MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

AN ANT COLONY OPTIMIZATION APPROACH FOR THE PROPORTIONATE MULTIPROCESSOR OPEN SHOP

ZEYNEP ADAK

Ph.D. THESIS

Department of Industrial Engineering

Thesis Supervisor
Prof. Dr. Serol BULKAN

Thesis CO-Supervisor
Asst. Prof. Dr. M. Övül ARIOĞLU

ISTANBUL, 2020

MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

AN ANT COLONY OPTIMIZATION APPROACH FOR THE PROPORTIONATE MULTIPROCESSOR OPEN SHOP

ZEYNEP ADAK
(724416004)

Ph.D. THESIS
Department of Industrial Engineering

Thesis Supervisor
Prof. Dr. Serol BULKAN

Thesis CO-Supervisor
Asst. Prof. Dr. M. Övül ARIOĞLU

ISTANBUL, 2020

ACKNOWLEDGMENT

I would like to express my greatest appreciation and gratitude to my supervisor Dr. Serol Bulkan for his support and guidance throughout the preparation of this thesis.

I would also like to thank the committee members Dr. Çiğdem Alabaş Uslu and Dr. Ayla Gülcü for their comments, suggestions, and assistance, and the defense committee members Dr. Ayhan Demiriz and Dr. Bahar Sennaroğlu for their time and suggestions in general.

Finally, my sincere appreciation is extended to my mother Dr. Gülseren Adak and my father Dr. Burhan Adak for their tireless patience, continuous encouragement, and great understanding during the whole of my PhD training and this thesis study.

September, 2020 **Zeynep Adak**

TABLE OF CONTENTS

ÖZET

ORANTILI ESNEK AÇIK ATÖLYE TİPİ ÇİZELGELEME İÇİN KARINCA KOLONİSİ OPTİMİZASYONU YAKLAŞIMI

Atölye çizelgeleme problemleri imalat ve hizmet sektörlerinin her birinde son derece geniş uygulama alanlarına sahiptir. Esnek açık atölye tipi çizelgeleme yaygın görülen atölye ortamları arasındadır. Birden fazla işlem istasyonu içeren bu atölye tipinde bu istasyonlardan en az biri aynı işlemi yapan paralel tezgahlara sahiptir. Bu işlem istasyonlarında tamamlanması gereken n tane iş bulunur ve işlerin istasyonları ziyaret etmede uymaları gereken bir rota kısıtı yoktur. Bu atölye tipi özellikle tıbbi teşhis test süreçlerinde, onarım ve bakım hizmetlerinde, denetim ve kalite kontrol işlemleri ve elektronik üretim süreçlerinde yaygın olarak bulunmaktadır. Ancak, bu atölye tipini çizelgeleme problemi literatürde çok az ilgi görmüştür. Son yıllarda araştırmaların sayısında artış görülmekle beraber, alan önemli ölçüde geliştirilmeye muhtaçtır. Bu tez çalışmasında, orantılı esnek açık atölye tipi ele alınmıştır. Burada orantılı ifadesi işlem istasyonlarının işlem sürelerinin her istasyon için sabit ve işten bağımsız olmasını ifade eder. Bu atölye tipini çizelgeleme problemi için bir karınca kolonisi algoritması önerilmiştir. Önerilen algoritma probleme uygun yeni ve çok etkili bir çözüm gösterimini temel alır. Algoritma ayrıca rassal arama ve yerel tarama (yerel aramaya benzer) rutinleri içerir. Geçmiş arama tecrübesinin ve probleme özel bilginin algoritmada kuvvetli ve etkin kullanımı özelleştirilmiş feremon iz bilgisi ve sezgisel bilgi yoluyla sağlanmıştır. Önerilen algoritma literatürden alınan 100 problemli bir problem seti kullanılarak test edilmiştir. Yapılan karşılaştırmalar önerilen algoritmanın bu problem tipi için literatürdeki en iyi algoritma olan dağınık arama ve yeniden yol bağlama (scatter search with path relinking) algoritmasından hem çözüm kalitesi bakımından hem de süre bakımından daha iyi olduğunu göstermiştir. Algoritmanın büyük boyutlu problemlerdeki başarısı ve bu çözüm kalitesine daha kısa sürede ulaşması bilhassa önemlidir.

ABSTRACT

AN ANT COLONY OPTIMIZATION APPROACH FOR THE PROPORTIONATE MULTIPROCESSOR OPEN SHOP

Shop scheduling problems have exceptionally wide application fields both in manufacturing and service sectors. Multiprocessor open shop is among common shop environments and it consists of at least two machine centers with one or more center having parallel machines for the same task. There are n jobs to visit the centers without a predefined route. The shop widely exists particularly in diagnostic medical testing, repair and maintenance services, inspection and quality control operations and electronics manufacturing processes. However, the problem gained little attention in the literature. There has been an increase in the number of researches in the field in recent years but still there is considerable room for improvement. In this thesis study, the proportionate multiprocessor open shop problem was considered where proportionate feature refers to processing times of machine centers being fixed and independent of the job. An Ant Colony Optimization algorithm was proposed for the problem. The algorithm is based on a very efficient novel solution representation of the problem. The proposed algorithm further employs random exploration and local exploration (analogous to local search) routines. Exploitation of search knowledge and problem-specific knowledge was incorporated with tailored uses of pheromone information and heuristic information, respectively. The algorithm was tested on 100 benchmark instances from the literature. Comparisons showed that it outperformed the current state-of-the-art scatter search with path relinking algorithm both in solution quality and computational time. Of particular importance is its performance in large-scale instances and the relatively short time it required to reach the high-quality results.

