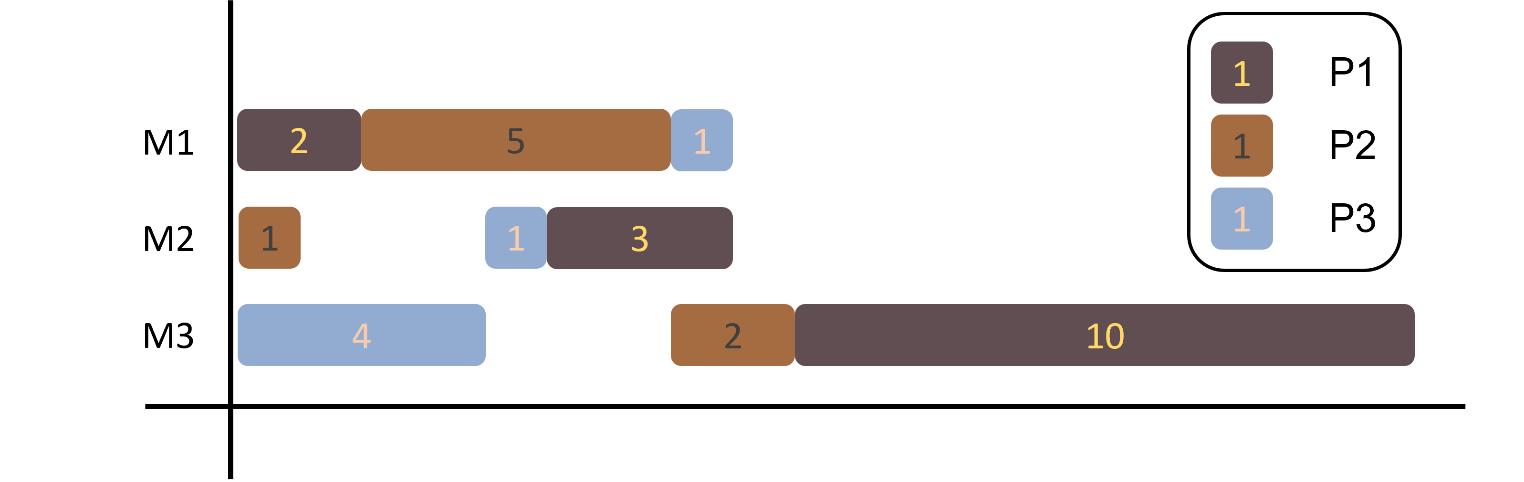
**图片包含 徽标

描述已自动生成**

**Outils d’Aide à la Décision**

**TP Numéro 2 : Job-Shop**



Ao XIE  
Chloé BERTHOLD

Responsable : Philippe LACOMME

Date : 02 nove. 2022

Campus des Cézeaux, 1 rue de la Chébarde, TSA 60125, 63178 Aubière CEDEX

Introduction

La question de l'ordonnancement de l'atelier est cruciale pour l'usine[1]. Le problème d’ordonnancement (*Job Shop Schedule Problem*, JSSP) est donc au cœur des systèmes d'atelier. Il s'agit d'un problème d'ordonnancement complexe typique présentant un degré élevé de complexité et de difficulté. Ce problème peut être décrit comme un ordonnancement rationnel de la séquence de traitement des tâches en fonction d'objectifs de production uniques ou multiples et des conditions environnementales de l'atelier, à condition que les contraintes soient satisfaites, et la détermination des heures de début et de fin de chaque processus en fonction de l'ordre obtenu, c'est-à-dire n pièces à traiter sur m machines[2]. Les contraintes à respecter lors de l'usinage de ces pièces sont les suivantes.

(1) Une pièce ne peut pas être usinée sur plus d'une machine en même temps.

(2) Une machine ne peut traiter qu'une seule pièce à la fois.

(3) Le processus d'usinage de la pièce doit satisfaire aux exigences du parcours de la pièce.

(4) Une fois qu'une pièce a été usinée dans une machine, le processus est ininterrompu.

Dans ce travail, nous avons écrit un algorithme en pipeline pour calculer le flux de travail le plus efficace en l'absence de temps de transport en utilisant le langage C++. Il y a huit fonctions incluses dans le programme, à savoir lire\_fichier(), bierwith(), vérifier\_vecteur(), evaluer(), recherche\_locale(), hashage(), permut() et grasp(). Dans ces fonctions, la fonction lire\_fichier() permet de lire une structure de graphe dans un fichier et de la stocker dans une instance en paramètre, la fonction bierwith() permet de générer un vecteur de Bierwith aléatoirement dans le champ adapte de la structure passée en paramètre, la fonction verifier\_vercteur() permet de vérifier que le vecteur de Bierwith de la solution passée en paramètre est bien construite et termine l’exécution si ce n’est pas le bonne situation, la fonction hashage() permet de calculer le hash de la solution passée en paramètre et la fonction permut() permet de réaliser une permutation simple du vecteur de Bierwith donne dans la structure de solution donnée en paramètre, le reste de la fonction sera expliqué dans l'article.

