A HYBRIDIZED APPROACH FOR SOLVING GROUP SHOP PROBLEMS (GSP)

TAN MU YEN

NATIONAL UNIVERSITY OF SINGAPORE

2005

A HYBRIDIZED APPROACH FOR SOLVING GROUP SHOP PROBLEMS (GSP)

TAN MU YEN
(B.Eng (Hons), NUS)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF INDUSTRIAL & SYSTEMS ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2005

Acknowledgement

I would like to express my most sincere gratitude to my supervisors, A/Prof Ong Hoon Liong and Dr Ng Kien Ming, for providing me with the opportunity to work on this project and for introducing me to the world of machine scheduling. While they have given autonomy in this research study, they were very enthusiastic and helpful in providing the much treasured support in dealing with both academic and administrative issues. Their patience as well as guidance throughout the project has benefited me significantly.

Table of Contents

List of Symbols

COP	Combinatorial optimization problem
P	Polynomial-time verifiable problem
NP	Non-deterministic polynomial-time verifiable problem
PLS	Polynomial-time local search problem
FSP	Flow shop problem
JSP	Job shop problem
MSP	Mixed shop problem
OSP	Open shop problem
GSP	Group shop problem
n	Number of jobs in the shop scheduling problem
m	Number of machines in the shop scheduling problem
M_j	A specific machine in the shop scheduling problem
J_i	A specific job in the shop scheduling problem
n_i	Number of operations in job J_i
O_{ij}	Operation from job J_i that is processed on machine M_j
p_{ij}	Processing time of operation O_{ij}
r_i	Release date of job J_i; time when the first operation of J_i becomes available for processing

μ_{ij}	Set of machines that is associated with operation O_{ij}
$f_i(t)$	Cost function that computes the cost of completing job J_i at time t
d_i	Due date of J_i; committed completion time of job J_i
w_i	Priority factor denoting the importance of job J_i relative to other jobs in the system
C_i	Completion time of job J_i
$\alpha\|\beta\|\gamma$	Graham's three field machine scheduling classification system
α	Parameter that specifies machine environment
β	Parameter that specifies job characteristics
γ	Parameter that specifies optimality criterion
$G=(V,C,D)$	Disjunctive graph representation that consists of a node set V, conjunctive arc set C and disjunctive arc set D
V	Set of nodes on the disjunctive graph that represents all the operations in the scheduling problem
C	Set of directed conjunctive arcs which reflect the precedence relations between the operations
D	Set of disjunctive arcs which are used to present disjunctive constraints that arises naturally in machine scheduling
$\succ\succ$	Next-follow relation specifies the relationship between two

	operations in a schedule
O	Set of finite operations in a shop scheduling problem
ψ_j	Set of operations which have to be processed on machine M_j
ξ_i	Set of operations which belong to job J_i
$(o,o') \in \preceq$	Partial order that specifies that the processing of operation o has to be completed before the processing of operation o' can begin
$\preceq_{pred}(o)$	Set of predecessors of an operation o
$\preceq_{succ}(o)$	Set of successors of an operation o
$m(o)$	Function denoting the machine on which an operation o has to be processed on
$j(o)$	Function denoting the job on which an operation o belongs to
$g(o)$	Function denoting the group which an operation o belongs to
$C_{\max}$	Maximum completion time or makespan
Γ	Set consisting of groups of operations
$N(s)$	Set of neighborhood moves of schedule s
$\overline{N(s)}$	Set of schedules $s \in N(s)$ that satisfies a pre-defined acceptance and admission criteria such that $\overline{N(s)} \subseteq N(s)$
$M[i]$	The machine that processes operation i.

$PM[i]$	The operation processed on $M[i]$ just before operation i, if it exists.
$SM[i]$	The operation processed after $M[i]$ just after operation i, if it exists.
$J[i]$	The job to which operation i belongs to.
$PJ[i]$	The operation belonging to job $J[i]$ that precedes operation i, if it exists.
$SJ[i]$	The operation belonging to job $J[i]$ that follows operation i, if it exists.
$G[i]$	The group to which operation i belongs to.
$PG[i]$	The operation belonging to group $G[i]$ that precedes operation i, if it exists.
$SG[i]$	The operation belonging to job $G[i]$ that follows operation i, if it exists.
e_i	Length of a longest path from node 0 to node i, excluding p_i
l_i	Length of a longest path from node i to node $N+1$, excluding p_i
S_P	Partial Schedule
O^+	List of unscheduled operations
t_{ec}	Earliest completion time of operation
t_{es}	Earliest start time of operation
S_C	Current schedule
S_{LB}	Local Best Schedule
S_{GB}	Global Best Schedule
$TList$	Tabu List

LB	Lower bound of shop scheduling problem
TS	Tabu search algorithm
SA	Simulated Annealing
GA	Genetic Algorithm
VNS	Variable Neighborhood Search
GLS	Guided Local Search
EC	Evolutionary Computation
ACO	Ant Colony Optimization
$\alpha(T)$	The geometric temperature function of simulated annealing method.
MaxRestart	The maximum number of iterations before a restart is initiated in the GSP scheduler during the diversification phase.
MaxBacktrackPoints	The maximum number of solutions that will be stored in the backtrack memory of the GSP Scheduler.
MaxDivIterations	The maximum number of iterations allowed in the diversification phase of the GSP Scheduler.
MinPercentDev	The minimum percentage deviation from the lower bound solution.
MaxIntIterations	The maximum number of iterations allowed for each backtracked solution in the intensification phase of the GSP Scheduler.
TBListLen	The tenure of the Tabu List in the GSP Scheduler.
NCIterations	The maximum number of iterations allowed before a neighborhood structure change in the diversification phase of the GSP Scheduler.
S_{BST}	The best solution obtained for the computational runs of a particular problem instance.

S_{AVG}	The average solution value obtained for all the computational runs of a particular problem instance.
T_{AVG}	The average computational time required to obtained the final solution for the computational runs of a particular problem instance.
c_v	The coefficient of variation of the solutions obtained through the computational runs for a particular problem instance.

List of Figures

List of Tables

Abstract

The objective of shop scheduling problems is to determine the optimal allocation of machines to jobs with respect to some specified criteria. As these problems have been commonly acknowledged as being difficult to solve, previous research efforts has focused mainly only on developing customized approaches for each of these classes of problems. However, in recognition of the prevalence of machine scheduling problems as well as industries' need for a single and robust algorithm for the differing scheduling scenarios, this thesis addresses the application of meta-heuristics approaches to tackle a generalized formulation of shop scheduling problems known as the Group Shop Problem (GSP) by developing a hybridized approach.

The proposed scheduling approach consists of two main phases, namely: the diversification phase and the intensification phase. In the diversification phase, the proposed algorithm incorporates features of simulated annealing and variable neighborhood search to diversify its search. Additionally, the algorithm adopts the use of tabu-lists from Tabu Search throughout to prevent cyclical search from arising. Backtrack memories are also implemented to store promising solutions that are found during the initial phase so that the search during the intensification phase will be limited to only these promising regions of the search space.

To evaluate its performance, the algorithm has been subjected to extensive computational experiments using a set of benchmark problems for comparison with other known approaches for solving GSP. Among many benchmark problems used, the famous WHIZZIKD97 group shop problem has also been included for the experiment. The empirical results show that the proposed algorithm produces solutions of comparable quality but with shorter processing time.

Chapter 1 Introduction

1.1 Overview

Scheduling is the science and art of allocating finite and scarce resources over time to perform a collection of tasks in a variety of situations, with differing resource capacities and technological constraints, so as to optimize one or more pre-defined objectives. While there was considerable research interest in this field at the beginning of the twentieth century with the works of prominent manufacturing pioneers such as Henry Gantt, it took many years for the first scheduling publications to appear in the industrial engineering and operations research literature. Since problems arising from manufacturing were the main source of motivation for the early development in the field of scheduling, the vocabulary of manufacturing was employed when describing scheduling problems. Thus, resources were usually denoted as machines and basic task modules were termed as jobs. In scenarios where jobs may consist of several elementary tasks that are interrelated by precedence constraints, such elementary tasks are referred to as operations.

The voluminous amount of related research results since 1950s, including Johnson (1954), have culminated in a more definitive scheduling theory, which embodies numerous mathematical models to characterize the various classes of

scheduling problems that range from those that involve single-stage models[1] to those that involve multi-stage models[2], from those of a deterministic nature to those of a stochastic nature and from those that are concerned with single objective optimization to those that are concerned with multiple objective optimization.

In the broader context, scheduling problems belong to a larger problem class, known as Combinatorial Optimization Problems (COPs), which are concerned with determining the "best" configuration from a set of parameters to achieve some pre-defined goals. Usually, the objective of COPs is to locate an entity, which can be an integer, a subset, a permutation or a graph structure, from a finite or possibly countable infinite set. (See Papadimitriou and Steiglitz, 1982). An important aspect of COPs is to determine its solvability. In particular, the landmark study by Karp (1972), on Computational Complexity Theory, demonstrated that many of the most commonly studied optimization problems can be reduced to a single underlying problem of known computational complexity.

Central to the theory of Computational Complexity, NP-completeness provides the required formalization to differentiate the easy problems from the difficult problems. In essence, there are two basic classes of problems namely: class P of tractable problems and class NP of *polynomial-time verifiable* problems. The class P is the class of decision problems that can be solved by a polynomial-time

[1] Single-stage model refers to model with either a single machine or a number of parallel machines.
[2] Multi-stage model is synonymous with shop scheduling models. Like single-stage model, every stage in the multi-stage model may consist of either a single machine or a number of parallel machines. However, the number of machines in each stage should be the same.

algorithm while the latter consists of those problems that can be solved by a non-deterministic polynomial-time algorithm. Within the Class NP, NP-Complete problems are the most difficult problems. At present, all known algorithms for NP-complete problems require time that is not bounded by a polynomial function of the problem's input size. See Papadimitriou (1993). Moreover, most COPs, in general, are difficult to solve in nature.

As research works on scheduling in 1970s were strongly influenced by the work of Karp (1972), the difficulty of scheduling problems can be gleaned from the complexity status of such problems as reported in works such as Applegate and Cook (1991), and Brucker (1998). Earlier notable works include Lenstra *et al*. (1977), and Lenstra and Rinnooy (1979), which focus mainly on the complexity hierarchy of scheduling problems. Through these works, it is becoming increasingly clear that except for rare cases where polynomial time algorithms are available to solve the specific problems to optimality, most scheduling problems are NP-hard in the ordinary sense or strongly NP-hard. Despite the substantial amount of research directed to complexity study, there remains scheduling problems whose computational complexities have yet to be ascertained.

Earlier scheduling techniques focused on finding exact solutions via the application of enumerative algorithms with elaborate and sophisticated mathematical constructs. Particularly, the Branch and Bound technique, which searches a dynamically constructed tree representing the solution space of all feasible schedule, is the main enumerative technique. However, the general

limitations of these enumeration techniques coupled with the results of complexity studies on scheduling problems prompted the search for better scheduling algorithms. By the end of 1980s, the use of approximation methods emerged as the next viable alternative. Such methods typically forego guarantees of an optimal solution for gains in speed. The earliest approximation algorithms made use of priority rules to assign priorities to all the operations which are available to be sequenced and then choose the operation with the highest priority for the schedule construction. (See Panwalkar and Iskander, 1977). Despite its ease of implementation and its low computational demand, these algorithms were not effective in generating quality solutions especially for problems of high dimensionality.

The need for better approximation algorithms fueled the development of many innovative techniques, including but not limited to Large Step Optimization (Martin *et al.*, 1992), Tabu Search (TS) (Glover, 1989 and Glover, 1990) and Simulated Annealing (SA) (Van Laarhoven *et al.*, 1989), to bridge the basic gaps found in those algorithms based on priority dispatch rules. These innovative algorithms, which combine basic heuristic methods in higher level frameworks aimed at exploring search space, are also known as meta-heuristics. Today, meta-heuristics are almost a *de facto* method for solving scheduling problems.

Research efforts in the field of scheduling will continue to remain relevant, if not more important, given the recent trends in both the manufacturing and services industries. See Ashby and Uzsoy (1995), and Pinedo (2002). In particular, shop

scheduling formulations represent theoretical efforts to simplify models of scheduling problems often arising in industrial settings. The popularity of such models has led to the rapid growth of the shop scheduling research literature. Typically, a shop scheduling problem will consist of n jobs with operations to be scheduled on m machines. Depending on the nature of the problem, there may or may not be precedence relationship between the operations on each job.

Though the previous decades of research have availed a compendium of both exact and approximate scheduling methods attuned to solving specific problems, the differing characteristics of the various shop scheduling problems and specialized nature of most methods do not facilitate easy adaptation for more generic applications. For example, a successful approach to tackle a particular class of job scheduling problem may not work very well when modified to tackle another class. Considering the prevailing industrial trends, an algorithm that is robust and works well on a wide range of shop scheduling problems will be most desired. This study focuses on the general shop scheduling problem called Group Shop Problem (GSP) first coined in Sampels *et al.* (2002).

1.2 Motivation Factors

The motivating factors for the present research proposal can be discerned from the following perspectives:

a. Firstly, it has been spurred by the increasing importance of scheduling functions in both manufacturing and service sectors. Contextual changes in these arenas have been evidently marked by both paradigm shifts in Supply Chain Management (SCM) models and technological improvements such as Flexible Manufacturing System (FMS) and Enterprise Resource Planning (ERP) systems. (See Handfield and Nichols, 2002). Along with these changes, business planners today face greater challenges in deciphering information and making decisions. Specifically, manufacturing planning and transportation scheduling, which are two key areas in SCM, will benefit from advances in scheduling methodology. The emergence of the various shop scheduling models and the continual development of associated solving strategies represent significant efforts undertaken by researchers not only to relieve business planners of the burden of performing the traditional secondary role of scheduling but also to give them additional leverage in operations management.

b. Secondly, from an academic perspective, scheduling is one of the fundamental areas of combinatorial optimization, and shop scheduling problems has been commonly acknowledged for being hard to solve optimally. Traditionally, research efforts in shop scheduling have been delineated into Flow Shop Problems (FSP) (Johnson, 1954), Job Shop Problems (JSP) (Fisher and Thompson, 1963), Mixed Shop Problems (MSP) (Masuda *et al.*, 1985) and Open Shop Problems (OSP) (Rock and

Schmidt, 1983). This division of research efforts has resulted in a myriad set of customized techniques that will perform well on a particular shop scheduling problem but will show unsatisfactory results when applied to other shop scheduling problems. Since Group Shop Problem (GSP) is a generalization of the classical JSP, MSP and OSP, investigation into its properties will likely lead to a generalized approach to solving the various classes of shop scheduling problems and thus meeting the industries' need for single and robust algorithm for the differing scheduling scenarios.

Advances in the design of scheduling algorithm design will also shed new insights into how solving strategies for other COPs, such as Traveling Salesman Problem (TSP) (Lawler *et al*., 1985) and Vehicle Routing Problem (VRP) (Laporte, 1991), can be enhanced. With better understanding of these approximate methods, better meta-heuristics can be developed.

1.3 Objectives and Scope

Given the generality of the GSP formulation, it is unlikely that the new algorithm will reach the performance of the state-of-the-art meta-heuristics approaches for more specific shop scheduling problems, which tend to be more restricted in problem definition. Therefore, the primary aim of this research is to develop an algorithm that is both scalable in its applications and robust in its performance over a wide range of GSP instances. To facilitate the design of a new GSP

scheduling algorithm, a comparative study of existing meta-heuristics will be essential.

The collection of benchmark problem instances for comparative analysis will also be an important task in this study to circumvent situations where good performance results are achieved due to coincidence. Presently, there are already many benchmark problem instances available for JSP and OSP to allow JSP and OSP instantiations of the GSP formulation to be tested out by the various approaches. Since GSP is a relatively new scheduling problem, the consolidation of "true" and "good" GSP benchmark instances will be challenging.

While dynamic[3] and stochastic versions of shop scheduling formulations show higher degree of industrial relevance (Righter, 1994, and Floudas and Pardalos, 2001), current research will only focus on deterministic GSP formulation since research in GSP is still in its infancy stage of development. Likewise, parallel computing implementation, multiple objectives optimization and parallel machines environments formulations will not be explored in this thesis. Rather, the focus will be on non-parallel implementation of a GSP scheduling algorithm for makespan optimization in single machine environment.

[3] Dynamic Scheduling is sometimes known as Reactive Scheduling.

1.4 Thesis Outline

Having dealt with the introduction of this research, which forms the first chapter, the organization of the subsequent chapters is as follows: Chapter 2 covers the theoretical background on COPs, meta-heuristics as well as the various deterministic models of shop scheduling problems and the prevalent methods for solving them. Following that, the approach and rationale for the design of the scheduling algorithm for GSP will be outlined in Chapter 3. The computational results and analysis will be addressed in Chapter 4. Finally, Chapter 5 concludes this thesis by summarizing the specific research issues that have been dealt with and also highlighting possible directions for future research.

1.5 Research Contribution

This study demonstrates the feasibility of devising an algorithm that is both scalable in its applications and robust in its performance on a wide range of GSP instances. Moreover, this study has shown that it is possible to devise a good scheduling algorithm that is easy to implement and yields solutions of good quality in a reasonable amount of time. This is illustrated through comparison with the computational results of other known approaches for solving GSP problems.

In the literature, most researchers tend to focus on making tactical improvements to existing meta-heuristics for solving specific shop scheduling problems. While

the implementation of good neighborhood definitions and extensive memory structures in search algorithms are important, it is imperative that researchers do not lose sight of the underlying features of the problem that they are solving. Therefore, this study attempts to construct an algorithm that aligns its search strategy based on known results about the search space of GSP and to incorporate an array of existing techniques from known meta-heuristics into the algorithm so as to achieve maximum effectiveness. The result of this is a hybridized approach for solving GSP.

Chapter 2 Literature Survey on Shop Scheduling

2.1 Overview

The significant amount of research efforts in the field of deterministic scheduling over the past four decades have led to the growth of scheduling models and related solving strategies. Given the astounding number and variety of scheduling models, a quick exposition of the entire scheduling landscape is not an easy task. However, this chapter attempts to create clarity for understanding the pertinent issues related to deterministic scheduling by elucidating the necessary theoretical foundations as well as key findings from existing research literature on shop scheduling. In particular, the basic scheduling framework and its related notation will also be briefly discussed. This will be followed by a general introduction to the various shop scheduling models, the disjunctive graph representation and the different types of schedules. An overview of local search techniques, meta-heuristics as well as the concept of fitness landscape will also be provided herein to establish the relevant context for an outline on the known approaches for shop scheduling. Finally, the topic on common neighborhood definition will serve as the concluding section.

2.2 Basic Framework and Notation

Common scheduling terminology makes a distinction between a sequence and a schedule. While a sequence is a permutation of a set of jobs on a given machine, a schedule consists of both the sequencing of jobs in time and the allocation of finite resources to the appropriate jobs within a machine setting, allowing for possible preemptions of jobs by other jobs that are released at later points in time. Similarly, the term scheduler is also differentiated from the term scheduling policy. Usually, a scheduler corresponds to an algorithm performing the function of generating schedules. On the other hand, a scheduling policy is a rule or a set of operating principles that prescribes the actions for a scheduler that is best suited to the current state of a typically stochastic system.

