École Nationale des Ponts et Chaussées

ATRSC : PROJET

GESTION OPTIMISÉ D'ATELIER

Élève :

Murilo COSTA CAMPOS DE MOURA

Enseignant :

Frédéric Meunier

2021-2022

Minimiser le temps de séjour moyen

Question 1

Pour avoir une borne inférieure nous allons relâcher la contrainte de disponibilité des travailleurs. Le problème devient alors simplement un job shop avec une règle de priorité FIFO. Vu que chaque instance difère seulement par rapport aux travailleurs, la même borne inférieure calculée sera appliqué à toutes les instances du problème.

Chaque machine du atelier m sera traitée comme un système $M/G/1$. Par le Théorème de Pollaczek–Khinchin nous avons le résultant suivant pour le temps d'attente dans ce type de système :

$$\mathbb{E}[W] = \frac{\lambda\,\mathbb{E}[U^2]}{2(1-\rho)} \tag{1}$$

Avec ce resultat, c'est possible de calculer l'espérance du temps de séjour moyen par type de produit t en utilisant la formule suivante, étant M_t l'ensemble des machines du parcours de t :

$$\mathbb{E}[S_t] = \sum_{M_t} \mathbb{E}[W_m] + \mathbb{E}[U_{tm}] \tag{2}$$

Finalement, nous allons tous les donnés nécessaires pour faire les calcules.

t	λ	Parcours	$\mathbb{E}[U_{tm}]$	$\mathbb{E}[U_{tm}^2]$
1	0.29	1,2,3,4,8	0.68, 0.40, 0.87, 0.26, 0.93	0.47, 0.17, 0.76, 0.07, 0.87
2	0.32	2,4,7	0.64, 0.76, 0.43	0.4, 0.57, 0.19
3	0.47	3,5,1	0.60, 0.46, 0.49	0.37, 0.21, 0.25
4	0.38	5,6,7,8	0.43, 0.66, 0.81, 0.28	0.19, 0.43, 0.66, 0.08

TABLE 1 – Inputs pour chaque type de produit t

Dans le tableau 2, les inputs sont aggregés pour chaque machine.

m	λ_m	$\mathbb{E}[U_m]$	$\mathbb{E}[U_m^2]$	ρ_m
1	0.76	0.63	0.40	0.48
2	0.61	0.52	0.29	0.32
3	0.76	0.61	0.42	0.47
4	0.61	0.52	0.33	0.32
5	0.85	0.47	0.22	0.40
6	0.38	0.66	0.43	0.25
7	0.70	0.64	0.45	0.45
8	0.67	0.56	0.42	0.37

TABLE 2 – Inputs pour chaque machine m

À partir de l'équation (1), le résultat suivant a été obtenu pour l'espérance du temps d'attente dans chaque machine.

m	1	2	3	4	5	6	7	8
$\mathbb{E}[W_m]$	0.30	0.13	0.30	0.15	0.16	0.11	0.28	0.23

Par l'équation (2), nous obtenons enfin les bornes inférieurs du temps de séjour moyen par type de produit.

t	1	2	3	4
$\mathbb{E}[S_t]$	4.228	2.375	2.300	2.951

Globalement, le temps moyen de séjour sera alors 2.869.

Question 2

Dans la travail de Rajendran et Holthaus [1], plusieurs règles de priorité sont testées à afin d'essayer d'optimiser le temps moyen de séjour. Parmi celles qui ont été simulées, celle qui a obtenu le meilleur résultat pour cette métrique est la règle PT+WINQ qui sera ensuite appelée juste comme WINQ. L'algorithme est le suivant :

Algorithm : WINQ

---

1 : **Input** : l'ensemble de travailleurs, de machines et produits en attente
2 : **Output** : un choix de machine pour le travailleur
3. Nous cherchons d'abbord l'ensemble des machines en attente que le travailleur est qualifié pour manipuler
4. Pour chaque machine de cet ensemble, nous calculons la somme des temps de traitement futurs des produits en attente
5. Nous choisissons envoyer le travailleurs à la machine dont la somme de ces temps est la plus courte.
End

Question 3

L'algorhitme a été implementé comme suit, au sein de la méthode algo de la classe Worker

