# การแก้ปัญหาการจัดตารางการผลิตในระบบไหลเลื่อนโดยใช้เวลาในการดำเนินงานน้อยที่สุดด้วยอัลกอริทึมการบรรจวบ

นางสาวอรรัมภา ศรีมงคลกุล



วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2557

# MINIMIZING MAKESPAN USING NODE-BASED COINCIDENCE ALGORITHM IN THE PERMUTATION FLOWSHOP SCHEDULING PROBLEM

Miss Ornrumpha Srimongkolkul


A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Engineering Program in Computer Engineering

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2014

| | |
|---|---|
| Thesis Title | MINIMIZING MAKESPAN USING NODE-BASED COINCIDENCE ALGORITHM IN THE PERMUTATION FLOWSHOP SCHEDULING PROBLEM |
| By | Miss Ornrumpha Srimongkolkul |
| Field of Study | Computer Engineering |
| Thesis Advisor | Professor Dr. Prabhas Chongstitvatana |

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree

.....Dean of the Faculty of Engineering

(Professor Dr. Bundhit Eua-arporn)

THESIS COMMITTEE

.....Chairman

(Assistant Professor Dr. Sukree Sinthupinyo)

.....Thesis Advisor

(Professor Dr. Prabhas Chongstitvatana)

.....External Examiner

(Dr. Chalermsub Sangkavichitr)

อรรัมภา ศรีมงคลกุล : การแก้ปัญหาการจัดตารางการผลิตในระบบไหลเลื่อนโดยใช้เวลาในการดำเนินงานน้อยที่สุดด้วยอัลกอริทึมการบรรจบ (MINIMIZING MAKESPAN USING NODE-BASED COINCIDENCE ALGORITHM IN THE PERMUTATION FLOWSHOP SCHEDULING PROBLEM) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: ศ. ดร. ประภาส จงสถิตย์วัฒนา, หน้า.

การจัดตารางการผลิตเป็นสิ่งที่สำคัญซึ่งส่งผลกระทบโดยตรงกับประสิทธิภาพในการดำเนินงานโดยรวมของโรงงานอุตสาหกรรม การจัดตารางการผลิตที่มีประสิทธิภาพสามารถลดเวลาในการผลิตซึ่งส่งผลให้โรงงานอุตสาหกรรมสามารถลดต้นทุนการผลิตและและจัดการทรัพยากรได้ดียิ่งขึ้น

ปัญหาการจัดตารางการผลิตในระบบไหลเลื่อนได้รับความสนใจจากภาคธุรกิจและอุตสาหกรรมมานานกว่าครึ่งศตวรรษ นักวิจัยได้ออกแบบอัลกอริทึมที่ซับซ้อนมากมายเพื่อช่วยแก้ปัญหาการจัดตารางการผลิตในระบบไหลเลื่อน และแสดงผลลัพธ์ที่มีประสิทธิภาพ อย่างไรก็ตามการออกแบบอัลกอริทึมในงานวิจัยจำเป็นต้องคำนึงถึงความซับซ้อนของอัลกอริทึมและเวลาที่ใช้ในการคำนวณเพื่อให้สามารถนำไปใช้ได้จริงในโรงงานอุตสาหกรรม

วิทยานิพนธ์ฉบับนี้นำเสนออัลกอริทึมการบรรจบเพื่อใช้ในการแก้ปัญหาการจัดตารางการผลิตโดยมีวัตถุประสงค์ในการใช้เวลาในการผลิตน้อยที่สุด ผลจากการทดลองอัลกอริทึมการบรรจบได้รับการพิสูจน์ว่าเป็นอัลกอริทึมที่มีประสิทธิภาพในการหาคำตอบที่ดีโดยใช้ทรัพยากรและระยะเวลาในการคำนวณต่ำ ในการแก้ไขปัญหาการจัดตารางการผลิตนี้ อัลกอริทึมการบรรจบได้เจอคำตอบที่ดีที่สุดเป็นจำนวน 10%ของคำตอบทั้งหมด และค่าเฉลี่ยของคำตอบห่างจากค่าที่ดีที่สุดคิดเป็น 0.98% นอกเหนือจากนั้น เวลาที่ใช้ในการหาคำตอบของอัลกอริทึมการบรรจบยังมีค่าน้อยกว่าเวลาที่ใช้ในการหาคำตอบของอัลกอริทึมอื่นๆที่นำมาใช้เปรียบเทียบในวิทยานิพนธ์



ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อ อ.ที่ปรึกษาหลัก .....

ปีการศึกษา 2557

## 5670462721 : MAJOR COMPUTER ENGINEERING

KEYWORDS: PERMUTATION FLOWSHOP / MAKESPAN / NODE-BASED COINCIDENCE ALGORITHM / PRODUCTION SCHEDULING

ORNRUMPHA SRIMONGKOLKUL: MINIMIZING MAKESPAN USING NODE-BASED COINCIDENCE ALGORITHM IN THE PERMUTATION FLOWSHOP SCHEDULING PROBLEM. ADVISOR: PROF. DR. PRABHAS CHONGSTITVATANA, pp.

Scheduling problem has always been an important problem in the industrial sectors since it creates huge impact on the overall performance of the manufacturing. Good scheduling can reduce overall production time which then leads to lower cost and good resource management.

The permutation flowshop scheduling (PFSP) is the classic scheduling problems that attracts both business and research area for almost half a century. On research side, a variety of complex algorithms have been introduced to solve the problems and provide high quality of solutions. Nevertheless, these algorithms will be useless if they fail to implement in practice where computational time and complexity of algorithm become an important issue of concern.

This research proposes a Node-Based Coincidence Algorithm (NB-COIN) for the permutation flowshop scheduling problems (PFSP) aimed at Makespan minimization. NB-COIN is proved to be an effective algorithm that can provide good quality solutions using small amount of time and resources. The results generated by NB-COIN are also better than other well-known algorithms in consideration. Based on the bench-mark data sets of Taillard, 10% of the solutions provided by the presented algorithm are optimal solutions. Moreover, the solutions found by NB-COIN are also achieve 0.96% gap from upper bound in average. More importantly, those solutions are found within a short time.



Department: Computer Engineering Student's Signature .....

Field of Study: Computer Engineering Advisor's Signature .....

Academic Year: 2014

# ACKNOWLEDGEMENTS

This research is one of the most challenging works I have ever accomplished in my life. It would never have been completed successfully without all supports, guidance, and kindness from the following people.

I would like to express my deepest gratitude to my supervisor, Professor Dr. Prabhas Chongstitvatana for enlighten me to the research area, and get my research start on the right direction from the beginning, provide me a valuable advice and open to all of my questions until I successfully finish my research. I would not have got through the difficult moment without his support.

I would like to give my sincere thanks to Dr. Warin Wattanapornprom for providing guidance about Node-based Coincidence Algorithm which is used on this research. I also thank him for motivating me to deliver this research and keep pushing me to finish the paper.

I would like to thank all thesis committees for being available on my thesis proposal date and thesis defending date at a short notice.

I would like to thanks all members of ISL (Intelligent System Laboratory) for being energetic to give me suggestions and answers me a questions about the research.

Finally, I would like to thank my family and friends for being supportive and encouraging me in both good and bad time.

CONTENTS

Page

Page

Page

# *TABLE OF FIGURE*

# *TABLE OF TABLE*

# *Chapter 1*

# *Introduction*

## 1.1 Background

In manufacturing sector, Scheduling has always been one of the important decision-making process that create high impact in term of effectiveness and efficiency of production process. Poor scheduling can lead to low productivity, higher cost and longer time consuming.

Flow shop is a production line that is widely used in many industry such as chemical, paper, circuit and glass industry. In the flow shop, the jobs process on a set of machines in the same order. In addition, each machine is either idle or occupied by only one job at the time with no pre-emption and interruption. The permutation of jobs for every machine can be scheduled in $(n!)^m$ solutions where n is the number of job and m is the number of machine. However, in the permutation flowshop scheduling problem(PFSP), passing any jobs is not allowed to simplify and reduce the solution's possibility into (n!). The objective of PFSP is divided into two main criteria; makespan and flowtime. The makespan is the completion time of the last jobs while flowtime is the summation of completion time of each job. The makespan criterion is well-known to lead to rapid turn-around of jobs, uniform utilization of resources and minimization of work-in-process inventory.

The permutation flow shop scheduling problem (PFSP) has become an interesting research topic for many researchers since Johnson[1] introduced in 1950s. Later, the complexity of PFSP is proved to be a NP-hard (Garey et al [2] and Lenstra et al [3]). Many heuristic optimization methods have been developed to achieve high quality solutions in a reasonable computational time such as Nawaz et al.[4], Palmer [5], Campbell et al. [6], Dannenbring [7], Taillard [8], Framinan et al. [9] and Framinan and Leisten [10]. However, those heuristics are not only failed to achieve the optimum solutions but also consume a lot of CPU time. Even the results given by the most

powerful heuristics, NEH, proposed by Nawaz et al. [4] are still far at almost 7% from the optimal value. When it comes to a conclusion that only heuristics may not capable enough to overcome the optimum solution for the PFSP, many researchers developed more complex methods, metaheuristics, such as tabu search [11-16], genetic algorithms (GAs) [15, 16], ant colony optimization [17-19], particle swarm optimization[20], iterated local search (ILS) [21]or the Estimation of Distribution Algorithm (EDA) [22]. Although these methods provide better results, a long computational time or a lot of resources is required. Later, the complexity of the algorithms is even enhanced by integrated two or more metaheuristics called the hybrid metaheuristics. This technique was used by G.I. Zobolas [23] and H. Liu [24] to achieve optimal solution with high computational speed.

