances réactives, une pénalité est ajoutée pour chaque contrainte qui n'est pas respectée par l'ensemble des variables testées, ce qui rend le coût de la fonction objectif beaucoup plus élevé si des points situés en dehors du domaine de faisabilité sont testés.

Comme dans le cas de l'algorithme génétique, l'algorithme d'évolution différentielle fourni par la bibliothèque `scipy` contient plusieurs paramètres qui peuvent être modifiés, parmi lesquels le nombre d'itérations, la taille de la population, le taux de mutation et le taux de croisement (recombinaison).

Pour tester l'algorithme, les paramètres ont été fixés à un nombre maximal de 35 itérations, une taille de population de 200, un taux de mutation compris entre 0,5 et 1 et un taux de croisement de 0,8. On peut ainsi lancer l'algorithme pour différents nombres de selfs, entre 1 et 4, et obtenir la valeur de la fonction objectif associée, dont les résultats compilés suivent ci-dessous :

Nombre de selfs (s)	Fonction Objectif
1	41.4734
2	35.3249
3	31.5224
4	31.641

TABLE 4 – Valeurs pour la fonction objectif en utilisant l'algorithme d'évolution différentielle

Pour cette configuration, le temps d'exécution estimé de l'algorithme varie entre 10 et 15 minutes. L'exécution de l'algorithme montre que l'ensemble des variables se situe toujours dans les limites définies, l'algorithme d'évolution différentielle, tel que fourni par la bibliothèque Scipy, permet une exécution rapide de l'algorithme d'optimisation, bien que, pour la fonction testée et avec

les paramètres imposés, il ne renvoie pas de solution optimale au problème, mais les points obtenus sont des points faisables du problème.

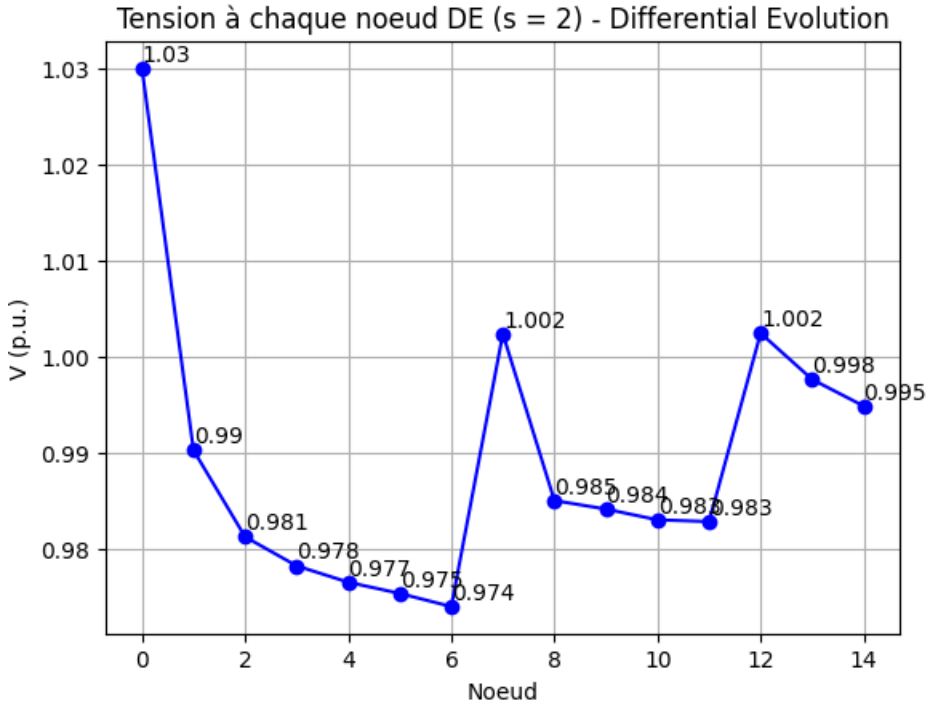

FIGURE 15 – Tension (en p.u) dans chaque noeud du réseau CIGRE modifié

Parmi les pistes d'amélioration de l'algorithme, on peut citer l'étude de l'algorithme pour différents paramètres, afin de trouver le meilleur ensemble de paramètres pour le problème, ainsi que la révision de la fonction objectif pour minimiser les temps de calcul et la valeur de la fonction objectif. Compte tenu du fait que les valeurs obtenues pour la fonction objectif sont supérieures d'un ordre de grandeur aux résultats obtenus avec les autres algorithmes, cela indique également une erreur possible dans les termes de pénalité inclus dans la fonction de coût, ce qui pourrait rendre artificiellement les solutions plus longues à trouver et l'ensemble optimal de points nécessiter plus d'itérations pour être atteint.

