Ordonnancement atelier sériel avec réglage

Introduction

Dans ce rapport je vais expliquer le problème d'ordonnancement de tâches dans un atelier sériel avec réglage ainsi que 4 métaheuristiques d'optimisation : algorithme aléatoire, de descente, génétique et recuit simulé.

Explication du problème

Tesla est un constructeur automobile révolutionnant le marché de l'automobile, grâces à ses voitures autonomes électriques. L'une des prouesses de l'entreprise californienne consiste en la fabrication de deux modèles différents (Tesla Model S, Model X) dans une seule et même usine. Pourtant les deux voitures sont de catégories différentes, berline pour l'un et VUS pour l'autre.

Les outils de productions sont néanmoins les mêmes.

Dans le processus de production, les machines constituant l'unité de production ont besoins de réglages pour passer d'un modèle à l'autre. Le problème d'ordonnancement atelier sériel modélise ce cas de figure. Il y a ici plus de tâches différentes que de machines.

Toutes nos instances comprennent 20 tâches (*n*) et 10 ou 5 machines (*m*). Chaque tâche contient le temps d'exécution qu'elle nécessite sur chaque machine. Les machines contiennent une matrice de taille $n \times n$ indiquant le temps de réglage nécessaires pour passer d'une tâche à l'autre. Toutes les tâches doivent passer de la machine 0 à la m dans l'ordre.

Objectif

Dans ce projet, l'objectif et de trouver un ordonnancement des tâches permettant minimiser le temps d'exécution de la commande. L'élément sur lequel les algorithmes vont travailler est une liste de tâche, indiqué dans l'ordre de passage.

Plan du document

Ce rapport va d'abord énoncer les informations pertinentes issus d'une recherche bibliographique scientifique. S'en suit un énoncé des différents choix techniques dans la construction des algorithmes, le choix des instances puis les résultats et analyse.

Revue littéraire

Après lecture d'articles scientifique (de Ruben Ruiz, Colin R.Reeves et Runwei Cheng) en sont sortis différents principes pour construire une heuristique d'optimisation performante:

- Génération NEH : Ruben Ruiz présente cette méthode de construction d'une solution initiale simple et efficace. Elle consiste à estimer le temps total d'exécution d'une tâche (sans prise en compte du temps de réglage sur les machines) et de les mettre par ordre décroissant de durée d'exécution.
- Runwei Cheng présente différentes manières de représenter ce problème, avec la présentation par frise chronologique des machines permettant une modélisation intuitive du problème.
- SB2OX : Cette méthode de croisement propose de garder les phénotypes similaires dans les deux chromosomes parents.

Choix des instances de Test

Parmi les 20 instances données dans le sujet, j'ai choisi de construire mes heuristiques sur 4 instances. J'en ai choisies deux de tailles 20 travaux 5 machines, les deux autres de 20 travaux et 10 machines. Le choix s'est dirigé vers celles ayant un optimal aux extrêmes. Les instances sont :

- SDST10_Tai012 – 10 machines / 20 travaux (optimal 1751)
- SDST10_Tai014 – 10 machines / 20 travaux (optimal 1465)
- SDST10_Tai010 – 5 machines / 20 travaux (optimal 1178)
- SDST10_Tai002 – 5 machines / 20 travaux (optimal 1401)

Métaheuristiques et description

Toutes les méthodes d'optimisations ont été conçues en C++, sous Visual studio 2015. Les méthodes de lecture du problème, et évaluation des solutions sont les mêmes. Après plusieurs tests, la méthode de construction utilisée prend un temps considérable pour construire la solution.

Cette étape où l'algorithme passe les tâches dans les machines les unes après les autres peut certainement être améliorée, avec de la programmation parallèle pour la consultation des temps de réglage sur les machines.

Algorithme aléatoire

Cet algorithme est le plus simple à comprendre. Il consiste à choisir aléatoirement une tâche et la permuter avec un autre. Il a été exécuté avec 1000 évaluations.

Descente

Cette méthode décrite par un enseignant de la RMIT, université de Melbourne. Ici, la solution initiale est générée avec la NEH. Ensuite, nous permutons les premières tâches jusqu'à les placer dans une position réduisant le temps de production pour recommencer par la tâche 0. La méthode est évaluée 500 fois, l'implémentation faite ici tombe rapidement dans un optima local.

Recuit Simulé

Le Recuit Simulé utilise le parcours de voisinage de l'algorithme de descente. Il apporte une amélioration, le parcours de voisinage précédente est efficace en début d'exécution mais montre vite ses limites. Les paramètres sont les suivants :

- Température : 1000. La fonction objectif initiale dépasse rarement les 2000. La température est proportionnelle à cet unité

- Alpha : 0.9 comme recommandé dans le cours

- Palier : 10. Après observation de l'algorithme de descente, 10 permutations représentent la plage optimale pour le palier.