In all scheduling problems, the number of jobs and machines are assumed to be finite. Typically, m machines $M_j\left(j=1,...,m\right)$ have to process n jobs $J_i\left(i=1,...,n\right)$. A job J_i consists of a number n_i of operations $O_{i1},...,O_{in_i}$ with each operation O_{ij} being assigned a processing requirement p_{ij}. If job J_i has only one operation $\left(n_i=1\right)$, J_i can be identified as O_{i1} with processing requirement of p_i. Sometimes, a release date r_i, on which the first operation of job J_i becomes available for processing, may be specified.

Furthermore, each operation O_{ij} is associated to a set of machines $\mu_{ij} \subseteq \{M_1,...,M_m\}$. In a dedicated machine environment, all μ_{ij} are one element sets. On the contrary, all μ_{ij} are sets equal to the set of all machines in a parallel machine environment and this allows problems in flexible manufacturing, where machines are equipped with different tools, to be formulated. Problems of this type are termed as scheduling problems in multi-purpose machine environments, where an operation can be processed on any machine equipped with the appropriate tool. As for multi-processor task scheduling problems, all machines in the set μ_{ij} are used simultaneously by O_{ij} during the entire processing period.

A cost function $f_i(t)$ is commonly included in the problem formulation to determine the cost of completing job J_i at time t. In many cases, a due date d_i, which represents the committed completion time of job J_i, and a weight w_i, which is a priority factor denoting the importance of job J_i relative to other jobs in the system, are used in defining $f_i(t)$.

Given the wide span of problem formulations subsumed under the general theory of scheduling, a comprehensive classification scheme will be essential. The three field $\alpha|\beta|\gamma$ classification system, which was introduced by Graham ***et al.*** (1979), is one such scheme that provides the basic notations required to characterize most scheduling problems in terms of machine environment α, job characteristics β

and optimality criterion γ. Brucker (1998) provides a systematic and detailed classification of scheduling problems.

The machine environment is characterized by a string $\alpha = \alpha_1 \alpha_2$ of two parameters such that $\alpha_1 \in \{\circ, P, Q, R, PMPM, QMPM, G, J, F, O, X\}$ is used to specify machine-operation models with $\circ$ denoting the empty symbol and $\alpha_2 \in Z^+$ is used to indicate the number of machines in this system. An overview of the possible parameter values of α_1 for specifying the various Machine-Operation models is as follows:

Table 2.1: Parameters for Specifying Machine-Operation Models

Parameters	Characteristics
$\alpha_1 = \circ$	• Each job must be processed on a dedicated machine.
$\alpha_1 \in \{P, Q, R\}$	• Each job can be processed on each of the machine. • $\alpha_1 = P$ for identical parallel machine environment where processing time p_{ij} of job J_i on M_j is equal to the processing time p_i of job J_i for all machines M_j. • $\alpha_1 = Q$ for uniform parallel machines environment where processing time p_{ij} of job J_i on M_j is equal to p_i / s_j with s_j specifying the speed of machine M_j for all machines M_j. • $\alpha_1 = R$ for unrelated parallel machines environment where processing time p_{ij} of job J_i on M_j is equal to p_i / s_{ij} given job-dependent speeds s_{ij} of M_j.
$\alpha_1 \in \{PMPM, QMPM\}$	• $\alpha_1 = PMPM$ and $\alpha_1 = QMPM$ denote multi-purpose machines with

	identical speeds and multi-purpose machines with uniform speeds respectively.
$\alpha_1 \in \{G, J, F, O, X\}$	• System is made up of a set of dedicated machines i.e. all μ_{ij} are one-element sets and a collection of jobs with each job J_i consisting of a set of operations $O_{i1},...,O_{in_i}$. • $\alpha_1 = G$ denotes the Group Shop Model where there are precedence relations between arbitrary operations. • $\alpha_1 = J$ denotes the Job Shop Model where precedence relations is of the form $O_{i1} \to O_{i2} \to ... \to O_{in_i}$ for $i = 1,...,n$ such that $\mu_{ij} \neq \mu_{i(j+1)}$ for $i = 1,...,n-1$. When $\mu_{ij} = \mu_{i(j+1)}$, the model will be labeled as Job Shop with Machine Repetition. • $\alpha_1 = F$ denotes the Flow Shop Model, which is a special case of Job Shop Model, where $n_i = m$ for $i = 1,...,n$ and $\mu_{ij} = \{M_j\}$ for $i = 1,...,n$ and $j = 1,...,m$. If jobs in a Flow Shop model are processed in the same order on each machine, then it is known as a Permutation Flow Shop model. • $\alpha_1 = O$ denotes the Open Shop Model, which is basically a Job Shop Model with the exception that there is no precedence relations between operations. • $\alpha_1 = X$ denotes the Mixed Shop Model, which is a combination of a Job Shop Model and an Open Shop Model.

On the other hand, the job characteristics are specified by a set β containing at most six elements $\beta_1, \beta_2, \beta_3, \beta_4, \beta_5$ and β_6. The tabulation below provides a brief summary of these parameters:

Table 2.2: Parameters for Specifying Job Characteristics

Parameters	Characteristics
β_1	• $\beta_1 = pmtn$ indicates that **preemption** (or job splitting) is allowed.
β_2	• This parameter is used to describe precedence relations between jobs. • $\beta_2 = prec$ corresponds to scheduling problems with precedence relations between jobs defined by an arbitrary cyclic directed graph. • Other values, include chains, intree, outtree, tree or series-parallel directed graph, are used to describe more restricted precedence structures.
β_3	• If $\beta_3 = r_i$, then release dates may be specified for each job.
β_4	• This parameter is used to specify any restrictions on the processing times or on the number of operations.
β_5	• If $\beta_5 = d_i$, then a deadline d_i is specified for each job J_i.
β_6	• This parameter is used to specify sets of jobs that must be grouped into batches for joint processing on machines. E.g. $p-batch$ and $s-batch$.

Like all other combinatorial optimization problems, the goal of a scheduling problem is often stated in the form of an objective function or performance measure. Very often, the optimization of a scheduling problem entails the search for a feasible solution which minimizes the performance measure. In this context, the performance measure is also known as a total cost function and this is indicated as γ in Graham's three field classification system. Thus, denoting the completion time of job J_i by C_i and its associated cost by $f_i(C_i)$, the two types of

total cost functions, bottleneck objectives and sum objectives, are defined respectively as follows:

$$f_{\max}(C) := \max\left\{ f_i(C_i) \middle| \, i = 1,...,n \right\} \tag{2.1}$$

$$\sum f_i(C) := \sum_{i=1}^{n} f_i(C_i) \tag{2.2}$$

While makespan $\gamma = \max\{C_i | i = 1,...,n\}$, total flow time $\gamma = \sum_{i=1}^{n} C_i$ and weighted (total) flow time $\gamma = \sum_{i=1}^{n} w_i C_i$ are often defined as objective functions, other functions are also possible. However, only some of the quantities belong to an important class of performance measures that are known as regular measures of performance, stated formally as follows:

Definition 2.1.

A performance measure Z is regular if:

a. The scheduling objective is to minimize Z.

b. The value of Z can increase only if at least one of the completion times in the schedule increases.

This definition is significant because it is usually desirable to restrict attention to a limited set of schedules called a dominant set. In this case, makespan is regular.

Definition 2.2.

A set R *is a dominant set of regular measures of performance if there exists a schedule* $S \in R$ *with completion time* C_j *and regular measure* Z *and* $S' \notin R$ *with completion time* C_j' *and regular measure* Z' *such that:*

a. $C_j \leq C_j'$ *for all* j.

b. $Z \leq Z'$ *for any regular measure.*

From the above definition, it is clear that a dominant set of schedules must also contain the optimal schedule.

2.3 Disjunctive Graph Representation

Graphical methods such as Gantt charts, see Porter (1968), are often employed to represent schedules. A Gantt chart is essentially a horizontal bar chart developed as a production control tool in 1917 by Henry L. Gantt, an American engineer and social scientist, which may be either *machine-orientated* or *job-orientated* in the context of machine scheduling. While these graphical tools are useful for visualization purposes, they lack the conciseness offered by mathematical constructs.

In particular, the disjunctive graph model of Roy and Sussmann (1964) provides a convenient means to represent feasible schedules for shop scheduling problems. It has replaced the solution representations by Gantt charts, which is useful in user interfaces to graphically depict a solution to a problem. When the objective function of the shop scheduling problem is regular, the set of feasible schedules represented in this way always contains an optimal solution to the problem. For any given instance of a shop scheduling problem, its definition is given below:

Definition 2.3.

A disjunctive graph representation $G=(V,C,D)$ *consists of a node set* V, *conjunctive arc set* C *and disjunctive arc set* D *such that:*

a. $V=\{O_{sce},O_{11},,...,O_{1n_i},...,O_{i1},...,O_{in_i},O_{snk}\}$ *is the set of nodes representing the operations of all jobs, where* O_{ij} *is the* j*-th operation on* $J_i(i=1,...,n)$, *with two additional dummy nodes* O_{sce} *(source) and* O_{snk} *(sink), to denote the start and end of a schedule.*

b. C *is the set of directed conjunctive arcs which reflect the precedence relations between the operations with the numbers on the arcs reflecting the processing times. If* u *and* v *are two operations with* p_u *and* p_v *as their respective processing requirement, there exists a conjunctive arc* (u,v) *with length*

p_u for every $u \in V$, $v \in V$ pair where u has to be processed before v. Moreover, there are conjunctive arcs, denoted by subset O, between the source and all operations without a predecessor and between all operations without a successor and the sink. Therefore, $C = \{A \cup B \cup O\}$ where $A = \bigcup(A_i : i \in J)$ for precedence relations A_i between operations of the same job J_i and $B = \bigcup(B_j : j \in M)$ for precedence relations B_j between operations on the same machine M_j.

c. D contains disjunctive arcs which are used to present disjunctive constraints that arise naturally in machine scheduling. Mathematically, this set can be represented as $D = E \cup F$ consisting of two distinct subsets of disjunctive arcs E and F where $E = \bigcup(E_i : i \in J)$ and $F = \bigcup(F_j : j \in M)$. Furthermore, there is a pair of disjunctive arcs (u,v) and (v,u), with lengths p_u and p_v respectively, in either E_i for each pair of operations belonging to the same job J_i which are not connected by a conjunctive arc or F_j for each pair of operations processed on the same machine M_j which are not connected by a conjunctive arc.

Evidently, the sets E and F are very similar: E decomposes into E_i subgraphs, one for each job J_i and F decomposes into F_j subgraphs, one for each machine M_j. Let $K = J \cup M$, where each element k of the set K is either a job or a machine. Hence, $D = \bigcup(D_k : k \in K)$ where $D_k = E_k$ if $k \in K \cap J$ and $D_k = F_k$ if

$k \in K \cap M$. Figure 2.1 provides an illustration of a disjunctive graph representation for a schedule with n jobs on m machines.

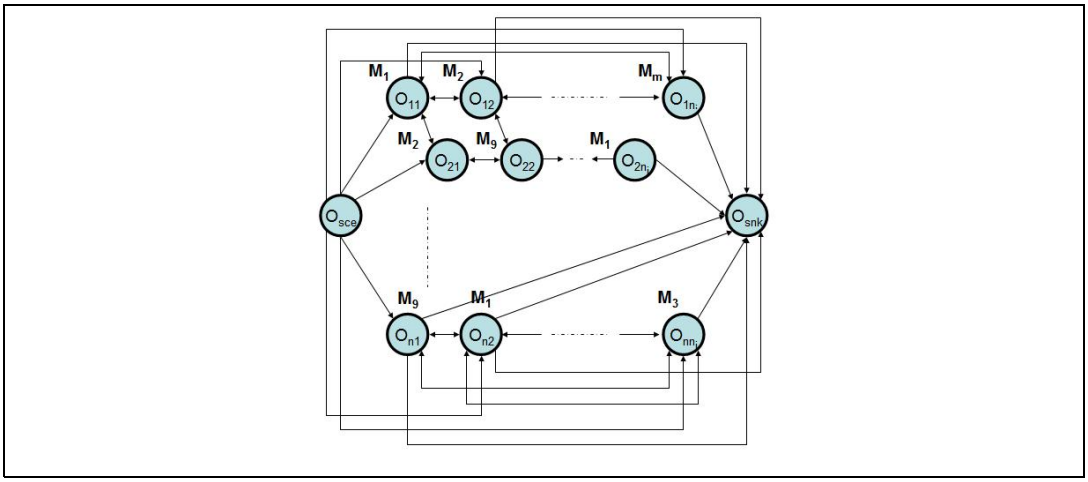


Figure 2.1: Pictorial of Disjunctive Graph Representation

Since the basic scheduling decision is to define an ordering between the operations connected through disjunctive arcs by turning these undirected disjunctive arcs into directed ones, the concept of selection Ω_k is therefore important.

Definition 2.4.

Ω_k is a set of directed disjunctive arcs, called fixed arcs, chosen from the D_k's such that it contains exactly one member of each disjunctive pair of D_k.

A feasible schedule can only be obtained from G when the selection is a complete selection.

Definition 2.5.

A selection $\Omega = \bigcup(\Omega_k : k \in K)$ *is a complete selection if:*

a. Each disjunctive arc has been fixed.

b. The resulting graph $G(\Omega) = (V, C \cup \Omega)$ *is acyclic.*

Given a complete selection Ω, a corresponding schedule S, which defines an order of operations for each job and each machine, may be constructed. For each path γ from vertex i to vertex j in $G(\Omega)$, define the length of γ to be the sum of lengths of arcs in that path.

2.4 Classification of Schedules

Since the taxonomy of schedules (Pinedo, 2002) is pivotal to the analysis of shop scheduling problems, a short discourse on the various classes of schedules will be essential in establishing the basic shop scheduling concepts.

Definition 2.6.

A schedule is called feasible if the precedence relations are maintained and the resource constraints are met.

In principle, there are an infinite number of feasible schedules for a shop scheduling problem since an arbitrary amount of idle time can be inserted at any machine between adjacent pairs of operations. Accordingly, there are various possible moves on a schedule, with respect to its representation on Gantt chart, which can be made to improve its viability in terms of any specific regular performance measure. Nevertheless, schedules are classified as semi-active, active, non-delay. The ensuing text outlines their respective definitions:

Definition 2.7.

A feasible schedule is called semi-active if no operation can be completed earlier without changing the order of processing on any one of the machines.

The start times of operations on each machine of a semi-active schedule are adjusted so that that there are no idle times between each operation on the same machine whenever possible. This form of adjustment, known as a local left-shift, is equivalent to moving an operation block to the left on the Gantt chart while preserving the operation sequence on the machine.

Definition 2.8.

A feasible schedule is called active if it is not possible to construct another schedule by changing the order of processing on the machines and having at least one operation finishing earlier and no operation finishing later.

Global left-shift refers to an adjustment in which some operation is begun earlier without delaying any other operation. An active schedule is a schedule in which no global left shift can be made. However, many semi-active schedules can often be compacted into the same active schedule through a series of global left-shifts. Clearly, the set of active schedules dominates the set of semi-active schedules. Therefore, it is sufficient to consider only active schedules when optimizing any regular measure of performance.

Definition 2.9.

A feasible schedule is called non-delay if no machine is kept idle while an operation is waiting for processing.

Since the number of active schedules still tends to be prohibitive, it is often convenient to focus on an even smaller subset of schedules known as non-delay schedules. All non-delay schedules are active schedules as no global left-shifting is possible for these schedules. However, many active schedules may not be non-delay schedules since requiring a schedule to be non-delay is equivalent to prohibiting unforced idleness. This implies that the number of non-delay schedules may be significantly less than the number of active schedules. The dilemma is that there is no guarantee that the set of non-delay schedules will contain an optimum.

2.5 Active Chain Concepts

Given that the set of active schedules is the smallest dominant set of schedules, it is sufficient to consider only active schedules for scheduling problems with regular performance measures. Therefore, it will make sense for some algorithms to search only in the active schedule space. To facilitate subsequent description of the active chain, the notion of the next-follow will be introduced here:

Definition 2.10.

A next-follow relation $\succ\succ$ is a relation between two operations o and o' in a schedule such that $o' \succ\succ o$ if and only if:

a. The starting time of o' is equal to the finishing time of o.

b. o is either the preceding operation of o' of the same job or the operation of a different job processed on the same machine as o'.

Given that an operation can at most next-follow two other operations, the formal definition of an active chain is as follows:

Definition 2.11.

An active chain of an operation o is a set of operations including o and an operation without a predecessor, such that:

a. Except for the operation without a predecessor, each operation in the set next-follows exactly one operation in the set.

b. Except for operation o, there is exactly one operation in the set that next-follows the operation for every operation in the set.

An example of an active chain is included in Figure 2.2. It is clear from the definition that the length of an active chain of operation o is the sum of the processing times of all operations in the chain. Therefore, if the earliest starting times of all jobs are all zero, the finishing time of o will be simply the length of its active chain. In this case, the makespan of the schedule is equivalent to the length of the longest active chain in the schedule.

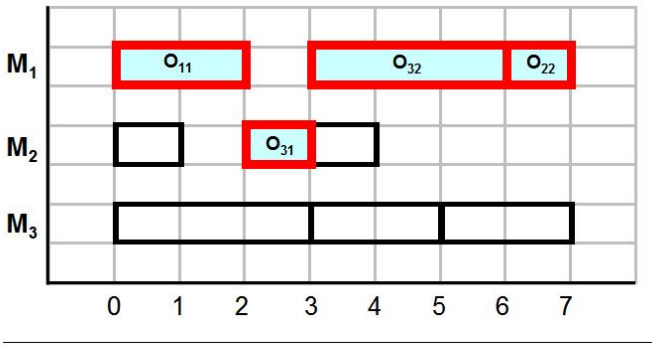


Figure 2.2: Pictorial of Active Chain

Based on this concept of active chain manipulation, Sun *et al.* (1995) proposed a scheduling algorithm for JSP and further proved that the necessary condition for

the makespan of an active schedule to be shortened is that at least one precedence relation between operations in one of the longest active chains of an active schedule is changed.

2.6 Group Shop Problem

The prevalence of many simplified models of scheduling problems often occurring in manufacturing and production settings signifies not only the growing needs for efficient scheduling tools but also marks the imperative to narrow the gap between academic research and industrial practice. However, as these simplified models are purely abstraction of real life problems, it is often difficult to fit real life problems into these very specialized formulations. Furthermore, stemming from the differing characteristics of the various Shop Scheduling problems, the repeated applications of a successful scheduling approach tailored for a particular problem type to another problem type often do not yield satisfactory results.

To bridge this gap, Sampels *et al.* (2002) advocated the use of a broad Shop Scheduling definition, which is known as Group Shop Problem (GSP), for the design and implementation of single and robust algorithm for the differing scheduling scenarios. Indeed, the GSP formulation generalizes several disparate formulations including Flow Shop Problem (FSP), Job Shop Problem (JSP), Open Shop Problem (OSP) and Mixed Shop Problem (MSP) succinctly into a simple canonical form outlined as follows:

Definition 2.12.