CLAIM FOR ORIGINALITY

This thesis study has the following important contributions to multiprocessor open shop (MPOS) and ant colony optimization literatures:

1. A novel very efficient permutation representation, *implicit-stage permutation*, of a feasible solution of the proportionate multiprocessor open shop problem was developed.
2. An ant colony optimization (ACO) approach was proposed for the problem for the first time.
3. The ACO algorithm used a *random exploration routine* to search for good solution characteristics very rapidly. This random solution generation has been largely avoided in ACO algorithms in the literature due to typically inferior solution quality results. However, the novel solution representation enabled moderate-quality random solutions which helped in accumulating knowledge about favorable solution components. This random exploration phase resulted in a new approach in solution construction mechanism of ACO.
4. The ACO algorithm employed an adopted *Most Work Remaining Heuristic* as the heuristic information and enabled strong exploitation of problem-specific knowledge. This is the first time this heuristic was tailored to use in proportionate MPOS problem.
5. A *local exploration (LE) engine* was incorporated in the algorithm. It served similar purposes as a local search but was not a local search algorithm. No neighborhood function was used. It mainly generated several different schedules around a single permutation. The proposed LE routine is a powerful approach in schedule generation from a permutation and it can be utilized in many different heuristic and metaheuristic scheduling algorithms.
6. The proposed algorithm was shown to be the new state-of-the-art algorithm for the proportionate MPOS problem considering the benchmark instances by Matta (2009). It reached new upper bounds and provably optimal solutions.
7. Lastly, a rational explanation about the optimality of results for 2-stage benchmark problem instances of Matta (2009) is supplied in this thesis study.

SYMBOLS

$\boldsymbol{a}$: Parameter in minimum pheromone limit

$\boldsymbol{\mathcal{A}}$: Set of ants in an ACO algorithm

$\boldsymbol{C_j}$: The time all operations of job j completed

$\boldsymbol{C_{max}}$: The time all operations of all jobs completed

$\boldsymbol{c_{best}}$: Cost of the best solution (global best or iteration best)

$\boldsymbol{c_{GB}}$: Cost of the best-so-far solution

$\boldsymbol{c_h}$: Cost of the solution generated by ant h

$\boldsymbol{c_{nn}}$: Cost of the TSP solution generated by Nearest Neighbor heuristic

$\boldsymbol{d_{\xi\varphi}}$: Distance between cities ξ and φ

$\boldsymbol{d_j}$: Desirability of job j

$\boldsymbol{f}(\cdot)$: Objective function of a problem

$\boldsymbol{h}$: Ant index

$\boldsymbol{i}$: Stage index

$\boldsymbol{j}$: Job index

$\boldsymbol{\mathcal{J}}$: Set of jobs

$\boldsymbol{k}$: Machine index

$\boldsymbol{L}$: Length of permutation

$\boldsymbol{m_i}$: Number of machines in stage i

$\boldsymbol{\mathcal{M}_i}$: Set of machines in stage i

$\boldsymbol{n}$: Number of jobs

$\boldsymbol{n_i}$: Number of remaining jobs still to be processed in stage i

N : Number of operations

$\boldsymbol{\mathcal{N}_\xi}$: Neighborhood of city ξ in TSP

$\boldsymbol{\mathcal{O}}$: Set of operations of all jobs

$\boldsymbol{\mathcal{O}_j}$: Set of operations of job j

$\boldsymbol{O_{ji}}$: Operation of processing of job j at stage i

$\boldsymbol{p_i}$: Processing time of stage i (proportionate shop)

$\boldsymbol{p_{ji}}$: Processing time of job j at stage i (identical parallel machines)

$\boldsymbol{p_{jik}}$: Processing time of job j at stage i in machine k

$\boldsymbol{p_{\xi\varphi}}$: Probability of moving to city φ while in city ξ in ACO for TSP

$\boldsymbol{q}$: A random number uniformly distributed between [0,1]

$\boldsymbol{q_0}$: Parameter to modulate exploration and exploitation level in ACS

$\boldsymbol{Q_{\xi\varphi}}$: Quality function

$\boldsymbol{r}$: Job index

$\boldsymbol{s}$: Number of stages

$\boldsymbol{s_{ji}}$: Start time of processing operation O_{ji}

$\boldsymbol{s_{\varphi}}$: Stage referred by implicit-stage representation φ

$\boldsymbol{S}$: Set of stages

$\boldsymbol{\vec{s}}$: Binary vector to represent a solution for a combinatorial problem

$\boldsymbol{\vec{s}_{best}}$: Solution vector for the best solution (global best or iteration best)

$\boldsymbol{t}$: Iteration index

$\boldsymbol{\mathcal{U}}$: Set of solutions generated in an iteration

$\boldsymbol{w}$: Stage index

$\boldsymbol{x_{jiw}}$: Binary variable that takes 1 if O_{ji} processed before O_{jw} and takes 0 otherwise

$\boldsymbol{y_{jik}}$: Binary variable that takes 1 if O_{ji} processed in machine k and takes 0 otherwise

$\boldsymbol{z_{jrik}}$: Binary variable that takes 1 if O_{ji} processed before O_{ri} in machine k and takes 0 otherwise

$\boldsymbol{\alpha}$: Parameter to determine influence of pheromone information

$\boldsymbol{\beta}$: Parameter to determine influence of heuristic information

$\boldsymbol{\gamma}$: Parameter in weighted pheromone summation rule

$\boldsymbol{\Delta\tau}$: Amount of added pheromone

$\boldsymbol{\eta_{\xi\varphi}}$: Heuristic information

H : Heuristic information matrix

θ : Parameter in local pheromone update

$\boldsymbol{\kappa}$: Number of ants in ACO algorithm

$\boldsymbol{\xi}$: Row index in pheromone matrix | City index | Position index in permutation

