Rapport : Job-Shop Scheduling Problem

BENKHALI KHALIL

October 8, 2024

1 Introduction

Dans ce rapport, nous présentons un programme écrit en C++ qui permet de lire les données d'un graphe représentant des jobs et des machines, et de calculer la machine ayant la durée cumulée la plus longue. Ce calcul est réalisé à partir d'un fichier texte contenant les informations sur les jobs et les machines. Le résultat est ensuite stocké dans une structure et affiché à l'utilisateur.

2 Description du problème

Le problème consiste à lire un fichier texte contenant des informations sur un ensemble de jobs et de machines. Chaque ligne du fichier (à partir de la deuxième) représente un job et contient des paires (machine, durée), qui indiquent sur quelle machine le job doit être exécuté et pendant combien de temps.

Le but est de calculer la somme des durées d'exécution pour chaque machine et d'identifier la machine qui a la durée totale la plus longue. Cette valeur est ensuite stockée dans une structure appelée `Instance`.

3 Réflexion et Analyse

Lors du développement de ce programme, j'ai dû réfléchir aux différents composants nécessaires pour résoudre efficacement le **Job Shop Scheduling Problem (JSSP)**. Voici les principales étapes de mon raisonnement :

1. **Compréhension du Problème** : J'ai d'abord analysé le format des données fournies dans le fichier texte, qui contient des informations sur les jobs et les machines, ainsi que les durées de chaque opération. Comprendre comment les jobs sont liés aux machines et aux durées était essentiel pour la structuration des données.

2. **Structuration des Données** : J'ai créé une structure `Instance` pour stocker les informations pertinentes, y compris le nombre de jobs et de machines, ainsi que des vecteurs pour les machines et les durées. Cela m'a permis d'organiser les données de manière à faciliter leur utilisation dans les différentes fonctions.

3. **Lecture des Données** : La fonction `lire_graphe` est conçue pour lire les données du fichier texte et remplir la structure `Instance`. J'ai utilisé des vecteurs dynamiques pour stocker les machines et les durées, ce qui m'a permis de gérer facilement les tailles variables de jobs et d'opérations.

4. **Génération du Vecteur de Bierwirth** : La fonction `V_Bierwirth` génère un vecteur représentant l'ordre des opérations. J'ai utilisé une technique de remplissage où chaque job apparaît plusieurs fois dans le vecteur, suivi d'un mélange aléatoire pour simuler un ordre d'exécution aléatoire.

5. **Planification et Évaluation** : La fonction `Evaluer` calcule les dates de début pour chaque opération en tenant compte de l'ordre défini par le vecteur de Bierwirth et des disponibilités des machines. J'ai pris soin de gérer les conflits d'accès aux machines pour garantir que les opérations ne se chevauchent pas.

4 Structure du programme

Le programme se compose des éléments suivants :

- Une structure `Instance` pour stocker les valeurs.
- Une fonction `lire_graphe()` qui lit le fichier texte et cumule les durées.
- Une fonction `V_Bierwirth()` qui génère un vecteur basé sur les machines et qui mélange ce vecteur aléatoirement.
- Une fonction `shuffle()` pour mélanger les éléments du vecteur.
- Une fonction `Evaluer()` qui calcule les dates de début des opérations et le plus long chemin.
- Une fonction `main()` qui appelle les fonctions ci-dessus et affiche les résultats.

5 Structure `Instance`

La structure `Instance` est définie comme suit :

```
struct Instance {
    int n_j;  // Nombre de jobs
    int n_m;  // Nombre de machines
    std::vector<std::vector<int>> machines;  // Machines associ es     chaq
    std::vector<std::vector<int>> durees;    // Dur es des op rations
    int opt;  // Dur e maximale cumul e sur une machine
};
```

Cette structure permet de stocker le nombre de jobs `n_j`, le nombre de machines `n_m`, et la somme des durées cumulées pour la machine la plus utilisée dans `opt`.

