Rapport recherche locale

Le code est basé sur la **métaheuristique** du recuit simulé. J’ai choisi cette méthode pour sa simplicité et sa capacité d’éviter des minimums locaux. Le pseudo-code de base est celui vu en cours. Le principe général est qu’on explore un voisinage et plus une valeur (la température) est élevée, plus la probabilité qu’on accepte une solution mauvaise est élevée. Plus on avance dans le processus, plus la température diminue et moins on a de chance d’accepter une mauvaise solution.

La fonction de résolution est appelée solve\_simulated\_annealing et prend en paramètre la température initiale et le pas entre chaque température.

Tout d’abord, on commence l’algorithme par une **solution initiale**. Cette solution correspond à la solution naïve qui ouvre tous les générateurs et associe les machines au générateur le plus proche. Cela nous assure de ne pas trouver un résultat pire que celle de la solution naïve. On associe cette solution à la variable best\_solution.

On initialise la variable current\_temperature à la température initiale choisie. L’algorithme utilise une température initiale élevée, soit de 300 pour favoriser l’exploration des voisins. Combinée avec le pas de 0.01, cela nous permettra aussi d’avoir un grand nombre d’itérations tout en restant au-dessous de la limite de 5 minutes pour rouler le code.

Ensuite, on cherche de nouvelles solutions selon le **critère d’arrêt** choisi. Le critère d’arrêt correspond au moment où la température actuelle franchi la barre du 0. Pour chaque itération, on trouve un voisin aléatoirement à partir de la solution actuelle (décrit plus bas). Ce voisin correspond à la variable new\_state. Ensuite, on détermine si on garde ou non cette nouvelle solution. La **fonction de sélection** correspond au processus suivant : (1) On calcule la variable delta qui correspond à différence entre le coût du voisin et celle de la solution actuelle. (2) Si cette valeur est inférieure, c’est-à-dire que la nouvelle solution a un coût plus bas que la solution actuelle, alors on remplace la solution actuelle pour celle du voisin. Sinon, on accepte ou non le voisin selon une probabilité donnée par la méthode is\_accepted . Cette probabilité dépend de la température actuelle et du delta. (3) Finalement, on vérifie si le coût de solution actuelle est inférieur au coût de notre meilleure solution à date. Si c’est le cas, on associe best\_solution à cette solution. À la fin de l’itération, on diminue la température actuelle selon la valeur alpha qui correspond au pas.

La **fonction du voisinage** est appelée find\_random\_neighbor. Elle choisit au hasard un générateur. Si celui-ci est fermé, elle appelle la fonction open\_generator\_and\_reassign\_devices qui va ouvrir le générateur et réassocier les machines aux générateurs ouverts les plus proche. Si le générateur est fermé, elle appelle la fonction close\_generator\_and\_reassign\_devices qui va fermer le générateur et réassocier les machines qui lui étaient associées selon le générateur ouvert le plus proche. Elle vérifie aussi qu’il y a toujours un générateur d’ouvert. Si le générateur choisi est le dernier ouvert, elle ouvre un autre générateur au hasard.

Le résultat est best\_solution.