MU4MEN01 – Projet d'Optimisation

AKIL Adam, BUCLET Zeca, NOCHÉ Kévin

10 janvier 2025

Table des matières

Introduction

Dans ce présent document, nous allons étudier un modèle représentant un train, sa consommation, sa batterie et comment optimiser deux choses :

— La capacité de ladite batterie.

— La chute de tension maximale aux bornes du train, sachant que nous ne devons pas atteindre moins de 500 V aux bornes du train.

Le modèle auquel nous nous intéressons a été codé en python (le code est joint au document), et prend en compte, de façon non-exhaustive, les déplacements du train, sa vitesse, son accélération, sa puissance consommée et la puissance de la LAC (ligne aérienne de contact). Le fichier `marche.txt`, qui nous a été remis dès le début du projet, indique les déplacements du train en fonction du temps. De rapides et simples calculs nous permettent de trouver la vitesse du train, puis son accélération, primordiales pour la suite du projet.

La fonction `Simulation()`

Sans Batterie

Avec Batterie

Monte-Carlo

Algorithme NSGA-II

L'algorithme génétique NSGA-II permet de trouver les meilleures solutions dans un contexte donné de façon assez efficace. Cet algorithme génère une population de taille N d'individus (les solutions), puis recherche au sein de cette population les « solutions non-dominées », et les range en suivant cette ordre :

— En premier, les solutions non-dominées, c'est-à-dire celles qui dominent toutes le autres, les meilleures de la population.

— En deuxième, les solutions non-dominées qui restent, en ayant retirées les solutions précédentes.

— En troisième, les solutions non-dominées restantes, en ayant retirées toutes les précédentes.

— On continue jusqu'à épuisement des individus dans la population de départ.

Ayant maintenant à notre diposition un classement des individus selon leurs performances, on sélectionne les 50% meilleurs dans ce classement, puis les croisons afin de créer la génération suivante. Puis le cycle recommence, pour un total de N itérations.

Il faut savoir que NSGA-II, demandant un classement des solutions selon leur dominance, doit utiliser une fonction qui classe les solutions selon cette ordre. Pour ceux qui regarderait notre code, il s'agit de la fonction `non_dominant_sort()`.

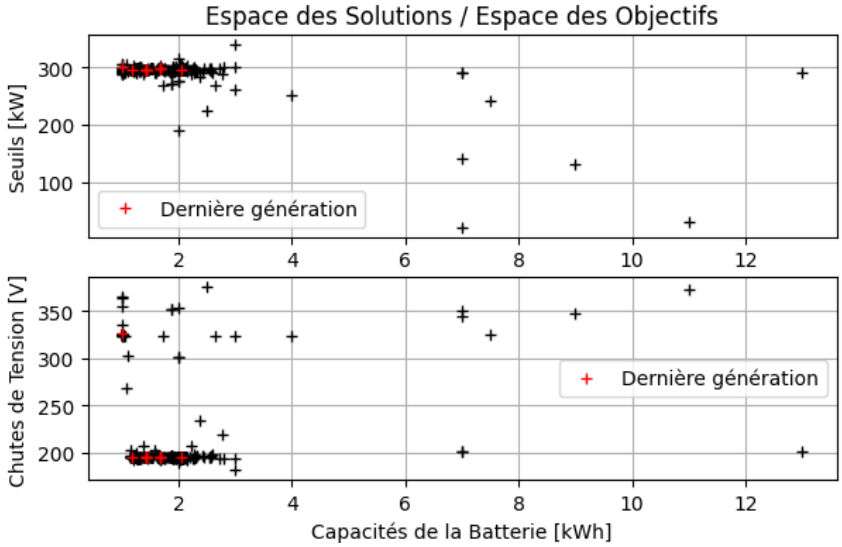

FIGURE 1 : Résultats de l'algorithme NSGA-II.

Comme affiché sur la figure 1, les solutions intéressantes pour notre batterie semblent se trouver aux alentours de 1 ou 2 kWh, pour un seuil vers les 300 kW et une chute de 200 V (le points rouges représentant la dernière génération créée par notre algorithme). Suivant la simulation, les résultats peuvent énormément changer, et il est conseiller de relancer plusieurs fois l'algorithme afin de trouver des solutions potentiellement meilleures. Par ailleurs, étant donné l'absence de « *crowding_distance* », nommé « critère de distance » en français, nos solutions tendent à se rassembler en un amas compact ; augmenter la mutation peut sensiblement améliorer ce problème.

Conclusion