6 Fonction `lire_graphe`

La fonction `lire_graphe()` prend en entrée le nom du fichier et la structure `Instance` et calcule la machine avec la durée cumulée maximale. Voici la définition de cette fonction :

```
void lire_graphe(const string& nom_fichier, Instance& instance) {
    ifstream fichier(nom_fichier);  // Ouvrir le fichier en mode lecture
    if (!fichier) {
```

```
        cerr << "Erreur lors de l'ouverture du fichier." << endl;
        exit(1);  // Quitter le programme en cas d'erreur
    }

    // Lire le nombre de jobs et de machines
    fichier >> instance.n_j >> instance.n_m;

    // Initialiser un tableau pour cumuler les dur es de chaque machine
    vector<int> durees_machines(instance.n_m, 0);  // Tableau des dur es ,

    // Lire les donn es des jobs et cumuler les dur es pour chaque machir
    for (int i = 0; i < instance.n_j; ++i) {
        for (int j = 0; j < instance.n_m; ++j) {
            int machine, duree;
            fichier >> machine >> duree;  // Lire la machine et la dur e
            durees_machines[machine] += duree;  // Cumuler la dur e pour
            instance.machines[i][j] = machine;  // Machine pour le job i,
            instance.durees[i][j] = duree;      // Dur e pour le job i, o
        }
    }

    // Trouver la machine avec la dur e cumul e maximale
    instance.opt = *max_element(durees_machines.begin(), durees_machines.en

    fichier.close();  // Fermer le fichier apr s lecture
}
```

6.1 Explication de la fonction

La fonction `lire_graphe()` commence par ouvrir le fichier texte contenant les données des jobs et des machines. Elle lit d'abord le nombre de jobs et de machines, puis cumule les durées pour chaque machine. À la fin, la fonction identifie la machine avec la durée cumulée maximale et stocke cette valeur dans `instance.opt`.

7 Fonction `V_Bierwirth`

La fonction `V_Bierwirth()` génère un vecteur basé sur le nombre de machines et mélange les valeurs. Voici la définition de cette fonction :

```
int* V_Bierwirth(Instance& instance) {
    int taille_vecteur = instance.n_j * instance.n_m;  // Taille correcte

    // Allouer dynamiquement de la m moire pour le vecteur de Bierwirth
    int* VB = new int[taille_vecteur];
    if (!VB) {
        std::cout << "Probl me lors de la g n ration de notre vecteur de
        return nullptr;  // Utiliser nullptr au lieu de Null
    }
```

```
    // Remplir le vecteur avec les jobs : chaque job apparait 'instance.n_m
    for (int i = 0; i < taille_vecteur; i++) {
        VB[i] = i / instance.n_m;  // Remplir le vecteur avec les jobs
    }

    // Convertir le tableau dynamique en vecteur pour le m langer
    std::vector<int> vec(VB, VB + taille_vecteur);  // Convertir le tableau
    shuffle(vec);  // M langer le vecteur

    // Remplir    nouveau le tableau avec les valeurs m lang es
    for (int i = 0; i < vec.size(); ++i) {
        VB[i] = vec[i];
    }

    return VB;  // Retourne le tableau m lang
}
```

7.1 Explication de la fonction

La fonction `V_Bierwirth()` commence par calculer la taille du vecteur en multipliant le nombre de jobs par le nombre de machines. Elle utilise `new` pour allouer de la mémoire pour le tableau `VB`. Ensuite, elle remplit le tableau avec des valeurs basées sur l'indice modulo le nombre de machines.

Après cela, le tableau est converti en `std::vector` pour utiliser la fonction `shuffle` pour le mélanger.

8 Fonction `shuffle`

La fonction `shuffle()` est utilisée pour mélanger les éléments du vecteur :

```
void shuffle(std::vector<int>& arr) {
    srand(static_cast<unsigned int>(time(0)));  // Initialisation de la gra
    for (int i = arr.size() - 1; i > 0; --i) {
        int j = rand() % (i + 1);
        std::swap(arr[i], arr[j]);  //  changer  arr[i] avec arr[j]
    }
}
```

Cette fonction parcourt le vecteur de la fin vers le début et échange chaque élément avec un autre élément choisi aléatoirement. Cela garantit une permutation aléatoire des éléments.