$\boldsymbol{\pi}$: A permutation (solution)

$\boldsymbol{\pi_{GB}}$: Global best permutation (solution)

$\boldsymbol{\pi_h}$: Permutation generated by ant h

$\boldsymbol{\rho}$: Evaporation rate

$\boldsymbol{\tau_{\xi\varphi}}$: Pheromone information

$\boldsymbol{\tau_{max}}$: Maximum pheromone limit

τ_{min} **:** Minimum pheromone limit

τ_0 **:** Initial pheromone values

T **:** Pheromone trails matrix

v **:** Number of cities in TSP

φ **:** Column index in pheromone matrix | City index | Implicit-stage representation

Φ **:** Random variable for the next solution component with random proportional rule probability distribution

ABBREVIATIONS

ACO **:** Ant Colony Optimization
ACS **:** Ant Colony System
AS **:** Ant System
CV **:** Coefficient of Variation
DE **:** Differential Evolution
FPTAS **:** Fully Polynomial Time Approximation Schemes
GA **:** Genetic Algorithm
GB **:** Global-Best
HCF **:** Hyper-Cube Framework
HPSO **:** Hybrid Particle Swarm Optimization
IB **:** Iteration-Best
ICA **:** Imperialist Competitive Algorithm
LB **:** Lower bound
LE **:** Local exploration
MA **:** Memetic Algorithm
MMAS **:** Max-Min Ant System
MILP **:** Mixed Integer Linear Programming
MIP **:** Mixed Integer Programming
MPOS **:** Multiprocessor Open Shop
MWRH **:** Most Work Remaining Heuristic
NN **:** Nearest neighbor heuristic for TSP
PTAS **:** Polynomial Time Approximation Scheme
SA **:** Simulated Annealing
SD **:** Standard Deviation
SS/PR **:** Scatter Search with Path Relinking
TS **:** Tabu Search
TSP **:** Travelling Salesman Problem

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

A shop is a collection of machines dedicated for certain tasks. A single machine shop is also possible and common. Additionally, operators in service sector are regarded as machines and the environment is modelled as a shop environment. Single machine, parallel machines, flow shop, job shop, open shop and numerous extensions of them are common shop settings one can face both in manufacturing and service facilities. Each differ mainly in the way jobs visit the machines. The general purpose in dealing with a machine shop is to create a feasible schedule that minimizes (or maximizes) one or more objectives. A schedule consists of the set of beginning and ending times for each machine to process each required job.

Open shop is a machine environment where jobs have no predefined routes to visit the machines. It is in contrary to a job shop -each job has its own route- and a flow shop -every job follows the same route. In an open shop, there are at least two machines each with its own task, and not all jobs are required to be processed by every machine. A machine can process a single job and a job can be processed in a single machine at a time. This form of an open shop is also referred as a classical open shop. Multiprocessor open shop (MPOS) is a generalization to the classical open shop and possesses machine centers (stages) that include parallel machines for the same task. It is also named as a flexible open shop, in line with its flexible job shop and flexible flow shop counterparts. MPOS was the machine environment considered in this study.

1.1. Application Areas of MPOS

MPOS environment is common in various industries. Health sector is a prominent one. Medical testing services carry out several tests on patients including X-ray exam, magnetic resonance imaging, computed tomography scan, blood draw, positron emission tomography scan, electrocardiogram, bone scan, echocardiogram, ultrasound imaging, bone survey, mammogram, pulmonary function test, barium enema and barium swallow (Matta, 2009). These examination processes also appear in emergency department laboratories (Azadeh et

al., 2014). The medical diagnostic process requires no order to take the tests, and some of the testing units may include more than one nurse/machine to carry out the test.

Another area of application for MPOS environment is automotive repair and maintenance shops. An auto garage serves several operations. Brake service and replacement; car electrical system; engine tune up and rebuild; wheel alignments; lube, oil and filter change; tire repair, rotation and change; transmission clutch service and cooling system service are among those operations. A car visits the repair shops in any order, and no two operations can be carried out at the same time on the car. Moreover, it is common to meet more than one processor (worker) at each specialized shop.

Electronics manufacturing, inspection and quality control operations are other industrial fields that one can encounter a MPOS setting.

1.2. Overview of Previous Research

Despite the wide application areas of the shop environment, the research on MPOS problem has been very limited and the field still requires to be explored and studied both theoretically and practically at various aspects. While referring to previous research, as well as the present study in MPOS scheduling, the 3-field notation by Graham et al. (1979) is used in this text. Definitions of this notation and of the shop features mentioned in this chapter can be found in Chapter 2.

The 2-stage nonpreemptive MPOS problem with l machines in each stage and makespan minimization objective, $O_2(P_l)||C_{max}$, was shown to be NP-complete even when there are 2 machines at each stage (Mao, 1995). Chen and Strusevich (1993) provided a heuristic approach for $O(P)||C_{max}$ that uses a heuristic for the parallel machine scheduling and a greedy heuristic for the classical open shop part of the problem. Their approximation algorithm had a worst-case bound of $1 + p_H$ with p_H being the worst-case bound of the heuristic used in parallel machine scheduling. They established a worst-case bound of strictly less than 2 for the 2-stage case. Schuurman and Woeginger (1999) showed that a dense schedule algorithm for the problem $O(P)||C_{max}$ had a worst-case ratio of 2. They also improved the worst-case ratio for the 2-stage case by providing a $(3/2 + \varepsilon)$ – approximation algorithm. Later, several

polynomial time approximation schemes (PTAS) were proposed for the problem. A PTAS is a $(1+\varepsilon)$ - approximation algorithm whose running time is polynomial on the size of the input. Following the PTAS by Jansen and Sviridenko (2000), an almost fully PTAS (FPTAS) -a PTAS whose running time is also polynomial on $1/\varepsilon$- was proposed by Sevastianov and Woeginger (2001) for $O(P)||C_{max}$. Kononov and Sviridenko (2002) took operation release times into account, $O(P)|r_{ji}|C_{max}$, in their PTAS.