A group shop problem, as depicted in Figure 2.3, consists of a finite set of operations O, *which can be partitioned into* m *subsets* $\psi = \{\psi_1,...,\psi_m\}$ *such that* $\bigcup_{j=1}^{m} \psi_j = O$ *and* n *subsets* $\xi = \{\xi_1,...,\xi_n\}$ *such that* $\bigcup_{i=1}^{n} \xi_i = O$, *together with a partial order*[4] $\preceq \subseteq O \times O$ *where* $\preceq \cap \xi_i \times \xi_j = \phi$ *for* $i \neq j$ *and a function* $\lambda : O \to \aleph$ *that assigns processing times to operations.*

ψ_j is the set of operations which has to be processed on machine M_j whereas ξ_i is the set of operations which belong to job J_i.

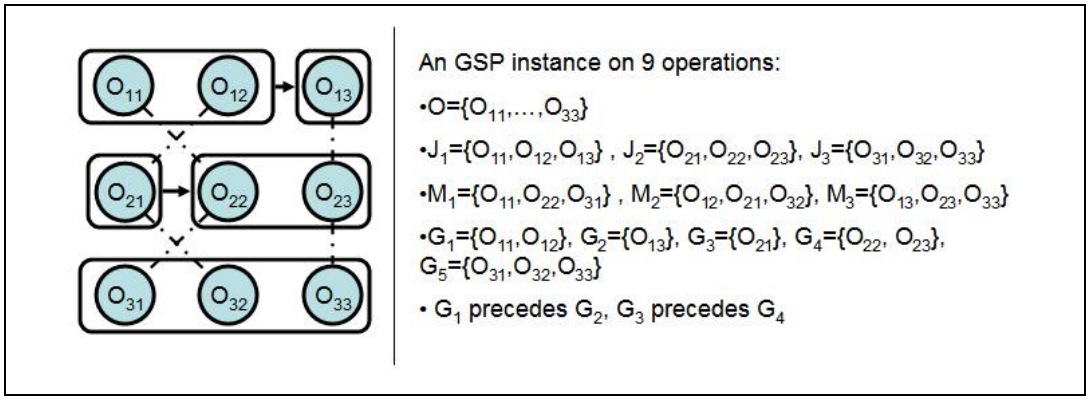


Figure 2.3: Pictorial of GSP Instance

With respect to the partial order defined above, $(o,o') \in \preceq$ (also denoted by $o \preceq o'$) means that the processing of operation o has to be completed before the

[4] A relation R is a partial ordering if it is a pre-order (i.e. it is reflexive (xRx), transitive ($xRyRz \Rightarrow xRz$) and anti-symmetric ($xRyRx \Rightarrow x = y$) with elements x and y for which neither xRy nor yRx. Comparing the partial order denoted by $o \preceq o'$ and the next-follow relation $o' \succ\!\succ o$, it is clear that only the former is transitive in nature whereas the latter imposes a more stringent requirement on the starting time of o' to be equal to the finishing time of o.

processing of operation o' can begin. Moreover, $(o,o') \in \preceq$ and $(o',o'') \in \preceq$ implies that $(o,o'') \in \preceq$. Therefore, the partial order defines the technological sequences with the set $\preceq_{pred}(o) = \{o' | (o',o) \in \preceq\}$ and the set $\preceq_{succ}(o) = \{o' | (o,o') \in \preceq\}$ known as the set of predecessors of an operation o and the set of successors of an operation o respectively.

Definition 2.13.

A feasible solution is a refined partial order $\preceq^* \supseteq \preceq$ *for which the restrictions* $\preceq^* \cap \psi_j \times \psi_j$ *and* $\preceq^* \cap \xi_i \times \xi_i$ *are total* $\forall i, j$

Denoting the machine on which an operation o has to be processed on as $m(o)$ and the job on which an operation o belongs to as $j(o)$, a feasible solution defines a sequential ordering of the operations in a job and on a machine such that either $(o,o') \in \preceq^*$ or $(o',o) \in \preceq^*$ for every pair of operations o, o' with $j(o) = j(o')$ or $m(o) = m(o')$. Furthermore, each machine can process at most one operation at a time, operations must be processed without preemption and operations belonging to the same job must be processed sequentially.

Definition 2.14.

The cost of a feasible solution is defined by:

$$C_{\max}(\preceq^*) = \max\left\{\sum_{o\in\eta}\lambda(o) \middle| \eta \text{ is an active chain in } (O, \preceq^*)\right\}$$

where $C_{\max}$ *is called the makespan of a solution such that the goal is to find a feasible solution that minimizes* $C_{\max}$.

Given a feasible schedule, the corresponding acyclic graph can be easily constructed. If the redundant arcs are removed, every operation, except node 0 and node $N+1$, has at most two immediate predecessors and at most two immediate successors. Moreover, there are also operations that can be postponed without increasing the makespan as well as those that cannot be postponed. The former is labeled as slack operations whereas the latter is known as critical operations. Accordingly, a critical path is an ordered sequence of critical operations, which can be further decomposed into subsequences of operations, called blocks.

Definition 2.15.

A block is a maximal subsequence of operations processed on the same machine or belonging to the same group, where the set of groups Γ *is defined as the basic unit of segregation for the partition* ξ *such that* $\Gamma = \{\Gamma_1,...,\Gamma_g\}$.

It should be noted that the restriction $\preceq \cap \xi_i \times \xi_i$ is total (i.e. means that there are precedence relations defined between operations for all operations belonging to the same job) in JSP and FSP, trivial (i.e. means that there are no precedence relations defined between operations for all operations belonging to the same job) in OSP, and either total or trivial for each i in MSP. For GSP, the segregation of partition ξ into groups Γ, with $g(o)$ denoting the group which an operation o belongs to, imposes a weaker restriction on $\preceq$ such that $\preceq \cap \Gamma_i \times \Gamma_i$ has to be trivial. Moreover, for o, o' from the same job with $o \in \Gamma_i$ and $o' \in \Gamma_j$ such that $i \neq j$, either $o \prec o'$ or $o \succ o'$ holds. Note also that the coarsest refinement $\Gamma = \xi$ (group sizes equal to job sizes) is equivalent to OSP and the finest refinement $\Gamma = \{\{o\} | o \in O\}$ (group sizes of 1) is equivalent to JSP. In short, for group shop scheduling, operations of each job can be partition into many groups, which may be of different sizes with precedence constraints applying only between groups of operations when these operations are scheduled.

2.7 Local Search

As most scheduling problems are NP-hard in nature, the use of approximate methods has been a popular approach for solving scheduling problems. In particular, local search algorithms have been widely used for attacking NP-hard problems. Similarly, these algorithms have also been employed and adapted extensively for scheduling problems. Therefore, it is important that a basic appreciation of local search algorithms is developed since these search techniques are often the basis of more complex scheduling algorithms.

While the use of local search algorithms has been reported as early as in Croes (1958), initial interest in local search algorithms decreased over time due to the lack of new conceptual development. Despite its practical usefulness then, the computational resources required for higher quality solutions and larger size problems were beyond the reach of the early years of computer science. The recent decade of renewed interest in local search algorithms is attributed to several factors as reported in Aarts and Lenstra (1997), of which, the understandability, flexibility and ease of implementation of local search algorithms, in comparison with exact algorithms, justify practical application of these algorithms in solving large problem instances. Together with parallel development in data structures to facilitate more efficient neighborhood solution search and exponential improvement in computer speed and memory availability, these have given much thrust to research in the design of local search algorithms.

Since the crux of local search algorithm design rests primarily on the definition of its neighborhood structure, the following formalization necessarily follows:

Definition 2.16.

A neighborhood structure is a function $N : S \mapsto 2^S$ *that assigns to every* $s \in S$ *a set of neighbors* $N(s) \subseteq S$. $N(s)$ *is also called the neighborhood of* s.

The choice of an appropriate neighborhood structure is often done in a problem specific manner as it critically determines the performance of the local search algorithm. Indeed, local search algorithms present an interesting trade-off between the size of neighborhood $N(s)$ and the efficiency of the search. Small neighborhoods are faster to evaluate but do not result in good moves consistently, which can retard the progress of search. Large neighborhoods may guarantee good moves only at high computational cost. Despite the large variety of neighborhood structures reported in the literature, there are few general principles to rely on for the structural design of neighborhoods.

While neighborhood structure stipulates the set of solutions that can be reached from s in a single step of a local search algorithm, it may be defined by either explicit enumeration of the set of possible neighbors or implicit definition of the set of possible local changes that may be applied to a solution. The former is termed as an explicit neighborhood while the latter is called an implicit neighborhood.

Very often, an implicit neighborhood requires some auxiliary structure, such as memory list, other than simply the variables and values for its characterization. Other neighborhood structures depend on either problem instance or randomization. Certainly, local search algorithms may have several different neighborhood structures to select from, in which case, the choice of the neighborhood to use at each state may be made on the basis of a fixed schedule, dynamic selection or a combination of both mechanisms.

As it is not always necessary to generate all possible elements of a neighborhood, the solution found by a local search algorithm may not be a globally optimal solution. It may only be optimal with respect to local changes.

Definition 2.17.

A local minimum is a solution s *such that* $\forall s' \in N(s): f(s) \leq f(s')$. *A local minimum is termed as a strict local minimum if* $\forall s' \in N(s): f(s) < f(s')$.

Since neighboring solutions are generated by a move-generation mechanism that selects and accepts from a pool of solutions according to some pre-defined criteria, the following definitions are essential.

Definition 2.18.

A move-generation mechanism generates the set of neighbors by changing one attribute or a combination of attributes of a given instance s. *A move-generation is a transition from a solution* s *to another* $s' \in N(s)$ *in one step (or iteration).*

Definition 2.19.

A candidate list of solutions $\overline{N(s)}$ *is a set of solutions* $s' \in N(s)$ *that satisfies some pre-defined acceptance and admission criteria such that* $\overline{N(s)} \subseteq N(s)$.

Moreover, to design a local search algorithm, the following choices need to be specified clearly:

a. Generation Mechanism. Rules that determine how (feasible) neighbors s' from $N(s)$ can be obtained.

b. Acceptance and Selection Strategy. This is sometimes termed as the pivoting rule, which determines which neighborhood solution s' from $\overline{N(s)}$ replaces the current one.

c. Stopping Test. These functions determine when the algorithm terminates a particular line of search. If a local search algorithm finds a satisfying assignment, then it can terminate. However, if the algorithm gets trapped in local minima or other parts of the search space from which escape is likely to be costly, there should be criteria for when to abandon search and restart from a different initial point.

Given the myriad of choices that a designer needs to specify when designing local search algorithms, it is not surprising that there exist many variants of local search algorithms. The most basic form for generation mechanism is the iterative improvement, commonly employed in hill-climbing techniques or steepest descent techniques[5]. Within this basic framework, design variation hinges on the pivoting rule with the first-improvement rule and the best-improvement rule being the most commonly used ones. While the first improvement rule repeatedly generates neighboring solutions of s but only returns the first lower cost solution, the best-improvement rule examines the whole neighborhood and returns the best solution at each step. The time needed to check the entire neighborhood at each step is known as the *check-out time*. Papadimitriou and Steiglitz (1982) provide more elaborate discussion.

Another interesting aspect of local search is complexity analysis. Conventional computational complexity classes offer easy means of classifying problems

[5] The term "hill-climbing" pertains to a maximization problem, but the equivalent descent method refers to minimization problems.

according to their difficulty, which are then solved by any known algorithm. Yet, these classes cannot capture the local search problems associated with usual local search algorithms. To address the question of worst-case complexity of local search algorithms, the class PLS of *polynomial-time local search problems* is introduced in Johnson *et al.* (1988). In general, local search algorithms in this class require polynomial time for each local search step.

2.8 Meta-Heuristics

It is easy to observe that one of the major drawbacks of iterative improvement local search is that it may stop at a poor quality local minimum. While one possibility is to restart the local search whenever some criterion is met so as to avoid being trapped in local minima, the increasing dimensionality of the search space with respect to problem size will inevitably make this approach less attractive. Moreover, the possible search space structures are not exploited with restarts from random initial solutions.

```
Procedure TabuSearch(s∈S)
     Determine initial candidate solution s;
     while (termination condition not satisfied)
          Determine set N of non-tabu neighbors of s;
          Choose a best improving solution s' in N;
          Update tabu attributes based on s';
          s=s';
     end
     return s;
End TabuSearch
```

Figure 2.4: Algorithmic Framework of Tabu Search

The inadequacies found in the iterative improvement local search have paved the way for the design and development of general heuristic methods, which are applicable to a wide range of COPs. The term meta-heuristics, which was first coined in Glover (1986) together with the introduction of Tabu Search, as outlined in Figure 2.8.1, to describe another kind of approximate algorithm that attempts to combine basic heuristic methods in higher level frameworks aimed at exploring search space is defined as follows:

Definition 2.20.

A meta-heuristic is a master strategy that governs the behavior of other subordinate heuristics through intelligent combination of concepts for exploring and exploiting the search spaces, including the possible use of learning strategies to structure information, so as to produce solutions beyond those that are normally generated through iterative improvement local search efficiently.

The generality of the definition encompasses numerous forms of approximate methods based on various interpretations of what constitutes “intelligent search”. Notably, the dichotomy between meta-heuristic orientation and heuristic orientation is distinct so as to accentuate the significance of the departure from classical heuristic design. Moreover, the emphasis on governance differentiates a meta-heuristic from a simple random restart procedure or a random perturbation procedure albeit the fact that these naive restarting and perturbation procedures

are sometimes classed as low-level meta-heuristics. These interpretations present design choices which in turn can be used for classification purposes. Depending on the features selected to differentiate between various meta-heuristics, several classifications are possible:

a. Nature Inspired vs. Non-nature Inspired. The origins of the algorithm provide one intuitive means of classifying meta-heuristics. Colorni *et al.* (1996) offers some insights to the importance and promise of utilizing natural and social analogies to derive meta-heuristics. Notably, nature inspired meta-heuristics tend to loosely model a phenomenon existing in nature. Not only are they non-deterministic and adaptive, they present implicitly a parallel structure (multiple agents). Genetic Algorithm (Winter *et al.*, 1995) is an example of nature inspired algorithms.

b. Population Based vs. Single Point Search. The manner in which the search is being conducted by the algorithm is another distinguishing characteristic. Tabu Search and Variable Neighborhood Search (VNS) (Mladenovic and Hansen, 1997) are examples of meta-heuristics based on single-point search, in which a single solution is manipulated at each step (or iteration) of the algorithm. On the contrary, the search process in population based meta-heuristics has the property of describing the evolution of a set of points in the search space. More importantly, the population in search space is brought about through conscious and

iterative treatment on sets of points. An example of a population based method is Ant Colony Optimization (ACO) (Dorigo and Stützle, 2004).

d. Trajectory vs. Discontinuous Methods. An essential difference between various meta-heuristics is whether there exists only one single search trajectory corresponding to a closed walk on the neighborhood graph or whether there are larger jumps in the neighborhood graph. Interestingly, trajectory methods, such as Simulated Annealing (Kirkpatrick *et al.*, 1983), as depicted in Figure 2.5, and Tabu Search, share the property of describing a trajectory in the search space during the search process. In fact, the search process of these methods can be seen as the evolution in (discrete) time of a discrete dynamical system as reported in Devaney (1989) and Bar-Yam (1997).

```
Procedure SimulatedAnnealing(s∈S)
     Determine initial candidate solution s;
     Set initial temperature T;
     while (termination condition not satisfied)
          Choose a neighbor s' of s probabilistically;
          If (s' satisfies probabilistic acceptance criterion)
               s=s';
          end
          Update T according to annealing schedule;
     end
     return s;
End SimulatedAnnealing
```

Figure 2.5: Algorithmic Framework of Simulated Annealing

d. Dynamic vs. Static Objective Function. Another form of differentiation is based on the manner in which the objective function is used in the algorithm. While most algorithms keep the objective function given in the problem formulation i.e. "as it is", some others like Guided Local Search (GLS) (Voudouris and Tsang, 1995), adjust[6] their objective functions during the search in order to escape local optima via search landscape transformation.

e. Single vs. Multiple Neighborhood Structures. Most meta-heuristics operate on single neighborhood structure whereas other meta-heuristics such as VNS uses a set of neighborhood structures to diversify the search process.

f. Memorizing vs. Memory-less Algorithms. The use of search history has also dichotomized the entire family of meta-heuristics into memorizing algorithms and memory-less algorithms. Typically, memory-less algorithms are Markovian in nature as the information they need is only the current state of the search process. Considering the use of memory as one of the fundamental elements of a powerful meta-heuristic, the design of appropriate memory structures has become an important consideration in meta-heuristic design. Consequently, there are a variety of ways in which memory can be exploited in meta-heuristics. Short term and long term

[6] Very often, these algorithms also incorporate information during the search process into the modification of the objective functions.

memory structures are predominantly used in many meta-heuristics. The former usually keeps track of recently performed moves, visited solutions or, in general, decisions taken. The latter forms an accumulation of synthetic parameters and indices about the search.

Other than the taxonomy for meta-heuristics, an important dimension to think about is the interplay between the degree of exploitation and the degree of exploration during the conducting of the search. The degree of exploitation, or sometimes known as degree of intensification, refers to the amount of effort directed to local search in the present region of the search space. In contrast, the degree of exploration, commonly referred to as degree of diversification, refers to the amount spent to search in distant regions of space, including the selection of solutions in a far region and the acceptance of a worsening solution, relative to the current solution, so as to gain the possibility of discovering new and better solutions. Since these two requirements are conflicting in nature, a reasonable tradeoff between them is imperative and they must be carefully tuned in each meta-heuristic depending on the problem. Additionally, trade-off between effort, in terms of computational time, and efficacy, in terms of quality of final solution, will also need to be considered.

2.9 Fitness Landscape

In the broadest sense, optimization can be conceived to occur in three interrelated number spaces. The search space contains the legal values of all elements that

can be entered into the objective function(s). On the other hand, an objective function is a set of operations on parameters while the function space contains the results of those operations. The last type of number space is one-dimensional and is termed as the fitness space. It contains the degrees of success with which patterns of parameters optimize the values in the function space, measured as goodness or error. Generally, each point in the parameter space maps to a point in the function space, which further corresponds to a point in the fitness space. Interestingly, direct mapping of search space to fitness space is possible in many cases.

Despite the limitations of conventional theoretical analysis, there is no doubt that the performance of meta-heuristics depends strongly on the configuration of these underlying spaces. This has led to the dominance of empirical methods for evaluation of approximate methods in many research works. Works such as Colletti and Barnes (2000), Grover (1992), and Gutin and Yeo (2001), which emphasize theoretical analysis of COPs and the approximate methods for solving them, are generally rare.

In the recent years, several research papers, such as Boese *et al.* (1994), Reeves (1999), and Mattfeld *et al.* (1999), have attempted statistical investigations on search spaces to either account for the performance of meta-heuristics or devise specific algorithmic variants to better exploit the known characteristics of the COPs. Works such as Weinberger (1990) and Stadler (1995) have demonstrated the apparent similarities between COPs search spaces and genotype space.

