IDSF II: Integrated Decision Support Framework and its Application for Dispatching
Policy based on Part Similarity

A thesis presented to
the faculty of
the Russ College of Engineering and Technology of Ohio University

In partial fulfillment
of the requirements for the degree
Master of Science

Jia Guo
March 2012
© 2012 Jia Guo. All Rights Reserved.
This thesis titled
IDSF II: Integrated Decision Support Framework and its Application for Dispatching
Policy based on Part Similarity

by

JIA GUO

has been approved for
the Department of Industrial and Systems Engineering
and the Russ College of Engineering and Technology by

__________________________________________
Dušan Šormaz
Associate Professor of Industrial and Systems Engineering

__________________________________________
Dennis Irwin
Dean, Russ College of Engineering and Technology
ABSTRACT

GUO, JIA, M.S., March 2012, Industrial and Systems Engineering

IDSF II: Integrated Decision Support Framework and its Application for Dispatching Policy based on Part Similarity

Director of Thesis: Dusan Sormaz

Computer simulation is increasingly considered as an effective method in academic research and industrial application. A framework that integrates simulation model building, running, and analyzing from given input data is expected to be extremely beneficial. This research improves Integrated Decision Support Framework (IDSF I) by redesigning the system structure to improve efficiency, reprogramming under Objects Oriented Modeling to increase flexibility of the framework for future expansion and further experimentation. A FMS is designed to carry the experimentation. The running performance of IDSF is examined, along with further exploration of the effects of various system parameters in terms of system performance. Parts similarity measure is also introduced into the system as criterion for dispatching policies and its performance is compared to classical policies. Procedure is applied on WIP in front of each machine, such that processing order of parts is recalculated repeatedly such that sequence of parts corresponds to most or least similar parts. Motivation for this approach is load balancing among machines, which further results in higher production rates.

Approved: _____________________________________________________________

Dušan Šormaz

Associate Professor of Industrial and Systems Engineering
TABLE OF CONTENTS

ABSTRACT .............................................................................................................................. 3
LIST OF TABLES .................................................................................................................... 6
LIST OF FIGURES .................................................................................................................. 7
1. INTRODUCTION ............................................................................................................ 9
2. LITERATURE REVIEW .............................................................................................. 11
   2.1 Flexible Manufacturing System ............................................................................ 11
   2.2 Scheduling in FMS ............................................................................................... 12
   2.3 Similarity ................................................................................................................ 14
   2.4 Integration of CAPP and FMS Simulation ............................................................. 15
3. PROBLEM STATEMENT AND OBJECTIVES ......................................................... 18
   3.1 The Requirement of Dynamic Decision Making .................................................. 18
   3.2 The Integration of FMS ......................................................................................... 19
   3.3 The Usage of Similarity/Dissimilarity .................................................................. 19
   3.4 Objectives ............................................................................................................... 20
      3.4.1 Integrated Decision Support Framework II ................................................... 20
      3.4.2 Introduction of New Dispatching Policies ....................................................... 22
4. METHODOLOGY ......................................................................................................... 23
   4.1 Architectural Improvement of IDSF II .................................................................. 23
   4.2 Object Oriented Model Structure of IDSF II ....................................................... 25
   4.3 Similarity/Dissimilarity Coefficient Based Dispatching Policy ............................. 27
5. IMPLEMENTATION .................................................................................................... 33
   5.1 Object Oriented Modeling ..................................................................................... 33
      5.1.1 FMS Group ..................................................................................................... 33
      5.1.2 Simulation Model Group .............................................................................. 38
      5.1.3 Control Group .............................................................................................. 41
## LIST OF TABLES

Table 1: Example of Dissimilarity Calculation Step 1 ..................................................... 30
Table 2: Step 1 Resorted Queue by Dissimilarity............................................................. 31
Table 3: Example of Dissimilarity Calculation Step 2 ..................................................... 31
Table 4: Step 2 Resorted Queue by Dissimilarity............................................................. 32
Table 5: List of Classes and Methods in FMS Group....................................................... 36
Table 6: List of Classes and Methods in Simulation Group............................................. 39
Table 7: List of Classes and Methods in Control Group .................................................. 42
Table 8: Running Parameters............................................................................................ 58
Table 9: P-value for all measures (all Model Type)......................................................... 62
Table 10: P-value for all measures (Model Type Random and Dynamic Best).............. 63
Table 11: P-value for all measures (Model Type Dynamic Best) .................................... 64
Table 12: Performance Improvement by Model Type...................................................... 67
Table 13: Performance Improvement by Routing Policy ................................................ 74
Table 14: Machine Utilization Increase by Inter-arrival Time ....................................... 75
Table 15: Total Queue Size Decrease by Batch Sizes .................................................... 77
LIST OF FIGURES

Figure 1: Manufacturing Model of IDSF [12] ................................................................. 16
Figure 2: Architecture of IDSF I ..................................................................................... 21
Figure 3: Proposed Architecture of IDSF II ................................................................. 24
Figure 4: Structure of Object ........................................................................................ 26
Figure 5: Structure of FMS Group ................................................................................. 34
Figure 6: Structure of Simulation Model Group ............................................................ 38
Figure 7: Structure of Control Group ............................................................................ 41
Figure 8: Main Window ................................................................................................. 44
Figure 9: Main Window (Initiation) .............................................................................. 45
Figure 10: Main Window (After Select Files) ............................................................... 46
Figure 11: Main Window (After Set Project Name) ....................................................... 47
Figure 12: Main Window (After Set Parameters) .......................................................... 48
Figure 13: File Loading Window .................................................................................. 50
Figure 14: Parameter Selection Window ...................................................................... 52
Figure 15: Similarity/Dissimilarity Modules in Simulation Model ............................... 53
Figure 16: FMS Configuration [12] ............................................................................. 55
Figure 17: Part Design [12] ......................................................................................... 56
Figure 18: Screenshot of IDSF Input Data ................................................................. 57
Figure 19: Screenshot of running parameters selection ............................................... 59
Figure 20: Screenshot of Output File .......................................................................... 60
Figure 21: Total Part Out of Model Types ................................................................. 65
Figure 22: Total WIP of Model Types ................................................................. 66
Figure 23: Average Total Waiting Time of Model Types ...................................... 66
Figure 24: Total Queue Size of Model Types ....................................................... 66
Figure 25: Average Total Machine Utilization of Model Types .............................. 67
Figure 26: Single Machine Utilization for Dynamic Best Model Type ..................... 69
Figure 27: Average System Performance of Dispatching Policies .......................... 70
Figure 28: Total Parts Out of Routing Policies .................................................... 72
Figure 29: Total WIP of Routing Policies ............................................................ 72
Figure 30: Average Total Waiting Time of Routing Policies ................................. 72
Figure 31: Total Queue Size of Routing Policies .................................................. 73
Figure 32: Average Total Machine Utilization for Routing Policies ....................... 73
Figure 33: Average Total Machine Utilization of Inter-Arrival Time 6 and 7 ........... 75
Figure 34: Total Queue Size for Batch Sizes ...................................................... 76
1. INTRODUCTION

The continuous improvement of system efficiency and productivity is increasingly necessary for the success of companies and their manufacturing systems. Mass production is no longer able to accommodate the dynamic changing market needs. Diversity and flexibility of the products have become the key factors of manufacturing systems. In the highly competitive global market nowadays, shorter production cycle, increased production output and improved system adaptability would greatly improve manufacturers’ ability to succeed.

Flexible Manufacturing System (FMS) is increasingly utilized in industry for its nature of high adaptability and responsiveness. Instead of traditional mass production, flow shop, and job shop system, FMS is able to produce smaller volume and higher variety of products without changing the basic structure of the system, by utilizing numerically controlled machines (Computer Aided Manufacturing), robots and automated transportation system. During the adaptation process of new products, the control module of FMS is able to make updated decision based on the current system status and available policy options. Moreover, improved quality and system efficiency can also be expected due to the increased flexible controllability of the digitalized system.

Many researchers contributed to the improvement of FMS with different focus. Some research focus on the utilization of scheduling theories in FMS, while others focus on process planning and etc. Computer Simulation has also been considered as an effective tool in understanding the complex nature of FMS and comprehensive issues
involved in system performance improvement. However, only a few researches have attempted to integrate different perspectives and tools into one platform.

In this research, a framework that integrates flexible manufacturing system with flexible scheduling, dynamic decision making, multi-products/processes, and computer simulation is developed. The purpose of this framework is to create a simulation platform for researchers and industrial users to integrate various scheduling policies, process planning theories and etc. New theories and their combination and comparison with existing policies can be included in the framework. By designing and running experimentations on the framework, more comprehensive understanding of FMS is expected to be continuously achieved. Dissimilarity Coefficient is also included in this research as one of the dispatching policies besides common policies such as First In First Out (FIFO), Shortest Processing Time (SPT), and etc. The motivation for such attempt is to achieve more balanced system work load, represented by higher total machine utilization, when parts with least similar process is chosen as succeeding operation choice.
2. LITERATURE REVIEW

This section is categorized into three subsections according to the notions introduced into this research, which are Flexible Manufacturing System, Scheduling in FMS, and Similarity.

2.1 Flexible Manufacturing System

A vast amount of publications have been focused on a variety of issues about Flexible Manufacturing System, including the request, definition, dimension, measurement, and other related subjects. It is important to have better understanding of the nature of FMS, as well as the notion of flexibility, in order to more effectively design and improve the integrated framework in this research.