---

```
import numpy as np
if METHOD == "WINQ":
    candidate_machines={}
    i = 0
    while (i < NBR_MACHINES):
        if sys.machines[i].awaiting and self.qualifications[i]:
            candidate_machines[i]=0
        i += 1

    for i in candidate_machines:
        #retrieves the awaiting product of the candidate machine
        product = sys.machines[i].service[0]
        #finds in which part of the rout the product is
        current_machine = ROUTES[product.type][product.current_step]
        remaining_time = np.sum(product.processing_times[current_machine:])
        candidate_machines[i] = remaining_time
```

```
    if candidate_machines != {}:
        best_machine = min(candidate_machines, key=candidate_machines.get)
```

Cette implémentation du algorhitme a obtenu les résultats suivantes, testés sur 20 replications de la simulation :

Instance	$\overline{S}$	Intervalle de confiance (95%)
1	5.3283	(5.2725, 5.3840)
2	4.1028	(4.0533, 4.1524)
3	3.8338	(3.8030, 3.8647)
4	2.7566	(2.7491, 2.7642)

TABLE 3 – Résultats avec la règle WINQ

Instance	$\overline{S}$	Intervalle de confiance (95%)
1	8.3884	(8.2453, 8.5315)
2	4.8903	(4.7874, 4.9931)
3	4.4004	(4.3618, 4.4391)
4	2.7920	(2.7840, 2.8000)

TABLE 4 – Résultats originales avec la règle FIRST

Nous pouvons constater que pour chacune des instances du problème la nouvelle règle WINQ a eu comme résultat un temps de séjour moyen plus petit que la simulation originale. Nous concluons alors que l'algorithme a réussi à minimiser cette métrique.

Équilibrer la charge des employés

Question 4

Comme dans n'importe quel instance chaque travailleur a des qualifications différentes et chaque type de produit, avec son propre taux d'arrivé, suit une route de machines particulières, c'est normal que certains travailleurs restent occupés plus longtemps que d'autres. D'autant plus que, initiallement, rien est fait pour repartir la charge plus équitablement.

Question 5

La stratégie sera d'implementer une règle de priorité Greedy pour la politique du choix de l'employé dans la salle d'attente. Maintenant avec cette règle, si plusieurs travailleurs sont qualifiés pour la même machine, celui dont l'utilisation est la plus faible jusqu'à présent sera choisi.

Question 6

Pour l'implementation de cette règle, la fonction worker_available sera changé. Sa nouvel version a été implementé comme suit :

```
def worker_available(self, machine):
    qualified_worker = []
```

```
for i in range(len(self.waiting)):
    if self.waiting[i].qualifications[machine.id]:
        qualified_worker.append(i)

utilisation={}
for j in qualified_worker:
    utilisation[j]= self.waiting[j].worked_time

if qualified_worker != []:
    best_worker = min(utilisation, key=utilisation.get)
    worker = self.waiting.pop(best_worker)
    machine.worker.append(worker)
    return True
else:
    return False
```

Un nouvel attribut appelé worked_time a été créé au sein de la classe Worker, dans le but de faciliter le calcul du temps d'utilisation en le gardant dans la mémoire. Sa valeur initiale est 0 et il est mis à jour à travers de cet extrait de code dans la classe Event_end_service.

```
self.machine.worker[0].worked_time
    +=(self.time-self.machine.worker[0].start_times[-1])
```

Finalement, les résultats obtenus étaient les suivants, en gardant la règle FIRST initiale :

	Instances			
Travailleur	1	2	3	4
1	0,7997	0,7183	0,6224	0,5077
2	0,782	0,78	0,3437	0,5077
3	0,6452	0,7799	0,3433	0,5077
4	0,8201	0,7683	0,7706	0,5077
5			0,3438	0,5078
6			0,6226	0,5077
sigma	0,07924	0,02940	0,18776	0,00004

TABLE 5 – Résultats avec la politique Greedy

	Instances			
Travailleur	1	2	3	4
1	0,7997	0,6882	0,6084	0,4832
2	0,782	0,8022	0,2994	0,4939
3	0,6452	0,7676	0,4023	0,555
4	0,8201	0,7885	0,7706	0,5495
5			0,329	0,4447
6			0,6366	0,52
sigma	0,07924	0,05098	0,19095	0,04218

TABLE 6 – Résultats avec la politique originale

Nous pouvons noter que, dans tous les instances (sauf la première où il n'y a pas des superpositon de qualification entre les travailleurs), la politique Greedy a fourni une

répartition de la charge plus équilibrée, ce qui peut être constaté par l'écart type plus petit.

1 Bibliographie

Références

[1] C. Rajendran and O. Holthaus, A comparative study of dispatching rules in dynamic flowshops and jobshops, European journal of operational research 116 (1999), no. 1, 156–170