Queyranne and Sviridenko (2002) provided new approximation bounds for $O(P)|r_j|\sum w_s C_s$ and $O(P)|r_j, pmtn|\sum w_s C_s$.

Lawler et al. (1982) considered preemptive MPOS with the objective of makespan minimization and with either single-operation or multiple-operation machines and provided optimal schedules for each case. They also considered machine speeds and provided a linear programming model for MPOS with unrelated parallel machines.

Matta (2009) considered a proportionate MPOS for the first time and provided two different mixed integer programming (MIP) formulations for the problem $O(P)|prop|C_{max}$. She developed a Genetic Algorithm (GA) and created a testbed of difficult instances. This testbed was later used to test a tabu search (TS) algorithm by Abdelmaguid et al. (2014) and a hybrid of the TS with particle swarm optimization (HPSO) by Abdelmaguid (2014) for the problem. Matta and Elmaghraby (2010), on the other hand, provided polynomial-time optimum solution algorithms for two very special classes of the problem having stages with balanced work-loads, $O(P)|bal., prop|C_{max}$. The proportionate shop was also studied by Zhang et al. (2019) to schedule medical examination laboratories, similar to Matta (2009), but with minimization of sum of job completion times objective, $O(P)|prop|\sum C_j$. The MIP model was supplied and GA, Simulated Annealing (SA) and HPSO algorithms were compared based on large-scale test instances adopted from flexible job shop instances of Hurink et al. (1994). GA was showed to perform better than the other two algorithms in terms of convergence and stability.

Abdelmaguid (2020) considered the general MPOS with unrelated parallel machines, $O(R)||C_{max}$, supplied a MIP formulation and developed a scatter search with path relinking (SS/PR) algorithm for its solution. New neighborhood search functions and solution

combination functions were also provided. The benchmark testbed of Matta (2009) was used to test the algorithm on proportionate MPOS with parallel identical machines which constituted a special case for the problem studied. The algorithm improved solution quality, at the expense of increased computational time.

The objective to minimize sum of job completion times was considered by Naderi et al. (2011), $O(P)||\sum C_j$. They proposed a MIP formulation, as well as a memetic algorithm (MA) for the problem. They showed a hybrid MA-SA to perform better than pure applications of each single metaheuristic in randomly generated small and large-sized instances.

Azadeh et al. (2014) modelled the scheduling of patients in an emergency department as a MPOS problem. They considered sum of weighted job completion times as the objective to minimize total waiting time of patients while taking the triage factor (urgency of a patient) into account. MIP model of the problem, $O(P)||\sum w_j C_j$, and a GA model were proposed. The GA increased the efficiency of the department compared to the actual system.

Goldansaz et al. (2013) considered independent set-up times and sequence-dependent removal times in a MPOS environment with makespan objective, $O(P)|ST_{sd}, ST_{si}|C_{max}$. A hybrid of imperialist competitive algorithm (ICA) with GA was proposed and tested on small to large-sized instances generated.

Bai et al. (2016) considered the MPOS problem with and without job release times to minimize makespan, $O(P)|r_j|C_{max}$ and $O(P)||C_{max}$, respectively. They employed the differential evolution (DE) algorithm for moderate scale problems and proved the asymptotic optimality of general dense scheduling algorithm for very large-scale ones.

Wang and Chou (2017) applied SA to a 4-stage problem with release times to minimize multi-objectives of makespan and total weighted tardiness, $O_4(P)\left|r_j\right|C_{max}, \sum w_j T_j$. They compared two types of SA in terms of Pareto-optimality using small and large-scale instances they generate.

More detailed information about the research on MPOS problem can be found in a recent review by Adak et al. (2020).

1.3. The Present Study

In this thesis study, the proportionate MPOS problem with the objective of makespan minimization, $O(P)|prop|C_{max}$, was considered. An Ant Colony Optimization (ACO) algorithm was proposed for the problem for the first time. A novel very efficient way of solution representation was developed and used in the algorithm. Solution construction phase of the algorithm included random complete solution generation as a new approach. This enabled random, hence fast, exploration of the solution space. Full and effective use of search knowledge (intensification or exploitation) was ensured with informed selection of pheromone information. Problem-specific knowledge was also incorporated with tailored heuristic information. At the end of the solution construction phase, a local exploration routine was proposed to generate different schedules from a single permutation. It had a similar effect on solution quality as a local search. The proposed algorithm was tested on Matta (2009) benchmark problem set and further analysis on the results were carried out. Lastly, performance of the proposed algorithm was compared with the current state-of-the-art algorithm for the problem in these instances.

1.4. Outline of Thesis

Multiprocessor open shop is defined in Chapter 2. Definitions, description of shop features and MIP formulation are supplied. Chapter 3 gives general information about ACO metaheuristic. MPOS environment studied in this thesis as well as the assumptions are stated in Chapter 4. In the same chapter, the proposed solution representation and the proposed algorithm are presented. Test results and analysis are given in Chapter 5. Comparison of the results with literature are also provided in that chapter. Chapter 6 further discuss the proposed algorithm and results. Lastly, Chapter 7 gives final conclusions about the study.

2. MULTIPROCESSOR OPEN SHOP PROBLEM

This chapter introduces a formal statement of a general MPOS problem. Related definitions and descriptions are given. An adapted MIP formulation from the literature is also supplied. The specific MPOS environment studied in this thesis is described and the associated assumptions are stated in Chapter 4.