Stecke defined FMS as an integrated system combining numerically controlled machines and automated material handling system that is able to a variety of part types simultaneously [18]. According to Correa, flexibility is a filter, and “absorber” of uncertainty, so that the perturbations from outside the system can be reduced [7]. Simon divided the reason of uncertainty into two categories, lack of information and lack of knowledge, which implies the need for effective information flow of the system for better decision making [14].

Skinner categorized flexibility into three dimensions according to the objects of variance, the process flexibility, the product flexibility and the product volume flexibility [15]. J. Browne et al. [4] classified flexibility in FMS into more specific eight categories: machine flexibility, process flexibility, routing flexibility, operation flexibility, production flexibility, volume flexibility, layout flexibility, and product flexibility. They
also pointed out that the total production flexibility is composed primarily of two main
categories: machine flexibility and routing flexibility. Flexibility in FMS was also
classified in another periodic way: Short-median-term flexibility and Long-term
flexibility, according to Barad and Sipper [1]. A ninth category of flexibility was also
added to the eight categories specified by Browne, transfer flexibility, which is the
capability of multiple machines in the system to process the same job.

Other researchers have also contributed a variety of classifications, but share the
main principles.

Previous research and classification of FMS demonstrated the main characteristics
of the FMS. The need for dynamic information flow for decision making and the high
responsiveness of the system is obviously existent, which implies the requirement of this
study about dynamic decision support system and its integration.

2.2 Scheduling in FMS

Flexible Manufacturing System implements various scheduling theories and
policies, including but not limited to routing selection policies, dispatching policies (also
known as part selection policies), process planning, and etc. Many researchers have been
working on such subjects in order to improve the overall system performance or meet
specific performance target. The integration of various scheduling policies and study of
their effectiveness or their combination’s effectiveness is one of the major purposes of
developing the framework in this research.

Chan has conducted several research experiments to study the impact of different
routing and dispatching policies. In a simulated FMS model [5], three routing policies,
No Alternative Routings (NAR), Alternative Routings Dynamic (ARD), and Alternative Routings Planned (ARP), along with four dispatching policies, Shortest Processing Time (SPT), Shortest Imminent Processing Time (SIPT), Longest Processing Time (LPT), and Longest Imminent Processing Time (LIPT). The result of the simulation runs indicated that the combination of ARP routing policy and SPT dispatching policy perform the best result for infinite system buffer. For finite buffer capacity, however, the performance of such combination is compromised. The experiment also implied that the dispatching policy of LIPT showed variation among different combinations with routing rules.

In another study of the impact of the dispatching policies conducted by Chan et al. [6], 14 dispatching policies, FCFS, FAFS, SPT, LPT, SPT x TOT, SPT/TOT, LPT x TOT, LPT/TOT, FOPNR, MOPNR, EDD, LDD, LWKR, and MWKR (see reference for acronym articulations), were considered along with three measures of performance, mean flow time, mean tardiness, and mean earliness. The FMS simulation model collected system information based on the three measures of performance at every decision point, which is after a certain number of parts. Based on the system measures, the dispatching policies that do not provide good performance are replaced by others to generate the best performance of the system. The condition of machine breakdown was also included in the study. Different objectives were also considered in the study. The results of the study confirmed that system with dynamic changing dispatching policies perform better than any single policy system.

Jeong et al. [11] have introduced Genetic Algorithm (GA) in a dynamic framework into FMS scheduling area. Several models have been provided to improve the
overall performance of FMS by dynamically collecting system data into GA and generating new schedules.

2.3 Similarity

Numerous researchers have conducted research about the utilization of the notion of similarity/dissimilarity in manufacturing system, many of which are for CF purposes.

In a methodology introduced by Saygin and Kilic [13], dissimilarity was employed for routing selection in FMS. The model assumed that production mix is unknown, thus offline scheduling would be impossible in such case. Dissimilarity between different routing options was calculated based on the number of machine types that are not common between the two routes divided by the total number of machine types. In FMS operation, the most dissimilar route would be chosen to achieve maximized system efficiency and capability utilization. A nonlinear integer programming formulation was developed to implement the dissimilarity routing selection logic. In a simulation experiment including FA (First Available), EPL (Equal Probability Loading), and DMM (Dissimilarity Maximization Method) as routing selection rules, and several dispatching policies, DMM performed the best result among the studied rules.

Hachicha et al. [9] proposed a CF method considering the similarity and the principal component analysis (PCA). A similarity coefficient matrix was built during the first phase. Based on the machine usage of different parts, a matrix containing such information was collected. A general standardization method was employed to generate the similarity coefficient of machines. PCA was implemented during the second phase to
assign machines and parts into different cells. A cluster algorithm was also used to accommodate exceptional machines and parts.

Jeon and other [10] improved the similarity coefficient to include machine failure condition as a parameter for alternative routing consideration. Garbie et al. [8] proposed an approach to introduce new parts into existing manufacturing cells based on similarity coefficient.

In the methodology proposed by Bhatnagar and Saddikuti [2], the capability of operators was also taken into consideration. Moreover, a big set of problems reported in research was introduced to the model as well as a group of widely used similarity coefficients, which were compared and analyzed during the study. Therefore, this study is especially valuable in terms of the research of similarity coefficient.

Most published studies about similarity/dissimilarity in FMS were devoted to CF or routing selection, but the utilization of similarity in dispatching policy remains to be explored, which implies the practical reason for the proposed study.

2.4 Integration of CAPP and FMS Simulation

Integrated Decision Support Framework for Flexible Manufacturing Systems (IDSF) was developed by Patel [12]. Three modules are included in this Framework. Alternative process plans are generated by Rule Based System developed by Sormaz et al. [16] [17]. A simulation model with Numerically Controlled Machines and Auto Guided Vehicles is developed to implement different logic, and test the impact of different combination of logics, routing policies and dispatching policies. The manufacturing model is shown in Figure 1. An inter-connective module, composed by
MS Excel and the VBA program within, is also introduced to transform original data to meet the requirement of different model types, and store model information as well.

Figure 1: Manufacturing Model of IDSF [12]

Four model types have been developed in this framework, Static Best, Random, Routing Dynamic Best, and Feature Focused Dynamic Best. The functionalities of these four models are listed below.

Static Best: - For the same part type, the same predetermined best process plan with lowest processing time will be selected at the beginning of simulation for all following parts.
Static Random: - At each process selection decision point (PSDP), a random process plan amongst predetermined alternative plans will be selected until the next PSDP.

Routing Dynamic Best: - At each PSDP, a process plan amongst predetermined alternative plans will be selected based on the current system status to achieve better system utilization until the next PSDP.

Feature Focused Dynamic Best: - At each PSDP, a new best process plan will be generated for the remaining operations based on the current system status to achieve better system utilization until the next PSDP.

Four dispatching policies are employed in this framework. FIFO, SPT, SIPT, LIPT. For each combination of all model types, dispatching policies, batch sizes, and performance parameters, a simulation model is generated from the model template. After running all simulation models, statistical analysis is performed to determine the model type and dispatching policy with better system performance. The result of this study showed that, Feature Focused Dynamic Best model achieved both higher throughput and better load balance among alternative machines.
3. PROBLEM STATEMENT AND OBJECTIVES

This study is driven by some practical problems in the implementation of FMS. Clear views of these problems are meaningful and necessary before introducing any ideas and methodologies.

3.1 The Requirement of Dynamic Decision Making

The requirement of the achievement of flexibility can be generally classified into two categories, the capability of the facilities to perform various jobs, and the responsiveness to implement such capability. Finding the effective logic to achieve such responsiveness is the essential task for the research of FMS.

According to some previous work in the literature review section, the notion of cybernetics is included among most of the definitions of flexibility, where continuous system adjustments need to be performed in real time based on feedbacks from the system. Thus, the achievement of flexibility should be based on the implementation of cybernetic principles. Feedback and adjustment based on feedback are crucial elements of the basic control loop, without which control loop would not be a control loop but a simple flow process. The very same logic can be introduced into FMS. Without the information collection about the system performance and decision making based on the collected information, FMS would be no more a flexible system but a tradition flow shop manufacturing system with much lower responsiveness. Therefore, Dynamic Decision Making is the essential component of FMS, the key element to achieve the basic goal of FMS.
3.2 The Integration of FMS

Considerable amount of research have been conducted to improve the performance of FMS from various perspectives. Though many have made progress in finding new methodologies to achieve higher flexibility, most current models lack the integrated framework to compose these improved methodologies and newly coming approaches into an integrated system. An integrated framework that is able to compose a variety of components of FMS, implement different methodologies, and test new approaches, is required for both academic research and practical purposes. The integration may include but not limited to, routing policies, dispatching policies, logic of operators and their randomness, system failure, operation buffer, transportation of parts, cost, and etc.

3.3 The Usage of Similarity/Dissimilarity

As summarized in the literature review section, the notion of similarity/dissimilarity has been widely used in manufacturing systems. In CF methodologies utilized today, similarity between machines and parts is the most basic element to determine the composition of manufacturing cells. Similarity has also been introduced into routing selection problems in some research, to improve the system utilization. But the usage of similarity/dissimilarity as dispatching policy remains unexplored, and probably some dispatching policies based on similarity are able to make new improvement of the performance of FMS.
3.4 Objectives

This study includes two main parts, which are the designing of IDSF II with significant structure improvement from the previous IDSF, and improved model logics and their integration with IDSF, including Dissimilarity Coefficient Based Dispatching Policy (DCBDP).

3.4.1 Integrated Decision Support Framework II

Due to its flexibility and responsiveness to dynamically changing system status, IDSF has considerably huge potential for future implementation in industrial field and academic research, such as the integration with other theoretical development in scheduling, process control, and CAM. However, the previous architecture of IDSF, shown in Figure 2, is not flexible and integrated enough for future development.