However, at the end of the day, all industrial world need is neither an optimum solution nor a complex method, but is a simple algorithm that can provide a reasonable solution in a short period of time. It is worth noting to implement such a complex algorithms that people in the field are hardly replicate or implement in the real world situation. The proposed method Node-Base Coincidence Algorithm(NB-COIN), is easy to implement and can provide high quality solution.

## 1.2 Research Purpose and Objectives

The objective of this research is to find the optimum solution in the makespan criteria of the Flowshop scheduling problem by Node Based Coincidence Algorithm (NB-COIN).

## 1.3 Research Scope

1. The proposed method is able to find the work's sequence in the permutation flowshop scheduling problem.
2. The work's sequence generated from Node Based Coincidence Algorithm is makespan minimization.
3. The CPU time is provided and use as the termination of the method.

## 1.4 Research Limitation

1. All jobs are independent and ready to enter to the machines.
2. The processing time of all jobs are provided upfront.
3. Every machine is ready to work and multitasking is prohibited.
4. The job cannot be divided.
5. The processing time of job on each machines cannot be changed during the production period
6. There is unlimited buffer between the machines.

## 1.5 Contribution

This research involved the development of an algorithm for the permutation flowshop scheduling problem. This improvement to warehousing performance would be beneficial for both manufacturing industry and academic research.

On the one side, the manufacturing can improve their flowshop scheduling by using this algorithm since it provides good quality of result using only small amount of resource and time. Such improvements could reduce costs and lead time of overall operations, and increase customer service levels by providing faster processing time which then lead to faster delivery. Therefore, the overall performance of manufacturing could be increased significantly.

On the other side, it can enlighten researchers to focus on practical algorithm that easy to use in real life where the quality, complexity and computational time are balances.

## 1.6 Research Structure

This research consists of six chapters.

**Chapter 1: Introduction**

This chapter provides the background of the research area and the importance of the problem. The research purpose, research objectives, research scope and limitations is identified.

**Chapter 2: The permutation Flowshop Scheduling Problem**

The research problem, the permutation flow shop scheduling, is determined in detail in Chapter 2 including, the models formulated and well-known objectives to the problem.

**Chapter 3: Well-known approaches for the permutation flowshop scheduling**

In this chapter, the well-known algorithms to solve the permutation flow shop scheduling are elaborated. This chapter includes well-known heuristic, metaheuristic and hybrid heuristic that has been applied to the problem.

**Chapter 4: Node-based Coincidence Algorithm**

The algorithm to solve research problem is discussed in this chapter, covering the characteristic of an algorithm, general procedure and example of an algorithm in optimization problem



**Chapter 5: Computational Experiment and Results**

The computational experiment and results are presented in this chapter. The design of experiments such as test instance and user define parameter are determined. The proposed algorithm is tested against well-known approaches for the research problem and will be analysed in two criteria; CPU time and performance.

**Chapter 6: Conclusion and Future work**

A summary of the research is presented in this chapter, together with limitations of research and suggestions for future work

# *Chapter 2*

# *The Permutation Flowshop Scheduling Problem*

The previous chapter set a background of the research. The purpose of the research, objectives of the research, research scope and limitation were identified. The need to implement an algorithm to solve the permutation flowshop scheduling problem was also highlighted.

In this chapter, the overview of flow shop and the permutation flowshop scheduling problem will be illustrated in order to provide a solid understanding of the important of research problem. Moreover, the model formulated and the well-known objectives for the problem will be determined. Finally, the complexity of the problem will be discussed.

## 2.1 Flow shop

Flow shop is a type of production line that has been widely used in many manufacturing and assembly facilities since it can produce variety of products from one set of machines. In the flow shop, each job processes on a set of machines in the same order so all jobs have to follow the same route. The processing time of each job for each machine is different. In addition, the machines are assumed to be set up in series and either idle or occupied by only one job at the time with no pre-emption and interruption.

In the general flow shop, it considers the operation of n job on m machine. It assumes that there is unlimited buffer all storage between the machines so all the job that has been completed in the upstream machine can be waited at the buffer which will not cause the delay or blocking. The manufacturing usually uses general flow shop when the size of products that need to process in the machines are physically small such as printed circuit boards or integrated circuits, it doesn't require large space or buffer between the machines which is also very easy to be stored at large quantity.

However, there are other kind of flow shop such as flow shop with blocking or zero-buffer flow shop, no wait flow shop and hybrid flow shop.

*2.1.1 Flow shop with zero buffer or blocking flow shop*

The Blocking issue happens when the buffer is full so the job that has been completed at the upstream machine cannot be released to the buffer. It needs to remain at the upstream machine until the buffer is free which then prevent the next job from beginning its processing. Flow shop with zero buffer is often used with the products that are physically large such as television set, car or copier require large buffer between two machines.

There are many approaches for blocking flow shop scheduling problem including both heuristics and metaheuristics. In heuristics, the constructive Greedy and NEH heuristic have been proved to be an effective heuristic to solve a problem[4]. These heuristic can rapidly yield feasible solution but it needs to trade off with the quality of the solution on large scale problem. On the other hand, Metaheuristics have proposed to increase the quality such as Genetic algorithm[25] and particle swarm optimization[26].

*2.1.2 No Wait flow shop (NWFSP)*

No-wait flow shop is one of flow shop production line where all jobs need to process on the machine continuously until completion at the last machine without interruption. Therefore, the job on the first machine sometime needs to be delayed to ensure that the whole production process meet no-wait restriction[27].

There are several No wait flow ship application in practical such as chemical processing, Steel production and plastic molding. Moreover, No wait flow shop is also adapted in the manufacturing to achieve Just In time operation system and robot cells to avoid waiting time in the production process[28].

The parameter for no wait flow shop is similar to general flow shop but it require some restriction where the starting time of job $j_i$ on machine $m_i$ need to be equal to the completion time of job $j_i$ on the upstream machine $m_{i-1}$ for each *i* and *j*.

No wait flow shop has been interested by many researcher and several metaheuristics has been proposed to achieve high quality of solution within the reasonable time. Genetic Algorithm was proposed by Aldowaisan and Allahverdi in 2003 [29]. Furthermore, other metaheuristics were also designed and implemented such as estimation of distribution algorithm (EDA) [30], Particle Swarm Optimization (DPSO) [31] and ant colony optimization (ACO)[32].

*2.1.3 Hybrid flow shop*

Hybrid flow shop is more complex than other flow shop since it involves parallel operation. In Hybrid flow shop, there are a number of stages in series with one or more number of identical machine in parallel at each stage. All jobs have to operate in the same route through the stage. Therefore, it needs to be focus on not only a permutation of jobs for each machine but also the assignment of job and the sequence of job on each machine.

Hybrid flow shop is used in several industrial sector such as glass, paper, steel and fabric industries[33].

This kind of flow shop has attracted several researchers. An example of solution approaches of hybrid flow shop are NEH heuristic[4], Tabu search algorithm [34], ant colony optimization[35] and quantum-inspired immune algorithm (QIA)[36].

## 2.2 The permutation Flowshop Scheduling Problem

The permutation flowshop scheduling Problem (PFSP) was presented by Johnson's seminal paper[1]. Since then, it has always been interested by many researchers and the number of literatures published to solve this problem has been increasing rapidly. Day and Hottenstein has reviewed PFSP in 1970[37]. Then, Dudek investigated the problem highlighting the solving problem strategy and diverse optimization criterions. Reisman et al [38] also provided a statistic review while comprehensive review and evaluation of permutation flow shop heuristic was done by Ruiz and Maroto[9, 39].

The permutation flowshop is a simple multistage scheduling problem. The model of flowshop include a set $J$ of $n$ jobs, $J=\{j_1,\ldots j_n\}$, and set $K$ of $m$ machines, $K=\{k_1,\ldots,k_m\}$. All

jobs have to visit to all machines in the same route. The flowshop process is determines in Figure 2.1.

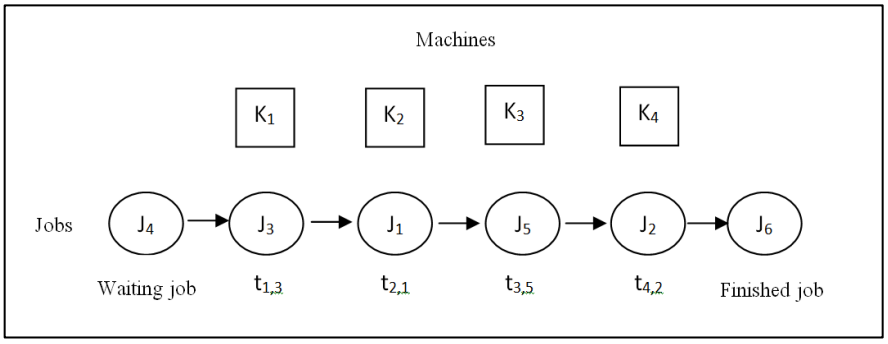


Figure 2.1 The production process of job sequence 6-2-5-1-3-4 entered to 4 machines in the PFSP

*2.2.1 Model Formulation*

n= number of jobs to be process

m = number of machines in the production line

J = a set J of n jobs where $J=\{j_1,\ldots, j_n\}$

$K$= *a set* of $m$ machines where $K=\{k_1,\ldots, k_m\}$

$t_{k,j}$ = the processing times of job $J$ on machine $K$ and

$C(k,j)$ = the completion time of job $J$ on machine $K$.

Thus, the completion time, $C(k,j)$ can be calculated as follows:

$$C(1,1) = t_{1,1} \tag{2.1}$$

$$C(1,j) = C(1,j-1)+t_{1,j},\ j=2,\ldots,n, \tag{2.2}$$

$$C(k,1) = C(k-1,1) + t_{k,1},\ k = 2,\ldots,m, \tag{2.3}$$

$$C(k,j) = max\{C(k,j-1),C(k-1,j)\}+ t_{k,j}, \tag{2.4}$$

*2.2.2 Well-known objectives for the permutation flowshop scheduling problem*

The common objective of PFSP is divided into two main criteria; makespan, total flowtime objectives. The makespan is the completion time of the last jobs while flowtime is the summation of completion time of each job.