2.1. Definition of a MPOS Environment

MPOS defines an environment with s stages and n number of jobs. A stage is a machine center with parallel machines. Stage i has m_i parallel machines and at least one stage has $m_i \geq 2$. This constitutes the multiprocessor part of the problem. Generally, the parallel machines in a stage can perform a single identical task. That is, a stage is responsible of a certain task type. There are rare studies, however, that allow machines in a stage to do tasks of other stages, called multi-purpose machines. The parallel machines in a stage may have different speeds. They are called *identical* if they run at the same speed, while machines with different speeds are called *uniform*. If the speed of the machines also depend on the job then the machines are named as *unrelated* machines. p_{jik} is the processing time of job j at stage i in machine k. If identical parallel machines are assumed, the processing time notation reduces to p_{ji}. Processing of job j at stage i is named as an operation and represented by O_{ji}. Jobs do not follow a predefined route to visit the stages, and this is the open part of the problem. Additionally, all jobs may not be required to visit all stages. It is generally assumed that one machine can process a single job, and a job can be processed by a single machine at a time.

2.2. Description of Several Shop Features

Several shop features that one can come across in MPOS research are defined in the following.

Preemption is the option to interrupt an operation in a machine and continue it on a later time not necessarily on the same machine. An environment that allows preemption is called *preemptive*. Otherwise, it is called *nonpreemptive*. Preemption brings flexibility to scheduling

of a shop environment and often allows polynomial-time optimum solution algorithms (see Section 1.2).

Table 2.1. Description of notations

α-field: Machine environment	
Notation	Description
$O(P)$	Multiprocessor open shop with identical machines in a stage
$O_2(P)$	Multiprocessor open shop having 2 stages, each with identical parallel machines
$O_2(P_l)$	Multiprocessor open shop having 2 stages, each with l identical parallel machines
$O_4(P)$	Multiprocessor open shop having 4 stages, each with identical parallel machines
$O(R)$	Multiprocessor open shop with unrelated parallel machines in a stage

β-field: Job characteristics	
Notation	Description
$pmtn$	Preemption allowed
r_j	Job j has release time r_j
r_{ji}	O_{ji} has release time r_{ji}
$prop$	Proportionate shop
$bal.$	Balanced shop
ST_{sd}	Sequence-dependent setup times
ST_{si}	Sequence-independent setup times

γ-fied: Optimality criterion	
Notation	Description
C_{max}	Makespan
$\sum C_j$	Sum of job completion times
$\sum w_j C_j$	Total weighted job completion time
$\sum w_s C_s$	Total weighted stage completion time
$\sum w_j T_j$	Total weighted tardiness

In a *proportionate* shop fixed stage processing times are present that do not change with job identity. That is, a stage processes any job in a fixed amount of time. Thus, the notation of p_{ji} further reduces to p_i. Certainly, in proportionate shops the parallel machines should be identical.

In a *balanced shop*, number of machines at each stage are fixed a priori so as to balance supply with demand. It is a construct introduced by Matta and Elmaghraby (2010) for a MPOS setting. It allows workload balance between stages and prevents any stage to constitute a bottleneck in the problem. Otherwise, such a stage would have decisive role in resulting makespan value.

Release time is the time when a job is allowed to start processing on any machine. No early than its release time a job can be considered for processing. Similarly, *operation release time* is also possible, and it restricts the earliest time an operation can be started. If release times are not of concern in an environment, then all jobs (/operations) are ready for processing at the start of the schedule.

Setup time is the time required for a machine to be prepared before it becomes ready for processing. It may be a cleaning process, a configuration, or various other operations on the machine. It is also named as independent setup time or sequence-independent setup time.

Sequence-dependent setup time or removal time is a similar setup time, but it depends on the exiting job from machine and the next entering job. Disengaging tools for a job, releasing a job from jigs and fixtures, dismantling fixtures, jigs and tools, inspecting and sharpening the tools are among common removal operations (Józefowska & Weglarz, 2006). Sometimes, these times are aggregated with the processing time of a job in the machine and the explicit removal times are ignored.

Although there are numerous other shop features modelled in shop scheduling problems, above-mentioned features are now sufficient for both reviewing the MPOS literature (see Section 1.2) and defining the environment in this study.

Once an environment is described with its features, one or more objective functions are considered while scheduling the shop. Most of previous research on the problem have been about minimizing *makespan*, the time all operations are completed. Total weighted stage completion times, weighted and unweighted sum of job completion times and total weighted tardiness have been other minimization objectives considered in MPOS literature.

The standard 3-field notation of Graham et al. (1979) used in this text is a shorthand representation of the shop environment and the objective function(s) of a problem at hand. The notation is in the form of $\alpha|\beta|\gamma$ where the α-field represents the machine environment, the β-field shows the job characteristics and the γ-fied is for the optimality criterion (objective function). Table 2.1 gives several notations that can appear in each of the fields and are relevant in this thesis study.

2.3. Mixed Integer Programming Formulation

Several mixed integer linear and nonlinear programming formulations were proposed in the literature for the MPOS problem with various environmental features. In this study, an adapted version of the Mixed Integer Linear Programming (MILP) formulation by Abdelmaguid (2020) for $O(R)||C_{max}$ is given. For other MIP mathematical models, the reader is referred to Matta (2009) for $O(P)|prop|C_{max}$, to Naderi et al. (2011) for $O(P)||\sum C_j$, to Goldansaz et al. (2013) for $O(P)|ST_{sd}, ST_{si}|C_{max}$, to Azadeh et al. (2014) for $O(P)||\sum w_j C_j$, and to Zhang et al. (2019) for $O(P)|prop|\sum C_j$.