Firstly, after the process plan generation, each one of the combination of all model types, routing policies, dispatching policies, batch size and other possible criteria, will be generated as an .xlsx file containing specific model information. In the previous study, 120 .xlsx files were generated before data can be loaded into simulation models, which is a waste of time and effort. Moreover, all model info files need to be manually loaded into simulation template for running, which require a large amount of effort and time to complete. Nevertheless, such redundancy is not necessary for the functionality of the framework, and therefore should be improved for the efficiency of future academic and practical implementation.
Secondly, in the existing framework, all codes for data transformation, model construction (VBA code for most of them) are inside the .xlsx files and simulation model template, based on the current theoretical elements built in the framework. It implies that simulation models and model information files are necessarily required to be modified.
when new methodologies and control logic are to be introduced into IDSF, which requires a significant amount of work, and is inefficient for future development and implementation. One important objective of this study is addressing such inefficiency.

3.4.2 Introduction of New Dispatching Policies

Four dispatching policies have been introduced in the previous IDSF for test purposes, which are FIFO, SPT, SIPT, and LIPT. Since these dispatching policies are developed in traditional manufacturing systems, new dispatching policies accommodating dynamic framework are desired.

Similarity/Dissimilarity has been one of the most significant factors considered in manufacturing systems. This study will introduce Dissimilarity Coefficient Based Dispatching Policy into the framework as a new dispatching policy (DCBDP). The hypothesis of DCBDP is that, when the most dissimilar part from the previous part is chosen to be processed, it is the least likely that it will be sent to the same machine sequence for the remaining operation, therefore total WIP will be reduced, total utilization of the machines can be increased, and throughput time will be reduced. A set of simulation experiments will be performed in the new IDSF II prototype to validate this hypothesis.
4. METHODOLOGY

In this study, a new framework, Integrated Decision Support Framework II, is designed for better model building efficiency and controllability. The new framework is designed under Object Oriented Modeling to achieve better flexibility in terms of system expansion. A new dispatching policy based on parts dissimilarity was also developed and integrated into the framework.

4.1 Architectural Improvement of IDSF II

The newly designed IDSF II improves the architecture of previous IDSF, to increase the efficiency of the working process and the compatibility of the framework with new elements. A control framework is developed in Visual Basic, with user interface for experimentation design, model info input, and simulation running control. Most of system functions will be redeveloped in Object Oriented Programming (OOP) within the new control framework, and interact with model info files, simulation model, and other elements of IDSF II. As a result, when introducing new simulation experiment parameters into the system, only certain relative methods need to be modified, while simulation model and other modules could remain the same functionality within the framework. The improved architecture is shown in Figure 3.
Figure 3: Proposed Architecture of IDSF II
As shown in the figure, the process plan is generated by CAPP and stored in .csv files. The procedure for generation of process plan is described by Patel [12], and its detail is beyond the scope of this thesis. The control framework represented by a user interface window takes the model info input and simulation experiment parameters from the operator. By the transformation function code within IDSF II, model info and process plan are combined and transformed to the input of a specific simulation model, which is one of the combinations of all experiment parameters. IDSF II further combines this input and the simulation template to generate a simulation model, save and run it to get results for replications and results for simulation run, stored in .csv files. IDSF II will keep generating new simulation models and run them until all combinations of running parameters are finished. This function is able to reduce the manual work for experiment running than the previous methodology in IDSF, which require operator to manually load the info of every model to simulation template.

4.2 Object Oriented Model Structure of IDSF II

In this research, the structure of IDSF II is developed under Object Oriented Programming environment to organize the functions of the system into interconnected classes, as shown in Figure 4. According to the functionality of each class, these classes are classified into three functional groups, which are Control Group, FMS Group, and Simulation Model Group.
Figure 4: Structure of Object
FMS Group contains Communication class, FMSModel class, Part class, Tool class, Machine class, Feature class, and Operation class. FMS Group is mainly used to store data from process planning system, such as machines, parts, tools, and operations, in an organized structure.

Simulation Model Group, composed by SimModel class and RunModel class, focuses on exporting FMS data into Simulation Model template, setting up various simulation models with different parameters.

Control Group, including User Interface class, Parameters class, Select Files class, and DDMS Project class, is responsible for controlling both groups above within each project.

See Implementation section for detailed explanation about each group and class.

4.3 Similarity/Dissimilarity Coefficient Based Dispatching Policy

In this study, various algorithms for similarity calculation from past research are evaluated to gain more comprehensive understanding about the concept of similarity.

JSC model [2] was introduced for similarity calculation in CF. The definition of this model is defined as Equation 1.

\[
\frac{\sum_j a_{ij} \times a_{kj}}{\sum_j a_{ij} \times \sum_j a_{kj}} - \frac{\sum_j a_{ij} \times \sum_j a_{kj}}{\sum_j a_{ij} \times \sum_j a_{kj}}
\]

\( a_{ij} = 1 \), if machine i is required for processing part j

\( = 0 \), otherwise

Noticeably, the section of \( \sum_j a_{ij} \times \sum_j a_{kj} \) in the denominator is designed to make the range of the result from 0 to 1, without which the result would be ranged from 0 to 0.5.
SCB model [2] is a simplified JSC model, where only the number of shared parts between two machines is taken consideration. The definition of this model is shown as Equation 2.

\[ \sum_j a_{ij} \times a_{kj} \]  

\( a_{ij} = 1, \text{ if machine } i \text{ is required for processing part } j \)  
\( = 0, \text{ otherwise} \)

SVC model [2] is also been used in CF machine similarity calculation. It introduced three levels of similarity between two machines. When an operation requires both machines, they are considered as similar by given “score” of 2. When an operation requires neither machines, the “score” is defined as 0, higher than the lowest similarity level of “score” -1 where an operation requires only one out of two machines. The definition of this model is shown in Equation 3.

\[ \sum_j \delta_v(a_{ij} + a_{kj}) \]  

\( a_{ij} = 1, \text{ if machine } i \text{ is required for processing part } j \)  
\( = 0, \text{ otherwise} \)
\[ \delta_v(a_{ij} + a_{kj}) = 2, \text{ if } a_{ij} = a_{kj} = 1 \]  
\( = 1, \text{ if } a_{ij} \neq a_{kj} \)  
\( = 0, \text{ otherwise} \)

In this research, an algorithm is employed for similarity/dissimilarity calculation for DCBDP. The similarity between two parts is defined as the number of machines they shared in their process. The similarity coefficient is calculated as the number of machine(s) shared by both parts’ process plans multiply by 2, then divided by the sum of
the number of machine(s) been used within each plan. The mathematical model is defined as Equation 4. The number 2 as a factor is designed to make the range of $S_{(a,b)}$ from 0 to 1.

$$S_{(a,b)} = \frac{M_s \times 2}{M_a + M_b} \quad (4)$$

$S_{(a,b)}$ Similarity between part $a$ and part $b$

$M_s$ Number of machine(s) shared by the process plans of part $a$ and part $b$

$M_a$ Number of machine(s) in the process plan of part $a$

$M_b$ Number of machine(s) in the process plan of part $b$

An example of the calculation of Similarity Coefficient is given below. Part A and Part B are two parts within the system, and each contains 4 process plans. $P_{A1}$, $P_{A2}$, $P_{A3}$, $P_{A4}$ are for Part A, and $P_{B1}$, $P_{B2}$, $P_{B3}$, $P_{B4}$ are for Part B.


$P_{B3}$ contains: Machine B, Machine D, Machine F.

Machine B and Machine D are shared machines between two plans, so $M_s$ Equals to 2.

Therefore, the similarity coefficient of $P_{A2}$ and $P_{B3}$ can be calculated as:

$$S_{(A2,B3)} = \frac{2 \times 2}{5 + 3} = 0.5$$

In this study, the logic of Similarity/Dissimilarity used as dispatching policy can be described as follow: at the end of every process step of every part, all waiting parts in the queue behind this part will be rearranged based on how similar/dissimilar they are compared to the leaving part. Therefore, as soon as this part leaves the machine, the
machine will pick the first one in the rearranged waiting queue, which is the most dissimilar part to the previous one.

An example is given in Table 1. Five parts are waiting in the queue of a machine that is currently processing a “Current Part”. Each part has been assigned a process plan when entering the system. The specific machine requirement of the process plan of each part is shown in the table. The similarity coefficient of every part compared to the current part on machine is calculated by the algorithm employed in this study. Based on the calculated similarity coefficient, then ranking of similarity is able to be produced and shown in the bottom row of the table.

### Table 1: Example of Dissimilarity Calculation Step 1

<table>
<thead>
<tr>
<th>Parts in Queue</th>
<th>P3</th>
<th>P5</th>
<th>P2</th>
<th>P1</th>
<th>P6</th>
<th>Current Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mach. required by current process plan</td>
<td>M2</td>
<td>M4</td>
<td>M1</td>
<td>M2</td>
<td>M1</td>
<td>M4</td>
</tr>
<tr>
<td>M3</td>
<td>M5</td>
<td>M3</td>
<td>M4</td>
<td>M3</td>
<td>M5</td>
<td></td>
</tr>
<tr>
<td>M4</td>
<td>M4</td>
<td>M5</td>
<td>M4</td>
<td>M4</td>
<td>M7</td>
<td></td>
</tr>
<tr>
<td>M6</td>
<td></td>
<td>M7</td>
<td></td>
<td></td>
<td>M5</td>
<td></td>
</tr>
<tr>
<td>M7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>M6</td>
<td></td>
</tr>
<tr>
<td>Similarity to Current Part</td>
<td>0.5</td>
<td>0.8</td>
<td>0.34</td>
<td>0.58</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>Ranking</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Based on the ranking of similarity coefficient, the queue will be resorted by descending sequence of the similarity coefficient ranking (or ascending sequence of the similarity coefficient). The new queue is shown in Table 2.