Despite its origin in theoretical biology, the notion of fitness landscape is central to COP's search space analysis. The metaphorical view of a genotype space of a species as a landscape, where related genotypes occupy nearby locations, states a model of genotype space as a mountainous region consisting of peaks, valleys, ridges and plateaus. The same model is applicable in the description of the evolutionary process of a species adapting to regions of higher fitness through natural selection. In the domain of optimization, fitness landscape is the topology of the fitness surface over the entire search space with the objective function playing the role of an artificial fitness function. This topology, which describes the shape of the search space as encountered by search algorithms, is defined as:

Definition 2.21.

A fitness landscape is a topology on S *which defines for each solution* $s \in S$ *a set of neighborhood solutions* $N(s) \in S$ *and assigns to every* $s \in S$ *a fitness value* $f(s)$ *governed by the mapping* $f : S \to \Re, s \mapsto f(s)$ *with respect to* N. *If arbitrary elements of* S *can be transformed into each other by a finite sequence of neighborhood moves,* N *is called a connected neighborhood.*

Though the present research in search space analysis seems rudimentary, the need for characterizing search space properties of COPs will continue to drive research efforts in this area. With better understanding of the underlying features of COPs, approximate methods will in turn be attuned to deliver better results.

2.10 Known Shop Scheduling Approaches

So far, most works on deterministic shop scheduling have centered on JSP, FSP, OSP and MSP. Among them, JSP has attracted the most research interest with the famous 10 jobs 10 machines instance formulated for the first time by Fisher and Thompson (1963) and subsequently solved by Carlier and Pinson (1989) using a Branch and Bound algorithm. Following this, other Branch and Bound algorithms have been proposed, including Applegate and Cook (1991), and Brucker *et al.* (1994), to improve computational performance. In addition, many simple heuristics, including the use of priority rules, were employed in the early stage. Among these, the Shifting Bottleneck Procedure (SBP), which was proposed by Adam *et al.* (1988), is known to be a simple and effective heuristic method as the algorithm builds up and improves a schedule via iterative construction of a single bottleneck machine problem. Nevertheless, the fundamental problems with SBP are its difficulties in performing re-optimization of schedules as well as the algorithm's tendencies in generating infeasible schedules.

Many of the search algorithms developed for $J\|C_{\max}$ between the late 1980s and early 1990s can be considered to be innovative. Examples include the SA approach by Larrhooven *et al.* (1992), TS Approach by Taillard (1994), TS Approach by Amico and Trubian (1993), as well as GA by Nakano and Yamada (1991). To date, the best known algorithm for $J\|C_{\max}$ is the TS Algorithm of

Nowicki and Smutnicki (1996a). Apparently, TS algorithms generally exhibit better empirical performance in comparison with those based on SA and those based on variants of SBP. Reviews by Vaessens *et al.* (1996) as well as Jain *et al.* (1999) provide good overview of the research development in this area.

As a special case of JSP, FSP also enjoyed considerable attention from researchers. However, most works focused on the Permutation Flow Shop Problem (PFSP), which refers to a FSP whose processing order of the jobs on machines is the same for every machine. Several methods have been proposed for PFSP in the literature. These include SA Algorithms of Osman and Potts (1989), and Ogbu and Smith (1990), TS Algorithms of Taillard (1990), and Nowicki and Smutnicki (1996b) as well as GA of Reeves (1995). In contrast, very few papers address the General Flow Shop Problem (GFSP).

Comparatively, there are lesser papers that deal with $O\|C_{\max}$. Among the few earlier heuristic methods published, the more notable ones are Rock and Schmidt (1983), Shimoys *et al.* (1994) as well as the generalized SBP approach by Ramudhin and Marier (1996). However, in recent times, local search based meta-heuristics have been developed that solve OSP more effectively. These include TS Approach by Liaw (1999a), Hybrid GA by Prins (2000), and Liaw (2000) as well as SA Approach by Liaw (1999b). As Liaw's algorithms for $O\|C_{\max}$ are based on makespan estimation, they perform extremely well on both benchmarks and randomly generated problems.

Recognizing that a multi-stage system may be a mixture of "pure" shops in practice, Masuda *et al.* (1985) introduced the concept of MSP and initiated the theoretical investigation of MSP. This was followed by an algorithm by Strusevich (1991), which improved upon the results of Masuda *et al.*. Thereafter, works of Shakhlevich *et al.* (1999) and Shakhlevich *et al.* (2000), presented some polynomial / pseudo-polynomial algorithms for solving some special cases and provided a survey for the recent results about the complexity of MSP respectively.

GSP was first introduced in the context of a mathematical competition organized by TU Eindhoven, Netherlands[7]. Although GSP shares many characteristics of MSP, it is generally considered a further extension of the MSP framework. Similar to MSP, there are little known works on GSP. Furthermore, GSP is an NP-hard problem since it is well known that its special cases, the JSP and the OSP, are both NP-hard. So far, there are only three working GSP algorithms: one based on ACO Approach (Blum, 2003) and two based on TS Approach (Sampel *et al.*, 2002 and Liu *et al.*, 2005).

The key contribution from Sampel *et al.*'s (2002) is the formulation of the first GSP neighborhood definition, which is an extension of the neighborhood definition by Nowicki and Smutnicki (1996a). Moreover, in the Sampel *et al.*'s TS Approach, there are three characterizing features, namely: the usage of dynamic tabu lists, the implementation of a restart mechanism and the incorporation of probabilistic

[7] See http://www.win.tue.nl/whizzkids/1997/

rules for neighborhood moves selection. Subsequently, using the same neighborhood definition from Sampel *et al.*, Blum (2003) effectively applied the ACO meta-heuristics on GSP despite less successful earlier attempts by other researchers, including Colorni *et al.* (1993), on other classes of shop scheduling problems. Particularly, Blum made tactical improvements to the pheromone model for moves selection. As for the Liu *et al.*'s TS Approach, it incorporates the use of a variety of neighborhood definitions that are an extension to those of Liaw's. (Liaw, 1999a and Liaw, 1999b). Unlike most other TS Approaches for Shop Scheduling Problems, two different types of tabu lists are implemented to store the entire machine routing for the job and entire job routing for the machine corresponding to the selected move. Based on the comparative study of Liu *et al.* (2005), the TS Approach by Liu outperforms the other two algorithms by obtaining good solutions for most of the GSP benchmark instances.

2.11 Common Neighborhood Definition

The quality of solutions obtained through local search heuristics strongly depends on the search neighborhood of the algorithm as the neighborhood definition determines how the algorithm traverses within the search space. Most of the modern scheduling neighborhood definitions are based on moves generation on critical paths as they are more likely to lead to improving moves. In fact, for every feasible schedule, there is at least one critical path which can be conceived as a series of either machine blocks i.e. consecutive operations belonging to the same

machine on a critical path or group blocks i.e. consecutive operations belonging to the same group on a critical path.

In order to understand how to formulate good neighborhood definitions, it is worthwhile to note that they often exhibit the following desirable features:

a. Correlativity. As far as possible, a neighboring solution should be highly correlated to its originator to facilitate a thorough exploration of search space.

b. Feasibility. Perturbations that always lead to feasible solutions avoid the computational effort needed to perform repair on generated infeasible solutions.

c. Improvability. If additional problem specific knowledge can be incorporated into the neighborhood definition, a move would have a better chance to obtain an improved solution value.

d. Size. The average size of neighborhood moves should be within reasonable bounds so as to prevent either premature termination of algorithm or excessive computational demand to be placed on the algorithm.

e. Connectivity. Promising areas of the search space may be excluded from the search process if there is no finite sequence of moves leading from an arbitrary solution to a global optimal solution.

JSP, being one of the most widely researched Shop Scheduling Problem, offers an array of different neighborhood structures. A quick survey of the existing JSP neighborhoods will provide insights for the design of GSP neighborhoods, which are in turn crucial ingredient of any GSP scheduler. For the explanation of the different neighborhood structures, the following definition is needed:

Definition 2.22.

*An **internal operation** is defined as an operation of a machine or group block that is neither the first nor the last operation in that block.*

V. Laarhoven *et al.* (1992) outlined a neighborhood which is based on the reversal of the processing orders of two swappable operations in any machine block of a schedule's critical path. The design of this neighborhood was based on two basic observations, namely:

a. The reversal of two swappable operations on a schedule's critical path will never lead to an infeasible schedule.

b. The reversal of two swappable operations that are not on a schedule's critical path will lead to either an infeasible schedule or a non-improving schedule.

The advantage of this neighborhood is that it is connected. However, as highlighted by Matsuo *et al.* (1988), many of the candidate moves under this neighborhood are non-improving in nature since the reversal of two swappable internal operations on a schedule's critical path can never improve a schedule. Therefore, the neighborhood of Matsuo *et al.* (1988) is defined as that of V. Laarhoven *et al.* (1989) but with the exclusion of swappable internal operations. Subsequently, Nowicki and Smutnicki (1996a) extended the idea of Matsuo *et al.* (1988) by excluding from their neighborhood definition the reversal of the first two operations of the first machine block on a schedule's critical path and the reversal of the last two operations of the last machine block on a schedule's critical path. Other noteworthy neighborhood definitions are outlined by D. Amico and Trubian (1993):

a. Permute the processing order of any two swappable operations at the beginning of a machine block on a schedule's critical path with a preceding operation processed on the same machine or permute the processing order of any two swappable operations at the end of a machine block on a schedule's critical path with a succeeding operation processed on the same machine.

b. Move any one operation from machine blocks, of size at least 2, in front of the first operation of the machine block or after the last operation of the machine block.

Against this backdrop, a generalization of the result derived for JSP in Brucker (1994) is provided in Blum (2003), as follows:

Theorem 2.1.

Let $\preceq^*$ *be a feasible solution to a GSP instance. If there is a solution* $\preceq^{*'}$ *with* $C_{\max}(\preceq^{*'}) < C_{\max}(\preceq^*)$, *then there is a machine block or group block* $\chi^i = o^i_{k_i} \succ\succ^{*'} \ldots \succ\succ^{*'} o^i_1$, *where* k_i *denotes the number of operations in* χ^i, *in the critical path* η^c *such that* $\exists o \in \chi^i,\ o \neq o^i_1$ *with* $o^i_1 \succ\succ^{*'} o$ *or* $\exists o \in \chi^i,\ o \neq o^i_{k_i}$ *with* $o \succ\succ^{*'} o^i_{k_i}$.

With this theorem, it is reasonable to define the neighborhood of a feasible solution $\preceq^*$ as follows: A feasible solution $\preceq^{*'}$ is a neighbor of $\preceq^*$ $(\in N(\preceq^*))$ if there is exactly one machine block or exactly one group block $\chi^i = o^i_{k_i} \succ\succ^* o^i_{k_i-1} \succ\succ^* \ldots \succ\succ^* o^i_2 \succ\succ^* o^i_1$ in a critical path η^c of $\preceq^*$ such that the order of o^i_1 and o^i_2 or the order of $o^i_{k_i-1}$ and $o^i_{k_i}$ is being swapped in $\preceq^{*'}$. See Appendix for the proof of this theorem.

2.12 Concluding Remarks

Thus far, this chapter has provided the basic schema for understanding the notations, definitions and general solving approaches for shop scheduling problems. From this literature survey, it appears that meta-heuristics are more effective than most other approaches. While these meta-heuristics have been applied extensively on different classes of scheduling problems, it is difficult to determine the relative merits of each of these meta-heuristics methods as the overall design principle of these methods are often based on tactical improvements of certain aspects of the algorithm, such as neighborhood structures and memory structure, and comparative analysis of algorithm performance are often based on experimental results.

Finally, another interesting trend that was uncovered in this literature survey is that there are increasingly more studies being conducted to establish the underlying nature of shop scheduling problems. In particular, the use of a fitness landscape provides a novel way of analyzing the characteristics of combinatorial optimization problems. Although it is necessary to take note of the different meta-heuristic techniques as well as the various local search neighborhood definitions applied to enhance the algorithm performance, insights on the nature of the shop scheduling search space are more important as they will enable better search strategies to be devised and the appropriate techniques to be incorporated into the design of the scheduling algorithms.

Chapter 3 GSP Scheduling Methodology

3.1 Overview

With the conceptual framework of the present study laid out in the preceding chapter, the current chapter sets out to elucidate the design of a GSP Scheduler. Apart from providing a sketch of the algorithm design, this chapter also provides the specific implementation details required to develop the algorithm, including but not limited to memory structures, neighborhood definitions and critical path computation methods. Other than addressing the key design considerations highlighted in the preceding chapters, this chapter will also cover all the relevant design rationales taken for the design of this algorithm.

3.2 Algorithmic Outline

A typical shop scheduling algorithm can generally be perceived to consist of two stages, namely, the schedule construction stage and the schedule improvement stage. Similarly, the proposed GSP Scheduler utilizes a simple list scheduling algorithm similar to the one proposed in Giffler and Thompson (1960) to randomly generate different initial solutions. Although any randomly generated solution may serve as the starting point for local search algorithms, constructive algorithms are typically used to generate good initial solutions before subsequent applications of local search algorithms. This often leads to the discovery of better quality local

minima in fewer number of steps compared to the direct use of local search algorithms without initial application of construction heuristics.

As outlined in Figure 3.1, the proposed algorithm starts off by first generating an initial schedule based on random selection of schedule type and priority rule with the *ConstructSchedule* Procedure. Due to the myopic nature of schedule construction phase, the constructed solutions need not be locally optimal with respect to some simple neighborhood. Therefore, this initial solution will be improved upon through the *OptimizeSchedule* Procedure.

```
Procedure GSP Scheduler
   Select ScheduleType randomly;
   Select PriorityRule randomly;

   InitialSchedule = ConstructSchedule(ScheduleType, PriorityRule);
   Optimal_Schedule = OptimizeSchedule(InitialSchedule);
End GSP Scheduler
```

Figure 3.1: Algorithmic Skeleton of GSP Scheduler

The potential of constructing such a hybrid scheduling system that is capable of integrating dispatching rules and search procedures has been substantiated by the computational results of the scheduling algorithms by Amico and Trubian (1993), Sun *et al.* (1995) as well as that by Liaw C. F. (1999a), which have been covered in the literature survey. In particular, such hybrid systems can be designed so that search procedures are implemented whenever for effectiveness is needed and dispatching rules are carried out when quick responses are

necessary. Since the objective of this research is to develop an algorithm for industrial applications, this approach has been incorporated as part of the algorithm to provide the necessary scalability and speed for real-life situations.

3.3 Schedule Construction

Other than being used to generate the initial solution for the GSP Scheduler Algorithm, the *ConstructSchedule* Procedure is also used iteratively within the *OptimizeSchedule* Procedure to provide further random starting points to increase the probability of finding better schedules within the solution space by the algorithm. In the current implementation, the *ConstructSchedule* Procedure is designed to generate both active schedule and non-delay Schedule using different priority rules. The following figure outlines the *ConstructSchedule* Procedure:

Procedure ConstructSchedule($S_P \in S$)

Initialize partial solution S_P;

Initialize list of unscheduled operations i.e. $O^+ = O^*$;

for($t = 1$ to $t = |O|$) **do**

Put operations, with predecessor in S_P, into O_t;

$O_t^{'}$ = Restrict(S_P, O_t);

o^* = Choose($O_t^{'}$);

Extend S_P by appending operation o^*;

$O^+ = O^+ \setminus \{o^*\}$;

end

```
    return S_P ;
End ConstructSchedule

* o denotes the list of all operations to be scheduled.
```

Figure 3.2: Algorithmic Skeleton of ConstructSchedule Procedure

Using the *Restrict* Procedure, a candidate list strategy is applied to further restrict the set O_t. Depending on the type of schedule desired, there are two major ways of implementing this function. For the generation of active schedules, the following version of *Restrict* Procedure should be used:

```
Procedure Restrict(S_P ∈ S , O_t)
    Calculate the earliest possible completion time t_ec for all o ∈ O_t ;
    Select machine M* with the minimal completion time t* ;
    Define O_t' as operations on M* with earliest start time t_es ≤ t* ;
End Restrict
```

Figure 3.3: Algorithmic Skeleton of Restrict Procedure for Active Schedule

However, if non-delay schedules are needed, the algorithm skeleton of the *Restrict* Procedure depicted in Figure 3.4 should be implemented.

```
Procedure Restrict(S_P ∈ S , O_t)
    Determine the earliest possible starting time t* for all o ∈ O_t ;
    Define O_t' as operations that can start at time t* ;
End Restrict
```

Figure 3.4: Algorithmic Skeleton of Restrict Procedure for Non-Delay Schedule

As for the *Choose* Procedure, it implements a policy for determining among the operations of O_t', the one to be scheduled next. The policy typically involves choosing operations with the highest weight, where weights are given by priority rules. Table 3.1 shows a selection of them.

Table 3.1: List of Priority Rules

Priority Rule	**Description**
Random	An operation is randomly chosen
EST	An operation with earliest starting time
EFT	An operation with earliest finishing time
SPT	An operation with shortest processing time
LPT	An operation with longest processing time
LWR	An operation with least work remaining in the job
MWR	An operation with most work remaining in the job
LTW	An operation with least total work in job
MTW	An operation with most total work in job

The rationale for the choice of only generating both active schedules and non-delay is clear as it allows the algorithm to focus its attention on a limited set of schedules of desirable quality, which in turn reduces the amount of time the algorithm needs for local search improvements. While the set of non-delay schedules is smaller than the set of active schedules, the GSP Scheduler randomly selects between both types for schedule generation since there is no guarantee that the set of non-delay schedules will contain an optimum as opposed to active schedules. Coupled with the use of different priority rules and the implementation of a schedule regeneration process within the

OptimizeSchedule Procedure, the dependency of the algorithm's performance on the initial seed will be reduced.

3.4 Search Strategy

While the smoothness of a search landscape, according to prevailing opinion in Gu and Huang (1994), Schneider *et al.* (1997), and Coy et al. (1998), is the central pre-requisite of successful search, Mattfeld *et al* (1999) has demonstrated that smoothness alone is not sufficient in order to explain the intractability of the hard JSP instances. In general, problems with local optima dispersed over the entire search space do not work well with local search methods while problems with smooth overall landscape tend to make adaptive search methods, such as GA, less effective. It is reasonable to further generalize the results of this study for GSP instances since JSP tends to be more difficult to solve compared to other shop scheduling problems.

Other than relying on the findings of Mattfeld *et al* (1999) as the primary basis for determining the choice between population based search approach and single-point based search approach for the implementation of the GSP Scheduler, the relative merits and shortcomings of both approaches have also been considered. While population based approaches, such as GA, tend to be highly efficient methods for intensification, they are often outperformed by purpose-built methods. See Reeves (1994) for further elaboration on a comparative analysis of the various meta-heuristics approaches. Furthermore, based on the results of the

literature survey in the preceding chapters on known scheduling approaches, most of the better scheduling algorithms are based on single-point based search approaches, such as Tabu Search. As such, the GSP Scheduler adopts the framework of single-point based search, which is in this case the Tabu Search method.

Among other findings, the same study by Mattfeld *et al* (1999) also revealed that local optima tend to be dispersed over the entire JSP search space and that harder JSP instances tend to possess smoother fitness landscape than easy instances. The authors further conjectured that these results may explain the success of hybrid methods, which combine sophisticated control with limited tabu walks. In other words, an algorithm, with a restart mechanism, will tend to perform better as it will be able to explore different parts of the search space so that it may be able to uncover better solutions elsewhere.