*Makespan minimization*

The makespan is the classic objective function that has always attracted many researchers. It is the finished time of the last job in the schedule. The performance of solutions is defined as the duration between the starting time of first job in the first machines and the finished time of last job in the last machine. The makespan criterion is well-known to lead to rapid turn-around of jobs, uniform utilization of resources and minimization of work-in-process inventory.

The makespan minimization is described as n/m/P/$C_{max}$. It consists of a set J of n jobs, $J=\{ j_1,\ldots, j_n\}$ and set *K* of *m* machines, $K=\{ k_1 ,\ldots, k_m\}$ Let $t_{k,j}$ denotes as the processing times of job *J* on machine *K* and *C(k,j)* be the completion time of job *J* on machine *K*.

Therefore, the makespan is denoted as

$$\text{Cmax} = \text{C(Km,Jn)} \tag{2.5}$$

*Total Flowtime minimization*

Total flowtime minimization focus on the total completion time. It has been an increasing objective functions for many researchers since minimizing total flow time can lead to the reduction in Work In Process (WIP) or in-process inventory and increase stability of machine utilization.

The total flowtime is defined n/m/P/Cj To calculate total flowtime the completion time (makespan) of each job need to be calculated. To calculate the total flowtime, the equation is defined as follow:

$$\text{TFT}=\sum_{k=1}^{n} C_{[k][m]} \tag{2.6}$$

$C_{[k][m]}$ denotes the completion time of job k on the last machine. The completion time of job k, $k \in \{1,2,\ldots,n\}$ on the machine i, $i \in \{1,2,\ldots,m\}$ can be denotes as $C_{[k]I}$ where $C_{[k]\,0}$ and $C_{[0]\,i} = 0$.The processing time of job j on machine i is denote to $t_{[k]\,j}$. Therefore, $C_{[k]j}$ is computed as follows:

$$C_{[k]j} = max\{ C_{[k]j-1}, C_{[k-1]\,j}\} + t_{[k]\,j} \tag{2.7}$$

*2.2.3 An Example of the permutation flowshop Scheduling Problem*

In the automotive parts manufacturing, there is 6 jobs and 4 machines. Each job requires different processing time on the machines as input time in Table 1. The best solution in term of makespan is obtained from the job's sequence 6-2-5-1-3-4 with a minimum makespan at 322 seconds.

| Job | Processing time ($t_{k,j}$) | | | | |
|---|---|---|---|---|---|
| | $K_1$ | $K_2$ | $K_3$ | $K_4$ | Total processing time (s) |
| $J_1$ | 25 | 45 | 52 | 40 | 162 |
| $J_2$ | 7 | 41 | 22 | 66 | 136 |
| $J_3$ | 41 | 55 | 33 | 21 | 150 |
| $J_4$ | 74 | 12 | 24 | 48 | 158 |
| $J_5$ | 7 | 15 | 72 | 52 | 146 |
| $J_6$ | 12 | 14 | 22 | 32 | 80 |

Table 2.1 The Processing Time Of Jobs on the Machines

*2.2.4 Complexity of Flowshop Scheduling Problem*

In a small problem where the machine number is less than 3, flow shop can be solved optimally. Since 1954, Johnson has proposed Johnson's algorithm to solve flowshop scheduling problem in 2 and 3 machines which can guarantee optimal[1]. However, many researchers has been proved that flowshop scheduling problem is np-hard problem when the number of machine is higher than 2 machines where it is extremely difficult to find the optimal result for all variables within a tolerable computation time since the run-time grows exponentially when the number of variables is increased, as when adding more machines.

For the flowshop scheduling problem (FSSP), the job can either process in all machine or pass another when the machine is busy and there is a queue so it may not

operate as first come first serve or no always need to follow the job sequence. In this case, the permutation of jobs for every machine can be scheduled in $(n!)^m$ solutions where *n* is the number of job and m is the number of machine. Hence, the maximum solutions can be extremely large even in a small instance. For example, in 10x10 instance, the maximum number of possible solutions is $(10!)^{10} = 3.96 \times 10^{65}$.

On the other hand, in the permutation flowshop scheduling problem(PFSP), passing any jobs is not allow to simplify and reduce the solution's possibility into (n!). However, Garey et all [2] was proved that the permutation flowshop scheduling is also be strongly np-hard when the number of machine exceed 3 machines.

Therefore, it is very important to limit search space to only best solutions by identifying the characteristics that optimal solution possess and focus only the sequence that contain those characteristics.

## 2.3 Conclusion

This chapter provides a solid understanding of flow shop and the permutation flow shop scheduling problem. Although there are several kind of flow shop, This research focuses on the classic flow shop which is permutation flowshop scheduling problem in makespan minimization.



It is clear that the permutation flowshop scheduling has interested many researchers and many literate has reviewed the methods to solve the problem. Since the PFSP is np-hard, the methods need to consider both computational time and quality of solutions.

In the next chapter, the well-known approaches for the permutation flowshop scheduling problem with makespan minimization will be illustrated. The stage of art of each method and its advantage and disadvantage will be discussed.

# *Chapter 3*

# *Solution Approaches for the Permutation Flowshop Scheduling Problem*

In this chapter, some well-known methods to solve PFSP will be described. The methods are as follows:

1. Two heuristic approaches: NEH and Greedy heuristics,
2. Two metaheuristic approaches: Any Colony System and Coincidence Algorithm and Hybrid Metaheuristic algorithm.

In addition, the strength and weakness of these methods will be also discussed in detail.

## 3.1 Heuristics

Heuristic is a category of problem solving approaches which are not guaranteed globally optimal solutions. However, these approaches were created to satisfy some acceptable goals.

### *3.1.1 The NEH Heuristic*

NEH algorithm [4] is a heuristic approach to solve PFSP proposed by Nawaz, Enscore and Ham in 1983. The objective of this algorithm is to minimize the makespan of the problem by iteratively taking new jobs into consideration. The main idea of this heuristic is that the high priority should be given to the job with more total processing time on all machine. In 1984, the performance of this algorithm was discussed by Park et. al. [40]. In addition, there are also some researches which consider this algorithm to be an efficient way for makespan minimization [8, 28].

Let $J_1, J_2, J_3, \dots, J_n$ are $n$ jobs to be considered in PFSP on $m$ machines and $P_{ik}$ is the processing time of job $k$ on machine $i$ where $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$. Briefly, NEH algorithm can be explained simply by these 3 steps as follows:

1. Step1: sort the jobs as a non-increasing sequence according to their total processing time.

   To begin with, the total processing time $T_k$ of each job is calculated as:

$$T_k = \sum_{i=1}^{m} P_{ik} \quad (3.1)$$

   Then all jobs $J_1, J_2, J_3, \dots, J_n$ are to be sorted to $J'_1, J'_2, J'_3, \dots, J'_n$ in the way that $T'_1 \geq T'_2 \geq T'_3 \geq \cdots \geq T'_n$.

2. Step 2: schedule first two jobs (which have two highest total processing times) to minimize the makespan as if there are only these two jobs.

In this step, only $J'_1$ and $J'_2$ are considered. The makespan of the sequence $J'_1 - J'_2$ and $J'_2 - J'_1$ are calculated. The one with the minimum value of makespan will be selected. Please note that the relative order between $J'_1$ and $J'_2$ will remain unchanged in the output (i.e. $J'_1$ before $J'_2$ or $J'_2$ before $J'_1$).

3. Step 3: iteratively take new job into consideration one at a time (from $k = 3$ to $n$) according to the sequence obtained from step 1.

Each $J'_k$ is iteratively taken into consideration in order. To put it simply, $J'_k$ is inserted into the previous selected sequence of length $k - 1$ in each possible position. For each insertion, its makespan is calculated. The sequence with the minimum makespan is selected. Then, the algorithm proceeds to its next iteration.

The complexity of this approach depends on the number of makespan computations. In step 2, makespan is computed two times. In step 3, for each $k^{th}$ iteration, makespan is to be computed $k$ times. Therefore, the number of makespan computation is:

$$2 + 3 + \cdots + n = \frac{n(n+1)}{2} - 1 \quad (3.2)$$

In other words, the complexity of this algorithm is $O(n^2)$.

Overall, the advantage of the NEH algorithm is that it can give reasonable results using only easily implemented algorithm. However, as discussed above, the

computational complexity can be unacceptable or the output can be even far from the optimal solution when the number of jobs is high.

*3.1.2 Greedy Heuristic*

In 2012, M. Ancau proposed two heuristic approaches based on greedy concept for PFSP [29] which are the constructive greedy heuristic (CG) and the stochastic greedy heuristic (SG). In this subsection, only stochastic greedy heuristic approach will be described as it was shown by the author that this approach is likely to give better solutions.

*Constructive Greedy Heuristic (CG)*

The constructive heuristic algorithm (CG) generates a job's sequence using two lists called job list and optimal schedule. A job list consist of n elements (j1; j2; :::; jn). Firstly, a pair of jobs from the job list will be selected and arranged to find the minimum completion time passing to the optimal schedule. Then, repeat the first step, however either increase the selected elements to k(n-k-1), k is the number of rounds, or pass to the optimal schedule in the relative position that minimize completion time.



*Stochastic greedy heuristic (SG)*

In the stochastic heuristic (SG), the job list consists of n random job's elements. In detail, stochastic greedy heuristic approach can be described as follows:

1. Randomly generate a permutation of jobs, $J_{\pi_1}, J_{\pi_2}, J_{\pi_3}, \dots, J_{\pi_n}$.
2. Pick $J_{\pi_1}$ and $J_{\pi_2}$ as the starting positions for an optimal sequence. Their relative position is also adjusted to minimize the makespan.
3. Select the next candidate $J_{\pi_3}$ and its relative position to be placed on the optimal sequence which minimize the makespan among all possible positions.
4. Repeat step 3 fo$J_{\pi_4}, J_{\pi_5}, \dots, J_{\pi_n}$r to obtain an instance of the optimal sequence.