**Table 2: Step 1 Resorted Queue by Dissimilarity**

<table>
<thead>
<tr>
<th>New Queue</th>
<th>P5</th>
<th>P1</th>
<th>P6</th>
<th>P3</th>
<th>P2</th>
<th>Machine</th>
</tr>
</thead>
</table>

After the queue is resorted, P2 will be selected by the machine because it has the least similarity with the previous “Current Part”. When P2 is about to be finished, a new round of resorting of the current 4 parts in queue will be initiated, as shown in Table 3. It should be notices that it is possible that new parts join the queue between these two steps. Sorting them at the end will include any new parts joining the queue.

**Table 3: Example of Dissimilarity Calculation Step 2**

<table>
<thead>
<tr>
<th>Parts in Queue</th>
<th>Mach. required by current process plan</th>
<th>Similarity to Current Part</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>P5</td>
<td>M4</td>
<td>0.4</td>
<td>3</td>
</tr>
<tr>
<td>P1</td>
<td>M2</td>
<td>0.29</td>
<td>4</td>
</tr>
<tr>
<td>P6</td>
<td>M1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>M2</td>
<td>0.5</td>
<td>2</td>
</tr>
</tbody>
</table>
Again, based on the ranking of similarity coefficient, the queue will be resorted by descending sequence of the similarity coefficient ranking. The new queue is shown in Table 4.

**Table 4: Step 2 Resorted Queue by Dissimilarity**

<table>
<thead>
<tr>
<th>New Queue</th>
<th>P6</th>
<th>P3</th>
<th>P5</th>
<th>P1</th>
<th>Machine</th>
</tr>
</thead>
</table>

Detailed explanation of how this logic is implemented in simulation model, see Section 5.
5. IMPLEMENTATION

In this section, modules involved in this research will be thoroughly explained, including the implementation of IDSF II in an object oriented programming environment, the design of user interface, and the implementation of similarity/dissimilarity coefficient based dispatching policy.

5.1 Object Oriented Modeling

IDSF II is designed under Object Oriented Modeling. The system structure can be classified into three different groups, including FMS Group, Simulation Model Group, and Control Group. The purpose and functionality of each class is explained as following.

5.1.1 FMS Group

FMS group, as shown in Figure 5, is designed to store FMS data in an interconnected hierarchical structure, therefore provides more organized data. FMS group also provides a variety of methods that can be called throughout the system for more flexible and capable data handling. It is also easy to be expanded by simply adding new Subs at certain target class when new logic of data handling is needed during the upgrade of the system. The functionality of each class is discussed as following. Detailed list and explanation of all methods of every class in FMS group is shown in Table 5.
Communication class is responsible of reading input data from supported format (in this case, MS Excel Sheet, but has the potential to be expanded into other formats), and transforming them into logically connected instances of different classes. Communication class also handles the communication with Control Group and Simulation Model Group, and supports the building of simulation models by providing access to FMS configuration data, such as machines, tools, parts, features, and etc.

FMSModel class represents a specific FMS configuration with machine set, part set, and tool set information. It is also the highest level of a FMS system, which contains different Hashtables to store system information. Once an FMSModel instance is created
by Communication class, system configuration data will be transformed and stored separately by “add” subs in corresponding Hashtables under the instance of FMSModel. Other functionalities are also given to this class, such as exporting machine or part names as string.

Machines class and Tools class are designed to store machines and tools in the system. This information can be accessed through FMSModel class.

Parts class is the hosting class of all types of parts within the system. A hashtable of features is created for each instance of Parts, and logic of reading and selecting non-duplicate features is also included when adding new features through the “addFeature” sub.

Similar to Parts class, each instance of Features class represents a specific feature of a part, and stores all the operations required to manufacture this feature. Except for the logic of selecting the non-duplicate operation, Features class is also responsible for calculating and storing the best operation among others in terms of processing time.

Operations class is the lowest level of FMS Group. Instances of Operations class store the basic information of every manufacturing operation, such as processing time, type of operation. It also correspond each operation to existing instances of Machines and Tools, to identify the machine on which the operation is to be performed and the needed tool for such operation.
Table 5: List of Classes and Methods in FMS Group

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication</td>
<td>Sub New(ByVal excelPath As String, ByVal proj As DDMSProject)</td>
<td>Create new instance of Communication, call subs to create new instance of FMSModel, set input data path, call subs to read through input data into memory</td>
</tr>
<tr>
<td></td>
<td>Sub getTotalInfo()</td>
<td>Record the total number of lines of input data</td>
</tr>
<tr>
<td></td>
<td>Sub getInputtoArray(ByVal line As Integer)</td>
<td>Get process info of specified line into array</td>
</tr>
<tr>
<td></td>
<td>Sub readArray ()</td>
<td>Loop through all input data and call methods to read data and store into FMS as instances</td>
</tr>
<tr>
<td>FMSModel</td>
<td>Sub New (ByVal com As Communication)</td>
<td>Create new instance of FMSModel, set parent Communication instance, reset all FMS indexes into 0</td>
</tr>
<tr>
<td></td>
<td>Sub initialize()</td>
<td>Create new Hashtables to store part, machine and tool info</td>
</tr>
<tr>
<td></td>
<td>Public Sub addPart(ByVal partName As String)</td>
<td>When part does not exist currently in corresponding Hashtable, create instance of Parts, and add to the Hashtable</td>
</tr>
<tr>
<td></td>
<td>Public Sub addMachine(ByVal machineName As String)</td>
<td>When machine does not exist currently in corresponding Hashtable, create instance of Machines, and add to the Hashtable</td>
</tr>
<tr>
<td></td>
<td>Public Sub addTool(ByVal toolName As String)</td>
<td>When tool does not exist currently in corresponding Hashtable, create instance of Tools, and add to the Hashtable</td>
</tr>
<tr>
<td></td>
<td>Function partNameToString () As String</td>
<td>Return all part names as string, separated by CrLf</td>
</tr>
<tr>
<td></td>
<td>Function machineNameToString () As String</td>
<td>Return all machine names as string, separated by CrLf</td>
</tr>
<tr>
<td></td>
<td>Function toolNameToString () As String</td>
<td>Return all tool names as string, separated by CrLf</td>
</tr>
<tr>
<td></td>
<td>Function getPartCount () As Integer</td>
<td>Return total number of parts in Hashtable</td>
</tr>
<tr>
<td></td>
<td>Function getMachineCount () As Integer</td>
<td>Return total number of machine in Hashtable</td>
</tr>
<tr>
<td></td>
<td>Function getToolCount () As Integer</td>
<td>Return total number of tool in Hashtable</td>
</tr>
<tr>
<td></td>
<td>Function findMachine(ByVal name As String) As Machines</td>
<td>Find and Return the instance of Machines by name as string</td>
</tr>
<tr>
<td></td>
<td>Function findPart (ByVal name As String) As Parts</td>
<td>Find and Return the instance of Parts by name as string</td>
</tr>
<tr>
<td></td>
<td>Function findTool (ByVal name As String) As Tools</td>
<td>Find and Return the instance of Tools by name as string</td>
</tr>
<tr>
<td>Class</td>
<td>Constructor Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Parts Class</td>
<td>Sub New (ByVal name As String, ByVal FMSModel As FMSModel)</td>
<td>Create new instance of Parts, set parent FMSModel instance, create new Hashtable to store features of this part</td>
</tr>
<tr>
<td></td>
<td>Sub addFeature (ByVal featureName As String)</td>
<td>When feature does not exist currently in corresponding Hashtable, create instance of Features, and add to the Hashtable</td>
</tr>
<tr>
<td></td>
<td>Public Function getFeatureCount () As Integer</td>
<td>Return total number of features in Hashtable</td>
</tr>
<tr>
<td></td>
<td>Public Function findFeature (ByVal name As String) As Features</td>
<td>Find and Return the instance of Features by name as string</td>
</tr>
<tr>
<td>Machines Class</td>
<td>Sub New (ByVal name As String, ByVal FMSModel As FMSModel)</td>
<td>Create new instance of Machines, set parent FMSModel instance</td>
</tr>
<tr>
<td>Tools Class</td>
<td>Sub New (ByVal name As String, ByVal FMSModel As FMSModel)</td>
<td>Create new instance of Tools, set parent FMSModel instance</td>
</tr>
<tr>
<td>Features Class</td>
<td>Sub New (ByVal name As String, ByVal part As Parts)</td>
<td>Create new instance of Features, set parent Parts instance, create new ArrayList to store operations of this feature, set index of best operation and best processing time as 0</td>
</tr>
<tr>
<td></td>
<td>Sub addOperation (ByVal processName As String, ByVal machineName As String, ByVal toolName As String, ByVal pTime As Double)</td>
<td>When operation does not exist currently in corresponding Hashtable, create instance of Operations, and add to the Hashtable. If the process time of this operation is smaller than the existing best process time, the processing time of this operation would be recorded as the best, and the index of this operation would be recorded as the best operation</td>
</tr>
<tr>
<td></td>
<td>Public Function getOperationCount () As Integer</td>
<td>Return total number of operation in ArrayList</td>
</tr>
<tr>
<td>Operations Class</td>
<td>Sub New (ByVal feature As Features, ByVal type As String, ByVal machine As Machines, ByVal tool As Tools, ByVal pTime As Double)</td>
<td>Create new instance of Operations, set parent FMSModel instance, set the instances of Tools and Machines required to perform this operation as properties of this operation</td>
</tr>
</tbody>
</table>
5.1.2 Simulation Model Group

Simulation Model Group contains two classes, SimModel and RunModel, as shown in Figure 6. The main functionality of this group is to set up simulation models based on different parameters, handle the running of each model as well as exporting the results. Detailed list of all classes and methods of this group is shown in Table 6.