Bien que l'algorithme d'évolution différentielle soit déjà présent dans la bibliothèque Scipy, la difficulté réside dans l'adaptation du problème, en ajoutant les contraintes comme pénalités dans la fonction objectif d'origine dans `fct_model_reseau`, étant donné que la définition des contraintes, comme effectué pour l'algorithme déterministe, il semble encore être en phase de développement, avec l'apparition multiple d'erreurs lors de la tentative d'utilisation de la

définition des contraintes telle que décrite par la documentation lors de la tentative d'application de l'algorithme à ce problème. De plus, on remarque que, pour différentes tailles de population, le temps que met chaque génération (itération) à être calculée croît significativement, ce qui rend l'exploration paramétrique encore plus difficile, puisque, en plus des paramètres de mutation, de croisement et de sélection, il existe différentes stratégies de mutation qui peuvent être choisies, telles que `best2bin`, `best2exp`, `randtobest1exp` et 17 autres techniques différentes, ce qui rend le temps d'évaluation significatif.

2.6 Comparaison des résultats obtenus

Nous allons donc pouvoir comparer les différents résultats obtenus par nos algorithmes.

Résultats	SLSQP	Réseau modifié	GA	DE
Temps	50min	50min	17min	10-15min
Puissance	2.238 MVAr	1.03 MVAr	2.25054 MVAR	31.5224 MVAR
Selfs	1	1	1	3

TABLE 5 – Comparaison entre les différents algorithmes d'optimisation

Nous constatons donc que l'algorithme déterministe est celui qui permet d'avoir le résultat le plus précis et de minimiser la puissance réactive des selfs. Ceci est cohérent avec la théorie. Les algorithmes déterministes renvoient un résultat plus optimisé que les algorithmes stochastiques. Cependant, dès qu'on augmente le nombre de selfs et donc le nombre de cas à étudier, le temps de résolution devient long. Les algorithmes stochastiques comme l'algorithme génétique nous donnent un résultat assez proche de la valeur obtenue par l'algorithme SLSQP. De plus, le temps de résolution reste beaucoup plus faible que pour le SLSQP lorsque le nombre de cas augmente.

Nous ne prenons pas en compte la valeur obtenue par l'algorithme d'évolution différentielle car il n'est pas encore opérationnel (sûrement un problème dans les paramètres de la modélisation ou dans le code correspondant).

Ici, notre réseau est de petite taille donc l'algorithme SLSQP reste faisable mais pour des réseaux de taille plus importante, il devient beaucoup trop long. Il faut donc privilégier les algorithmes stochastiques.

3 Conclusion

Dans cette étude, nous sommes partis d'un réseau CIGRE contenant plusieurs producteurs d'énergie, en plus de la centrale principale, que nous avons désignés par l'acronyme GED. Le réseau CIGRE, tel qu'il était fourni par l'exemple du paquet pandapower, ne contenait initialement aucun nœud dont les tensions ne respectaient pas la restriction d'être comprises entre 0.95 et 1.05 p.u. Afin que l'étude réalisée dans le présent document puisse avoir lieu, le réseau a dû être légèrement modifié en introduisant des GED plus importants à certains nœuds du réseau original.

À partir de ce réseau, la tâche consistant à trouver le meilleur positionnement des selfs (éléments de caractère inductif) pour réguler les niveaux de tension en chaque point du réseau impliquait ensuite l'utilisation de deux types d'algorithmes : un algorithme déterministe et deux algorithmes stochastiques (ou méta-heuristiques) basés sur les caractéristiques évolutives de la biologie des populations. Ces algorithmes ont permis de trouver la meilleure configuration de selfs à installer dans le réseau afin de l'adapter aux normes.

Les résultats obtenus par les algorithmes déterministes et l'algorithme génétique (classe stochastique) ont permis d'obtenir des résultats cohérents avec le réseau et ses points critiques, c'est-à-dire les points où les GED ont une puissance excédentaire, provoquant une surtension dans le réseau. Les algorithmes ont donc réussi à converger vers une solution minimale en fonction du nombre de selfs que l'on souhaite installer.