5. Repeat step 2 - 4 by using$J_{\pi_i}$ and $J_{\pi_{i+1}}$ as the starting positions where $i$ is an integer in range $[2, n-1]$.
6. Repeat step 1 - 5 for $S_{max}$ times while keeps tracking of the best optimal sequence. Finally, the one with the minimum makespan is returned.

As confirm by the experiment [29], this approach provides better results compared to the result from NEH algorithm significantly. However, the drawback of this approach is its longer time consumption.

## 3.2 Metaheuristics

In the previous section, some heuristic approaches related to PFSP were presented. Notice that, heuristic approaches are problem-dependent approaches which cannot be adapted to other problems. In this section, some metaheuristics approaches, which are problem-independent approaches, will be described.

### *3.2.1 Ant Colony System*

This idea was first introduced by Dorigo [41] . Then, in 2004, Rajendran proposed a variation of Ant Colony Optimization (ACO) application for PFSP [19] which is extended from the M-MMAS algorithm proposed by Stutzle in 1998 [21].

The general concept of ACO is based on the behaviour of how ants build and find their optimal paths. This mechanism is guided by a chemical substance called pheromone. Briefly, the algorithm will be executed back and forth between path generation phase and pheromone update phase. Each position along the way from the start to end positions are associated with their pheromones (in [19], these are called trail intensities). In the path generation phase, a path is generated based on those trail intensities. When the generation is completed, its value of objective function of the path will be computed. The value is then used in the pheromone update phase to make the next path generation phase likely to generate optimal solutions. Figure 3.1 illustrate a behaviour of ants in searching for the shortest path from nest to food source

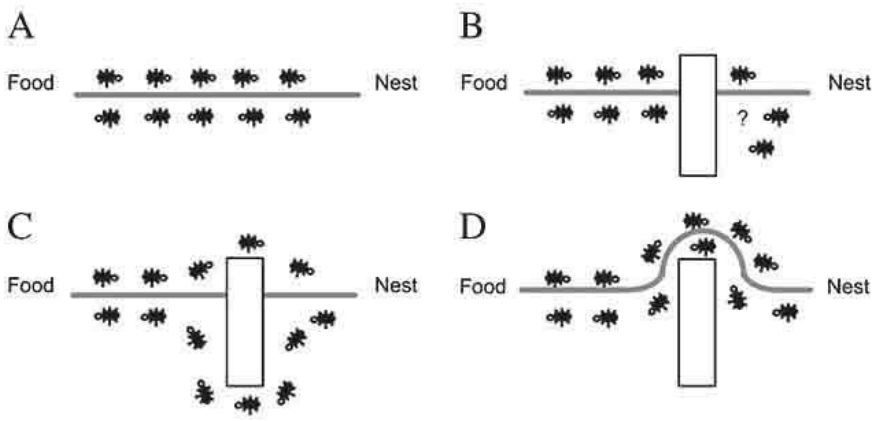


Figure 3.1 Ant's behavior when searching for food source

The mentioned approach proposed by Rajendran is called PACO [19]. Regarding to the approach, the path of ants or seed sequence is treated as the sequence of jobs. The sequence is to be iteratively updated through the ACO process. The algorithm can be briefly described as follows (More detail can be found in [19]):

1. Initialize seed sequence using the solution from NEH algorithm mentioned in section 3.1.1.
2. Refine the initial seed sequence using the following job-index-based local search procedure three times:
    a) For each job index $i$, insert the job in each other possible positions while keep the relative positions of other jobs to be the same.
    b) Choose the best sequence to update the current seed sequence.
    c) Repeat the above step for all possible values o $i$ f (all jobs).
3. Initialize each trail intensity $\tau_{ik}$ (the desire of placing job $i$ in position $k$).
4. Construct a new ant-sequence by selecting jobs from the unscheduled lists in order from the positions 1 to $n$. The selection is done according to a uniform random $u \in [0,1]$,
    a) If $u \leq 0.4$: the first unscheduled job in the best sequence is selected.

b) Else if $u \leq 0.8$: a job with maximum value o$T_{ik}$f among the first five unscheduled jobs in best sequence is selected. $T_{ik}$ can be computed by:

$$T_{ik} = \sum_{q=1}^{k} \tau_{iq} \quad (3.3)$$

c) Else: a job is selected according to the probability distribution:

$$p_{ik} = \frac{T_{ik}}{\sum_{l} T_{lk}} \quad (3.4)$$

, where $l$ is a member of the set of all indices the first five unscheduled jobs in best sequence.

5. Refine the generated sequence using the job-index-based local search procedure three times.
6. Update each trail intensity $\tau_{ik}$ according to the generated sequence.
7. Repeat the step 4 – 6 for a certain number of times (e.g. 40).
8. Final best sequence is then refined by job-index-based swap scheme.

The ACO algorithm was shown to be an effective algorithm. The main advantage is that the reasonable solution can be generated only within a small amount of time. However, the drawback of this approach is that the result is highly depended on the parameters setting.

*3.2.2 Coincidence Algorithm*

Coincidence algorithm (COIN) was introduced by Wattanapornphom et al in 2009 [42]. It is one of the evolutionary algorithm that use source of model to incremental learning from the previous candidate solution. Specially, COIN is not only train the data from positive feedback but also use negative feedback to avoid bad building block and reduce search space to focus only good solutions.

The algorithm consists of 6 steps. It is simply starts with initialize parameters and the joint probability matrix, H. Each element, Hxy, in the matrix denotes to the probability of y found in the absolute position x. Then, the candidate solution is generated in sequence to ensure that only valid permutations are sampled. In the third step, the candidate solution is evaluated by its objective or fitness value. All candidates are sorted from best to worst solution and divided into two groups of candidates, better group and worst group from the top and bottom C% of the rank. Then, the joint probability matrix is updated from these two groups as reward and punishment in the fifth step. Finally, repeat all steps until the termination condition is met. Figure 3.2 shows the process of NB-COIN.

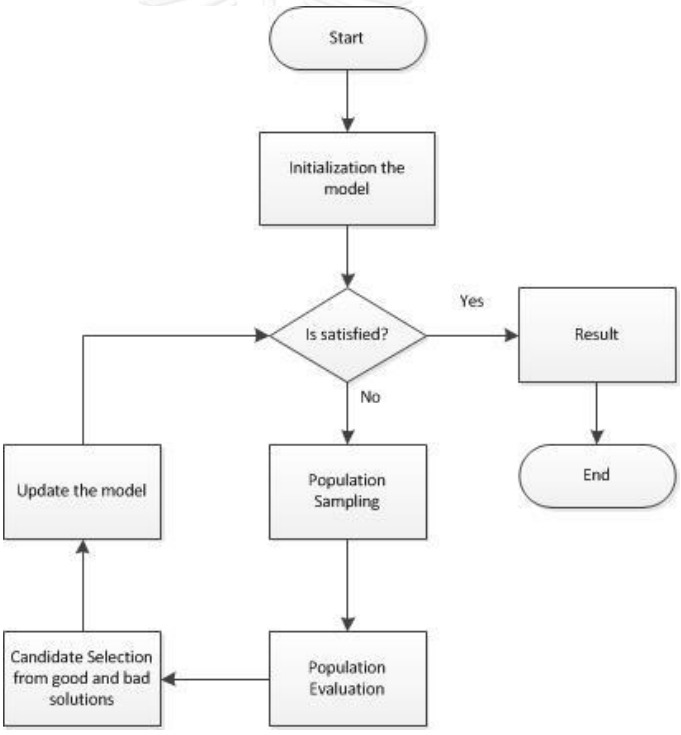


Figure 3.2 General procedure of Coincidence Algorithm

The complexity of COIN relies on the problem and candidate size in each generation. In the initialization stage, the complexity is $O(n^2)$. Population sampling require computational time at $O(mn^2)$ with $O(mn)$ space while Sorting and Candidate selection consumes $O(mlogm)$ and the update stage requires $O(mn^2)$.

The performance of COIN and NB-COIN was proven in many combinatorial optimization problems including path finding [42], production line sequencing and balancing [43], and puzzle [44, 45]. It is proved to be an effective algorithm that provides good quality solutions within small amount of computational time and resources. Also, the implementation of COIN is simple and require only a few user define parameters. Therefore, while other methods tend to either consume a lot of computational time and use complex operation to achieve high quality of solutions or use small amount of resources but provide lower quality, COIN becomes an alternative simple method which can provide competitive solutions in using less resources and time.

### 3.3 Hybrid Metahuristics Algorithm

The hybrid metaheuristic approaches are the approaches which combine metaheuristic approaches with other optimization techniques such as local search.

An example of metaheuristics approaches is the method proposed by Zobolas [23] in 2009. The method is based on well-known Genetic Algorithm (GA) [15, 16] and Variable Neighbourhood Search (VNS) [46]. Briefly an overview of the procedure of this approach is as follows:



1. Initial population generation: As the same as in traditional GA algorithm, the first step is to initialize the population. Let $i\_pop$ is the size of population, initial population can be partitioned as follows (the parameter $a$ is a real number between 0 and 1):
2. solution obtained from NEH [47],
    a) 1 solution obtained from CDS [6],
    b) 1 solution obtained from Palmer [5],
    c) 1 solution obtained from Gupta [48],
    d) $i\_pop * a - 4$ solutions sampled from GRNEH (Greedy Randomized procedure based on the NEH heuristic) and

e) $i\_pop * (1 - a)$ solutions generated randomly.