Although, unrelated parallel machines were considered in the original MIP model, identical parallel machines are assumed in the formulation here. This is due to the environment studied in this thesis, and for the sake of a more clear and simple representation. Another distinction of the model presented here from the original one is that all jobs are assumed to visit all stages. Thus, sets of stages and jobs that defined processing requirements are removed in the present formulation.

The MILP model is presented next, where the sets, indexes, labels, parameters, and decision variables are defined first. It is a model for $O(P)||C_{max}$ MPOS setting. The proportionate property that is imposed in this study can be easily incorporated by simply making $p_{ji} = p_i$ in the model.

Sets:

$\mathcal{S}$	Set of stages
$\mathcal{J}$	Set of jobs

$\mathcal{O}_j = \{O_{j1}, O_{j2}, \dots, O_{jn}\}$ Set of operations of job $j \in \mathcal{J}$

$\mathcal{O} = \bigcup_{j\in\mathcal{J}} \mathcal{O}_j$ Set of operations of all jobs

$\mathcal{M}_i$ Set of machines in stage $i \in \mathcal{S}$

Indexes and labels:

i, w Stage indexes $= \{1, 2, \dots, s\}$; stages $i, w \in \mathcal{S}$

j, r Job indexes $= \{1,2, \dots, n\}$; jobs $j, r \in \mathcal{J}$

k Machine index, $= \{1,2, \dots, m_i\}$; stage $i \in \mathcal{S}$

O_{ji} Operation of processing job $j \in \mathcal{J}$ in stage $i \in \mathcal{S}$

Parameters:

p_{ji} Processing time of operation $O_{ji} \in \mathcal{O}$

Decision variables:

$$x_{jiw} = \begin{cases} 1 & O_{ji} \text{ is processed before } O_{jw} \\ 0 & \text{otherwise} \end{cases} \qquad j \in \mathcal{J};\ \ i, w \in \mathcal{S};\ \ w > i$$

$$y_{jik} = \begin{cases} 1 & O_{ji} \text{ is processed on machine } k \\ 0 & \text{otherwise} \end{cases} \qquad j \in \mathcal{J};\ \ i \in \mathcal{S};\ \ k \in \mathcal{M}_i$$

$$z_{jrik} = \begin{cases} 1 & O_{ji} \text{ is processed before } O_{ri} \text{ on machine } k \\ 0 & \text{otherwise} \end{cases} \qquad j, r \in \mathcal{J};\ i \in \mathcal{S};\ k \in \mathcal{M}_i$$

s_{ji} Start time of processing operation O_{ji}

C_j The time all operations of job $j \in \mathcal{J}$ completed

C_{max} The time all operations of all jobs completed; $= \max_{j\in\mathcal{J}} C_j$

Minimize C_{max} (2.1)

subject to:

$C_{max} \geq C_j \quad \forall j \in \mathcal{J}$ (2.2)

$C_j \geq s_{ji} + p_{ji} \quad \forall O_{ji} \in \mathcal{O}_j \quad \forall j \in \mathcal{J}$ (2.3)

$\sum_{k\in M_i} y_{jik} = 1 \qquad \forall O_{ji} \in \mathcal{O}$ (2.4)

$s_{ji} \geq s_{jw} + p_{jw} - \mathbb{M}x_{jiw} \quad \forall i, w \in \mathcal{S} : w > i \quad \forall j \in \mathcal{J}$ (2.5)

$s_{jw} \geq s_{ji} + p_{ji} - \mathbb{M}(1 - x_{jiw}) \qquad \forall i, w \in \mathcal{S} : w > i \quad \forall j \in \mathcal{J}$ (2.6)

$z_{jrik} + z_{rjik} \leq \frac{1}{2}(y_{jik} + y_{rik}) \quad \forall i \in \mathcal{S} \quad \forall k \in \mathcal{M}_i \quad \forall j, r \in \mathcal{J} : r > j$ (2.7)

$$z_{jrik} + z_{rjik} \geq y_{jik} + y_{rik} - 1 \quad \forall i \in \mathcal{S} \quad \forall k \in \mathcal{M}_i \quad \forall j, r \in \mathcal{J} : r > j \tag{2.8}$$

$$s_{ri} \geq s_{ji} + p_{ji} - \mathbb{M}\left(1 - \sum_{k \in \mathcal{M}_i} z_{jrik}\right) \quad \forall i \in \mathcal{S} \quad \forall j, r \in \mathcal{J} : j \neq r \tag{2.9}$$

$$x_{jiw}, y_{jik}, z_{jrik} \in \{0,1\} \quad \forall i, j, k, r, w \tag{2.10}$$

In the MILP formulation, (2.1) states the objective function as makespan minimization. Constraint (2.2) defines the makespan by its relationship with completion time of each job, where the completion times are defined by constraint (2.3). Constraint (2.4) introduces the requirement that every job to be processed by a single machine in each stage. (2.5) and (2.6) together are disjunctive constraints to associate starting time of operations with the order a job would visit the stages. The machine sequences in a stage are defined by constraints (2.7) and (2.8), and together with (2.9) the disjunctive relationship is constructed. Lastly, (2.10) defines the binary decision variables.

3. ANT COLONY OPTIMIZATION

This chapter gives general background information about Ant Colony Optimization (ACO). The ACO field extended dramatically since its first introduction in 1992. A Web of Science search of "Ant colony optimization" in the title returns 4,128 results as of September, 2020. Certainly, not all aspects of the field are covered in this chapter. Rather, the underlying principles of the approach, main building blocks of the algorithm and several variants of ACO that are relevant in this study are explained here. For more detailed information about the algorithm, reader is referred to the book by Dorigo and Stützle (2004). Additionally, an overview of ACO algorithms and the latest developments in the algorithm can be found in a recent chapter by the authors (Dorigo & Stützle, 2019).