Therefore, to be inline with these findings, the proposed GSP Scheduler attempts to strike a balance between exploration and exploitation, in relation to the search within the entire solution space, at the strategic tier by delineating the *OptimizeSchedule* Procedure into two distinct phases:

a. Diversification Phase. In this phase, a move is selected to be applied on the current schedule. If there are no local moves possible for any step or if there is no improvement to the local search after *MaxRestart* iterations, a 'restart' will be initiated to generate a random schedule. At

each step, the schedule is compared to both the *local best solution* and *global best solution*. Promising solution and its associated tabu memory are stored in the 'backtrack' memory for use in the intensification phase.

The basis for moves selection has been implemented as the *SelectRModelmoves* Procedure, depicted in Figure 3.5, provides a two-stage moves selection mechanism. In the first stage, the best-improvement rule is applied to determine the best possible move on the basis of lowest makespan value in the current iteration. Although this rule yields better quality solution for each iteration, it requires greater computational time than the first improvement rule, which does not require all the neighborhoods of the current solution to be generated at one time but generate them one at a time until the first lower cost neighborhood solution. As such, for computational efficiency, the makespan corresponding to each of the possible local moves are estimated prior to the selection of moves in the *SelectRModelmoves* Procedure. As for stage two, a probabilistic acceptance criterion, which has been adapted from the Simulated Annealing Algorithm, is applied to determine if the current selected non-tabu move should be accepted so that the schedule can be adjusted.

```
Procedure SelectRModelmoves( S_C ∈ S , (u,v) )
   Estimate makespan for all local search moves of S_C ;
   Select local move (u,v) with best estimated makespan, C'_max ;
   if( (u,v) is non-tabu)
      Compute δ = C'_max − C_max where C_max is the makespan of S_C ;
      if  δ ≤ 0
         return (u,v);
      else
         Generate uniform random number x ∈ (0,1);
         if  x < exp(−δ/T)
            return (u,v);
         end
         return NULL;
      end
   else if( C'_max < C*_max where C*_max is the global best makespan)
      Compute δ = C'_max − C_max where C_max is the makespan of S_C ;
      if  δ ≤ 0
         return (u,v);
      else
         Generate uniform random number x ∈ (0,1);
         if  x < exp(−δ/T)
            return (u,v);
         else
            return NULL;
         end
      end
   else
      return NULL;
   end
End SelectRModelmoves
```

Figure 3.5: Algorithmic Skeleton of SelectRModelmoves Procedure

The rationale behind this approach is twofold: (1) to ensure that the final solution is independent of the starting solution and (2) to prevent the algorithm from being trapped in a local minimum. Moreover, a geometric cooling function is implemented to allow moves to be more readily accepted at the start of the algorithm and more frequently rejected towards the end of the diversification phase. The relationship between the cooling function, denoted by $\alpha(T)$, and the current temperature value, denoted by T, is defined as follows:

$$\alpha(T) = aT \tag{3.1}$$

According to Reeves (1994), most reported successes in the literature use values of a that are between 0.8 and 0.99. Similarly, the GSP Scheduler has adopted the use of these values.

b. Intensification Phase. The solutions, which are stored in the back-track memory, are restored for more refined search using a deterministic acceptance criterion for local moves. A maximum of *MaxBacktrackPoints* solutions will be stored in the backtrack memory for further search during the intensification phase. In this phase, the makespans corresponding to each of the possible local moves are estimated for the selection of moves using the *SelectDModelmoves* Procedure. Similar to *SelectRModelmoves*

Procedure, the best-improvement rule is applied as well. Figure 3.6 outlines the *SelectDModelmoves* Procedure below:

```
Procedure SelectDModelmoves( S_C ∈ S , (u,v) )
   Estimate makespan for all local search moves of S_C ;
   if(number of tabu moves = number of possible moves)
      Select oldest tabu move (u,v) from list;
      Update TList until selected move becomes non-tabu;
      return (u,v);
   else
      Select local move (u,v) with best estimated makespan, C'_max ;
      if( (u,v) is non-tabu)
         return (u,v);
      else if( C'_max < C*_max where C*_max is the global best makespan)
         return (u,v);
      else
         Select next best non-tabu local move (u,v);
         return (u,v);
      end
   end
End SelectDModelmoves
```

Figure 3.6: Algorithmic Skeleton of SelectDModelmoves Procedure

During the diversification phase, it is possible that more than *MaxBacktrackPoints* number of promising solutions may be generated for storage. However, in such cases, only the best *MaxBacktrackPoints* number of promising solutions with the best makespan will be stored.

The basic idea behind this search strategy is to identify as many promising solutions as possible from different parts of the search space at the onset during diversification phase so that better ones among these promising solutions can be selected for further improvement search in the intensification phase. Figure 3.7 provides the pseudo-code for *OptimizeSchedule* Procedure.

```
Procedure OptimizeSchedule( S_i ∈ S )
   Initialize current schedule S_C = S_i ;
   Initialize local best schedule S_LB = S_i ;
   Initialize global best schedule S_GB = S_i ;
   Initialize Temperature T for Simulated Annealing;
   Initialize Tabu List TList ;
   while(TerminationCondition1 != true)
      Generate local moves from S_C+ based on current neigbourhood;
      if(SelectedMove = SelectRModelMoves( S_C ,MoveList, T , TList )
         AdjustSchedule( S_C , SelectedMove);
         UpdateTabuList( TList );
         ComputeMakespan( S_C );
         if( S_C ≤ S_LB )
            S_LB = S_C ;
            if( S_C ≤ S_GB )
               S_GB = S_C ;
            end
         end
      end
      if(RestartCondition = true)
         Save S_C and its TList onto backtracking memory;
```

```
            S_C = RegenerateSchedule();
        end
        Update T based on geometric cooling function;
    end
    while(TerminationCondition2 != true)
        Load schedule from backtrack memory into S_C and TList;
        while(TerminationCondition3 != true)
            Generate local moves from S_C;
            if(SelectedMove = SelectDModelMoves(S_C,MoveList,T,TList)
                AdjustSchedule(S_C, SelectedMove);
                UpdateTabuList(TList);
                ComputeMakespan(S_C);
                if(S_C ≤ S_LB)
                    S_LB = S_C;
                    if(S_C ≤ S_GB)
                        S_GB = S_C;
                    end
                end
            end
        end
    end
    return(S_GB);
End OptimizeSchedule
+ In the diversification phase, local moves may be generated based on any of the
three different neighbor structures.
```

Figure 3.7: Algorithmic Skeleton of OptimizeSchedule Procedure

In this algorithm, the best solution found is always compared with the lower bound of the problem instance, which is in turn computed using the formula outlined in Taillard (1994):

$$LB = \max\left\{\max_{j}\left\{\sum_{i=1}^{n} p_{ij}\right\}, \max_{i}\left\{\sum_{j=1}^{m} p_{ij}\right\}\right\} \quad (3.2)$$

However, the drawback of using any form of lower bound formulation is that the best known makespan of a problem instance may be way higher than the lower bound makespan value. Therefore, it is important to define good termination conditions to mitigate the effects of unnecessary iterations taken by the algorithm that may arise from the above-mentioned situation.

Furthermore, three different termination conditions have been defined for two distinct phases of the algorithm. In the diversification phase, *TerminationCondition1* is satisfied whenever the total number of iterations has reached a pre-specified number of iterations, i.e. *MaxDivIterations* or the best solution obtained so far is less than *MinPercentDev* of the lower bound of the problem instance. Once this condition is met, the algorithm will proceed with the next phase of the algorithm. At each iteration step of the intensification phase, the algorithm will check if there is any schedule left in the backtrack memory for further schedule improvement and it will also determine if the current solution is equal to the lower bound of the problem instance. In particular, *TerminationCondition2* will be set to 'true' if there is no schedule left in the backtrack memory or if the current solution value is equal to the lower bound of the problem instance or if the optimum solution is found. When this happens, the algorithm will come to a halt. For each of the backtracked solution, local search will be performed up to a maximum of *MaxIntIterations* iterations. Whenever the

number of counts of local search performed on the current backtracked solution equals to *MaxIntIterations* or when the current solution value is equal to the lower bound of the problem instance or when the optimum solution is found, *TerminationCondition3* will be set to 'true'. This will trigger GSP Scheduler to terminate local search on the current backtracked solution so that it can retrieve the next schedule in the backtrack memory for further refinement.

3.5 Memory Structures

Two forms of memory have been implemented in this algorithm to achieve the two broad objectives of intensifying the search in "good" regions of the search space and diversifying the search towards unexplored regions:

a. Short Term Memory. A variable length tabu list memory structure has been designed to prevent the algorithm from revisiting recently generated solutions. A solution s' is considered forbidden if the current solution s can be transformed into s' by applying one of the moves in the tabu list. This list works in a First-In-First-Out (FIFO) manner. Each time a move (u,v), is made, its inverse (v,u) will be added to the tabu list. If the tenure, or length, of the tabu list is too short, its role of cycle prevention will not be effective. Conversely, if the tenure of a tabu list is too long, there will be too many restrictions imposed on the search. In the current implementation, the tenure of tabu List, i.e. *TBListLen*, is equal to the square root of the total number of critical operations in the current best

local schedule. However, it should also be noted that it is possible to override the tabu-status of moves when they lead to new solutions which are better than the best found so far. This is performed by means of aspiration level conditions. Moreover, in the event when all possible moves in a particular iteration are all tabu, the algorithm will pick the oldest possible move from the list and populate the list with replications of the "youngest" move until the selected move becomes non-tabu.

b. Long Term Memory. At the same time, to allow promising solutions uncovered during the diversification phase to be examined later, a back-track memory has been implemented as a form of long term memory to allow storage of the promising schedules and their associated tabu lists. The storing of the associated tabu list prevents the repetition of the same search history in the intensification phase of the algorithm. Since the local optima of shop scheduling problems tend to be dispersed within the search space, a long term memory of sufficient memory length will increase the chances of the optimum solution being located during the search.

3.6 Neighborhood Definitions

In an interesting study by Jain *et al* (2000), it was shown that the restrictive nature of the Nowicki and Smutnicki (NS) neighborhood (Nowicki and Smutnicki, 1996a) causes the initialization procedure of the scheduling algorithm to have substantial influence over the quality of the best solution found by the algorithm. Particularly,

Jain *et al* (2000) pointed out that one of the promising areas to explore for the design of scheduling algorithm is the application of multiple neighborhoods as the most restrictive neighborhood may not necessarily be the best. As such, it was proposed that multi-neighborhood system would apply NS neighborhood as the core, because it can be searched quickly and efficiently, and then some of the other neighborhoods at other levels in order to provide the necessary diversification. Other similar research works, such as Mladenovic and Hansen (1997), and Mladenovic and Hansen (2001), on the concept of Variable Neighborhood Search also examined how systematic change of neighborhoods within a possibly randomized local search algorithm can improve the performance of the algorithm. In view of these works, the proposed GSP Scheduler utilizes a total of four different neighborhood definitions to varying degrees. To describe the various neighborhoods, the notations in Table 3.2 are needed:

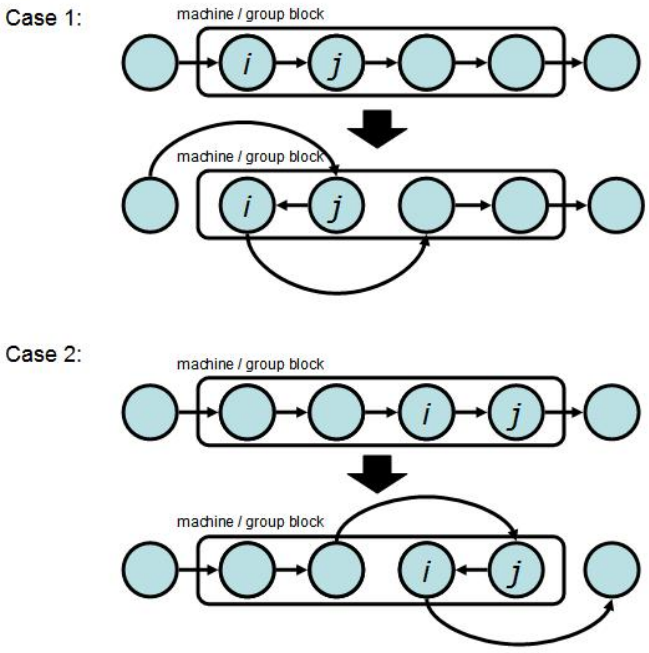
Table 3.2: Neighborhood Definition Notations

Symbol	Description
$M[i]$	The machine that processes operation i.
$PM[i]$	The operation processed on $M[i]$ just before operation i, if it exists.
$SM[i]$	The operation processed after $M[i]$ just after operation i, if it exists.
$J[i]$	The job to which operation i belongs to.
$PJ[i]$	The operation belonging to job $J[i]$ that precedes operation i, if it exists.
$SJ[i]$	The operation belonging to job $J[i]$ that follows operation i, if it exists.

$G[i]$	The group to which operation i belongs to.
$PG[i]$	The operation belonging to group $G[i]$ that precedes operation i, if it exists.
$SG[i]$	The operation belonging to group $G[i]$ that follows operation i, if it exists.

Using these notations, the definitions of the neighborhood used in the proposed GSP Scheduler are as follows:

Table 3.3: GSP Scheduler Neighborhood Definitions

NEIGHBORHOOD TYPE	NEIGHBORHOOD DEFINITION
$N0$	Swap i and j if both operations are either the first two operations of a machine / group block or the last two operations of a machine / group block, as follows: 

$N1$	With the exception of the first machine / group block and the last machine / group block, swap i and j if both operations are either the first two operations of a machine / group block or the last two operations of a machine / group block. For the first machine / group block, only the last two operations are swappable, while, only the first two operations are swappable for the last machine / group block.
$N2$	Assume that i is the first operation on a machine or group block. - Swap ($PM[i]$, i) if i is on a machine block. - Swap ($PG[i]$, i) if i is on a group block.
$N3$	Assume that j is the last operation on a machine or group block. - Swap (j, $SM[j]$) if j is on a machine block. - Swap (j, $SG[j]$) if j is on a group block.

The deployment of these neighborhood structures within the algorithm is as follows:

a. $N0$, $N2$ and $N3$ will be used during the diversification phase with $N0$ as the core. For every *NCIterations* iteration, one of the three neighborhood structures will be selected and applied once. Thereafter, the algorithm will revert back to the use of $N0$ for local search.

b. As for the intensification phase, the algorithm will make use of $N1$, which is a more restrictive version of $N0$, for local search. However, in the event that no further moves are possible in the current iteration for a particular backtracked solution, the algorithm will make a random selection among $N1$, $N2$ and $N3$ for generation of other feasible moves. Likewise, the algorithm will revert back to the use of $N1$ after a move has been applied on the given schedule.

In comparison with the neighborhood structures implemented in Liu *et al.*'s Algorithm (2005), the proposed neighborhood structures are much simpler in nature as they do not allow the exchange of the positions of more than two operations simultaneously. Nevertheless, it is still possible for the GSP Scheduler to achieve the same local search move sequence as in Liu *et al.*'s Algorithm (2005) when the different neighborhoods structures outlined above are applied sequentially. Unlike the Liu *et al.*'s Algorithm (2005), the use of such simple neighborhood definitions in the proposed algorithm does not impose the need for complex memory structures to store tabu moves. In particularly, for the Liu *et al.*'s Algorithm (2005), complete machine routings and job routings needs to be stored and to be compared whenever a local move is applied. This can be computationally intensive. Evidently, this approach avoids the pitfalls of having a single restrictive neighborhood definition as well as the complexities of implementing elaborate neighborhood structures.

3.7 Critical Path Determination

Another important aspect of the scheduling algorithm is the method used for the determination of the critical path. As mentioned in the preceding chapter, a critical path is an ordered sequence of critical operations whose length is equal to the makespan of the schedule. Since local searches are often performed on critical paths, the efficiency of the critical path determination approach will have a direct impact on the performance of the scheduling algorithm. This thesis adopts the topological-sequence method presented in Liu and Ong (2002). Typically, for any feasible schedule, every one of its operation i satisfies the following equations:

Definition 3.1.

$$e_i = \max\left\{e_{PM[i]} + p_{PM[i]}, e_{PJ[i]} + p_{PJ[i]}\right\}$$

where e_i and p_i = 0 for all undefined indices i.

Definition 3.2.

$$l_i = \max\left\{e_{SM[i]} + p_{SM[i]}, e_{SJ[i]} + p_{SJ[i]}\right\}$$

where e_i, p_i and l_i = 0 for all undefined indices i.

Most methods proposed in the literature, including Taillard (1994) and Bellman (1958), rely on the computation of e_i (where e_i denotes the length of a longest

path from node 0 to node i, excluding processing time of operation i, p_i) and l_i (where l_i denotes the length of a longest path from node i to node $N+1$, excluding processing time of operation i, p_i) so as to determine critical paths of schedules. However, in this study, the following approach is used:

Step 1. Compute the in-count value (the number of predecessors) of each node.

Step 2. Find the topological sequence of the N operations as follows:

a. Select node 0 as the first node on the topological order list.

b. Decrement the in-count value for each of the immediate successor nodes of the selected node by 1.

c. Select any of the unselected nodes which have a zero in-count value. Put this node as the next node on the topological order list.

d. Repeat Steps b and c until all nodes are selected.

Step 3. Starting from the setting $e_0 = 0$, calculate the e_i values of all nodes in the topological sequence according to Definition 3.1.

Step 4. Starting from the setting $l_{N+1} = 0$, calculate the l_i values of all nodes in the reversed topological sequence according to Definition 3.2.

Step 5. Determine the critical path by selecting operations satisfying the condition of $e_i + p_i + l_i = C_{\max}$.

3.8 Makespan Estimation Method

It is clear that the processing order of operations on critical paths needs to be modified in order to improve a current schedule. However, to do this effectively, the algorithm needs to be able to estimate the effects of all possible moves for a particular schedule based on the selected neighborhood definition so as to determine the most promising move for that particular iteration. In addition to that, the algorithm also needs to be able to determine the feasibility of the adjusted schedule as a move may produce infeasible schedules.

Although exact makespan computation methods as outlined in Section 3.7, and exact feasibility tests, including the labeling algorithm outlined in Adam *et al.* (1988), are available, the GSP Scheduler adopts an approach similar to that in Amico and Trubian (1993), and Liaw (1999a) to improve its computational efficiency. Particularly, the makespan estimation technique for the various neighborhood definitions has been incorporated into the design of *SelectRModelmoves* Procedure and *SelectDModelmoves* Procedure. Table 3.4

provides the details on how the estimated makespan values can be computed for the respective neighborhood definitions used in the GSP Scheduler.