Algorithme Génétique

Cet algorithme est le plus fascinant à concevoir et mettre en place. Voici ses caractéristiques :

- Génération de la population initiale : Combinaison de NEH et descente, on choisit ici les 5 premières tâches pour permutation.
- Selection : La sélection des parents est un mélange de sélection par roulette et aléatoire. La population initiale est généralement très homogène. Une sélection par roulette génère une duplication de l'individu allant vers une convergence néfaste pour le fonctionnement de l'algorithme.
  La sélection est aléatoire pour les premières générations. Ensuite, un choix par roulette wheel est effectuée. Le basculement se fait lorsque le nombre de génération produit est égal à la taille de la population
- Croisement : Une méthode de croisement Ox (avec une séparation à 33% et 66%) est effectuée. Ce choix se justifie par des résultats fructueux lors d'un précédent laboratoire

Résultats et Analyse

Heuristique	AG	Aléatoire	Descente	Recuit Simule
Deviation moyenne par rapport à l'optimum	12,59	12,09	21,11	18,54
Ecart type	2,24	3,55	3,79	1,19
Deviation maximum par rapport à l'optimum	15,49	14,20	24,64	20,06
Deviation minimale par rapport à l'optimum	10,56	6,78	17,06	17,54
Nb de fois que l'optimum est atteint	0	0	0	0
Nb de fois que l'heuristique obtiens la meilleure solution	2	2	0	0
Nb de fois que l'heuristique obtiens la pire solution	0	0	4	0

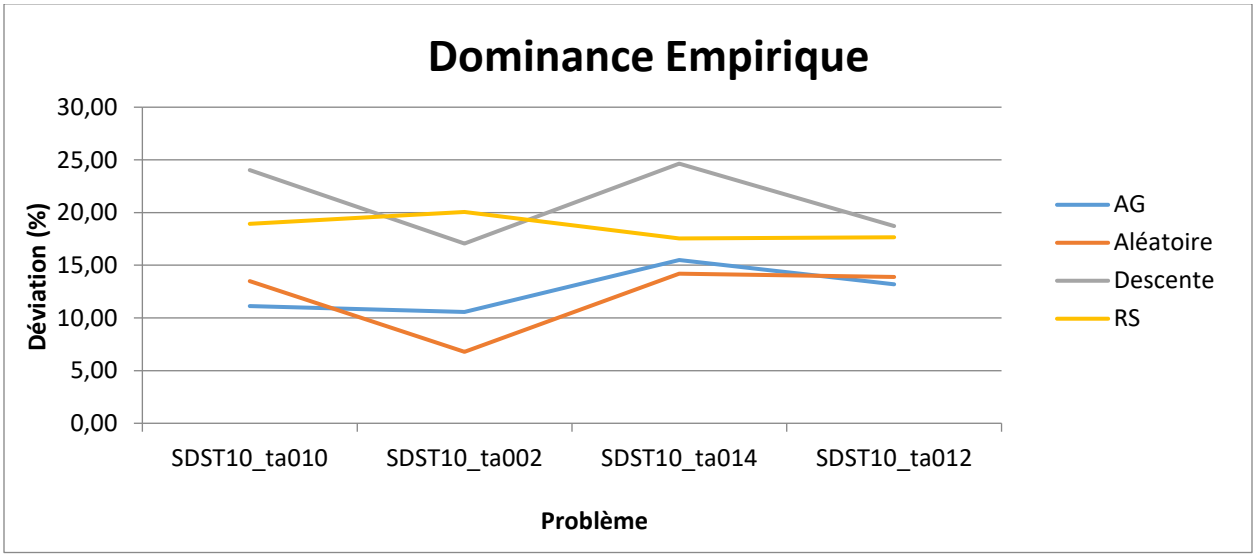
Tableau 1 : Résumé des résultats obtenus

Sur le tableau ci-dessus nous observons un surclassement de l'algorithme génétique et aléatoire face à l'algorithme de descente et de recuit simulé.

Cette remarque s'explique par le fait que la NEH est une heuristique localisée. Au vu de la taille du problème et de l'évolution des résultats au cours de l'execution, il apparaît que l'AG est certainement plus efficace sur les instances de plus grande taille.

L'aléatoire donne ici un résultat aussi bon que l'AG. Nous pourrions asseoir nos observations en effectuant des tests sur des instances de taille plus grande.

Le graphique de la dominance empirique montre que les heuristiques restent malgré tout performantes, avec une déviation maximale de 25% pour l'algorithme de descente. L'execution des méthodes n'ayant pas été chronométrée, il n'y as pas de données quantifiable à analyser. Néanmoins, l'algorithme génétique est le plus rapide et performant, avec plus de 1000 instances testées en environ 15 minutes.



Conclusion

Dans ce travail, il est rendu compte d'une méthodologie de compréhension, synthétisation et mise en place d'une métaheuristique pour un problème donné. Il est intéressant de voir à quel point il est important d'assimiler des notions issues des recherches scientifiques avant de mettre au point un quelconque outil d'optimisation. Dans la production rendue avec ce travail, les défis ont été de modéliser le problème d'une manière non seulement intuitive mais surtout performante, afin de tirer la pleine puissance des outils informatiques.

Pour conclure, ces travaux démontrent avec clarté un surclassement des Algorithmes génétiques sur les autres méthodes, aussi bien sur la rapidité d'exécution que sur les résultats obtenus. Le fait que la solution aléatoire soit également performante s'apparente plus à un hasard des instances qu'as un résultat méthodique et générique.

Références

R. Ruiz, C. Maroto, J. Alcatraz *Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics (2005)* European Journal of Operational Research 165, 34-54

R. Ruiz, C. Maroto, J. Alcatraz *Two new robust genetic algorithms for the flowshop scheduling problem (2005)*

R. Cheng, M. Gen, Y Tsujimura *A tutorial survey of job-shop scheduling problems using genetic algorithms – I (1996)*