3.1. Origins and Basis of ACO

Ant algorithms are based on ideas that inspire from foraging behaviors of real ants. Ants leave a chemical substance on the ground, called pheromone, while they are searching for food. Among the ants that find the food source, the one who uses the shortest path would return earliest to the nest, increasing the pheromone levels in this shortest path. As a certain path has increased levels of pheromone, then it is more likely to be chosen by the other ants to reach to a food source. This behavior of ants leads to discovery of the shortest path to the food source, and it inspired researchers to adapt it for the solution of combinatorial optimization problems which led to a class of ant algorithms. The building blocks of this approach include: 1) Collective search for a solution, 2) leaving a trace in the memory, 3) updating the memory to include good solution characteristics, 4) construction of a solution based on collective knowledge.

The first ant algorithm was called Ant System and proposed by Dorigo (1992) (Dorigo et al., 1991, 1996). A number of different improved ant algorithms were followed until a general framework was introduced as ACO (Dorigo & Caro, 1999; Dorigo et al., 1999) to describe the common structure of an ant algorithm. "ACO is a metaheuristic in which a colony of artificial ants cooperate in finding good solutions to difficult discrete optimization problems"

(Dorigo & Stützle, 2004). An artificial ant can be considered as an imaginary agent in a computer program that is responsible for carrying out algorithmic components.

The general structure of an ACO algorithm is defined next, and several ACO variants are described in the following subsections.

3.2. General Structure of an ACO Algorithm

An ACO algorithm consists of three main phases: *Solution construction*, *Pheromone update*, and *Central actions*. Specification of each algorithm element, the ordering, and the interaction between them vary among different ACO algorithms. A general description of these algorithmic components is supplied in the following.

3.2.1. Solution construction

A group of solutions are constructed concurrently by artificial ants in *Solution construction* part. An artificial ant builds a solution by adding one solution component at every step. Choice of the solution component to be added is based on a stochastic decision-making process that uses pheromone information and heuristic information. *Pheromone information* is the collective memory of the ant colony that stores previous good/bad solution characteristics. A good solution component is reflected in the memory by high pheromone levels, while a bad one would have decreased levels of pheromone. *Heuristic information* is a problem-specific knowledge that allows building higher quality solutions. When an ant ends up with building its solution, it assesses the solution quality (that is the objective function is calculated).

3.2.2. Pheromone update

Pheromone is added to good solution components to increase their selection probability in future solution constructions. The choice of the solutions for pheromone update and the amount of pheromone to be added are part of the algorithm design. Pheromone is reflected as an added value in memory regions for that specific favorable solution pieces. Additionally, there is a process called *pheromone evaporation* that restricts too rapid accumulation of pheromone and convergence of the algorithm to a suboptimal/local best objective value.

3.2.3. Central actions

These are procedures that are performed centrally by the algorithm and not by individual ants. One prominent example is applying local search at the end of solution construction, either to each single solution or to only the best one. Also, the selection of the ant that will deposit pheromone is another example of a central action.

3.3. Travelling Salesman Problem

To formally state how an algorithm works, it is best to show the definitions and steps of the algorithm over a representative example. Travelling Salesman Problem (TSP) is such a sample problem for ACO, which was used as the application problem in the first ant algorithm and as a test problem in the numerous ones followed. Different ACO algorithms given in this section are presented using TSP setting and terminology. However, fundamental concepts of the algorithm were adapted to various other combinatorial problems as well in the literature.

TSP is the problem of constructing a shortest tour for a salesman that will visit several cities exactly once and will return to starting point of the tour. TSP is an archetype of NP-Hard combinatorial optimization problems, and it has gained significant attention in the literature (Lawler et al., 1985). There are υ number of cities, and $d_{\xi\varphi}$ is the distance between cities ξ and φ. A solution π to the problem is a tour, and can be represented as a permutation of υ cities. If the road between cities ξ and φ is traversed in the solution, that is if $\xi = \pi(i)$ and $\varphi = \pi(i+1)$, or vice versa, then (ξ, φ) is said to be a *solution component* and shown as $(\xi, \varphi) \in \pi$. The objective function of a permutation π is computed by the function $f(\pi)$ as in (3.1), and the solution that gives the minimum function value is the optimum solution.

$$f(\pi) = \sum_{i=1}^{\upsilon-1} d_{\pi(i)\pi(i+1)} + d_{\pi(\upsilon)\pi(1)} \tag{3.1}$$

where $\pi(i)$ is the i^{th} city in the permutation.

To construct a TSP solution, an artificial ant each time adds a next city to its partial solution to complete a tour. This next city is chosen stochastically among still unvisited cities. The

starting city is chosen randomly and is different for every ant. Pheromone trails $\tau_{\xi\varphi}$ for TSP is defined as the desirability of choosing city φ to move to while sitting in city ξ. Heuristic information is generally in favor of close cities and defined as $\eta_{\xi\varphi} = 1/d_{\xi\varphi}$. During implementation of an ACO algorithm, pheromone trails $\tau_{\xi\varphi}$ and heuristic information $\eta_{\xi\varphi}$ are stored in $\upsilon \times \upsilon$ matrices of T and H, respectively.

3.4. Coverage of the Section

In the sequel, some prior ACO algorithms that are relevant in this study are elaborated. Namely, those ACO implementations are *Ant System (AS)*, *MAX-MIN AS (MMAS)*, *Ant Colony System (ACS)* and the *hyper-cube framework (HCF) for ACO*. Additionally, a pheromone evaluation rule, the *pheromone summation rule*, that is useful in scheduling problems is presented at the end.