Table 3.4: Estimated Makespan Values for GSP Scheduler

NEIGHBORHOOD TYPE	ESTIMATED MAKESPAN VALUE
$N0$	$e'_j = \max\{e_{PJ[j]} + p_{PJ[j]}, e_{PM[i]} + p_{PM[i]}\}$ $e'_i = \max\{e_{PJ[i]} + p_{PJ[i]}, e'_j + p_j\}$ $l'_i = \max\{l_{SJ[i]} + p_{SJ[i]}, l_{SM[j]} + p_{SM[j]}\}$ $l'_j = \max\{l_{SJ[j]} + p_{SJ[j]}, l'_i + p_i\}$ $C'_{\max} = \max\{e'_i + p_i + l'_i, e'_j + p_j + l'_j\}$
$N1$	
$N2$	Swap ($PM[i]$, i) if i is on a machine block and $j = SM[i]$ $e'_i = \max\{e_{PJ[i]} + p_{PJ[i]}, e_{PM[PM[i]]} + p_{PM[PM[i]]}\}$ $e'_{PM[i]} = \max\{e_{PJ[PM[i]]} + p_{PJ[PM[i]]}, e'_i + p'_i\}$ $e'_j = \max\{e_{PJ[j]} + p_{PJ[j]}, e'_{PM[i]} + p'_{PM[i]}\}$ $l'_j = \max\{l_{SJ[j]} + p_{SJ[j]}, l_{SM[j]} + p_{SM[j]}\}$ $l'_{PM[i]} = \max\{l_{SJ[PM[i]]} + p_{SJ[PM[i]]}, l'_j + p_j\}$ $l'_i = \max\{l_{SJ[i]} + p_{SJ[i]}, l'_{PM[i]} + p_{PM[i]}\}$ $C'_{\max} = \max\{e'_i + p_i + l'_i, e'_{PM[i]} + p_{PM[i]} + l'_{PM[i]}, e'_j + p_j + l'_j\}$ Swap ($PG[i]$, i) if i is on a group block and $j = SG[i]$ $e'_i = \max\{e_{PM[i]} + p_{PM[i]}, e_{PG[PG[i]]} + p_{PG[PG[i]]}\}$ $e'_{PG[i]} = \max\{e_{PM[PG[i]]} + p_{PM[PG[i]]}, e'_i + p'_i\}$ $e'_j = \max\{e_{PM[j]} + p_{PM[j]}, e'_{PG[i]} + p'_{PG[i]}\}$

	$l'_j = \max\{l_{SM[j]} + p_{SM[j]}, l_{SG[j]} + p_{SG[j]}\}$ $l'_{PG[i]} = \max\{l_{SM[PG[i]]} + p_{SM[PG[i]]}, l'_j + p_j\}$ $l'_i = \max\{l_{SM[i]} + p_{SM[i]}, l'_{PG[i]} + p_{PG[i]}\}$ $C'_{\max} = \max\{e'_i + p_i + l'_i, e'_{PG[i]} + p_{PG[i]} + l'_{PG[i]}, e'_j + p_j + l'_j\}$
$N3$	Swap (j , $SM[j]$) if j is on a machine block and $i = PM[j]$ $e'_i = \max\{e_{PG[i]} + p_{PG[i]}, e_{PM[i]} + p_{PM[i]}\}$ $e'_{SM[j]} = \max\{e_{PG[SM[j]]} + p_{PG[SM[j]]}, e'_i + p'_i\}$ $e'_j = \max\{e_{PG[j]} + p_{PG[j]}, e'_{SM[i]} + p'_{SM[i]}\}$ $l'_j = \max\{l_{SG[j]} + p_{SG[j]}, l_{SM[SM[j]]} + p_{SM[SM[j]]}\}$ $l'_{SM[i]} = \max\{l_{SG[SM[i]]} + p_{SG[SM[i]]}, l'_j + p_j\}$ $l'_i = \max\{l_{SG[i]} + p_{SG[i]}, l'_{SM[i]} + p_{SM[i]}\}$ $C'_{\max} = \max\{e'_i + p_i + l'_i, e'_{SM[j]} + p_{SM[j]} + l'_{SM[j]}, e'_j + p_j + l'_j\}$ Swap (j , $SG[j]$) if j is on a group block and $i = PG[j]$ $e'_i = \max\{e_{PM[i]} + p_{PM[i]}, e_{PG[i]} + p_{PG[i]}\}$ $e'_{SJ[j]} = \max\{e_{PM[SG[j]]} + p_{PM[SG[j]]}, e'_i + p'_i\}$ $e'_j = \max\{e_{PM[j]} + p_{PM[j]}, e'_{SG[i]} + p'_{SG[i]}\}$ $l'_j = \max\{l_{SM[j]} + p_{SM[j]}, l_{SG[SG[j]]} + p_{SG[SG[j]]}\}$ $l'_{SG[i]} = \max\{l_{SM[SG[i]]} + p_{SM[SG[i]]}, l'_j + p_j\}$ $l'_i = \max\{l_{SM[i]} + p_{SM[i]}, l'_{SG[i]} + p_{SG[i]}\}$ $C'_{\max} = \max\{e'_i + p_i + l'_i, e'_{SG[j]} + p_{SG[j]} + l'_{SG[j]}, e'_j + p_j + l'_j\}$

In this algorithm, these estimation techniques are used primarily for the purpose of moves selection. Once a selected move is applied on the current schedule, the

exact makespan will be computed using the method outlined in the previous section. This is to ensure that errors arising from such estimations are always kept within a reasonable margin while computational requirements for moves selection are minimized.

3.9 Schedule Regeneration

In line with the overall strategy to initiate search in different parts of the search space, the restart mechanism, which consists of (1) saving the current schedule and its associated tabu list into backtracking memory and (2) regenerating another schedule as the next starting point for local search, has been implemented as part of the GSP Scheduler. Essentially, the *RegenerateSchedule* Procedure, outlined in Figure 3.7, offers two modes of schedule regeneration, namely: *'Full Restart'* and *'Partial Restart'*, and it alternates randomly between the two modes during the algorithm runtime. In the *'Full Restart'* mode, the algorithm selects randomly the type of schedule and the priority rule before it constructs a schedule from scratch. This is similar to the *ConstructSchedule* Procedure. As for the '*Partial Restart*' mode, the algorithm will select randomly operation blocks on the critical path for reshuffling based on '*shortest processing time first'* rule, '*longest processing time first'* rule or '*random'* rule.

Depending on the nature of the GSP problem, the ratio of group blocks to machine blocks on the critical path of the schedule may be different. Hence, the probabilistic rule for selecting operations' blocks on the critical path for reshuffling

in the *'Partial Restart'* mode needs to be biased to the nature of the GSP instance. To do this, this thesis defines and proposes the following characteristic for differentiating between different GSP instances:

Definition 3.3.

A group factor of a GSP instance is defined as the ratio between the number of groups to the average group length where group length is equal to the number of operations belonging to the same group.

Based on this definition, a JSP instance will have its group factor equal to the total number of operations in the instance (i.e. '*MaxGroupFactor'*) while a similar OSP instance with the same number of tasks and same corresponding processing times will have its group factor equal to the number of operations in the instance divided by the number of tasks per job (i.e. '*MinGroupFactor'*). Accordingly, the probabilistic rule is defined as follows:

Definition 3.4.

Probability of Selecting Group Blocks is defined as:

a. *0, if group factor is equal to MaxGroupFactor.*

b. *0.5, if group factor is equal to MinGroupFactor.*

c. $0.5 \times \frac{MaxGroupFactor - GroupFactor}{MaxGroupFactor - MinGroupFactor}$, *otherwise*

By incorporating this probabilistic rule into the process of selecting operations blocks on the critical path for reshuffling in the *'Partial Restart'* mode, the algorithm will be able to adapt better to the differing nature of GSP instances.

3.10 Concluding Remarks

It is clear that the design of the GSP Scheduler is based on a search strategy derived from the generalization of the findings of JSP search space analysis for GSP instances. The clear articulation of the search strategy enables all the relevant tactical considerations, such as those on memory structures and neighborhood definitions, to be applied in the implementation of this algorithm. Other than incorporating known features from other scheduling algorithms, such as simulated annealing approach for move selection, critical path determination method and makespan estimation techniques, the GSP Scheduler also improvises a probabilistic schedule regeneration mechanism that is unique to this algorithm. This will not only ensure that the search techniques are consistent with the overall search strategy but should also improve the overall performance of the algorithm in terms of its effectiveness and efficiency.

Chapter 4 Computational Experiments

4.1 Overview

With the conceptual framework of the present study laid out in the preceding chapters, computational experiments were carried out with the GSP Scheduler on a series of benchmark problems to gauge the performance of the algorithm in relation to other known GSP scheduling algorithms. This chapter sets out to discuss experimental inputs, the settings, and the results obtained.

4.2 Experimental Inputs

To evaluate the performance of the GSP Scheduler in this study, the algorithm is applied on several established benchmark instances taken from the literature. As benchmark problems provide a common platform on which algorithms can be tested and gauged, problem instances of different dimensions and grades of difficulty should be selected so that the capabilities as well as the limitations of an algorithm will be revealed when it is applied on such problems. It should be noted that most benchmark problems have only integer processing times with a rather small range. While such instances may seem to have very little practical usefulness, Amar and Gupta (1986) have indicated that real life scheduling problems are easier to solve than simulated ones regardless of the type of algorithm used.

A total of 41 problem instances have been selected for use in the computational experiments. This is the same set of benchmark problems that has been used for computational experiments by Liu *et al.* (2005), Blum (2003) and Sampel *et al.* (2002). By adopting this set of benchmark problems, it will be possible to benchmark the performance of the GSP Scheduler against other GSP scheduling algorithms. Moreover, this set of benchmark problems offers a good mix of 'square' instances, which refers to problem instances where the ratio of the number of jobs to the number of machines is equal to one, and 'rectangular' instances, which refers to problem instances where the ratio of the number of jobs to the number of machines is greater than one. Numerous studies, including but not limited to Ramudhin and Marier (1996), and Watson *et al.* (2003), have shown that 'square' instances are generally harder to solve than 'rectangular' instances.

Among these problem instances, WHIZZKID is the only established GSP instance as it was subjected to a mathematics competition in the Netherlands in 1997. It consists of 20 jobs comprising 197 operations on 15 machines and these operations can be sub-partitioned into 124 groups. As for the rest of the GSP benchmark instances, they can be generated from prominent JSP problems, such as Fisher and Thompson's FT10 (10 machines and 10 jobs; see Fisher and Thompson, 1963), Lawrence's LA38 (15 machines and 15 jobs; see Lawrence, 1984) and Adam *et al.*'s ABZ7 (15 machines and 20 jobs; see Adam *et al.*, 1988), by introducing groups of various lengths into the jobs. In particular, FT10 is a prominent JSP instance, whose optimality was not determined for more than

twenty years before the optimality of one solution was proved by Carlier and Pinson (1989).

The GSP instances derived from FT10, LA38 and ABZ7 are denoted by FT10_XX, LA38_XX and ABX7_XX respectively, where XX is the group length. For example, the GSP instance generated from LA38 with group length of 5 is denoted by LA38_05. When the group length is 1, these GSP instances (FT10_01, LA38_01 and ABBZ7_01) are the original JSS instances. As the group length increases, these instances get closer the OSP instances. However, these GSP instances become the original OSP instances when the group length is equal to the number of machines in the respective problem instance.

Another important aspect of experimental inputs is the choice of parameter values. Typically, the values of these parameters tend to have a significant effect on the algorithm performance. In the absence of a more rigorous approach for the determination of these values, the algorithm parameters are empirically selected based on the application of the GSP Scheduler on other randomly selected problem instance to ensure a reasonable trade-off between running time and solution quality. The various parameter values for the GSP Scheduler to be used in the various computational experiments are found in Table 4.1.

Table 4.1: Selected Algorithm Parameter Values

Parameter	Values
MaxRestart	5% of '*MaxDivIterations*'
MaxBacktrackPoints	5 to 10
MaxDivIterations	50000 to 80000
MinPercentDev	1% to 5%
MaxIntIterations	10000 to 20000
TBListLen	Square root of critical path length of the current best local best schedule
NCIterations	10% of '*MaxDivIterations*'

Generally, the parameters are varied across different classes of GSP problems, such as FT10 Class, LA38 Class, ABZ7 Class and WHIZZKID Class, since the characteristics of these problems are substantially different from each other i.e. different number of machines, different number of jobs and different number of total operations. However, the settings are maintained for GSP instances of the same class as the proposed algorithm is able to adjust its search mechanism in accordance to the differing nature of GSP instances of the same class.

4.3 Empirical Results

The GSP Scheduler was coded in Visual C++ 6.0 environment and was tested on an ACER Laptop with Pentium IV Processor 1.5 GHz CPU, with 256 MB RAM running on Microsoft Windows XP Operating System. Using the specified

parameter values, the algorithm was applied on all the 41 GSP instances. For each GSP instance, there were a total of 20 runs so that the following can be computed:

a. Best Solution. Out of the 20 runs on the same problem instance, the best solution, S_{BST}, will be recorded.

b. Average Solution. Corresponding to each problem instance, the mean value of 20 final solutions obtained will be registered as S_{AVG}.

c. Average Computational Time. Similarly, T_{AVG} is captured for each problem instance so that the average case performance of the algorithm can be ascertained.

d. Coefficient of Variation. To measure the dispersion of solutions obtained from the 20 runs of each problem instance, a dimensionless metric is computed. The formula is as follows:

$$c_v = \frac{\mu}{\sigma} \times 100 \tag{4.1}$$

where μ is the mean of all the final solutions obtained in the 20 runs while σ is the standard deviation of all the solutions obtained in the 20 runs.

As the experiments for the other algorithms were performed on different processors, the computational times reported in these algorithms cannot be used for direct comparison. In particular, the experiments for Sampel *et al.*'s TS Algorithm and Blum's ACO Algorithm were performed on a 1.1 GHz AMD Athlon CPU under Linux while the experiments for Liu et al.'s TS Algorithm were performed on a 1.8 GHz Pentium IV CPU with 256 MB RAM under Microsoft Windows XP. Therefore, by assuming the inverse relationship between processing speed of the CPU and its computational time, the adjusted computational times are recomputed based on the following formula:

$$T' = \frac{S}{S'} \times T \tag{4.2}$$

where T denotes the reported computational time while S and S' refer to the CPU speed used in the original experiment and the CPU speed used in this current computational experiment respectively.

Overall, the GSP Scheduler achieved relatively good results as it was able to obtain the best known makespan values for 22 out of 41 problem instances. As for the problem instances whose best known makespan values were not obtained by the GSP Scheduler, both the their best solution values and their average solution values found by the GSP Scheduler were well within 2% deviation from the best known makespan value as shown in Table 4.2. The value of the best

known makespan for each instance is also included in the table. However, those problem instances whose optimality are unknown are enclosed in brackets.

In line with the observation that 'square' instances are generally harder to solve than 'rectangular' instances in studies, such as Ramudhin and Marier (1996), and Watson *et al.* (2003), the GSP Scheduler yielded better on ABZ7 Class and WHIZZKID Class than on FT10 Class and LA38 Class. Particularly, 45% of the best known solution obtained by the GSP Scheduler was from the ABZ7 Class.

By denoting the GSP Scheduler as Algorithm I, Liu *et al.*'s TS Algorithm as Algorithm II, Blum's ACO Algorithm as Algorithm III and Sampel *et al.*'s TS Algorithm as Algorithm IV, two different tables, namely Table 4.3 and Table 4.4, were tabulated for the comparison of best case performance and average case performance respectively.

Table 4.2: Solution Quality for GSP Scheduler

INSTANCE	BEST KNOWN MAKESPAN	ACTUAL SOLUTION VALUE		DEVIATION FROM BEST KNOWN (%)	
		S_{BST}	S_{AVG}	S_{BST}	S_{AVG}
FT10_01	930	**930**	938.4	0.0	0.9
FT10_02	(872)	875	894.8	0.3	2.6
FT10_03	(827)	830	838.9	0.4	1.4
FT10_04	(782)	790	798.8	1.0	2.1
FT10_05	(745)	747	750.7	0.1	0.4
FT10_06	(725)	**725**	738.6	0.0	1.9
FT10_07	(686)	690	700.4	0.6	2.1
FT10_08	(655)	**655**	660.1	0.0	0.8
FT10_09	(655)	**655**	656.2	0.0	0.2
FT10_10	(655)	**655**	655.8	0.0	0.1
LA38_01	1196	1212	1218.0	1.3	1.8
LA38_02	(1106)	1118	1122.5	1.1	1.5
LA38_03	(1049)	**1049**	1060.1	0.0	1.1
LA38_04	(993)	1000	1003.4	0.7	1.0
LA38_05	(990)	1000	1003.2	1.0	1.3
LA38_06	(961)	970	979.8	0.9	2.0
LA38_07	(956)	965	976.9	0.9	2.2
LA38_08	(943)	951	960.5	0.8	1.9
LA38_09	(962)	**962**	975.9	0.0	1.4
LA38_10	(979)	**979**	1004.8	0.0	2.6
LA38_11	(979)	980	1008.0	0.1	3.0
LA38_12	(943)	960	961.7	1.8	2.0
LA38_13	(943)	**943**	955.6	0.0	1.3
LA38_14	(943)	**943**	950.8	0.0	0.8
LA38_15	(943)	**943**	953.3	0.0	1.1
ABZ7_01	656	670	678.6	2.1	3.4
ABZ7_02	(641)	642	650.1	0.2	1.4
ABZ7_03	(611)	**611**	616.3	0.0	0.9
ABZ7_04	(609)	610	611.5	0.2	0.4
ABZ7_05	(638)	**638**	638.5	0.0	0.1
ABZ7_06	(600)	**600**	602.7	0.0	0.4
ABZ7_07	(567)	579	583.2	2.1	2.9
ABZ7_08	(564)	564	574.6	2.3	0.4
ABZ7_09	(577)	**577**	582.6	0.0	1.0
ABZ7_10	(612)	**612**	613.0	0.0	0.2
ABZ7_11	(610)	**610**	614.7	0.0	0.8
ABZ7_12	(592)	**592**	596.0	0.0	0.7
ABZ7_13	(581)	**581**	586.5	0.0	0.9
ABZ7_14	(562)	**562**	563.2	0.0	0.2
ABZ7_15	(556)	**556**	556.0	0.0	0.0
WHIZZKID	469	**469**	476.9	0.0	1.7

Table 4.3: Comparison of the Algorithms' Best Case Performance

INSTANCE	BEST KNOWN MAKESPAN	ALGORITHM I		ALGORITHM II		ALGORITHM III		ALGORITHM IV	
		S_{BST}	T_{AVG}	S_{BST}	T_{AVG}	S_{BST}	T_{AVG}	S_{BST}	T_{AVG}
FT10_01	930	**930**	67.5	**930**	74.8	**930**	94.9	**930**	50.8
FT10_02	(872)	875	54.9	**872**	78.6	**872**	74.6	875	48.5
FT10_03	(827)	830	63.4	**827**	75.8	834	93.8	**827**	84.5
FT10_04	(782)	790	69.9	784	81.0	791	83.7	**782**	69.6
FT10_05	(745)	747	56.8	**745**	73.6	754	62.8	748	57.9
FT10_06	(725)	**725**	54.0	**725**	75.6	725	84.8	729	61.5
FT10_07	(686)	690	54.8	**686**	54.8	694	67.8	696	57.4
FT10_08	(655)	**655**	27.8	**655**	15.1	**655**	42.5	657	58.1
FT10_09	(655)	**655**	29.8	**655**	1.0	**655**	0.8	**655**	5.9
FT10_10	(655)	**655**	2.4	**655**	0.6	**655**	0.6	**655**	1.2
LA38_01	1196	1212	209.5	1202	235.4	1228	934.4	**1196**	605.4
LA38_02	(1106)	1118	186.5	**1106**	225.5	1126	838.1	**1106**	594.1
LA38_03	(1049)	**1049**	240.2	**1049**	218.4	1061	798.0	**1049**	690.4
LA38_04	(993)	1000	223.9	**993**	239.9	1019	707.8	**1002**	651.6
LA38_05	(990)	1000	226.4	**990**	249.2	1003	817.6	**990**	630.4
LA38_06	(961)	970	228.9	**961**	256.7	977	931.6	974	949.8
LA38_07	(956)	965	232.9	**956**	253.2	970	821.2	961	623.6
LA38_08	(943)	951	220.1	**943**	202.8	955	952.2	954	777.4
LA38_09	(962)	**962**	213.7	**962**	254.8	962	624.4	966	430.5
LA38_10	(979)	**979**	216.6	990	231.2	**979**	904.1	982	96.7
LA38_11	(979)	980	216.1	991	242.6	**979**	803.5	985	292.7
LA38_12	(943)	960	217.3	**943**	142.9	947	811.3	960	789.7
LA38_13	(943)	**943**	180.9	**943**	11.8	**943**	55.0	**943**	792.4
LA38_14	(943)	**943**	157.1	**943**	9.0	**943**	29.1	**943**	595.9
LA38_15	(943)	**943**	90.7	**943**	8.8	**943**	12.5	**943**	141.8