![Figure 6: Structure of Simulation Model Group](image)

SimModel class is responsible of configuring the general system of simulation models without touching specific parameters that define the differences between each single model to be ran in the experiment. A template from an assigned location is loaded when an instance of SimModel is created. Basic system elements are transferred into Simulation model, such as machines, parts, tools, distance, etc.; general experimental parameters are also transferred into simulation model, including inter-arrival time of parts, replication length, number of alternative plans, number of replications, etc.; a number of variables and sets are generated within the simulation model to assist the functionalities of experiment running. Two .csv files are generated inside the experiment folder to store running results of every model to be generated and run.
After receiving order and specific configuration parameters from the Control Group, RunModel class is designed to generate the final Arena model of the given configuration, and give the “run” order. The functionalities of RunModel class include but not limit to, adjusting model based on the given model type/routing policy type/dispatching policy type to meet the logic requirement, generating process plans for Random/Dynamic Best model type and marking the best one with lowest processing time, saving models after run is finished, attaching results to .csv files of the entire experimentation, etc..

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation Model</td>
<td>Sub New (ByVal modelTemplatePath As String, ByVal proj As DDMSProject)</td>
<td>Create new instance of SimulationModel, create new Arena application and open model template at given path, set parent DDMSProject instance</td>
</tr>
<tr>
<td></td>
<td>Sub transferInputData()</td>
<td>Transfer FMS system data into arrays for simulation model to access</td>
</tr>
<tr>
<td></td>
<td>Sub setParameterStrings()</td>
<td>Set model parameter names into string arrays</td>
</tr>
<tr>
<td></td>
<td>Sub retrieveRunParameters()</td>
<td>Retrieve simulation running parameters from parent class</td>
</tr>
<tr>
<td></td>
<td>Sub retrieveDistance()</td>
<td>Retrieve distances between machines from input data</td>
</tr>
<tr>
<td></td>
<td>Sub transferInterarrivalTime()</td>
<td>Retrieve inter-arrival times from parent class</td>
</tr>
<tr>
<td></td>
<td>Sub insertMachine()</td>
<td>Insert machines into simulation model</td>
</tr>
<tr>
<td></td>
<td>Sub transferPartNames()</td>
<td>Transfer part names into simulation model</td>
</tr>
<tr>
<td></td>
<td>Sub transferPartsAndMachinesInSet()</td>
<td>Transfer parts and machines into sets in simulation model</td>
</tr>
<tr>
<td></td>
<td>Sub transferSequenceIndexInVariable( )</td>
<td>Transfer sequence index into variables in simulation model</td>
</tr>
<tr>
<td></td>
<td>Sub createCurrentPartTypeVariable()</td>
<td>Create variable set of &quot;CurrentPartType&quot; and &quot;Current PlanNumber&quot; based on the total number of machines</td>
</tr>
<tr>
<td></td>
<td>Sub transferAltPlansInVariable()</td>
<td>Transfer number of alternative plans into simulation model</td>
</tr>
<tr>
<td></td>
<td>Sub createSequenceSwitchVariable()</td>
<td>Transfer more model parameters into simulation model</td>
</tr>
</tbody>
</table>
Table 6 (Continued): List of Classes and Methods in Simulation Group

<table>
<thead>
<tr>
<th>Sub Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transferProcessInfoInSequence()</td>
<td>Create sequence set in simulation model</td>
</tr>
<tr>
<td>transferDistanceInfo()</td>
<td>Transfer distance information into simulation model</td>
</tr>
<tr>
<td>transferResourceAndMachineQueueInformation()</td>
<td>Create machine resources and queues in simulation model</td>
</tr>
<tr>
<td>insertMachineQueueInAdvancedSet()</td>
<td>Insert machine queue in advanced set</td>
</tr>
<tr>
<td>createDispatchingVariable()</td>
<td>Create Dispatching Variable</td>
</tr>
<tr>
<td>createCSVPath()</td>
<td>Create the path for CSV files for recording running results</td>
</tr>
<tr>
<td>writeCSVTitle()</td>
<td>Create CSV files at given path, and write parameter titles</td>
</tr>
<tr>
<td>New(ByVal arena As Arena.Application, ByVal model As Arena.Model, ByVal parentProj As DDMSProject)</td>
<td>Create new instance of RunModel, set simulation application, set running parameters</td>
</tr>
<tr>
<td>transferData()</td>
<td>Transfer FMS data into variables for future use</td>
</tr>
<tr>
<td>insertParameters()</td>
<td>Insert running parameters into simulation model</td>
</tr>
<tr>
<td>transferProcessInfoInSequence()</td>
<td>Transfer process info in sequence into simulation model</td>
</tr>
<tr>
<td>transferProcessInfo()</td>
<td>Generate and transfer process plans into simulation model</td>
</tr>
<tr>
<td>calculateSumProcessingTime()</td>
<td>Calculate Total Processing Time for each process plan</td>
</tr>
<tr>
<td>getBestPlan()</td>
<td>Record Best Process Plan</td>
</tr>
<tr>
<td>transferBestPlanInVariable()</td>
<td>Transfer Best Process Plans into variables in Simulation Model</td>
</tr>
<tr>
<td>transferSumPPTimeInVariable()</td>
<td>Transfer Total Process Times into simulation model</td>
</tr>
<tr>
<td>transferDispatchingVariable()</td>
<td>Transfer Dispatching Type into simulation model</td>
</tr>
<tr>
<td>transferRoutingPolicy()</td>
<td>Transfer Routing Policy into simulation model</td>
</tr>
<tr>
<td>changingQueueType()</td>
<td>Change Queue Type based on dispatching policy</td>
</tr>
<tr>
<td>batchQuietRun()</td>
<td>Determine batch run</td>
</tr>
<tr>
<td>setFileNames()</td>
<td>Set file names for this simulation model</td>
</tr>
<tr>
<td>saveFile()</td>
<td>Create folder for each simulation model and save related files</td>
</tr>
<tr>
<td>calculateTotalWIP()</td>
<td>Calculate total working in progress</td>
</tr>
<tr>
<td>beginRunModel()</td>
<td>Begin running simulation</td>
</tr>
</tbody>
</table>
5.1.3 Control Group

Control Group, as shown in Figure 7, provides user interface, allow user to set up the configuration and control the running of experimentations. It is also the commanding group to set up FMS system and experimentation based on given information by user, as well as running the experimentation. Detailed list of all classes and methods of this group is shown in Table 7.

![Figure 7: Structure of Control Group](image)

User Interface class, Parameters class and Select Files class are based on three UI windows, containing necessary functionalities to transform input parameters into variables, and save them in the system. More details will be given in section 5.2.

DDMSProject class is the control class of each configured experiment. It is the parent class of Communication class in FMS Group and SimModel/RunModel classes in Simulation Model Group. All communication between these three classes, (or between FMS Group and Simulation Model Group), are conducted through DDMSProject class.
For every experiment, an instance of DDMSProject class is created. All experiment configurations, such as model types involved, dispatching policies involved, etc, are stored as variables/arrays within the instance. After configuring FMS system and having a general simulation model template through SimModel class, several loops are introduced to loop through each element of each parameter, to control the RunModel class to generate every simulation model of every combination of parameters, as well as running these models.

Table 7: List of Classes and Methods in Control Group

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserInterface</td>
<td>Private Sub FormUserInterface_Load</td>
<td>Create new instance of DDMSProject when form is loaded</td>
</tr>
<tr>
<td></td>
<td>Private Sub ButtonSelectFiles_Click</td>
<td>Show Loading File window</td>
</tr>
<tr>
<td></td>
<td>Private Sub ButtonParameters_Click</td>
<td>Show parameters selection window</td>
</tr>
<tr>
<td></td>
<td>ButtonFMS_Click</td>
<td>Call methods on DDMSProject to build FMS, create model info recording file</td>
</tr>
<tr>
<td></td>
<td>Private Sub ButtonSimModel_Click</td>
<td>Call methods on DDMSProject to build simulation model</td>
</tr>
<tr>
<td></td>
<td>Private Sub ButtonRun_Click</td>
<td>Call methods on DDMSProject to build specific models, combine all running results from each model into two results files</td>
</tr>
<tr>
<td></td>
<td>Private Sub ButtonProjName_Click</td>
<td>Set project name, create results files for this project</td>
</tr>
<tr>
<td>Select Files</td>
<td>Private Sub ButtonSelectDataFile_Click</td>
<td>Open dialog to choose input data file path</td>
</tr>
<tr>
<td></td>
<td>Private Sub ButtonSelectTemplateFile_Click</td>
<td>Open dialog to choose simulation template file path</td>
</tr>
<tr>
<td></td>
<td>Private Sub OK_Button_Click</td>
<td>Read data file locations into system</td>
</tr>
<tr>
<td></td>
<td>Private Sub Cancel_Button_Click</td>
<td>Cancel data file selection</td>
</tr>
<tr>
<td>Parameters</td>
<td>Private Sub DialogParameters_Load</td>
<td>Reset all parameters when dialog is loaded</td>
</tr>
<tr>
<td></td>
<td>Private Sub OK_Button_Click</td>
<td>Loop through all user selections, read and store all parameters</td>
</tr>
</tbody>
</table>
5.2 Design of User Interface

User interface design is one of main purposes of this study, the key element to achieve better control and handling of the framework. Through user interface, operator of IDSF II should be able to flexibly configure experiments by changing system parameters, such as model type, dispatching policy, batch size, inter-arrival time and etc, to test the influential degree of each factor, therefore obtains a more optimized model in general to serve real world operations. Three user interface forms, including Main Window, Parameter Selection Window, and File Loading Window, were created to achieve such goal.