Table des matières

[I. Fonctions de développement - 4 -](#_Toc118486702)

[i. La Procédure evaluer() - 4 -](#_Toc118486703)

[ii. La Procédure recherche\_locale() - 5 -](#_Toc118486704)

[iii. La Procédure GRASP() - 6 -](#_Toc118486705)

[II. Etudes de cas - 7 -](#_Toc118486706)

[i. Description du problème - 7 -](#_Toc118486707)

[ii. Validation de l'algorithme - 8 -](#_Toc118486708)

[Conclusion - 9 -](#_Toc118486709)

[Références bibliographiques - 10 -](#_Toc118486710)

**Table des figures**

[Figure 1 La Procédure evaluer() - 4 -](#_Toc118482747)

[Figure 2 La Procédure recherche\_locale() - 5 -](#_Toc118482748)

[Figure 3 La Procédure GRASP() - 6 -](#_Toc118482749)

[Figure 4 L’Etude de cas - 7 -](#_Toc118482750)

[Figure 5 Résultats obtenus - 8 -](#_Toc118482751)

# Fonctions de développement

Dans cette section, nous expliquons et analysons la fonction d'évaluation du graphe, la fonction de génération du graphe à partir de la séquence, la fonction de recherche locale et la fonction GRASP pour l'algorithme JSSP conçu. La première sous-section décrit la fonction evaluer(), la deuxième sous-section décrit la fonction recherche\_locale (), et la troisième sous-section décrit la fonction GRASP().

## La Procédure evaluer()

La fonction evaluer() permet de calculer les dates de début de toutes les opérations et indique leur père dans les champs de la structure de solution passée en paramètre par rapport au graphe donne en paramètre et d’un vecteur de Bierwith détermine à l’avance, L'algorithme est présenté à la figure 1.

图形用户界面, 文本, 应用程序

描述已自动生成

Figure La Procédure evaluer()

## La Procédure recherche\_locale()

La fonction recherche\_locale() permet de réaliser une amélioration de type descente à partir d’une première solution calculée grâce au vecteur de Bierwith renseigne dans la structure de solution donné en paramètre. Cette fonction est utilisée pour assister la fonction GRASP pour la raison que dans la fonction GRASP il est nécessaire de trouver le nœud enfant le plus proche dans chaque nœud, ce que fait cette fonction, l'algorithme est présenté à la figure 2.

图片包含 表格

描述已自动生成

Figure La Procédure recherche\_locale()

## La Procédure GRASP()

*Greedy Random Adaptive Search Procedure* (GRASP) est une métathéorique, Il s'agit d'un algorithme d'optimisation classique et efficace. L'algorithme est divisé en deux parties, la première étant la phase de construction et la seconde la recherche locale[3]. Nous avons vu le deuxième partie dans la fonction recherche\_locale(). Cette sous-section est donc pour la première partie. Par conséquent, la partie principale de l'algorithme consiste à parcourir le graphe et à enregistrer la solution optimale pour chaque sommet, l'algorithme spécifique est présenté dans la figure 3.

文本

中度可信度描述已自动生成

Figure La Procédure GRASP()

# Etudes de cas

Dans cette section, nous résolvons un problème de traitement de dix objets dans cinq machines en utilisant des algorithmes déjà réalisés. Dans la première sous-section, le problème spécifique est décrit, tandis que dans la deuxième sous-section, la réponse de notre algorithme à la solution du problème est décrite.

## Description du problème

Dans cette sous-section, nous présentons un problème pratique d'ordonnancement de l'atelier dans lequel cinq machines sont utilisées pour produire dix pièces. Les valeurs exactes du problème sont présentées dans la figure 4. Chaque ligne représente une pièce, les valeurs dans les cercles indiquent le numéro de série de la machine dans la séquence d'usinage, et les valeurs sur les lignes horizontales après les cercles indiquent le temps nécessaire pour cette étape d'usinage.

图示

描述已自动生成

Figure L’Etude de cas

## Validation de l'algorithme

Dans cette sous-section, nous utilisons les différentes fonctions écrites pour obtenir le résultat présenté à la figure 5. Ce résultat est divisé verticalement en trois parties : le résultat le plus à gauche est le vecteur et le parent de chaque nœud obtenu à l'aide de l'algorithme de Bierwith, la réponse du milieu est le vecteur et le parent obtenus en effectuant une recherche locale basée sur les résultats de l'algorithme de Bierwith, et la plus à droite est la réponse finale basée sur les deux premiers résultats et obtenue à l'aide de l'algorithme GRASP.

背景图案

描述已自动生成电脑屏幕的照片

低可信度描述已自动生成图片包含 表格

描述已自动生成

Figure Résultats obtenus

# Conclusion

Pour le problème JSSP, nous avons utilisé avec succès l'algorithme de Bierwith ainsi que l'algorithme GRASP pour obtenir la solution optimale dans ce travail. Huit fonctions différentes ont été utilisées pour mettre en œuvre les deux algorithmes et les fonctions mentionnées.

# Références bibliographiques

[1] X. Shao et C. S. Kim, « An Adaptive Job Shop Scheduler Using Multilevel Convolutional Neural Network and Iterative Local Search », *IEEE Access*, vol. 10, p. 88079‑88092, 2022, doi: 10.1109/ACCESS.2022.3188765.

[2] Y. Yu, « A Research Review on Job Shop Scheduling Problem », *E3S Web Conf.*, vol. 253, p. 02024, 2021, doi: 10.1051/e3sconf/202125302024.

[3] T. A. Feo et M. G. C. Resende, « A probabilistic heuristic for a computationally difficult set covering problem », *Oper. Res. Lett.*, vol. 8, no 2, p. 67‑71, avr. 1989, doi: 10.1016/0167-6377(89)90002-3.