Table 4.3 (Cont.): Comparison of the Algorithms' Best Case Performance

INSTANCE	BEST KNOWN MAKESPAN	ALGORITHM I		ALGORITHM II		ALGORITHM III		ALGORITHM IV	
		S_{BST}	T_{AVG}	S_{BST}	T_{AVG}	S_{BST}	T_{AVG}	S_{BST}	T_{AVG}
ABZ7_01	656	670	209.5	667	331.3	676	757.0	661	556.4
ABZ7_02	(641)	642	186.5	**641**	324.4	**641**	930.1	**641**	151.2
ABZ7_03	(612)	**611**	240.2	612	331.1	612	848.2	612	120.8
ABZ7_04	(609)	610	223.9	**609**	321.6	**609**	91.4	**609**	34.6
ABZ7_05	(638)	**638**	226.4	**638**	324.6	**638**	10.6	**638**	11.4
ABZ7_06	(600)	**600**	228.9	**600**	293.5	**600**	31.7	**600**	43.0
ABZ7_07	(567)	579	232.9	**567**	391.2	**567**	648.3	**567**	524.5
ABZ7_08	(564)	564	220.1	577	329.5	577	59.0	577	25.9
ABZ7_09	(577)	**577**	213.7	**577**	342.1	**577**	44.9	**577**	21.6
ABZ7_10	(612)	**612**	216.6	**612**	323.9	**612**	41.9	**612**	11.3
ABZ7_11	(610)	**610**	216.1	**610**	310.2	**610**	13.0	**610**	9.8
ABZ7_12	(592)	**592**	217.3	**592**	315.6	**592**	26.5	**592**	19.1
ABZ7_13	(581)	**581**	180.9	**581**	302.2	**581**	13.2	**581**	37.3
ABZ7_14	(562)	**562**	157.1	**562**	305.8	**562**	18.5	**562**	54.9
ABZ7_15	(556)	**556**	90.7	**556**	18.1	**556**	4.0	**556**	58.1
WHIZZKID	469	**469**	746.8	**469**	821.4	486	924.4	474	727.6

Table 4.4: Comparison of the Algorithms' Average Case Performance

INSTANCE	BEST KNOWN MAKESPAN	ALGORITHM I		ALGORITHM II		ALGORITHM III		ALGORITHM IV	
		S_{AVG}	c_v	S_{AVG}	c_v	S_{AVG}	c_v	S_{AVG}	c_v
FT10_01	930	938.4	0.44	933.2	0.28	939.5	0.71	930.6	0.20
FT10_02	(872)	894.8	0.76	873.4	0.22	887.5	0.96	880.1	0.30
FT10_03	(827)	838.9	0.48	831.8	0.42	853.1	1.25	838.3	0.32
FT10_04	(782)	798.8	0.38	791.1	0.73	805.6	0.86	796.0	0.39
FT10_05	(748)	750.7	0.34	748.5	0.85	764.1	0.90	758.1	4.74
FT10_06	(725)	738.6	0.64	727.4	0.40	736.6	0.65	745.3	0.49
FT10_07	(686)	700.4	0.79	692.9	0.56	702.5	0.87	709.7	0.55
FT10_08	(655)	660.1	0.32	655.0	0.00	655.1	0.05	660.5	0.42
FT10_09	(655)	656.2	0.34	655.0	0.00	655.0	0.00	655.0	3.44
FT10_10	(655)	655.8	0.29	655.0	0.00	655.0	0.00	655.6	3.88
LA38_01	1196	1218.0	0.66	1206.3	0.38	1235.5	0.34	1200.0	6.16
LA38_02	(1106)	1122.5	0.47	1107.6	0.14	1148.8	0.89	1112.3	0.33
LA38_03	(1049)	1060.1	0.29	1049.0	0.09	1068.4	0.37	1054.9	0.31
LA38_04	(1002)	1003.4	0.85	996.8	0.27	1028.7	0.60	1011.3	0.07
LA38_05	(990)	1003.2	0.75	996.0	0.43	1019.3	0.83	1011.4	0.15
LA38_06	(961)	979.8	0.34	970.7	0.91	985.6	0.50	981.2	0.41
LA38_07	(956)	976.9	0.42	962.5	0.56	980.5	0.61	971.0	0.00
LA38_08	(943)	960.5	0.40	948.1	0.44	970.5	0.77	962.8	0.18
LA38_09	(962)	975.9	0.43	963.7	0.31	981.3	0.80	987.0	2.86
LA38_10	(979)	1004.8	0.73	999.4	0.95	987.8	0.48	1012.1	0.64
LA38_11	(979)	1008.0	0.67	996.3	0.32	985.5	0.52	1015.6	0.87
LA38_12	(943)	961.7	0.27	945.9	0.52	951.9	0.33	965.5	0.59
LA38_13	(943)	955.6	0.74	943.0	0.00	943.0	0.00	946.6	2.70
LA38_14	(943)	950.8	0.31	943.0	0.00	943.0	0.00	943.6	2.71
LA38_15	(943)	953.3	0.79	943.0	0.00	943.0	0.00	943.6	2.36

Table 4.4 (Cont.): Comparison of the Algorithms' Average Case Performance

INSTANCE	BEST KNOWN MAKESPAN	ALGORITHM I		ALGORITHM II		ALGORITHM III		ALGORITHM IV	
		S_{AVG}	c_v	S_{AVG}	c_v	S_{AVG}	c_v	S_{AVG}	c_v
ABZ7_01	656	678.6	0.72	671.0	0.46	683.3	0.47	668.4	1.59
ABZ7_02	(641)	650.1	0.48	641.0	0.00	645.0	0.47	641.0	1.96
ABZ7_03	(612)	616.3	0.71	612.0	0.00	612.8	0.47	612.3	0.48
ABZ7_04	(609)	611.5	0.10	609.0	0.00	609.0	0.00	616.0	0.77
ABZ7_05	(638)	638.5	0.15	638.0	0.00	638.0	0.00	640.7	0.21
ABZ7_06	(600)	602.7	0.60	600.0	0.00	600.0	0.00	601.5	0.61
ABZ7_07	(567)	583.2	0.44	567.0	0.14	569.9	0.00	571.0	0.20
ABZ7_08	(577)	574.6	0.69	577.0	0.40	577.0	0.00	596.3	0.30
ABZ7_09	(577)	582.6	0.66	577.0	0.70	577.0	0.00	599.0	0.32
ABZ7_10	(612)	613.0	0.29	612.0	0.00	612.0	0.00	631.5	0.39
ABZ7_11	(610)	614.7	0.73	610.0	0.00	610.0	0.00	631.6	4.74
ABZ7_12	(592)	596.0	0.69	593.1	0.64	592.0	0.00	606.0	0.49
ABZ7_13	(581)	586.5	0.29	583.7	0.95	581.0	0.00	584.0	0.55
ABZ7_14	(562)	563.2	0.32	562.0	0.00	562.0	0.00	566.4	0.42
ABZ7_15	(556)	556.0	0.00	556.0	0.00	556.0	0.00	556.4	3.44
WHIZZKID	469	476.9	1.34	478.2	1.63	494.6	0.84	479.8	3.88

By comparing the results obtained by the GSP Scheduler with that of Blum's ACO Algorithm and Sampel *et al.*'s TS Algorithm, the GSP Scheduler outperforms them in most problem instances in three aspects: the best solution found, the coefficient of variation and the average computational time. Particularly, the solutions obtained via the GSP Scheduler were of comparable quality with respect to those obtained via the Liu *et al.*'s TS Algorithm. However, in most cases, the GSP Scheduler requires less computational time than other known GSP algorithms. Although the coefficients of variation for a number of the solutions obtained by the GSP Scheduler tend to be higher that that of the Liu *et al.*'s TS Algorithm, they are still within the acceptable margin of less than 1.0% for most cases.

Another interesting observation that was made in this computational experiment, which is in line with the observation made in Sampel *et al.* (2002), is that as the structure of the problem instance approaches that of OSP, it tends to be easier to solve. This is evident from the lower computational times and lower coefficients of variations associated with those problem instances with higher group lengths. Since the structure of the WHIZZKID problem is quite similar to that of a JSP, it is considered one of the toughest problems in the set of benchmarks. Therefore, it is not surprising that the optimal solution is only obtained by the GSP Scheduler and the Liu *et al.*'s TS Algorithm.

In essence, the computational experiments demonstrated that the GSP Scheduler is not only efficient but also robust as it can be applied to the MSP, OSP and JSP successfully.

4.4 Effect of Fitness Function on Algorithm Performance

Most algorithms, especially simple iterative search methods, face the problem of being unable to navigate their search out of either 'plateaus' or local optima as the choice of their next algorithm step is often based on the fitness of their current point as well as the corresponding fitness of the neighboring points. To circumvent such problems, Duvivier *et al.* (1991) advocated the use of enriched fitness function so as to improve solution quality. In particular, Duvivier *et al.* (1991) proposed that the enriched fitness function should consist of a main criterion and a secondary criterion. This will allow any algorithm to differentiate solution points that have the same value for the main criterion.

As a separate trial on the GSP Scheduler, the fitness function, as recommended by Duvivier *et al.* (1991), was incorporated into the *SelectDModelmoves* Procedure within the intensification phase of the GSP Scheduler leaving the rest of the algorithm intact. The assumption for this is that the GSP Scheduler is likely to be navigating either on 'plateaus' or on local optima during this phase. Therefore, the use of the enriched fitness function within this phase of the GSP Scheduler may help to improve the overall algorithm performance. The fitness function used is outlined as follows:

$$f(x) = K_2 \times K_1 \times C_{\max}(x) + K_2 \times H_2(x) + C_{op}(x) \tag{4.3}$$

where x denotes a schedule, $K_1 = M \times D$ denotes the product of the number of machines and the sum of all durations of all the operations of the instance, $K_2 = J \times M$ denotes the product of the number of machines and the number of jobs, $C_{\max}(x)$ denotes the makespan value of the schedule, $H_2(x)$ denotes the summation of the square of the completion time of the last operation performed on each machine and $C_{op}(x)$ denotes the number of critical operations found in the schedule.

Using the same experimental settings, the GSP Scheduler was applied on all the 41 GSP instances. Table 4.5 summarizes the results of this trial by computing the average percentage improvements achieved in terms of $\mathbf{S_{BST}}$, $\mathbf{S_{AVG}}$, $\mathbf{T_{AV}}$ and $\mathbf{c_v}$ for the different families of problem instances by the GSP Scheduler using the fitness function against the results obtained by the original algorithm without fitness function in Section 4.3.

Table 4.5: Effect of Fitness Function on GSP Scheduler's Performance

PROBLEM INSTANCE FAMILY	AVERAGE IMPROVEMENT PERCENTAGE (%)			
	S_{BST}	S_{AVG}	T_{AVG}	c_v
FT10	0.29	0.11	18.7	-5.2
LA38	-0.10	-0.06	35.1	18.6
ABZ7	0.01	0.02	20.2	4.5
WHIZZKID	0.00	-0.07	19.0	-2.7

The results did not provide substantial evidence to indicate that there are significant improvements in terms of overall solution quality. Moreover, since the computation of the fitness value requires that other information of the amended schedule to be known, makespan estimation techniques cannot be used in

conjunction with fitness function in the intensification phase. Consequently, the average computational times required by the GSP Scheduler, which uses the fitness function, increased rather significantly.

4.5 Concluding Remarks

In this chapter, extensive experiments were carried out to gauge the performance of the GSP Scheduler on a set of benchmark problems. As other known GSP scheduling algorithms for GSP have used this set of problems for similar computational experiments, these results can be compared to determine the relative strengths / weaknesses of the respective scheduling algorithms for GSP. Overall, the GSP Scheduler outperforms Sampel *et al.*'s TS Algorithm and Blum's ACO Algorithm in terms of solution quality and computational time. At the same time, the performance of the GSP Scheduler is also comparable to that of Liu *et al.*'s TS Algorithm as it is able to produce solutions of similar quality but with a shorter processing time. Notwithstanding that, unlike the Liu *et al.*'s TS Algorithm, the GSP Scheduler does not utilize elaborate neighborhood definitions, which in turn requires extensive tabu list structures for machine routing / job routing. Therefore, the GSP Scheduler is easier to implement for real life application compared to the Liu *et al.*'s TS Algorithm.

Chapter 5 Conclusions

5.1 Overview

In this present study, a methodology has been proposed to solve the GSP. Through the generalization of known findings about the distribution of local optima within the search space of JSP, a hybridized algorithm for solving GSP has been devised by incorporating known features of other popular meta-heuristics approach. The algorithm has been subsequently subjected to various computational experiments and the comparisons of empirical results have also been made with other known GSP algorithms to determine their relative strengths / weaknesses. This chapter concludes the thesis, by providing a brief review of the present study, demarcating its contributions, outlining the research issues addressed and finally proposing suggestions for future research directions.

5.2 Group Shop Scheduling: A Review

A typical shop scheduling algorithm can generally be perceived to consist of two stages, namely, the schedule construction stage and the schedule improvement stage. Most schedulers utilize a variety of techniques, which can range from simple dispatching functions to complex meta-heuristics, for both stages. In the case of the GSP scheduler, the schedule improvement can be further divided into the diversification phase and intensification phase. Likewise, many useful

features of other well-known meta-heuristics have been incorporated as part of the GSP Scheduler.

In the implementation of shop scheduling algorithms, most state-of-the-art methods perform local search on the schedule. Particularly, in the cases where the objective is to minimize schedule makespan, neighborhood moves for local search are often selected from either machine blocks or group blocks on the critical paths of schedules. While a more restrictive neighborhood definition will reduce the number of non-improving moves for the scheduling algorithm, it also has the effect of increasing the dependency of the final solution quality on the starting solution. Thus, the choice of neighborhood structures to use for local search and the selection mechanism for local moves are both crucial steps in the design of good GSP scheduling algorithms.

With the exception of Blum's ACO and Sampels *et al.*'s TS, neighborhood move estimation techniques were utilized during local search for selecting feasible moves on a critical path as opposed to the exact computation of schedule makespan corresponding to each of the possible moves prior to move selection. As such techniques have shown to be more computationally efficient, they have been incorporated as part of the GSP Scheduler.

5.3 Main Contribution Of The Present Study

The main contribution of the present study is the revelation of the potential and possibility of developing an algorithm that is both scalable in its applications, robust in its performance on a wide range of GSP instances and relatively easy to implement. In comparison with computational results of other known approaches for solving GSP problems, the proposed algorithm is able to produce solutions of comparable quality but with shorter processing times. Notwithstanding that, the proposed algorithm is easier to implement as it requires less elaborate neighborhood definitions and less complex memory structures as opposed to the Liu *et al.*'s algorithm (2005).

This study also demonstrates how known results about the search space of JSP can be utilized for the implementation of GSP scheduling algorithm. Moreover, it highlights the importance of devising the search strategy of the algorithm to suit the underlying characteristics of the search space so that maximum effectiveness can be achieved. Apart from implementing an array of techniques from existing meta-heuristics, new features, such as the probabilistic rule for selecting the blocks for reshuffling, have also been developed.

5.4 Future Work

Although research efforts should continue to be directed towards the improvement of the tactical aspects of GSP scheduling algorithms, such as the development of more efficient lower bound estimates of GSP as well as the development of more powerful neighborhood structures, two potential areas remains to be explored, namely: (1) the hybridization of GA technique with the GSP Scheduler and (2) the re-deployment of the fitness functions within the GSP Scheduler. Since GAs are generally known to be good intensification methods, the incorporation of GA technique into the intensification phase of the GSP Scheduler may improve the algorithm performance in terms of solution quality. As for the re-deployment of the fitness function, it may be worthwhile to consider the use of fitness functions in conjunction with the hybridization of GA technique with the GSP Scheduler as fitness functions may be more suitable for population-based meta-heuristics, such as GA.

From a broader perspective, two key research directions may become more dominant in future. Firstly, more search space analysis should be performed on shop scheduling problems, particularly GSP, to unveil further properties of the GSP landscape so as to improve scheduling algorithm design. Particularly, further research studies may wish to consider investigating the relationship between the properties of GSP landscape, such as the distribution of local optima, and the parameters of GSP instances, such as the group length. Notwithstanding

that, the development of methodologies for scheduling dynamic and stochastic GSP remains an area of immense potential as these methodologies will provide practitioners with more relevant tools for industrial applications.

References

[1] Aarts E. and Lenstra J.K., editors, (1997), Local Search in Combinatorial Optimization, Wiley-Interscience.

[2] Adam J., Balas E. and Zawack D. (1988), The Shifting Bottleneck Procedure for Job Shop Scheduling, Management Science, Vol. 34, pp. 391 – 401.

[3] Amar A. D. and Gupta J. N. D. (1986), Simulated versus Real Life Data Testing the Efficiency of Scheduling Algorithms, IIE Transactions Vol. 18, pp. 16 – 25.

[4] Amico D. and Trubian M. (1993), Applying Tabu Search to the Job Shop Scheduling Problem, Annals of Operations Research, Vol. 41, pp. 231 – 252.

[5] Applegate D. and Cook W. (1991), A Computational Study of the Job-Shop Scheduling Problem, ORSA Journal on Computing, Vol. 3, pp. 149 – 156.

[6] Ashby J. R. and Uzsoy R. (1995), Scheduling and Order Release in a Single-stage Production System, Journal of Manufacturing, Vol. 14, pp. 290 – 306.

[7] Bar-Yam Y. (1997), Dynamics of Complex Systems, Studies in Nonlinearity. Addison-Wesley.

[8] Bellman R. E. (1958), On A Routing Problem, Quarterly Applied Mathematics, Vol. 16, pp. 87 – 90.

[9] Blum C. (2003), An ACO Algorithm to tackle Shop Scheduling Problems, Submitted to European Journal of Operational Research.

[10] Boese K. D., Kahng A. B. and Muddu S. (1994), A New Adaptive Multi-Start Technique for Combinatorial Global Optimization, Operations Research Letters, Vol. 16, pp. 101 – 113.