5.2.1 Main Window

The Main Window, as shown in Figure 8, is the basic platform for user operation. By clicking the buttons of different functionalities, a simulation system/experiment with specific configuration is created step by step. Main Window is also the “display board” of all parameters included in this specific configuration, as well as the path of input files. The functionality of each element (button) within the Main Window is explained as follow.
When IDSF II is started, the Main window is loaded. An instance of the DDMSP project, discussed in section 5.1.3, is also created along with the opening of the program, representing a new framework to be generated and run throughout the entire running life of the program. Noticeably, the displays of model parameters are invisible while the Main window is initially loaded, as shown in Figure 9, since none of these parameters has been selected. As the operation progresses, however, the parameters will be displayed in sequence as they are selected one after another. The same logic also applies to buttons. Select Files button is the only enabled button when the Main Window is initially loaded, but other buttons will be enabled one after another, partially for the
purpose of “guiding” the user through the correct sequence in the process of system configuration.

![Figure 9: Main Window (Initiation)](image)

When button *Select Files* is clicked, the File Loading Window, discussed in section 5.2.2, is loaded for user to select the input data file and simulation model template files. Once these files are selected, the paths of these files will be displayed on the Main Window, and the corresponding segments of the display will be enabled. The button of *Set Project Name*, as well as the text box for project name input is also enabled for the
user to use for the next step. The screenshot of the Main Window, after *Select Files* button is clicked and needed files are selected, is shown in Figure 10.

![Main Window (After Select Files)](image)

**Figure 10: Main Window (After Select Files)**

Once the button of *Set Project Name* is clicked after a project name is given in the text box, a project folder with the given name will be created under the directory where the input data files are from, which will be used to store all files in this experiment. The button cannot be clicked if no project name is typed in the text box, and if such situation occurs, a message box of “Project name cannot be empty!” will appear to prevent the user
moving to the next step. Two csv files will also be created under such project folder, with the name “ProjectName-output.csv” and “ProjectName-summary.csv”, to store all model running results for analysis. The project name segment of the display will be enabled accordingly with the given project name displayed. The button of Set Parameters is enabled as follow. The screenshot of the Main Window, after Set Project Name button is clicked, is shown in Figure 11.

![Main Window (After Set Project Name)](image)

**Figure 11: Main Window (After Set Project Name)**
When the button of *Set Parameters* is clicked, the Parameter Selection Window, discussed in section 5.2.3, will appear on top of the Main Window, for user to select parameters and configure the system, as shown in Figure 14. Once these parameters been determined, they will be displayed on the Main Window, as well as been transformed and stored as variables/arrays of the instance of DDMSProject for future use. The button of *Build FMS* is enabled following these actions. The screenshot of the Main Window, after *Set Parameters* button is clicked and parameters are set, is shown in Figure 12.

![Figure 12: Main Window (After Set Parameters)]
The buttons of Build FMS, Build Sim Model, and Run Model are corresponded with the classes of Communication, SimModel and RunModel, respectively. They are displayed one after another as soon as the previous action is completed. Build FMS button creates an instance of communication, and generates a unique FMS system based on the configuration information obtained from both the input file and the instance of DDMSPProject, the parent class.

Similarly, Build Sim Model button creates an instance of SimModel, and generates a general simulation model based on the simulation model template of the given path, as well as information transferred from both parent class DDMSPProject and the FMS system generated in the previous action.

The button of Run Model triggers the functionality of DDMSPProject to loop through all simulation configurations based on the input information, and creates instances of RunModels one by one. The created instances pass the specific information of each model to the general simulation model generated in the previous action, to form finalized simulation models, and give the “Run” order.

Technically, these three buttons which handle the building and running of the system, can be combined into a single button with all functionalities built in. However, since the Integrated Decision Support Framework II is expected to be utilized as a research platform, the separation of “general system” building and “specific model building and running” is highly beneficial for researchers to verify that no error occurs in the “general system” building before full scale simulation run is initiated.
5.2.2 File Loading Window

File Loading Window, as shown in Figure 13, is designed for the user to select input data file and simulation template file. When the button of Select is clicked, only the corresponding file type would be displayed for user to choose, therefore minimize the chance of false file selection. The paths of these files would be stored as strings in the system for future activities to open the corresponding files, as soon as this window is closed.

![DialogLoadingFiles](image)

Figure 13: File Loading Window
5.2.3 Parameter Selection Window

The Parameter Selection Window, as shown in Figure 14 is designed for the user to set parameters, and configure the system as desired. Users can check the desired feature in checkboxes or type in numbers in textboxes to set up the system. As soon as the OK button is clicked, all selected parameters will be transfer and stored in variables/arrays within the instance of DDMSProject class, for future access. At least one parameter in each category has to be selected, otherwise a message box would appear to remind the user and prevent the window from closing. A “BatchRun” option is given at the end of selection, for user to decide if animation of the simulation is required. In large scale experiments, such animation can significantly increase the system time needed to run simulation, therefore this “BatchRun” option is unchecked as default.
5.3 The Application of Dissimilarity in Simulation Model

Similarity/Dissimilarity calculation has been implemented in Arena simulation model. Point of calculation is determined to correspond to the most realistic situation and it happens at the end of processing of features, at the end of each machine operation. Modules involved in Similarity/Dissimilarity are shown in Figure 15. The process of Similarity/Dissimilarity application is explained step by step as follow. Only modules related to Similarity/Dissimilarity are included.
Step 1

As soon as an operation is finished from the Process module, the part will move from the Process module to the Decide module, to decide if Similarity/Dissimilarity is the desired dispatching policy for the next job. Noticeably, although in simulation model this part has been moved forward through a variety of logical processes, since machine and resource have not yet been released, in real life, this operation has not yet been finished, and the part is still mounted on the machine.

Step 2

If Similarity/Dissimilarity is the dispatching policy for the system, Decide module will send the part through related process. In the Assign module, this part’s Part Type and the current Process Plan will be recorded as “current”, as a reference for the system to determine how similar each part waiting in queue is, compare to the “current” one. The Duplicate module creates a duplicated part as a “signal” to trigger the rearrangement of parts waiting in queue, while the original part follows the regular process to release the machine and resource and exit the system.

Figure 15: Similarity/Dissimilarity Modules in Simulation Model
Step 3

When the “signal” part arrives at Pickup module, all parts waiting in queue will be picked up. At the Dropoff module, the “signal” part will be disposed after accomplishing its responsibility, and all parts picked up in the previous action will be dropped off one by one, and be sent through the VBA module for Similarity Coefficient calculation and assignment.

Step 4

The VBA module calculates and assigns a similarity coefficient attribute to each part passing through. When parts reach the waiting queue at the Seize module, they will be accepted and arranged by the value of their similarity coefficient attribute. Since the higher the value, the more similar the part is to the “current” part, Similarity/Dissimilarity can be achieved by adjusting the sorting rule to “Highest Value” or “Lowest Value”. When these activities are finished, the “current” part will release the machine and resource, which marks the end of this operation in real life, and the part, will leave the machine heading for the next destination. Once the machine is released, the first part in queue will be picked up as the next job, which has already been sorted by previous activities as the most similar/dissimilar part.
6. EXPERIMENTATION

In this thesis, experimentation is designed for three purposes, which are 1) testing IDSF II and its running performance, 2) the effectiveness of Dissimilarity Coefficient Based Dispatching Policy, as well as 3) continued exploration of the effectiveness of various factors involved in FMS in terms of achieving the best system performance. To test the performance of the newly designed IDSF II, including the user interface, as well as other two purposes of this experimentation, a FMS system had been designed as follow.

6.1 FMS Design and Input Data

The tested FMS includes 8 machines connected by AGVs, which are CncDrillSlow, CncDrillFast, CncVMillSlow, CncHMmillSlow, CncVMillMid, CncHMmillMid, CncVMillFast, and CncHMmillFast. The layout of FMS configuration is shown in Figure 16.

Figure 16: FMS Configuration [12]
6 parts are to be produced by this FMS, including AES94, Bracket, NetEx, Plate, Slider, USC. Each part requires a number of operations that need to be performed on different machines. All parts and the number of features are shown in Figure 17.

<table>
<thead>
<tr>
<th>Name</th>
<th>AES94</th>
<th>Plate</th>
<th>Netex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td><img src="image1.png" alt="Model AES94" /></td>
<td><img src="image2.png" alt="Model Plate" /></td>
<td><img src="image3.png" alt="Model Netex" /></td>
</tr>
<tr>
<td>Number of features</td>
<td>10</td>
<td>11</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>USC</th>
<th>Slider</th>
<th>Bracket</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td><img src="image4.png" alt="Model USC" /></td>
<td><img src="image5.png" alt="Model Slider" /></td>
<td><img src="image6.png" alt="Model Bracket" /></td>
</tr>
<tr>
<td>Number of features</td>
<td>6</td>
<td>18</td>
<td>11</td>
</tr>
</tbody>
</table>

**Figure 17: Part Design [12]**

Detailed operations requirement of these parts as well as the capable machines to perform such operation is generated by Rule Based System [16] [17], and recorded into a .csv file as the input data of IDSF II. Screenshot of a portion of the input data is shown in Figure 18.
6.2 Running Parameters

In order to test the Dissimilarity Coefficient Based Dispatching Policy, and continuously explore the impacts of different factors on the system, the running parameters are chosen to meet these goals.