9 Fonction `Evaluer`

La fonction `Evaluer()` calcule les dates de début des opérations et le plus long chemin :

```
void Evaluer(const string &mon_fichier, Instance& instance) {
    // Lire le graphe depuis le fichier
```

```
lire_graphe(mon_fichier, instance);

int taille_vecteur = instance.n_j * instance.n_m; // Nombre total d'op

// G n rer le vecteur de Bierwirth
int* VB = V_Bierwirth(instance);

// Stocker les dates de d but pour chaque op ration
std::vector<int> ES_ij(instance.n_j * instance.n_m, 0);  // Date de d

// Variable pour le plus long chemin
int plus_long_chemin = 0;

// Parcourir les jobs dans l'ordre du vecteur de Bierwirth
for (int i = 0; i < taille_vecteur; i++) {
    int job = VB[i];  // Job    traiter

    // Pour chaque op ration du job, on calcule la date de d but
    for (int op = 0; op < instance.n_m; op++) {
        int machine = instance.machines[job][op];
        int duree = instance.durees[job][op];

        // Trouver la date de d but la plus t t pour cette op ration
        int date_debut = 0;
        if (op > 0) {
            // Si ce n'est pas la premi re op ration du job, il faut
            date_debut = ES_ij[job * instance.n_m + op - 1] + instance.
        }

        // Mettre    jour la date de d but pour cette op ration
        ES_ij[job * instance.n_m + op] = date_debut;

        // Calculer la fin de cette op ration
        plus_long_chemin = std::max(plus_long_chemin, date_debut + dure
    }
}

// Afficher les r sultats
std::cout << "Dates de d but des op rations :" << std::endl;
for (int i = 0; i < taille_vecteur; i++) {
    std::cout << ES_ij[i] << " ";
}
std::cout << std::endl;

// Afficher le plus long chemin
std::cout << "Le plus long chemin est :" << plus_long_chemin << std::e

delete[] VB;
```

}

9.1 Explication de la fonction

La fonction `Evaluer()` commence par lire les données du fichier et remplit la structure `Instance`. Elle génère ensuite le vecteur de Bierwirth et initialise un vecteur pour stocker les dates de début des opérations.

Pour chaque opération, la fonction calcule la date de début en tenant compte de la disponibilité de la machine et des opérations précédentes. À la fin, elle affiche les dates de début et le plus long chemin.

10 Fonction `main`

La fonction `main()` appelle les autres fonctions et affiche les résultats :

```
int main() {
    Instance instance;  // Cr er une instance de la structure

    // Appeler la fonction lire_graphe pour remplir les informations
    lire_graphe("La01.txt", instance);

    // Afficher la valeur de opt (la dur e cumul e maximale)
    std::cout << "La plus longue dur e sur une machine est :" << instance

    // G n rer le vecteur de Bierwirth
    int* vecteur = V_Bierwirth(instance);

    // Afficher le vecteur g n r
    for (int i = 0; i < instance.n_j * instance.n_m; ++i) {  // Correction
        std::cout << vecteur[i] << " ";
    }
    std::cout << std::endl;

    // Tester la fonction Evaluer avec un fichier texte contenant les donn
    Evaluer("La01.txt", instance);

    // Lib rer la m moire
    delete[] vecteur;  // Utiliser delete[] pour lib rer la m moire allo
    return 0;
}
```

10.1 Explication de la fonction

La fonction `main()` crée une instance de la structure `Instance` et appelle la fonction `lire_graphe()` pour charger les données. Elle affiche ensuite la durée cumulée maximale sur une machine. Le vecteur de Bierwirth est généré et affiché, suivi de l'appel à la fonction `Evaluer()` pour calculer les dates de début et le plus long chemin.

11 Techniques Clés Utilisées

- **Structures de Données** : Utilisation de la structure `Instance` et des vecteurs dynamiques pour organiser et gérer les données.
- **Gestion de la Mémoire** : Allocation dynamique et libération de mémoire pour gérer les vecteurs et tableaux de manière efficace.
- **Lecture de Fichiers** : Utilisation de `ifstream` pour lire les données depuis un fichier texte.
- **Mélange de Vecteurs** : Application de l'algorithme de Fisher-Yates pour mélanger le vecteur de Bierwirth de manière aléatoire.
- **Calcul de Dates et Gestion des Conflits** : Méthodes pour calculer les dates de début des opérations en tenant compte des disponibilités des machines et des dépendances entre les opérations.

12 Conclusion

Ce programme permet de lire un fichier texte représentant des jobs et des machines et de calculer la somme des durées sur la machine la plus utilisée. Grâce à la fonction `lire_graphe()`, le programme remplit directement la structure `Instance` avec la durée maximale, ce qui permet d'obtenir facilement le résultat à partir de la fonction `main()`. La fonction `V_Bierwirth()` génère un vecteur mélangé de manière aléatoire qui peut être utilisé pour d'autres calculs dans le cadre du problème de Job-Shop.