[11] Brucker P., Jurisch B. and Sievers B. (1994), A Branch and Bound Algorithm for Job Shop Scheduling Problem, Discrete Applied Mathematics, Vol. 49, pp. 107 – 129.

[12] Brucker P. (1998), Scheduling Algorithms (2nd Edition), Springer Verlag, Berlin.

[13] Carlier J. and Pinson E. (1989), An Algorithm for Solving the Job Shop Problem, Management Science, Vol. 35, pp. 164 – 176.

[14] Colletti B. W. and Barnes J. (2000), Linearity in the Traveling Salesman Problem, Applied Mathematics Letters, Vol. 13, pp. 27 - 32.

[15] Colorni A., Dorigo M., Maffioli F., Maniezzo V., Righini G. and Trubian M. (1996), Heuristics from Nature for Hard Combinatorial Optimization Problems, International Transactions in Operational Research, Vol. 3, pp. 1 – 21.

[16] Colorni A., Dorigo M., Maffioli F., Maniezzo V. and Trubian M. (1993), Ant Systems for Job-shop Scheduling, Belgian Journal of Operations Research, Statistics and Computer Science, Vol. 34 (1), pp. 39 – 54.

[17] Coy S. P., Golden B. L., Runger G. C. and Wasil E. A. (1998), See The Forest Before The Trees: Fine-Tuned Learning And Its Applications To The Traveling Salesman Problem, IEEE Transactions on Systems, Man and Cybernetics, Part A, Vol. 28 (4), pp. 454 – 464.

[18] Croes G. A. (1958), A Method for Solving Traveling Salesman Problem, Operations Research, Vol. 6, pp. 791 – 812.

[19] Devaney R. L. (1989), An Introduction to Chaotic Dynamical Systems, Addison-Wesley, Second Edition.

[20] Dorigo M. and Stützle T. (2004), Ant Colony Optimization, The MIT Press.

[21] Duvivier D., Preux Ph., Fonlupt C. and Robilliard D. (1991), The Fitness Function And Its Impact On Local Search Methods, IEEE International Conference on Systems, Man and Cybernetics, Vol. 3, pp 2478 – 2483.

[22] Fisher H. and Thompson G. L. (1963), Probabilistic Learning Combinations of Local Job Scheduling Rules, Industrial Scheduling, Prentice Hall, Englewood Cliffs, NJ, pp. 225 – 251.

[23] Floudas C. A. and Pardalos P. M. (2001), Stochastic Scheduling, Encyclopedia of Optimization, Vol. 5, pp. 367 – 372.

[24] Giffler B. and Thompson G. L. (1960), Algorithms for Solving Production Scheduling Problems, Operations Research, Vol. 8, pp. 487 – 503.

[25] Glover F. (1986), Future Paths for Integer Programming and Links to Artificial Intelligence, Computer and Operations Research, Vol. 13, pp. 533 – 549.

[26] Glover F. (1989), Tabu Search – Part I, ORSA Journal on Computing, Vol. 1, pp. 190 – 206.

[27] Glover F. (1990), Tabu Search – Part II, ORSA Journal on Computing, Vol. 2, pp. 4 – 32.

[28] Graham R. L., Lawler E. L., Lenstra J. K. and Rinnooy Kan A. H. G. (1979), Optimization and Approximation in Deterministic Scheduling and Sequencing: A Survey, Annals of Discrete Mathematics, pp. 287 – 326.

[29] Grover L. K. (1992), Local Search and the Local Structure of NP-complete Problems, Operations Research Letters, Vol. 12, pp. 235 – 243.

[30] Gu J. and Huang X. (1994), Efficient Local Search with Search Space Smoothing: A Case Study Of The Traveling Salesman Problem (TSP), IEEE Transactions on Systems, Man and Cybernetics, Vol. 24 (5), pp. 728 – 735.

[31] Gutin G. and Yeo A. (2001), TSP Tour Domination and Hamilton Cycle Decompositions of Regular Digraphs, Operations Research Letters, Vol. 28, pp. 107 – 111.

[32] Handfield R. B. and Nichols E. L. (2002), Supply Chain Redesign – Transforming Supply Chains into Integrated Value Systems, Prentice Hall.

[33] Jain A. S. and Meeran S. (1999), Deterministic Job-Shop Scheduling: Past, Present and Future, European Journal of Operational Research, Vol. 13, pp. 390 – 434.

[34] Jain A. S., Rangaswamy B. and Meeran S. (2000), New and 'Stronger' Job-Shop Neighborhoods: A Focus on the Method of Nowicki and Smutnicki (1996), Journal of Heuristics, Vol. 6, pp. 457 – 480.

[35] Johnson S. M. (1954), Optimal two- and three-stage production schedules with setup times included, Naval Research Logistics Quarterly, Vol. 1, pp. 61 - 68.

[36] Johnson D. S., Papadimitriou C. H. and Yannakakis M. (1988), How Easy is Local Search?, Journal of Computer System Science, Vol. 37, pp. 79 – 100.

[37] Karp R. M. (1972), Reducibility among Combinatorial Problems, Miller R. E. and Thatcher J. W., editors, Numerical Solution of Markov Chains, Plenum Press, New York.

[38] Kirkpatrick S., Gelatt C. D. and Vecchi M. P. (1983), Optimization by Simulated Annealing, Science, Vol. 220, pp. 671 – 680.

[39] Laporte G. (1991). The Vehicle Routing Problem: An Overview of Exact and Approximate Algorithms, Montréal : Centre for Research on Transportation.

[40] Laarhoven Van P. J. M. and Aarts E. H. L. (1989), Simulated Annealing: Theory and Applications, Mathematics and its Applications Series, Kluwer Academic Publishers, Dordrecht, The Netherlands.

[41] Laarhoven Van P. J. M, Aarts E. H. L and Lestra J. K. (1992), Job Shop Scheduling By Simulated Annealing, Operations Research, Vol. 22, pp. 629 – 638.

[42] Lawler E. L., Lenstra J. K., Rinnooy Kan A. H. G., and Shmoys D. B., editors, (1985), The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, John Wiley & Sons.

[43] Lawrence S. (1984), Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Technique, GSIA, Carnegie Mellon University.

[44] Lenstra J. K., Rinnooy Kan A. H. G. and Brucker P. (1977), Complexity Machine Scheduling Problems, Annals of Discrete Mathematics, Vol. 1, pp. 343 – 362.

[45] Lenstra J. K. and Rinnooy Kan A. H. G. (1979), Computational Complexity of Discrete Optimization Problems, Annals of Discrete Mathematics, Vol. 4, pp. 121 – 140.

[46] Liaw C. F. (1999a), A Tabu Search Algorithm for the Open Shop Scheduling Problem, Computers & Operations Research, Vol. 26, pp. 109 – 126.

[47] Liaw C. F. (1999b), Applying Simulated Annealing to the Open Shop Scheduling Problem, IIE Transactions, Vol. 31, pp. 457 – 465.

[48] Liaw C. F. (2000), A Hybrid Genetic Algorithm for the Open Shop Scheduling Problem, European Journal of Operational Research, Vol. 124, pp. 28 – 42.

[49] Liu S. Q. and Ong H. L. (2002), A Comparative Study of Algorithms for the Flow Shop Problem, Asia Pacific Journal of Operational Research, Vol. 19, pp. 205 – 222.

[50] Liu S. Q., Ong H. L. and Ng K. M. (2005), A Fast Tabu Search Algorithm for Group Shop Scheduling Problem, Advances in Engineering Software, Vol. 36, pp. 533 – 539.

[51] Martin O., Otto S. W. and Felten E. W. (1992), Large-step Markov Chains for TSP incorporating Local Search Heuristics, Operations Research Letters, Vol. 11, pp. 219 – 224.

[52] Masuda T., Ishii H. and Nishida T. (1985), The Mixed Shop Scheduling Problem, Discrete Applied Mathematics, Vol. 11, pp. 175 – 186.

[53] Matsuo H., Shu C. J., Sullivan R. S. (1988), A Controlled Search Simulated Annealing Method for the General Jobshop Scheduling Problem, Working

paper 03-04-88, Dept. of Management, Graduate School of Business, University of Austin, Texas.

[54] Mattfeld D. C., Bierwirth C. and Kopfer H. (1999), A Search Space Analysis of the Job Shop Scheduling Problem, Annals of Operations Research, Vol. 86, pp. 441 – 453.

[55] Mladenovic N. and Hansen P. (1997), Variable Neighborhood Search, Computers Operations Research, Vol. 24, pp. 1097 – 1100.

[56] Mladenovic N. and Hansen P. (2001), Variable Neighborhood Search: Principles and Applications, European Journal of Operations Research, Vol. 130, pp. 449 – 467.

[57] Nakano R. and Yamada T. (1991), Conventional Genetic Algorithm for Job Shop Scheduling Problems, Proceedings of the 4th International Conference on Genetic Algorithms and their Applications, pp. 474 – 479.

[58] Nowicki E. and Smutnicki C. (1996a), A Fast Taboo Search Algorithm for the Job Shop Problem, Management Science, Vol. 42, pp. 797 – 813.

[59] Nowicki E. and Smutnicki C. (1996b), A Fast Taboo Search Algorithm for the Permutation Flowshop Problem, European Journal of Operational Research, Vol. 91, pp. 160 – 175.

[60] Ogbu F. A. and Smith D. K. (1990), Simulated Annealing for the Permutation Flowshop Problem, OMEGA, Vol 19, pp. 64 – 67.

[61] Osman I. H. and Potts C. N. (1989), Simulated Annealing for Permutation Flowshop Scheduling, OMEGA, Vol. 17, pp. 551 – 557.

[62] Panwalkar S. S. and Iskander W. (1977), A Survey of Scheduling Rules, Operations Research, Vol. 25, pp. 45 – 65.

[63] Papadimitriou C. H. and Steiglitz K. (1982), Combinatorial Optimization - Algorithms and Complexity, Dover Publications, Inc., New York.

[64] Papadimitriou C. H. (1993), Computational Complexity, Addison Wesley.

[65] Pinedo M. (2002), Scheduling: Theory, Algorithms and Systems, Prentice Hall, pp. 21 to 25.

[66] Porter D. B. (1968), The Gantt Chart as applied to Production Scheduling and Control, Naval Research Logistics Quarterly, Vol. 15, pp. 311 – 317.

[67] Prins C. (2000), Competitive Genetic Algorithm for the Open Shop Scheduling Problem, Mathematical Methods of Operations Research, Vol. 52, pp. 389 – 411.

[68] Ramudhin A. and Marier P. (1996), The Generalized Shifting Bottleneck Proecdure, European Journal of Operational Research, Vol. 93, pp. 34 – 48.

[69] Reeves C. R., editor, (1994), Modern Heuristics Techniques for Combinatorial Problems, Advanced Topics in Computer Science Series, Blackwell Scientific Publications.

[70] Reeves C. R. (1995), A Genetic Algorithm for Flowshop Sequencing, Computers & Operations Research, Vol. 22, pp. 5 – 13.

[71] Reeves C. R. (1999), Landscapes, Operators and Heuristics Search, Annals of Operations Research, Vol. 86, pp. 473 – 490.

[72] Righter R. (1994), Stochastic Scheduling, Academic Press, San Diego.

[73] Rock H. and Schmidt G. (1983), Machine Aggregation Heuristics in Shop-Scheduling, Methods of Operations Research, Vol. 45, pp. 303 – 314.

[74] Roy B. and Sussmann B. (1964), Les Problèmes D`ordonnancement Avec Contraintes Disjonctives, SEMA, Note D.S. No. 9, Paris.

[75] Sampels M., Blum C., Mastrolilli and Rossi-Doria O. (2002), Metaheuristics for Group Shop Scheduling, Proceedings from Parallel Problem Solving from Nature - PPSN VII: 7th International Conference, Granada, Spain, Vol. 2439/2002, pp. 631 – 640.

[76] Schneider J., Dankesreiter M., Fettes W., Morganstern I., Schmid M. and Singer J.M. (1997), Search Space Smoothing for Combinatorial Optimization Problem, Physica A, Vol. 243, pp. 77 – 112.

[77] Shakhlevich N. V., Sotsko Y. N. and Werner F. (1999), Shop Scheduling Problems with Fixed and Non-Fixed Machine Orders of the Jobs, Annals of Operations Research, Vol. 92, pp. 281 – 304.

[78] Shakhlevich N. V., Sotsko Y. N. and Werner F. (2000), Complexity of Mixed Shop Scheduling Problems: A Survey, European Journal of Operational Research, Vol. 120, pp. 343 – 351.

[79] Shmoys D. B., Stein C. and Wein J. (1994), Improved Approximation Algorithms for Shop Scheduling Problems, SIAM Journal on Computing, Vol. 23, pp. 617 – 632.

[80] Stadler P. F. (1995), Towards A Theory Of Landscapes, Technical Report SFI-95-03-030, Santa Fe Institute.

[81] Strusevich V. A. (1991), Two Machine Super Shop Scheduling Problem, Journal of Operational Research Society, Vol. 42, 479 – 492.

[82] Sun D., Batta R. and Lin L. (1995), Effective Job Shop Scheduling Through Active Chain Manipulation, Computers Operations Research, Vol. 22, pp. 159 - 172.

[83] Taillard E. (1990), Some Efficient Heuristic Methods for the Flowshop Sequencing Problem, European Journal of Operational Research, Vol. 47, pp. 65 – 74.

[84] Taillard E. (1994), Parallel Taboo Search Techniques for the Job Shop Scheduling Problem, Operations Research Society of America Journal on Computing, Vol. 6, No. 2, pp. 108 – 117.

[85] Vaessens R. J. M., Aarts E. H. L. and Lenstra J.K. (1996), Job Shop Scheduling By Local Search, INFORMS Journal on Computing, Vol. 8, pp. 302 – 317.

[86] Voudouris C. and Tsang E. (1995), Guided Local Search, Technical Report CSM-247, Department of Computer Science, University of Essex.

[87] Watson J. P., Beck J. C., Howe A. E. and Whitley L. D. (2003), Problem Difficulty for Tabu Search in Job-Shop Scheduling, Journal of Artificial Intelligence, Vol. 143, pp. 189 – 217.

[88] Weinberger E. (1990), Correlated And Uncorrelated Fitness Landscapes And How To Tell The Difference, Biological Cybernetics, Vol. 63, pp. 325 – 336.

[89] Winter G., Periaux J. and Galan M., editors, (1995), Genetic Algorithms in Engineering and Computer Science, John Wiley & Son Ltd.

Appendix

Proof of Theorem 2.1.

Let $\preceq^*$ *be a feasible solution to a GSP instance. If there is a solution* $\preceq^{*'}$ *with* $C_{\max}\left(\preceq^{*'}\right) < C_{\max}\left(\preceq^*\right)$, *then there is a machine block or group block* $\chi^i = o^i_{k_i} \succ\succ^{*'} \ldots \succ\succ^{*'} o^i_1$, *where* k_i *denotes the number of operations in* χ^i, *in the critical path* η^c *such that* $\exists o \in \chi^i$, $o \neq o^i_1$ *with* $o^i_1 \succ\succ^{*'} o$ *or* $\exists o \in \chi^i$, $o \neq o^i_{k_i}$ *with* $o \succ\succ^{*'} o^i_{k_i}$.

Proof:

Let η^c *be a critical path in* $\preceq^*$ *such that* $\chi^i_M = o^i_{m_i} \succ\succ^{*'} \ldots \succ\succ^{*'} o^i_1$ *denote the* i*-th machine block on* η^c *while* $\chi^j_G = o^j_{g_j} \succ\succ^{*'} \ldots \succ\succ^{*'} o^j_1$ *denote the* j*-th group block on* η^c. *Moreover, let* k_M *and* k_G *represent the total number of machine blocks and group blocks respectively.*

Assume that if there is a feasible solution $\preceq^{*'}$ *with* $C_{\max}(\preceq^{*'}) < C_{\max}(\preceq^{*})$ *and no operation of any machine block or group block of* η^c *is in* $\preceq^{*'}$ *processed before the first operation of the corresponding block or after the last operation of the corresponding block, then the relation* $\preceq^{*'}$ *must contain:*

$$\forall i \in \{1,\ldots,k_M\}$$

$$o_1^i \preceq^{*'} o_l^i \;\; \forall l \in \{1,\ldots,m_i\} \text{ and } o_l^i \preceq^{*'} o_{m_i}^i \;\; \forall l \in \{1,\ldots,m_i\}$$

$$\forall j \in \{1,\ldots,k_G\}$$

$$\bar{o}_1^j \preceq^{*'} \bar{o}_l^j \;\; \forall l \in \{1,\ldots,g_i\} \text{ and } \bar{o}_l^j \preceq^{*'} \bar{o}_{g_i}^j \;\; \forall l \in \{1,\ldots,g_i\}$$

Thus, $\preceq^{*'}$ *contains an active chain*

$$o_{m_{k_M}}^{k_M} \succ\!\succ^{*'} \ldots \succ\!\succ^{*'} u_2^{k_M} \succ\!\succ^{*'} o_1^{k_M} \succ\!\succ^{*'} \ldots \succ\!\succ^{*'} o_{m_1}^1 \succ\!\succ^{*'} u_{m_1-1}^1 \succ\!\succ^{*'} \ldots u_2^1 \succ\!\succ^{*'} o_i^1$$

where $u_2^i,\ldots,u_{m_i-1}^i$ *is a permutation of* $o_2^i,\ldots,o_{m_i-1}^i$*, and an active chain*

$$\bar{o}_{g_{k_G}}^{k_G} \succ\!\succ^{*'} \ldots \succ\!\succ^{*'} \bar{u}_2^{k_G} \succ\!\succ^{*'} \bar{o}_1^{k_G} \succ\!\succ^{*'} \ldots \succ\!\succ^{*'} \bar{o}_{g_1}^{-1} \succ\!\succ^{*'} \bar{u}_{g_1-1}^{-1} \succ\!\succ^{*'} \ldots \succ\!\succ^{*'} \bar{u}_2^{-1} \succ\!\succ^{*'} \bar{o}_i^{-1}$$

where $\bar{u}_2^j,\ldots,\bar{u}_{g_j-1}^j$ *is a permutation of* $\bar{o}_2^j,\ldots,\bar{o}_{g_j-1}^j$.

By identifying $u_1^i = o_1^i$, $u_{m_i}^i = o_{m_i}^i$, $\bar{u}_1^j = \bar{o}_1^j$ *and* $\bar{u}_{g_j}^j = \bar{o}_{g_j}^j$,

it leads to

$$C_{\max}\left(\preceq^{*'}\right)=\sum_{i=1}^{k_M}\sum_{l=1}^{m_i}\lambda\left(o_l^i\right)=\sum_{i=1}^{k_M}\sum_{l=1}^{m_i}\lambda\left(u_l^i\right)=C_{\max}\left(\preceq^{*}\right)$$

and

$$C_{\max}\left(\preceq^{*'}\right)=\sum_{j=1}^{k_G}\sum_{l=1}^{g_j}\lambda\left(\overline{o}_l^j\right)=\sum_{j=1}^{k_G}\sum_{l=1}^{g_j}\lambda\left(\overline{u}_l^j\right)=C_{\max}\left(\preceq^{*}\right)$$

which is a contradiction to the assumption.