Three model types, including Best, Random and Dynamic Best, are chosen for the purposes of comparing their efficiency. Five Dispatching policies, FIFO, LIPT, SIPT, SPT and DCBDP are considered to compare and evaluate the effectiveness of Dissimilarity Coefficient Based Dispatching Policy in terms of increase in total machine utilization. Machine Utilization and Queue Size are the two routing policies (performance
criteria) implemented in the experiment for Dynamic Best model in terms of process plan selection. Batch sizes 10, 100 and 200 are considered before each decision making point, to study the potential impact of batch sizes in terms of total system efficiency. 10 replications are run for each model on a time scale of 2000 hours of warm up and 6000 hours of actual running, which is equal to the work load of one entire year with two shifts. Two groups of experimentation are run with part inter-arrival time 7 and 6 minutes, to verify the hypothesis of that FMS would have better total machine utilization with more loaded system. Since Routing Policy (Performance Criteria) is only valid for Dynamic Best Model, and Batch Size is not valid for Static Best, therefore the total number of simulation models built is 100. The selection of running parameters is shown in Table 8.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Model Type</th>
<th>Dispatching Policy</th>
<th>Routing Policy</th>
<th>Batch Size</th>
<th>Inter-arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Static Best, Random, Dynamic Best</td>
<td>FIFO, SPT, SIPT, LIPT, DCBDP</td>
<td>MU, Queue Size</td>
<td>10, 100, 200</td>
<td>6, 7</td>
</tr>
<tr>
<td>Total Model Built</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The screenshot of the selection of running parameters are shown in Figure 19.
Running Output

The results of simulation running are recorded by the IDSF II output module within each simulation model after each replication, into a .csv file. After all model runs are finished, all results are summarized into two .xlsx files ending with “-outputs” and “-summary”, where “-outputs” includes results from each replication, and “-summary” only contains the average figures from every simulation model.

For various analysis purposes, the recorded results include the following categories: Total Number Out of each part and the system, Total Working In Progress,
Machine Utilization for each machines and the average figure for the entire system, average Queue Size (number in queue) for each machine as well as the entire system. The running parameters of each simulation model are also recorded to identify specific combination of parameters. A number of analyses of the results are performed based on the recorded running results, to determine validity of the hypothesis of this research. The screenshot of a portion of the output file is shown in Figure 20.

![Figure 20: Screenshot of Output File](image-url)
7. ANALYSIS OF THE RESULT

7.1 Running Performance of IDSF II

One of the major contributions of this research is the development of IDSF II, including a much more efficient, flexible and integrated structure, as well as the redevelopment of the framework under Object Oriented Programming, by which further expansion of the framework becomes easier and leaner. The newly designed user interface for system set up is also a major improvement that needs to be tested in the experimentation.

The running of experimentation proved the performance of the newly designed IDSF II. The initial input data and simulation model template can be successfully loaded, and the selection or input of running parameters were performed without error occurring. Once the running of the experiment initiated, the framework start building simulation models automatically based on the combination of running parameters. When the running of one model is completed, the next simulation model automatically started to be built and run by the framework. The cycle continued until all model building and running were completed. Running results were exported successfully as designed into output files.

A number of errors occurred during the debugging stage, but they were successfully corrected.

The running performance proved the success of the redesigning of IDSF. More experimentation and further expansion for both industrial and academic purposes can be implemented based on the framework.
7.2 ANOVA Test

In this research, ANOVA test is performed to gain better understanding of the effect of each policy, as well as their combination, over the recorded system performance measures. Null hypothesis of the ANOVA tests is defined as no effect of single policy or their combination over the performance measure, while the acceptance of alternative hypothesis confirm the existence of such effect. When P-value < 0.05, null hypothesis is rejected and alternative hypothesis is accepted.

Since the policy of Batch Size is only valid for model type Random and Dynamic Best, and the Routing Policy is only valid for model type Dynamic Best, several ANOVA tests are performed to generate more accurate results.

<table>
<thead>
<tr>
<th>Source</th>
<th>Throughput</th>
<th>Total WIP</th>
<th>Avg. Total Waiting Time</th>
<th>Avg. Total Queue Size</th>
<th>Avg. Total Machine Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Type</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Dispatching</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Model Type*Dispatching</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The P-value for all measures and all model types are shown in Table 9. Since Batch Size and Routing Policy is not valid for all model types, only Model Type and Dispatching Policy, as well as their combination, are taken into consideration. As shown in the table, all P-values are equal to 0. Therefore we can reject the null hypothesis and
conclude that both Model Type and Dispatching Policy have statistically significant effect over all performance measures. The same conclusion is also valid for their combination.

Table 10: P-value for all measures (Model Type Random and Dynamic Best)

<table>
<thead>
<tr>
<th>Source</th>
<th>P-Value/Performance Measure</th>
<th>Throughput</th>
<th>Total WIP</th>
<th>Avg. Total Waiting Time</th>
<th>Avg. Total Queue Size</th>
<th>Avg. Total Machine Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Type</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.327</td>
</tr>
<tr>
<td>Dispatching</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Batch Size</td>
<td></td>
<td>0.5</td>
<td>0.937</td>
<td>0.321</td>
<td>0.937</td>
<td>0.024</td>
</tr>
<tr>
<td>MT*D</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MT*BS</td>
<td></td>
<td>0.836</td>
<td>0.978</td>
<td>0.598</td>
<td>0.978</td>
<td>0.107</td>
</tr>
<tr>
<td>D*BS</td>
<td></td>
<td>1</td>
<td>1</td>
<td>0.668</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

In order to include Batch Size as a factor in the ANOVA test, more ANOVA tests are performed for Model Type Random and Dynamic Best, since Batch Size is only valid for these two Model Types. The P-values for all measures in this scenario is shown in Table 10. For most measures except for the Averaged Total Machine Utilization under the effect of the combination of Dispatching Policy and Batch Size, the P-values are larger than 0.05. Therefore we can accept the null hypothesis and conclude that Batch Size does not have statistically significant effect over system performance.
Table 11: P-value for all measures (Model Type Dynamic Best)

<table>
<thead>
<tr>
<th>Source</th>
<th>P-Value/Performance Measure</th>
<th>Throughput</th>
<th>Total WIP</th>
<th>Avg. Total Waiting Time</th>
<th>Avg. Total Queue Size</th>
<th>Avg. Total Machine Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatching</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Batch Size</td>
<td>0.088</td>
<td>0.001</td>
<td>0.088</td>
<td>0.258</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Routing</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D*BS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D*R</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BS*R</td>
<td>0</td>
<td>0.16</td>
<td>0</td>
<td>0</td>
<td>0.54</td>
<td></td>
</tr>
</tbody>
</table>

However, the next scenario unveils that Batch Size does have effect under the Model Type of Dynamic Best. The P-values for all measures under Dynamic Best model type is shown in Table 11. From the values in the table, it can be concluded that Batch Size does have effect over Throughput (Total Parts Out), which is the most important measure of system performance. When combining Batch Size and Routing Policy or Dispatching Policy, the same conclusion can also be draw from the test results. Other effect of Batch Size, as well as its combination with other policies are also shown in the table with P-values smaller than 0.05.

7.3 Analysis of the Effects of FMS Policies

In previous research [12], a number of experimentation and analyses have been performed to determine the effects of different policies included in the system. Based on the results achieved, more experiments and analysis are implemented in this thesis to further explore the effects of various FMS policy parameters, including Model Types,
Dissimilarity Coefficient Based Dispatching Policy, Batch Size, Routing Policy, and Inter-Arrival Time. Results of these studies are explained as follow.

### 7.3.1 Model Type Analysis

Three Model Types, Best, Random, Dynamic Best, are included in the designed experimentation for the purpose of performance comparison. With inter-arrival time of 7 minute, the result of performance in terms of total part out, total working in progress, average total waiting time, total queue size, average total machine utilization, are shown in Figure 21, Figure 22, Figure 23, Figure 24, Figure 25, respectively. The charts on the left side separate the results by not only model types, but also dispatching policies, while the charts on the right side show the average number for all dispatching policies for each model type.

![Figure 21: Total Part Out of Model Types](image)

Figure 21: Total Part Out of Model Types
Figure 22: Total WIP of Model Types

Figure 23: Average Total Waiting Time of Model Types

Figure 24: Total Queue Size of Model Types
The average results for each model type clearly showed that in all categories of measurement, Dynamic Best performed significantly better than Random, and Random produced better results than Best. The results separated by dispatching policies also verified this conclusion, with the same performance comparison under every dispatching policy, with the exception of Random model type had slightly better machine utilization performance than Dynamic Best model type under SIPT dispatching policy. Quantified improvement by model types is summarized in Table 12.

