Planification Dynamique des Ateliers avec DQN (Deep Q-Network)

Contexte et Importance du Sujet

Dans de nombreuses industries, telles que la production manufacturière, les travaux sont souvent organisés sous forme d'**ateliers**, où plusieurs machines effectuent différentes étapes (ou opérations) sur des produits ou des pièces. L'objectif principal de la planification dans ces environnements est de **minimiser les retards** tout en utilisant efficacement les ressources disponibles, telles que les machines, et en répondant aux demandes dynamiques des clients.

Cependant, cette tâche est complexe en raison de plusieurs facteurs :

- **Concurrence entre les travaux :** Plusieurs tâches peuvent avoir besoin des mêmes machines.
- **Arrivées dynamiques :** Les nouveaux travaux arrivent de manière aléatoire, rendant la planification fixe inefficace.
- **Contraintes des dates limites :** Les produits doivent être terminés avant une date précise.

Problème clé : Comment concevoir un système intelligent capable de planifier les opérations dans un atelier dynamique tout en minimisant les retards globaux et en s'adaptant à des perturbations constantes ?

Problématique

Objectifs de planification

L'objectif est de minimiser le **taux moyen de retard** des travaux, c'est-à-dire le temps dépassant les dates limites spécifiées.

Contexte dynamique

- Les travaux arrivent selon un processus aléatoire basé sur une **loi de Poisson**.
- Les opérations d'un travail doivent être traitées dans un ordre spécifique (ordre séquentiel).
- Chaque opération peut être effectuée sur une ou plusieurs machines compatibles.

Contraintes

- Une machine ne peut effectuer qu'une seule tâche à la fois.
- Les machines doivent être attribuées aux opérations de manière à maximiser l'efficacité globale tout en respectant les dates limites.

Pourquoi une approche basée sur le DQN ?

Les approches classiques, telles que les règles heuristiques (SPT, EDD, etc.), sont limitées dans leur capacité à s'adapter à des environnements dynamiques et incertains. Un agent d'apprentissage par renforcement profond (DQN) peut apprendre à optimiser les décisions en fonction des états dynamiques du système.

Formulation du Problème

Travaux et opérations

Chaque travail J_i contient n_i **opérations** qui doivent être effectuées dans un ordre spécifique $O_{i,1}, O_{i,2}, \ldots, O_{i,n_i}$.

- Chaque opération $O_{i,j}$ peut être réalisée sur une ou plusieurs **machines compatibles** $M_k \in \mathcal{M}_{i,j}$.
- La durée de traitement de $O_{i,j}$ sur une machine M_k est donnée par $p_{i,j,k}$.

Dates limites des travaux

La date limite D_i pour un travail J_i est calculée comme suit :

$$D_i = A_i + \left(\sum_{j=1}^{n_i} \bar{p}_{i,j} \right) \cdot \text{DDT}$$

où :

- A_i est le moment où le travail J_i arrive dans l'atelier.
- $\bar{p}_{i,j}$ est le temps moyen de traitement de l'opération $O_{i,j}$ sur toutes les machines compatibles.
- **DDT (Due Date Tightness)** est un paramètre indiquant la flexibilité des dates limites.

Environnement dynamique

1. **Arrivée des travaux :** Les nouveaux travaux arrivent de manière aléatoire selon une loi exponentielle avec un paramètre λ, qui représente le temps moyen entre deux arrivées consécutives.
2. **Actions possibles :** À chaque instant de décision, l'agent peut choisir quelle **opération** attribuer à quelle **machine disponible**, en s'appuyant sur des règles heuristiques ou une politique apprise.

3. **Récompenses :** La récompense est basée sur le **taux de retard** :

$$R_t = -(\text{taux de retard actuel} + \text{taux de retard attendu})$$

Approche Basée sur le DQN

Structure de l'Agent

L'agent DQN utilise un réseau neuronal pour approximer une fonction $Q(s, a)$, qui évalue la qualité d'une action a dans un état s.

Entrées (espace des états)

L'état s contient des informations sur l'atelier, telles que :

- Le **taux actuel de retard** des travaux et des opérations.
- Les **taux d'avancement** des travaux (opérations terminées, temps écoulé).
- Les **dates limites** et **temps restants** des travaux et des opérations.
- Les **charges actuelles des machines**.

Sorties (espace des actions)

Les actions possibles consistent à :

- Choisir une opération spécifique à attribuer à une machine particulière.
- L'agent peut utiliser des heuristiques pour prendre des décisions rapides, telles que :
  - **SPT (Shortest Processing Time)** : choisir l'opération la plus rapide.
  - **EDD (Earliest Due Date)** : choisir l'opération avec la date limite la plus proche.
  - **MCR (Minimal Critical Ratio)** : minimiser le ratio entre temps restant et durée restante.

Entraînement

- L'agent est entraîné à travers des épisodes simulés où il explore différentes stratégies pour minimiser les retards.
- Un **replay buffer** est utilisé pour réutiliser les expériences passées, ce qui améliore la stabilité de l'apprentissage.
- La méthode d'**epsilon-greedy** est utilisée pour équilibrer exploration et exploitation.

Processus d'Entraînement

1. **Simuler l'environnement :** Créez un atelier avec un ensemble initial de travaux, puis ajoutez dynamiquement de nouveaux travaux selon une loi de Poisson.
2. **Collecte de données :** Pour chaque épisode, l'agent prend des décisions, reçoit des récompenses et met à jour sa politique.
3. **Optimisation des hyperparamètres :** Utilisez une bibliothèque comme **Optuna** pour optimiser les paramètres du modèle, notamment :
   - La taille des couches cachées.
   - Le taux d'apprentissage.
   - La taille des mini-lots.
4. **Évaluation :** Comparez les performances de l'agent avec des règles heuristiques classiques.

Livrables Attendus

1. **Code complet :**
   - Simulateur de l'atelier.
   - Implémentation de l'agent DQN.
   - Visualisation des décisions (diagrammes de Gantt).
2. **Analyse des résultats :**
   - Réduction progressive des retards au fil des épisodes.
   - Comparaison avec les heuristiques classiques.
3. **Rapport détaillé :**
   - Description des défis et des solutions apportées.
   - Analyse des performances.

Pourquoi ce Projet est Pertinent ?

- **Applications réelles :** Optimisation industrielle, logistique, et gestion des ressources.
- **Approche innovante :** Combiner apprentissage par renforcement et heuristiques classiques.
- **Développement de compétences clés :**
  - Programmation (Python, PyTorch).
  - Apprentissage par renforcement.
  - Simulation et optimisation.

Bonne chance dans cette aventure d'optimisation dynamique !