D'autre part, le troisième algorithme, d'évolution différentielle (classe méta-heuristique), n'a pas impliqué de changements brusques dans la fonction objectif à minimiser par rapport à l'algorithme génétique, où il a utilisé des pénalités pour augmenter la valeur de la fonction de coût en cas de non-respect des contraintes, et, même étant un algorithme efficace et capable d'être exécuté de manière opportune. En plus d'être implémenté par défaut dans la bibliothèque Scipy, il n'a pas présenté des valeurs du même ordre de grandeur que les valeurs trouvées pour les deux autres algorithmes testés, bien qu'à la fin de son optimisation (lorsqu'il atteint le nombre maximum d'itérations), il renvoie des résultats cohérents qui respectent toutes les contraintes du réseau, comme souhaité, bien qu'il n'ait pas le coût minimum.

Entre-temps, il est possible d'obtenir une gamme pertinente de résultats en considérant tous les algorithmes mis en œuvre, chacun ayant ses qualités et ses défauts. L'un des principaux défauts est la nécessité d'ajuster de manière exhaustive les paramètres dans le cas des algorithmes stochastiques, en particulier pour l'évolution différentielle, où des éventuelles pénalités inadaptées à la fonction

peuvent causer une inefficacité apparente de l'algorithme.

Ainsi, sur la base de ces résultats, certains points d'amélioration sont envisagés, tels que la réalisation d'études sur le comportement d'autres paramètres dans la convergence vers la valeur optimale de la fonction de coût, parmi lesquels le nombre de générations et la taille de la population. Il est également envisagé une éventuelle reformulation du problème dans le cas de l'algorithme d'évolution différentielle et un réglage fin des limites pour les puissances qui peuvent être installées, en restreignant les possibilités. Par conséquent, il sera possible d'avoir une convergence plus rapide vers un minimum global de la fonction.

Références

[1] Mathieu BARRETTE : Methode de comparaison statistique des performances d'algorithmes Évolutionnaires. http://espace.etsmtl.ca/id/eprint/156/1/BARRETTE_Mathieu.pdf, 2008.

[2] Boris BERSENEFF : Réglage de la tension dans les réseaux de distribution du futur. *Université de Grenoble*, 2010.

[3] Houari BOUDJELLA : Controle des puissances et des tensions dans un réseau de transport au moyen de dispositifs facts. https://www.memoireonline.com/02/08/956/m_controle-puissances-tensions-reseau-transport-dispositifs-facts-svc19.html, 2008.

[4] Richard A. CLARKE.

[5] Fraunhofer IEE et University of KASSEL : Pandapower documentation - cigre networks. https://pandapower.readthedocs.io/en/v2.0.1/networks/cigre.html?highlight=cigre#medium-voltage-distribution-network-with-pv-and-wind-der, 2016-2019.

[6] David KNIGHT : The self-resonance and self-capacitance of solenoid coils : applicable theory, models and calculation methods., 05 2016.

[7] Alberto REATTI et Francesco GRASSO : Solid and litz-wire winding non-linear resistance comparison. volume 1, pages 466 – 469 vol.1, 02 2000.

[8] Benjamin H. WATERS, Brody J. MAHONEY, Gunbok LEE et Joshua R. SMITH : Optimal coil size ratios for wireless power transfer applications. *In 2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2045–2048, 2014.

[9] Rafal P. WOJDA : *Winding Resistance and Winding Power Loss of High-Frequency Power Inductors*. Thèse de doctorat, Wright State University, 2012.

Appendices

Pseudocode - Évolution Differentielle

Algorithm 1 Differential Evolution Pseudocode

Set Initial parameters (N, F, C, G) and bounds
Create storage vectors (f, P, V, U) for function, population, mutants and trial
Generate a random initial population in the feasible space Ω
for $g = 1$ to $g = G - 1$ **do**
  for $n = 1$ to $n = N$ **do**
    Mutation :
    Select 3 mutually exclusive vectors from the family P_g : X_1,X_2,X_3
    Generate a mutant vector $v_n^{(g+1)} = X_1 + F\,(X_2 - X_3)$
    Crossover :
    Create a set of index $\mathbb{I}$
    for $i = 1$ to $i = D$ **do**

$$u_{n,i}^{(g+1)} = \begin{cases} v_{n,i}^{(g+1)} \text{ , if } rand \leq C \text{ or } i = \mathbb{I} \\ x_{n,i}^{(g)} \text{ , otherwise} \end{cases}$$

    Selection
    for $i = 1$ to $i = D$ **do**
      Update values $\rightarrow \mathrm{u}_{n,i}^{(g+1)} = \max\left(m_i, \min\left(M_i, u_{n,i}^{(g+1)}\right)\right)$
      Greedy selection

$$x_{n,i}^{(g+1)} = \begin{cases} u_n^{(g+1)} \text{ , if } f\left(u_n^{(g+1)}\right) < f\left(x_n^{(g)}\right) \\ x_n^{(g)} \text{ , otherwise} \end{cases}$$