3. Population improvement: In this step, the population is updated in the manner of traditional GA. The detail of each phase is as follows:
   a) Tournament selection approach proposed in [49] is used for parents selection,
   b) Crossover is done by the two-point crossover (version I) proposed by Murata [33],
   c) Shift mutation operators proposed in [50]is also applied.
4. Intensification phase using VNS: In this step, shaking function is to be processed in order to avoid locally optimal issues (i.e. a solution selected for intensification is replaced with the best solution from another neighbourhood if some criterion are met).
5. Population renewal: To avoid local optima, ages of the solutions are also taken into consideration. Some old solutions are to be replaced with respect to some random factors.
6. Repeat step 2 – 4 for $t\_max$ times

This approach performs quite well in some cases such as when the number of jobs is low (less than 50). Anyway, there is also a drawback of this approach that the computational speed is still low.

### 3.4 Conclusion

In this chapter, some state of the art approaches to solve PFSP was discussed including heuristic, metaheuristic and hybrid heuristic.

Firstly, heuristic approaches were presented. NEH algorithm is simple algorithm and can give a good approximation for the optimal solutions. However, its computational complexity is high and, it can subject to local optima issues. Moreover, the heuristics approach based on stochastic and constructive greedy heuristics is also illustrated.

Even though it can generate better solutions compare to NEH, it still takes long time to process. Then, two metaheuristic approaches were presented. PACO algorithm for PFSP was proved to be efficient algorithm but its problem is the sensitivity with parameter setting. Alternatively, Coincidence algorithm can be used which can provide a good quality of solution using small amount of resources. Finally, a hybrid metaheuristic approach was presented. It is a combination of GA and VNS approaches. However, its computational complexity is still high.

Next chapter will identify the methodology of the research which is Node based Coincidence Algorithm. The characteristic of the algorithm will be discussed together with its advantages and disadvantages. Then the procedure of the algorithm for the PFSP with makespan minimization will be determined.

# *Chapter 4*

# *Node-Based Coincidence Algorithm*

In the previous chapter, the well-known approaches for the permutation flowshop scheduling problem were determined. It includes both heuristics, metaheuristic and hybrid metaheuristics approaches. The advantage and disadvantage of each approach solution was identified.

As discussed in the previous chapter, there is always a trade-off between the complexity of the solution approach, computational time and the quality of the result. The higher complexity of the algorithm tends to contribute better quality of results and/or consume longer computational time. Therefore, it is worth looking for solution approaches that able to provide acceptable solution with a simple implementation in a short period of time.

This chapter will introduce new alternative approach for the permutation flowshop scheduling problem called Node-based Coincidence Algorithm which is used as a methodology of this research. The characteristic of the algorithm will be discussed to set the solid understanding of the algorithm and its applications. In addition, the general procedure of the algorithm on the PFSP with makespan minimization will be explained in details.



## 4.1 Characteristic of Node-based Coincidence Algorithm

Node Based Coincidence Algorithm(NB-COIN) is adapted from Coincidence Algorithm(COIN) proposed by Wattanapornprom W. et al. in 2009 [42]. Both COIN and NB-COIN belongs to Estimation of Distribution Algorithm(EDA) class [22] that represented the model in joint probability matrix as Markov Chain where the candidate solutions are developed by using model or knowledge extracted from previous candidate solutions. The main characteristic of these algorithms is the incremental learning from both good and bad solutions to avoid bad building block and expand more diverse solutions. However, instead of representing the element as an adjacent

pair called coincidence in the traditional COIN, NB-COIN adopts the node based representation from Node Histogram Sampling Algorithm (NHBSA) [51] where each element denotes as an independent node.

There are many advantages of NB-COIN. First, it is an algorithm that easy to implement or replicate. It neither contains complex operation nor requires advance mathematic formulation. Furthermore, NB-COIN can provide good quality of solutions using small amount of computational time due to its incremental learning. Moreover the algorithm learns from both positive and negative knowledge.

The performance of NB-COIN was proven in many combinatorial optimization problems including flowshop scheduling problems with total flowtime minimization[52] and order acceptance problems [53]. In the flowshop scheduling problems with total flowtime minimization, it is proved to be an effective algorithm that can provide a good quality of solutions with an average of 1.7% different from the best known result. Moreover, the performance of NB-COIN in order acceptance problem is also competitive with other metaheuristic algorithms such as Genetic Algorithm and NHBSA.

## 4.2 General Procedure of NB-COIN for the PFSP

In the permutation flowshop scheduling problem, the population or candidate solution of NB-COIN represents the sequence of jobs. The objective function or fitness value is the time from the starting of the first job until all jobs completed. The detail of each stage can be explained as follow:

### *4.2.1 Initialization*

In this step, the joint probability matrix *H(X,Y)* is generated. The matrix consists of *n×n* elements where *n* is the number of the jobs waiting to enter flowshop production line. Each column(Y) present the probability of job Y in the position X of job sequence. To make a fair start, all elements contain the same probability at $\frac{1}{n}$. The joint probability matrix in the initialize stage is shown in Fig. 4.1.

| Position \ Node | $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ | $Y_5$ |
|---|---|---|---|---|---|
| $X_1$ | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| $X_2$ | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| $X_3$ | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| $X_4$ | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| $X_5$ | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |

Figure 4.1 The joint probability matrix based on the problem size of 5

*4.2.2 Population Sampling*

There are three steps to generate a job sequence in NB-COIN. First, a sequence of job position is randomly generated. Then, the job sequence (Y) is sampling from the position's sequence until desired population size is reached.

Figure 4.2 presents an example of probabilistic model and the population sampling process. In this example, the position's sequence is sampled as

$$X_2 - X_4 - X_1 - X_5 - X_3$$

Position's Sequence

| $X_2$ | $X_4$ | $X_1$ | $X_5$ | $X_3$ |
|---|---|---|---|---|

| Position \ Node | $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ | $Y_5$ |
|---|---|---|---|---|---|
| $X_1$ | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| $X_2$ | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| $X_3$ | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| $X_4$ | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| $X_5$ | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |

Cadidate

| $Y_4$ | $Y_1$ | $Y_2$ | $Y_5$ | $Y_3$ |
|---|---|---|---|---|

Figure 4.2 The example of solution and its probabilistic model

*4.2.3 Population Evaluation*

To calculate the fitness value, makespan ($C(K_m,J_n)$), the candidate solution is examined using the formula 2.1-2.5 mentioned in Chapter 2.

The calculation can by using the tabl $n \times n$ where n is the number of job to calculate the completion time $C(K_m,J_n)$ where K is a set of machine and J is a set of job. Figure 4.3 show a table for makespan calculation.

| | | | | |
|---|---|---|---|---|
| $C(K_1,J_1)$ | $C(K_1,J_2)$ | $C(K_1,J_3)$ | $C(K_1,J_{n-1})$ | $C(K_1,J_n)$ |
| $C(K_2,J_1)$ | $C(K_2,J_2)$ | $C(K_2,J_3)$ | $C(K_2,J_{n-1})$ | $C(K_2,J_n)$ |
| $C(K_3,J_1)$ | $C(K_3,J_2)$ | $C(K_3,J_3)$ | $C(K_3,J_{n-1})$ | $C(K_3,J_n)$ |
| $C(K_{m-1},J_1)$ | $C(K_{m-1},J_2)$ | $C(K_{m-1},J_3)$ | $C(K_{m-1},J_{n-1})$ | $C(K_{m-1},J_n)$ |
| $C(K_m,J_1)$ | $C(K_m,J_2)$ | $C(K_m,J_3)$ | $C(K_m,J_{n-1})$ | $C(K_m,J_n)$ |

Figure 4.3 Makespan table

*4.2.4 Candidate Selection*

According to the special characteristic of NB-COIN, it trains the population from both positive and negative knowledge. The algorithm sorts the population from best to worst by their fitness value. Then, the two sub-population groups, good and poor population, are selected to update the joint probability matrix. The size of sub-population groups denoted by the selection pressure(C%).

*4.2.5 The matrix update*

The joint probability matrix $H(X,Y_j)$ is updated by rewarding good solutions and punishing poor solutions. To reward the good solutions, the probability of node $X_iY_j$ is increased by $\frac{k}{n}$ while other nodes in the same row $X_i$ are decreased by $\frac{k}{n^2}$. The parameter $k$ is the learning step, $n$ is the problem size and $r_{ij}$ denotes the total number of good coincidence $H_{ij}$. The reward equation is shown below.

$$H_{ij}(t+1) = H_{ij}(t) + \frac{k}{n}\left(r_{ij}(t+1)\right) - \frac{k}{n^2}\left(\sum_{i=1}^{n} r_{ij}\,(t+1)\right) \quad (4.1)$$

On the other hands, the poor candidates are punished by scattering the probability $\frac{k}{n^2}$ to other node in the same row where *k* is learning step and *n* is the size of problem. $P_{ij}$ is the total number of coincidence from the poor solutions. The punishment equation is illustrated as follow

$$H_{ij}(t+1) = H_{ij}(t) - \frac{k}{n}\left(p_{ij}(t+1)\right) + \frac{k}{n^2}\left(\sum_{i=1}^{n} p_{ij}\,(t+1)\right) \quad (4.2)$$

There are two main important parameters in the matrix update stage; learning step *k* and the minimum probability value in the joint probability matrix $H(X,Y_j)$. The learning step, *k* in NB-COIN is a very important parameter that contributes directly to the quality of the result. High learning step can fasten the learning rate and increases the speed to get good quality of result but it trades off with the lower diversity of solutions. Moreover, the minimum probability value in the joint probability matrix $H(X_i,Y_j)$ need to be set to ensure that the probability of any element is higher than 0 so there is still an opportunity to sampling population from any elements in the matrix.

For the permutation flowshop scheduling problem, the diversity of solution is very important to avoid local trap of optima. Therefore, it is better to use small amount of learning step (less than 0.05) to maintain the diversity of solutions and increase the change to escape from local trap of optima.



