Rapport sur l'Algorithme Heuristique $1|ri|\bar{C}$

Introduction

L'algorithme heuristique $1|ri|\bar{C}$ est conçu pour résoudre le problème d'ordonnancement sur une machine unique où chaque tâche est associée à un temps de disponibilité (r_i) et un temps de traitement (p_i). L'objectif principal est de minimiser le **temps moyen de complétion**, noté $\bar{C}$, défini comme :

$$\bar{C} = \frac{\sum_{i=1}^{n} C_i}{n},$$

où C_i est le temps de complétion de la tâche T_i.

Description du Problème

Le problème peut être décrit par les paramètres suivants :

- n : Nombre total de tâches.
- r_i : Temps de disponibilité de la tâche T_i, c'est-à-dire le moment où T_i peut être prise en charge.
- p_i : Temps de traitement de la tâche T_i, c'est-à-dire la durée nécessaire pour exécuter T_i.

L'objectif est de déterminer un ordre d'exécution des tâches afin de minimiser le temps moyen de complétion $\bar{C}$.

Fonctionnement de l'Algorithme

L'algorithme heuristique suit une approche gloutonne. Les étapes principales sont décrites ci-dessous :

Étapes de l'Algorithme

1. **Initialisation** :
   - Définir le temps courant t comme le plus petit temps de disponibilité : $t = \min(r_i)$.
   - Déterminer l'ensemble des tâches disponibles à t.
2. **Sélection de la tâche** :

- Parmi les tâches disponibles ($r_i \leq t$), sélectionner celle ayant le plus petit temps de traitement (p_i).

3. **Exécution de la tâche** :
   - La tâche sélectionnée est exécutée à partir de t.
   - Mettre à jour le temps courant : $t = t + p_i$.
   - Retirer la tâche de l'ensemble des tâches restantes.
4. **Mise à jour de l'ensemble des tâches disponibles** :
   - Si aucune tâche n'est disponible ($r_i > t$), avancer le temps courant au prochain temps de disponibilité : $t = \min(r_i)$.
5. Répéter les étapes jusqu'à ce que toutes les tâches soient exécutées.
6. **Calcul de la complétion** :
   - Enregistrer le temps de complétion C_i de chaque tâche.
   - Calculer $\bar{C} = \frac{\sum_{i=1}^{n} C_i}{n}$.

Pseudo-code de l'Algorithme

Le pseudo-code de l'algorithme est le suivant :

```
def scheduling_heuristic(n, release_times, processing_times):
    tasks = list(range(n))
    t = min(release_times)
    som = 0
    schedule = []

    while tasks:
        available_tasks = [i for i in tasks if release_times[i] <= t]
        if not available_tasks:
            t = min(release_times[i] for i in tasks)
            continue

        task = min(available_tasks, key=lambda x: processing_times[x])
        schedule.append((task, t))
        t = max(t + processing_times[task],
                min((release_times[i] for i in tasks if i != task), default=0))
        som += t

        tasks.remove(task)

    avg_completion_time = som / n
    return schedule, avg_completion_time
```

Figure 1: Représentation du pseudo-code de l'algorithme.

```
Entrées :
  - n : Nombre de tâches
  - r : Temps de disponibilité des tâches
  - p : Temps de traitement des tâches

Initialisation :
  - t = min(r)
  - som = 0
  - T = {T1, T2, ..., Tn} (ensemble des tâches non exécutées)

Tant que T n'est pas vide :
  - Trouver les tâches disponibles R = {Ti  T : ri  t}
  - Si R est vide :
      - t = min(ri pour Ti  T)
  - Sinon :
      - Sélectionner Tj  R avec p_j minimal
      - Ajouter (Tj, t) au Résultat
      - t = t + pj
      - Supprimer Tj de T
      - som += t

Sortie :
  - Résultat (ordre des tâches et temps de début)
  - C = som / n
```

Exemple d'Exécution

Considérons $n = 3$ tâches avec les paramètres suivants :

$$r = [0, 1, 2], \quad p = [3, 2, 1].$$

Tâche	Temps de disponibilité (r_i)	Temps de traitement (p_i)
T_1	0	3
T_2	1	2
T_3	2	1

Table 1: Paramètres des tâches.

Étapes de l'algorithme :

1. Initialisation : $t = 0$.
2. À $t = 0$, seule T_1 est disponible.
3. Exécution de T_1 : $t = 3$, $C_1 = 3$.
4. À $t = 3$, T_2 et T_3 sont disponibles. Sélection de T_3.
5. Exécution de T_3 : $t = 4$, $C_3 = 4$.

6. Exécution de T_2 : $t = 6$, $C_2 = 6$.

Résultat :

$$\bar{C} = \frac{C_1 + C_2 + C_3}{3} = \frac{3 + 6 + 4}{3} = 4.33.$$

Complexité

- **Complexité de sélection** : $O(n)$ pour trouver la tâche avec le plus petit temps de traitement.
- **Complexité globale** : $O(n^2)$ car l'algorithme itère n fois, et chaque itération effectue une recherche dans $O(n)$.

Conclusion

L'algorithme $1|ri|\bar{C}$ fournit une solution approchée pour le problème d'ordonnancement. Bien qu'il ne garantisse pas une solution optimale, il est efficace pour des instances de taille modérée et constitue une bonne base pour des approches plus avancées.