### Table 12: Performance Improvement by Model Type

<table>
<thead>
<tr>
<th></th>
<th>Best</th>
<th>Random</th>
<th>Dyn.</th>
<th>Random Improve over Best</th>
<th>Dynamic Improve over Best</th>
<th>Dynamic Improve over Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Part Out</td>
<td>17372</td>
<td>26804</td>
<td>33858</td>
<td>54%</td>
<td>95%</td>
<td>26%</td>
</tr>
<tr>
<td>Total WIP</td>
<td>16753</td>
<td>7582</td>
<td>504</td>
<td>55%</td>
<td>97%</td>
<td>93%</td>
</tr>
<tr>
<td>Avg. Total Waiting Time</td>
<td>5830</td>
<td>2460</td>
<td>513</td>
<td>58%</td>
<td>91%</td>
<td>79%</td>
</tr>
<tr>
<td>Total Queue Size</td>
<td>16751</td>
<td>7578</td>
<td>500</td>
<td>55%</td>
<td>97%</td>
<td>93%</td>
</tr>
<tr>
<td>Avg. Total MachUti</td>
<td>24%</td>
<td>49%</td>
<td>49%</td>
<td>101%</td>
<td>102%</td>
<td>0%</td>
</tr>
</tbody>
</table>
This study also improved the conclusion from previous research [12], where the advantages of Dynamic Best model over Random model were not significant. The reason for that insignificance was believed to be an error of variable type definition in the previous model. Discovered in this research, the error could cause the Dynamic Best system to choose the same process plan for a specific part, throughout the running. This failure of selecting alternative plans could cause significant system conjunction, therefore negatively impact the total performance of Dynamic Best model type. Since the error is corrected in the redeveloped IDSF II, Dynamic Best model type performed better than Random model type as expected.

7.3.2 Single Machine Utilization Analysis

In order to have a better understanding of the running status of the FMS, analysis of single machine performance was performed. The result of single machine utilization for every dispatching policy, under the model type of Dynamic Best, is shown in Figure 26.
Figure 26: Single Machine Utilization for Dynamic Best Model Type

From the chart, an obvious fact can be learned, which is that the machine of CncHMillFast was always loaded with operations, in any model under Dynamic Best model type, despite that one of the major purposes of such model type is to avoid this very scenario.

One of the possible explanations for this scenario is unbalanced system load. All operations on parts were determined and assigned to certain types of machines by rule based system. This process was separated from the design of FMS, or, to be specific, the number of machines required in FMS to accommodate the load. Therefore, the specific load of parts used in this experimentation may require large amount of milling process to manufacture. Under this scenario, although 4 alternative plans were generated, this CncHMillFast machine may have to be included in all alternative plans because of the amount of milling process required.
The consequence of this unbalanced work load is that the effectiveness of all other policies may be compromised. If every part has to move through the bottle neck and be significantly delayed, the appearance of the differences of other policies would be reduced.

7.3.3 Performance of Dissimilarity Coefficient Based Dispatching Policy

Dissimilarity Coefficient Based Dispatching Policy is included in the experiment, along with other 4 dispatching policies for comparison. The results of average system performance are shown in Figure 27. Only Dynamic Best model type was studied because of its high responsiveness. The inter-arrival time in this scenario is 6.

Figure 27: Average System Performance of Dispatching Policies
The performance results showed that DCBDP did not yield significantly higher total output or total machine utilization. It also performed average level in terms of average total waiting time. For total queue size, DCBDP did have shorter queue than FIFO and LIPT, but SIPT and SPT had similar performance to DCBDP.

The possible reason for the lower performance than expectation could be due to the unbalanced work load explained in the previous section. For instance, DCBDP is expected to produce higher total machine utilization because the part in queue with least similar route, compare to the previous part, would be selected. But if one machine is 100% loaded throughout the running, the improvement in machine utilization on other machines would be evened compare to other dispatching policies. More experimentation is required to be effectively implemented after the system being adjusted to a more balanced status.

7.3.4 Routing Policy Analysis

Two routing policies, one based on total machine utilization while the other based on total queue size, are designed for Dynamic Best model type as the determination of the selection of process plans. The performance of these two routing policies, with respect to total part out, total working in progress, average total waiting time, total queue size, and average total machine utilization, are shown in Figure 28, Figure 29, Figure 30, Figure 31, and Figure 32, respectively. In this case, the inter-arrival time was set as 6.
Figure 28: Total Parts Out of Routing Policies

Figure 29: Total WIP of Routing Policies

Figure 30: Average Total Waiting Time of Routing Policies
The results shown in figures clearly indicate that when total queue sizes are considered as the performance criteria in routing selection, the overall system performance receives significant improvement. Results under specific dispatching policies also verified this conclusion, where Queue Size routing policy performs better in every measurement under all dispatching policies, with the exception that LIPT produce smaller average total waiting time under the routing policy of Machine Utilization. Quantified improvement is shown in Table 13.
Table 13: Performance Improvement by Routing Policy

<table>
<thead>
<tr>
<th></th>
<th>MU</th>
<th>QueueSize</th>
<th>QueueSize Improve over MU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Parts Out</td>
<td>34231.14</td>
<td>37046.31</td>
<td>8%</td>
</tr>
<tr>
<td>Total WIP</td>
<td>5708.06</td>
<td>3209.39</td>
<td>44%</td>
</tr>
<tr>
<td>Avg. Total Waiting Time</td>
<td>3775.07</td>
<td>2231.75</td>
<td>41%</td>
</tr>
<tr>
<td>Total Queue Size</td>
<td>5703.81</td>
<td>3204.53</td>
<td>44%</td>
</tr>
<tr>
<td>Avg. Total MachUti</td>
<td>51%</td>
<td>58%</td>
<td>14%</td>
</tr>
</tbody>
</table>

7.3.5 Inter-Arrival Time Analysis

This experimentation included two groups of experiments with different part inter-arrival time, 6 and 7 minutes. The hypothesis is as follow. For smaller inter-arrival time, the system is supposed to be more loaded. Therefore FMS should be able to further increase the total machine utilization by distribute more parts to other capable machines besides the best machine choice. The result of average total machine utilization is shown in Figure 33. Both Random and Dynamic Best model types are evaluated since they both contain functionalities to utilize alternative process plans.
The result shown in the chart provided sufficient evidence for the hypothesis to be accepted. Quantified improvement is shown in Table 14. The average total machine utilization of 6 minutes inter-arrival time increased by 9% under Random model type, compare to 7 minutes. Dynamic Best model shows 12% increase.

Table 14: Machine Utilization Increase by Inter-arrival Time

<table>
<thead>
<tr>
<th></th>
<th>7 Inter-Arr Time</th>
<th>6 Inter-Arr Time</th>
<th>Increase of 6 over 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>49%</td>
<td>53%</td>
<td>9%</td>
</tr>
<tr>
<td>Dyn</td>
<td>49%</td>
<td>54%</td>
<td>12%</td>
</tr>
</tbody>
</table>
7.3.6 Batch Size Analysis

Three Batch Sizes, 10, 100 and 200, are included in the experimentation. The hypothesis of the included batch size study is that, when decision of routing selection is made on shorter time interval, such decision should be more effectively reflective of the real-time system status. Based on the experiment results, Batch Size is confirmed to have significant effect over Total Queue Size, when combined with Queue Size as routing policy. The results of Total Queue Size of different Batch Sizes, divided by Dispatching Policies, are shown in Figure 34. In this scenario, Model Type, Routing Policy and Inter-Arrival Time are set as Dynamic Best, Queue Size, and 7 minutes, respectively.

As shown in the figure, simulation models with Batch Size 10 have much shorter queues compare to larger Batch Size. Quantified improvement is shown in Table 15.
### Table 15: Total Queue Size Decrease by Batch Sizes

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>100</th>
<th>200</th>
<th>Decrease of 10 over 100</th>
<th>Decrease of 10 over 200</th>
<th>Decrease of 100 over 200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dissim</td>
<td>19.45</td>
<td>113.23</td>
<td>310.92</td>
<td>83%</td>
<td>94%</td>
<td>64%</td>
</tr>
<tr>
<td>FIFO</td>
<td>18.42</td>
<td>138.96</td>
<td>368.24</td>
<td>87%</td>
<td>95%</td>
<td>62%</td>
</tr>
<tr>
<td>LIPT</td>
<td>26.27</td>
<td>212.47</td>
<td>433.61</td>
<td>88%</td>
<td>94%</td>
<td>51%</td>
</tr>
<tr>
<td>SIPT</td>
<td>23.39</td>
<td>126.65</td>
<td>167.25</td>
<td>82%</td>
<td>86%</td>
<td>24%</td>
</tr>
<tr>
<td>SPT</td>
<td>20.56</td>
<td>120.96</td>
<td>316.69</td>
<td>83%</td>
<td>94%</td>
<td>62%</td>
</tr>
</tbody>
</table>
In this research, IDSF II is successfully designed and implemented under Object Oriented Modeling. This major improvement enables more flexibility in terms of simulation model building. Additional theories, policies and concepts can be introduced into the framework, such as including operators into the system to reduce the differences between simulation model and reality. System failure can also be included for the same purpose. By adjusting or adding objects into the system, more routing policies, dispatching policies and etc., can be tested by simulation. The design of user interface in this research is also beneficial in future expansion of the system.

Similarity Coefficient is included in IDSF II as an additional dispatching policy to compare with Dissimilarity Coefficient as well as other dispatching policies. However, due to unrealistically high output number, the result is not reported in this thesis. The reason for such error is expected to be certain programming issues. More debugging work can be performed to solve potential problems and expand the experimentation to consider similarity coefficient as dispatching policy.

Certain results in this research are compromised by unbalanced system load. Addressing this issue in future research would lead to better understanding about the hypothesis made in previous work. In general, a more dynamic FMS, which includes larger number of machines, parts, and number of alternative plans, is expected to reduce the probability of outliers, therefore increase the effectiveness of simulation study, and better serve academic experimentation and industrial implementation.
9. REFERENCES