## 4.3 Conclusion

This chapter has provided an overview of research methodology, the Node-based Coincidence Algorithm. The characteristic of the algorithm has been identified. The advantages of the algorithm have been discussed together with successful examples of its application in other combinatorial optimization problems. Finally, the procedure of NB-COIN has also explained in details.

According to the discussion in this chapter, NB-COIN is a competitive method that is easy to implement and able to provide high quality of solutions. It uses simple probabilistic matrix to generate solutions and updates the solution by using both good and bad samples to improve solutions and avoid bad solutions and it also reduces search space.

In the next chapter, Node-based Coincidence Algorithm will be used to solve the permutation flowshop scheduling problem and compared its performance against well-known approaches mentioned in Chapter 3.

# *Chapter 5*

# *Computational Environment and Results*

The previous chapter discussed the methodology of the research, Node-based Coincidence algorithm. The characteristic of the algorithm, its advantages and disadvantage is determined. Also, the procedure of NB-COIN on the permutation flowshop scheduling problem was illustrated.

This chapter will identify the computational experiment and discusses the computational result. To begin with, the design of computational experiment such as computer resources, the test instance and parameter setting will be shown. Then, the result of the research will be provided and discussed in details to test the effectiveness of the research methodology.

## 5.1 Design of computational Experiment

The proposed algorithm, Node Based Coincidence Algorithm(NB-COIN), was coded in C++ and run on MS Windows 7 using Intel Core i5 450M, 2.40GHz and 4GB of RAM.

### *5.1.1 Test Instance*



This research uses 40 instances of Taillard benchmark[8] where the number of job $n \in \{20,50\}$ and the number of machine $m \in \{5, 10, 20\}$.

Therefore, the test instance were selected and represented in four sets; 20×5, 20×10, 20×20, and 50×5, to determine the efficiency and performance of NB-COIN in the PFSP. Each set consist of 10 instances.

### *5.1.2 User Define Parameter*

There are several parameters that need to be defined for NB-COIN. For this research problem, the parameters have been set up and shown on table 5.1.

| Parameters | Value |
|---|---|
| Population Size | 500-1000 |
| Generation Size | 100-300 |
| Cutting percentage (C%) | 5-10% |
| Learning Step K | (<0.05) |
| Maximum probability | 0.8 |
| Minimum probability | `0.1*(1/Population size-3);` |

Table 5.1 Parameter setting for NB-COIN on PFSP

## 5.2 Results

The proposed algorithm was tested according to two different criteria; computational time and performance.

### *5.2.1 Computational time*

The CPU time obtained from NB-COIN were compared against the powerful metaheuristics such as ant colony systems[18] and the hybrid metaheuristic proposed by G.I. Zobolas[23] in 2009. For the CPU time, 5, 15, 25 and 100 seconds were allocated to four sets; 20×5, 20×10, 20×20 and 50×5. In Table 5.2, the results obtained from all groups of instance are summarized. The computational time of NB-COIN is superior when the number of job is 20 especially in 20×5 instance. It is twice faster than the hybrid metaheuristic and the ACS in 20×5 problem. Furthermore, the speed of hybrid metaheuristic is slower than NB-COIN by 5 and 15 seconds in 20×10 and 20×20 while the proposed algorithm is slightly slower than the ACS problem by 3 seconds and 9 seconds. However, the computational speed of NB-COIN decreases when the number of job exceeds 50.

| Instances | CPU time (Second) | | |
|---|---|---|---|
| | Hybrid Metaheuristic | ACS | NB-COIN |
| 20×5 | 10 | 11 | 5 |
| 20×10 | 20 | 12 | 15 |
| 20×20 | 40 | 16 | 25 |
| 50×5 | 25 | 44 | 100 |

Table 5.2 The computational speed

Since the permutation flowshop scheduling problem is np-hard. It is clear that the computational time of NB-COIN increases exponentially when the number of machine increases. It can be seen that the computational speed of NB-COIN is sensitive to the number of job more than the number of machine. From figure 5.1, the number of job creates a huge impact on the computational time of NB-COIN. When increasing the number of machine, the computational speed of NB-COIN is slower by few seconds but the speed is change dramatically when the number of job is increased.

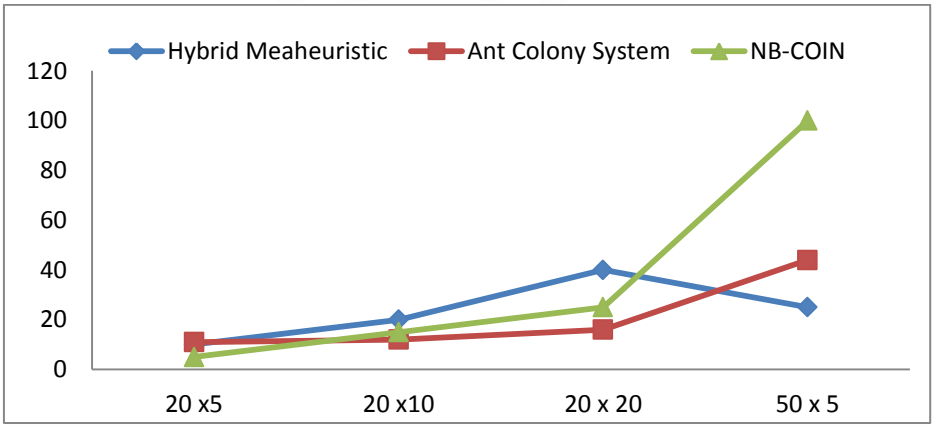


Figure 5.1 The graph shows computational time consumed by NB-COIN over other compared metaheuristics

*5.2.2 The Performance Analysis*

In this section, the solutions acquired from NB-COIN were tested on the Taillard benchmark against the upper bound. Although NB-COIN achieved the upper bound only a few solutions, it is essential to mention that NB-COIN runs on the PC and find the high quality solution in short CPU time while the upper bound are generally generated by branch and bound techniques and runs on a powerful workstations for extended time periods.

Moreover, the performance of NB-COIN was measured by the quality of solution. It is the percentage of gap between the makespan from our algorithm and the upper bound(UB) of Taillard. Each instance was run 5 times. To calculate is the percentage of gap, the equation is shown as follow;

$$Gap\ (\%) = \frac{C_{max} - UB}{UB} \times 100 \tag{5.1}$$

The results of three comparison methods; NEH, CG and SG are adopted from the original paper proposed by Nawaz et al. [4] and M. Ancau[29] to compare against NB-COIN. Overall, we found that NB-COIN performs far better than the NEH and the constructive greedy in all problem sizes while it is slightly superior the SG in the small size of problem (20×5). Moreover, NB-COIN provides a wide variety of solution that share the same quality.

Table 5.3 shows the result of 20×5 instance. NB-COIN not only found an optimum solution but the average gap is also a lot lower than both NEH and CG. However, comparing with SG algorithm, the average gap is a bit higher but NB-COIN performs better in term of the number of good solutions.

| Instances | UB | NEH | CG | SG | NB-COIN | Gap% | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | NEH | CG | SG | NB-COIN |
| Ta001 | 1278 | 1286 | 1286 | 1278 | 1294 | 0.626 | 0.626 | 0 | 1.252 |
| Ta002 | 1359 | 1365 | 1367 | 1366 | 1363 | 0.442 | 0.589 | 0.515 | 0.294 |
| Ta003 | 1081 | 1159 | 1141 | 1097 | 1090 | 7.216 | 5.550 | 1.480 | 0.833 |
| Ta004 | 1293 | 1325 | 1358 | 1306 | 1304 | 2.475 | 5.027 | 1.005 | 0.851 |
| Ta005 | 1235 | 1305 | 1301 | 1244 | 1244 | 5.669 | 5.344 | 0.729 | 0.729 |
| Ta006 | 1195 | 1228 | 1224 | 1210 | 1210 | 2.762 | 2.427 | 1.255 | 1.255 |
| Ta007 | 1239 | 1278 | 1264 | 1251 | 1251 | 3.148 | 2.018 | 0.968 | 0.968 |
| Ta008 | 1206 | 1223 | 1268 | 1206 | 1206 | 1.410 | 5.141 | 0 | 0 |
| Ta009 | 1230 | 1291 | 1277 | 1253 | 1253 | 4.959 | 3.821 | 1.870 | 1.870 |
| Ta010 | 1108 | 1151 | 1144 | 1117 | 1120 | 3.880 | 3.250 | 0.812 | 1.083 |
| | | | | | Average | 3.258 | 3.379 | 0.863 | 0.913 |

Table 5.3 Performance comparison of Taillard's 20×5 instance

The quality of solutions in the 20×10 and 20×20 problem are shown in Table 5.4 and Table 5.5. Since the performance of CG and SG algorithm for the instance where $m \in \{10,20\}$ are not report by M. Ancau[29], NB-COIN is solely tested with the NEH. The results show that the average gap of NB-COIN is over triple times better than the NEH in both size of problems.

| Instances | UB | NEH | NB-COIN | Gap% | |
|---|---|---|---|---|---|
| | | | | NEH | NB-COIN |
| Ta011 | 1582 | 1680 | 1599 | 6.195 | 1.074 |
| Ta012 | 1659 | 1729 | 1679 | 4.219 | 1.205 |
| Ta013 | 1496 | 1557 | 1518 | 4.077 | 1.471 |
| Ta014 | 1377 | 1439 | 1392 | 4.502 | 1.089 |
| Ta015 | 1419 | 1502 | 1433 | 5.850 | 0.987 |
| Ta016 | 1397 | 1453 | 1417 | 4.008 | 1.432 |
| Ta017 | 1484 | 1562 | 1513 | 5.256 | 1.954 |
| Ta018 | 1538 | 1609 | 1575 | 4.616 | 2.406 |
| Ta019 | 1593 | 1647 | 1608 | 3.390 | 0.942 |
| Ta020 | 1591 | 1653 | 1617 | 3.897 | 1.634 |
| | | | Average | 4.601 | 1.419 |

Table 5.4 Performance comparison of Taillard's 20×10 instance

| Instances | UB | NEH | NB-COIN | Gap% | |
|---|---|---|---|---|---|
| | | | | NEH | NB-COIN |
| Ta021 | 2297 | 2410 | 2323 | 4.919 | 1.132 |
| Ta022 | 2099 | 2150 | 2119 | 2.430 | 0.953 |
| Ta023 | 2326 | 2411 | 2349 | 3.654 | 0.989 |
| Ta024 | 2223 | 2262 | 2242 | 1.754 | 0.855 |
| Ta025 | 2291 | 2397 | 2314 | 4.627 | 1.004 |
| Ta026 | 2226 | 2349 | 2243 | 5.526 | 0.764 |
| Ta027 | 2273 | 2362 | 2300 | 3.915 | 1.188 |
| Ta028 | 2200 | 2249 | 2235 | 2.227 | 1.591 |
| Ta029 | 2237 | 2320 | 2276 | 3.710 | 1.743 |
| Ta030 | 2178 | 2277 | 2200 | 4.545 | 1.010 |
| | | | Average | 3.731 | 1.123 |

Table 5.5 Performance comparison of Taillard's 20×20 instance

As seen in Table 5.6, NB-COIN performs very well in Taillard's 50×5 instance. It is clear that the SG provides slightly better results in this size of problem. However, NB-COIN found more optimum solutions than SG and has lower average gap than both the NEH and CG algorithm. In addition, in each instance, although the average gap of SG algorithm is slightly lower than NB-COIN but the SG consumes more CPU time at almost double.

| Instances | UB | NEH | CG | SG | NB-COIN | Gap% | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | NEH | CG | SG | NB-COIN |
| Ta031 | 2724 | 2733 | 2761 | 2724 | **2724** | 0.330 | 1.358 | 0 | **0** |
| Ta032 | 2834 | 2843 | 2889 | 2848 | 2848 | 0.317 | 1.941 | 0.494 | 0.494 |
| Ta033 | 2621 | 2640 | 2674 | 2622 | 2640 | 0.725 | 2.022 | 0.038 | 0.725 |
| Ta034 | 2751 | 2782 | 2782 | 2782 | 2771 | 1.127 | 1.127 | 1.127 | 0.727 |
| Ta035 | 2863 | 2868 | 2908 | 2863 | **2863** | 0.175 | 1.572 | 0 | **0** |
| Ta036 | 2829 | 2850 | 2863 | 2840 | 2835 | 0.742 | 1.202 | 0.389 | 0.212 |
| Ta037 | 2725 | 2758 | 2781 | 2732 | 2739 | 1.211 | 2.055 | 0.257 | 0.514 |
| Ta038 | 2683 | 2721 | 2780 | 2701 | 2704 | 1.416 | 3.615 | 0.671 | 0.783 |
| Ta039 | 2552 | 2576 | 2595 | 2562 | 2565 | 0.940 | 1.685 | 0.392 | 0.510 |
| Ta040 | 2782 | 2790 | 2787 | 2784 | **2782** | 0.287 | 0.180 | 0.072 | **0** |
| | | | | | Average | 0.727 | 1.676 | 0.343 | 0.396 |

Table 5.6 Performance comparison of Taillard's 50×5 instance

Overall, it is clear that 10% of the solution found by NB-COIN is likely to be an optimal solution with 0% from upper bound where 3 out of 4 solutions are found in large instance (50x5). The gap averaging over all test instances is 0.96% from the upper bound. Figure 5.2 show the gap percentage of NB-COIN from the optimum value or upper bound in 4 set of test instances.



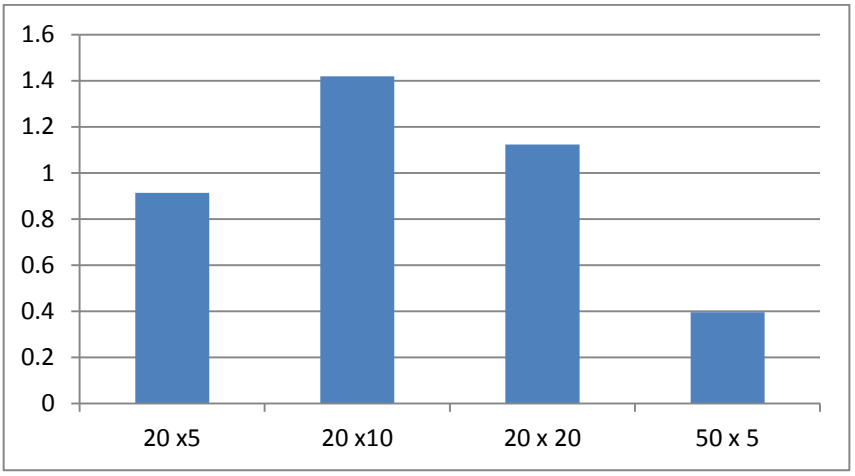


Figure 5.2 The performance of NB-COIN in term of gap % from upper bound

## 5.3 Conclusion

This chapter presented the computational environment and result for NB-COIN in the permutation flowshop scheduling problem. The proposed method was tested on a set of 40 Taillard instance. The experiment results are compared with powerful heuristics and metaheuristic such as Ant Colony system, NEH and constructive and stochastic greedy.

The results show that NB-COIN is an effective algorithm that can provide optimal solutions by 10% of the test instances. Moreover the average gap between the solutions and upper bound is also very low at 0.96%. The computational time of NB-COIN is also short which outperforms well-known complex algorithms such as ant colony system and hybrid metaheuristics in the small instance. However, the computational time of COIN is sensitive to the number of job so the computational time is increased dramatically when increasing the job number.

The next chapter will draw a conclusion of this research as well as provides a suggestion for future work.

# *Chapter 6*

# *Conclusion and Future Work*

The previous chapter presented the experimental design and computational result of this research.

This chapter will provide an overall conclusion to the research, based on the objective that has been achieved. The future work will also be suggested.

## 6.1 Conclusion

In this research, Node-based Coincidence algorithm was implemented and tested on the problem to find optimum solutions in the makespan minimization of the permutation flowshop scheduling.

The permutation flowshop scheduling problem is proved to be np-hard problem when the number of machine is higher than 2 machines. It is very difficult to find an optimal solution since the computational time will increase exponentially when the number of machine increases. Therefore, it is necessary to use heuristic or metaheuristic to solve the problem.



NB-COIN adopts an idea of incremental learning from the estimation of distribution Algorithm which belongs to Evolutionary algorithm class. It makes use of positive and negative knowledge to rapidly improve the solution. With this negative feedback, NB-COIN can reduce solution search space to focus only on good sampling. Also, the procedure of NB-COIN is very simple with only a few user defined parameters so it is very easy to replicate and implement.

On the experiment and results, NB-COIN is tested with a set of 40 Taillard instances and proved to outperform heuristic such as NEH and Greedy (CG and SG), the metaheuristic such as ant colony system and hybrid metaheuristic. The proposed method was tested in two criteria, computational time and performance.

1. The computational time

   In this criteria, NB-COIN outperforms At Colony system and Metaheuristic in small instances where the number of job is 20 jobs. However, the computational time of NB-COIN will be far higher than these two algorithms when the number of job increases to 50.

2. Performance

   The performance of NB-COIN is tested with NEH and greedy heuristics. The experiment shows that NB-COIN successfully achieves 4 optimal solutions which is 10% of the instance. Other the solutions of NB-COIN are also very close to the optimal value, at only 0.96% from the upper bound in average.

Hence, NB-COIN is an outstanding method that is easy to apply in the real world situation where computational time and quality solution is preferred.

.

## 6.2 Future Research

This research can be expanded and improved in the future work in various ways.

1. Implement NB-COIN in other flow shop scheduling problem

   There are many flow shop scheduling problems such as blocking flowshop scheduling problem, no wait flowshop scheduling problems and hybrid floshop scheduling problems. Since NB-COIN has proved to be an effective tool for the permutation flowshop scheduling problems, it is worth trying on other kind of flow shop problems.

2. Implement NB-COIN in other scheduling problems.

   NB-COIN also can be implemented and tested in other scheduling problems such as job shop scheduling problems and multi-processing scheduling problems.

3. Consider multi-objective in the permutation flowshop scheduling problems.

Some researchers consider 2 to 3 objectives on the permutation flowshop scheduling problems in order to cope with the practical environment where only 1 objective is not enough. Therefore, it is interesting to test NB-COIN on the multi-objective problems such as the combination of flow time and make span minimization.

4. Combine NB-COIN with some source of local search or convert to hybrid COIN.

   The performance of COIN can be increased by combining other methods and uses it as a local search in order to avoid local trap of optima.

# REFERENCES


[1] S. M. Johnson, "Optimal two-and three-stage production schedules with setup times included," *Naval research logistics quarterly,* vol. 1, pp. 61-68, 1954.

[2] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Mathematics of operations research,* vol. 1, pp. 117-129, 1976.

[3] J. K. Lenstra, A. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," *Annals of discrete mathematics,* vol. 1, pp. 343-362, 1977.

[4] M. Nawaz, E. E. Enscore, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega,* vol. 11, pp. 91-95, 1983.

[5] D. Palmer, "Sequencing jobs through a multi-stage process in the minimum total time--a quick method of obtaining a near optimum," *OR,* pp. 101-107, 1965.

[6] H. G. Campbell, R. A. Dudek, and M. L. Smith, "A heuristic algorithm for the n job, m machine sequencing problem," *Management science,* vol. 16, pp. B-630-B-637, 1970.

[7] D. G. Dannenbring, "An evaluation of flow shop sequencing heuristics," *Management science,* vol. 23, pp. 1174-1182, 1977.

[8] E. Taillard, "Some efficient heuristic methods for the flow shop sequencing problem," *European journal of Operational research,* vol. 47, pp. 65-74, 1990.

[9] J. M. Framinan, R. Leisten, and R. Ruiz-Usano, "Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimisation," *European Journal of Operational Research,* vol. 141, pp. 559-569, 2002.

[10] J. Framinan and R. Leisten, "An efficient constructive heuristic for flowtime minimisation in permutation flow shops," *Omega,* vol. 31, pp. 311-317, 2003.

[11] J. Grabowski and M. Wodecki, "A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion," *Computers & Operations Research,* vol. 31, pp. 1891-1909, 2004.

[12] E. Nowicki and C. Smutnicki, "A fast tabu search algorithm for the permutation flow-shop problem," *European Journal of Operational Research,* vol. 91, pp. 160-175, 1996.

[13] C. R. Reeves, "Improving the efficiency of tabu search for machine sequencing problems," *Journal of the Operational Research Society,* pp. 375-382, 1993.

[14] J.-P. Watson, L. Barbulescu, L. D. Whitley, and A. E. Howe, "Contrasting structured and random permutation flow-shop scheduling problems: search-space topology and algorithm performance," *INFORMS Journal on Computing,* vol. 14, pp. 98-123, 2002.

[15] C. R. Reeves, "A genetic algorithm for flowshop sequencing," *Computers & operations research,* vol. 22, pp. 5-13, 1995.

[16] C. R. Reeves and T. Yamada, "Genetic algorithms, path relinking, and the flowshop sequencing problem," *Evolutionary computation,* vol. 6, pp. 45-60, 1998.

[17] T. Stützle, "An ant approach to the flow shop problem," in *Proceedings of the 6th European Congress on Intelligent Techniques & Soft Computing (EUFIT'98)*, 1998, pp. 1560-1564.

[18] F. Ahmadizar, F. Barzinpour, and J. Arkat, "Solving permutation flow shop sequencing using ant colony optimization," in *Industrial Engineering and Engineering Management, 2007 IEEE International Conference on*, 2007, pp. 753-757.

[19] C. Rajendran and H. Ziegler, "Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs," *European Journal of Operational Research,* vol. 155, pp. 426-438, 2004.

[20] M. F. Tasgetiren, Y.-C. Liang, M. Sevkli, and G. Gencyilmaz, "A particle swarm optimization algorithm for makespan and total flowtime minimization in the

permutation flowshop sequencing problem," *European Journal of Operational Research,* vol. 177, pp. 1930-1947, 2007.

[21] T. Stützle, "Applying iterated local search to the permutation flow shop problem," *FG Intellektik, TU Darmstadt, Darmstadt, Germany,* 1998.

[22] B. Jarboui, M. Eddaly, and P. Siarry, "An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems," *Computers & Operations Research,* vol. 36, pp. 2638-2646, 2009.

[23] G. Zobolas, C. D. Tarantilis, and G. Ioannou, "Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm," *Computers & Operations Research,* vol. 36, pp. 1249-1267, 2009.

[24] H. Liu, L. Gao, and Q. Pan, "A hybrid particle swarm optimization with estimation of distribution algorithm for solving permutation flowshop scheduling problem," *Expert Systems with Applications,* vol. 38, pp. 4348-4360, 2011.

[25] V. Caraffa, S. Ianes, T. P. Bagchi, and C. Sriskandarajah, "Minimizing makespan in a blocking flowshop using genetic algorithms," *International Journal of Production Economics,* vol. 70, pp. 101-115, 3/21/ 2001.

[26] S.-W. Lin and K.-C. Ying, "Minimizing makespan in a blocking flowshop using a revised artificial immune system algorithm," *Omega,* vol. 41, pp. 383-389, 4// 2013.

[27] L.-Y. Tseng and Y.-T. Lin, "A hybrid genetic algorithm for no-wait flowshop scheduling problem," *International Journal of Production Economics,* vol. 128, pp. 144-152, 11// 2010.

[28] J.-Y. Ding, S. Song, J. N. D. Gupta, R. Zhang, R. Chiong, and C. Wu, "An improved iterated greedy algorithm with a Tabu-based reconstruction strategy for the no-wait flowshop scheduling problem," *Applied Soft Computing,* vol. 30, pp. 604-613, 5// 2015.

[29] T. Aldowaisan and A. Allahverdi, "New heuristics for no-wait flowshops to minimize makespan," *Computers and Operations Research,* vol. 30, pp. 1219-1231, 2003.

[30] L. Wang and C. Fang, "An effective estimation of distribution algorithm for the multi-mode resource-constrained project scheduling problem," *Computers and Operations Research,* vol. 39, pp. 449-460, 2012.

[31] Q. K. Pan, M. Fatih Tasgetiren, and Y. C. Liang, "A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem," *Computers and Operations Research,* vol. 35, pp. 2807-2839, 2008.

[32] Q. K. Pan, M. Fatih Tasgetiren, P. N. Suganthan, and T. J. Chua, "A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem," *Information Sciences,* vol. 181, pp. 2455-2468, 2011.

[33] Q.-K. Pan, L. Wang, J.-Q. Li, and J.-H. Duan, "A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation," *Omega,* vol. 45, pp. 42-56, 6// 2014.

[34] E. Nowicki and C. Smutnicki, "The flow shop with parallel machines: A tabu search approach," *European Journal of Operational Research,* vol. 106, pp. 226-253, 1998.

[35] K. Alaykýran, O. Engin, and A. Döyen, "Using ant colony optimization to solve hybrid flow shop scheduling problems," *International Journal of Advanced Manufacturing Technology,* vol. 35, pp. 541-550, 2007.

[36] Q. Niu, T. Zhou, and S. Ma, "A quantum-inspired immune algorithm for hybrid flow shop with makespan criterion," *Journal of Universal Computer Science,* vol. 15, pp. 765-785, 2009.

[37] J. E. Day and M. P. Hottenstein, "Review of sequencing research," *Naval Research Logistics Quarterly,* vol. 17, pp. 11-39, 1970.

[38] A. Reisman, A. Kumar, and J. Motwani, "Flowshop scheduling/sequencing research: a statistical review of the literature, 1952-1994," *Engineering Management, IEEE Transactions on,* vol. 44, pp. 316-329, 1997.

[39] R. Ruiz and C. Maroto, "A comprehensive review and evaluation of permutation flowshop heuristics," *European Journal of Operational Research,* vol. 165, pp. 479-494, 9/1/ 2005.

[40] Y. B. Park, C. D. Pegden, and E. E. Enscore, "A survey and evaluation of static flowshop scheduling heuristics," *The International Journal of Production Research,* vol. 22, pp. 127-141, 1984.

[41] M. Dorigo, "Optimization, learning and natural algorithm," Phd., DEI, Politecnico di Milano, Italy, 1992.

[42] W. Wattanapornprom, P. Olanviwitchai, P. Chutima, and P. Chongstitvatana, "Multi-objective Combinatorial Optimisation with Coincidence algorithm," in *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, 2009, pp. 1675-1682.

[43] P. Chutima and N. Kampirom, "A multi-objective coincidence memetic algorithm for a mixed-model U-line sequencing problem," *International Journal of Advanced Operations Management,* vol. 2, pp. 201-248, 01/01/ 2010.

[44] R. Sirovetnukul, P. Chutima, W. Wattanapornprom, and P. Chongstitvatana, "The effectiveness of hybrid negative correlation learning in evolutionary algorithm for combinatorial optimization problems," in *Industrial Engineering and Engineering Management (IEEM), 2011 IEEE International Conference on*, 2011, pp. 476-481.

[45] K. Waiyapara, W. Wattanapornprom, and P. Chongstitvatana, "Solving Sudoku puzzles with node based Coincidence algorithm," in *Computer Science and Software Engineering (JCSSE), 2013 10th International Joint Conference on*, 2013, pp. 11-16.

[46] P. Hansen and N. Mladenovi**Ć**, "Variable neighborhood search: Principles and applications," *European journal of operational research,* vol. 130, pp. 449-467, 2001.

[47] M. Nawaz, E. E. Enscore Jr, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega,* vol. 11, pp. 91-95, 1983.

[48] J. N. Gupta, "A functional heuristic algorithm for the flowshop scheduling problem," *Operational Research Quarterly,* pp. 39-47, 1971.

[49] J. H. Holland, *Genetic algorithms*. New York: Scientific American, 1992.

[50] R. Ruiz, C. Maroto, and J. Alcaraz, "Two new robust genetic algorithms for the flowshop scheduling problem," *Omega,* vol. 34, pp. 461-476, 2006.

[51] S. Tsutsui, "Node Histogram vs. Edge Histogram: A Comparison of Probabilistic Model-Building Genetic Algorithms in Permutation Domains," in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, 2006, pp. 1939-1946.

[52] O. Srimongkolkul and P. Chongstitvatana, "Application of Node Based Coincidence algorithm for flow shop scheduling problems," in *Computer Science and Software Engineering (JCSSE), 2013 10th International Joint Conference on*, 2013, pp. 49-52.

[53] W. Wattanapornprom, L. Tieke, W. Wattanapornprom, and P. Chongstitvatana, "Application of node based estimation of distribution algorithms for solving order acceptance with capacity balancing problems by trading off between over capacity and under capacity utilization," in *Computer Science and Engineering Conference (ICSEC), 2014 International*, 2014, pp. 238-243.

# VITA

Miss Ornrumpha Srimongkolkul was born on January, 24, 1991 in Bangkok, Thailand. She achieved her bachelor degree in Computer Engineering at Cholulongkorn University in 2013. After the graduation, she decided to pursue her master degree in computer engineering at Chulalongkorn University.