3.5. Ant System

AS was proposed by Dorigo (1992) and it was the first ACO algorithm. In this subsection, pheromone initialization, solution construction and pheromone update procedures of the AS are described.

3.5.1. Pheromone initialization

Before an ant can construct a solution, pheromone trails, $\tau_{\xi\varphi}$, should be initialized. AS assigns the τ_0 values given in (3.2). τ_0 refers to initial pheromone entries.

$$\tau_{\xi\varphi} = \tau_0 = \frac{\kappa}{c_{nn}} \tag{3.2}$$

where κ is the number of ants, and c_{nn} is the objective function value (cost) of the solution obtained by nearest neighbor (NN) heuristic. NN is a greedy heuristic which selects the closest city as the next city to move.

A wise selection of initial pheromone values is important to prevent the search process from converging towards low-quality solutions generated in early iterations. Additionally, initial

pheromone entries should not obstruct the incorporation of the search experience which might occur in cases of high τ_0 choices for a long time until the pheromone memory is updated completely.

3.5.2. Solution construction

An ant constructs a solution city-by-city in a stepwise manner using the probabilistic rule given in (3.3). That is, in each step, the ant on city ξ moves to a next unvisited city φ with a probability $p_{\xi\varphi}$.

$$p_{\xi\varphi} = \frac{\tau_{\xi\varphi}^{\alpha}\eta_{\xi\varphi}^{\beta}}{\sum_{k\in\mathcal{N}_\xi}\tau_{\xi k}^{\alpha}\eta_{\xi k}^{\beta}}\ ,\ \text{if}\ \varphi \in \mathcal{N}_\xi \tag{3.3}$$

where $\mathcal{N}_\xi$ is the set of cities still unvisited by the ant standing on city ξ, also named as the neighborhood, and α and β are parameters that state the influences of search experience (pheromone information) and problem-specific knowledge (heuristic information) on a solution construction, respectively. Indeed, neighborhoods $\mathcal{N}_\xi$ and the probabilities $p_{\xi\varphi}$ are defined ant-specific, i.e. $\mathcal{N}_\xi^h$ for ant h. However, for the sake of simplicity ant index is dropped in the formulations in this text, keeping in mind that every ant generates a different solution by its own, having its individual solution construction memory till completing the full permutation and calculating the objective function.

The definition of the neighborhood restricts the process to result with only feasible solutions. The stochastic rule of (3.3) assigns higher probabilities of selection to cities with elevated pheromone levels and with higher heuristic desirability. If β is taken to be null, then heuristic information is not present, and the next city is decided based only on pheromone trails. This, however, may lead to inferior solution quality as a *stagnation* case may occur as a result of rapid pheromone accumulation. Stagnation is referred to a situation where the solution construction process would always result with the same solution. In that case, no exploration of the search space would take place, and the algorithm is stagnated.

3.5.3. Pheromone update

Once the solution construction phase is completed and κ ants produced κ number of different solutions, pheromone trails are updated to reflect the latest search experience on the collective memory. First, before adding any pheromone, some of the previously accumulated pheromone is evaporated from all entries $\tau_{\xi\varphi}$ of the pheromone matrix as in (3.4). This evaporation avoids excessive pheromone accumulation and prevents the process to get stagnated. It is a kind of forgetting, and solution components that are not chosen by the ants become increasingly less pheromone intense.

$$\tau_{\xi\varphi} \longleftarrow (1-\rho)\tau_{\xi\varphi}, \quad \forall(\xi,\varphi) \in \mathrm{T} \tag{3.4}$$

where ρ is a parameter called the evaporation rate, $0 < \rho \leq 1$. Followingly, every ant deposits pheromone to solution components of its solution as:

$$\tau_{\xi\varphi} \longleftarrow \tau_{\xi\varphi} + \Delta\tau_{\xi\varphi}^{h}, \quad \forall(\xi,\varphi) \in \pi_h \tag{3.5}$$

The amount of pheromone $\Delta\tau_{\xi\varphi}^{h}$ ant h deposits is a function of its objective function value and is defined for TSP by

$$\Delta\tau_{\xi\varphi}^{h} = \begin{cases} 1/c_h, & \text{if } (\xi,\varphi) \in \pi_h \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

where c_h is the cost (tour length) of the solution π_h generated by ant h. Thus, the higher the solution quality, the more pheromone is deposited. Once all ants finish their pheromone updates, the resulting pheromone entries would become as in (3.7).

$$\tau_{\xi\varphi} \longleftarrow \tau_{\xi\varphi} + \sum_{h=1}^{\kappa} \Delta\tau_{\xi\varphi}^{h}, \quad \forall(\xi,\varphi) \in \pi_h \tag{3.7}$$

Common solution components among ant solutions would reach higher levels of pheromone, being more desirable in future solution generations.

3.6. MAX-MIN Ant System

MMAS (Stützle & Hoos, 1997; 2000) greatly improved the performance of the earlier ACO algorithms, namely AS and its variants. Opposed to AS where all ants add pheromone, MMAS requires only the ant with the best solution quality to update the pheromone trails. This best can be chosen to be iteration-best or global-best. As their names imply, an iteration-best (IB) solution is the one with the lowest objective value among the solutions generated in an iteration, while a global-best (GB) is the best over all iterations that have been completed until that GB.

Allowing only a best solution to deposit pheromone would result in too rapid pheromone accumulation in the solution components that are common in best solutions and would lead to a situation of stagnation in solution construction process. This is due to the high desirability of good solution components forcing every ant to build the same but suboptimal solution. To avoid this very possible stagnation case, MMAS imposes limits on the amount of pheromone that each entry in the pheromone matrix can have. The algorithm does not neither allow too low pheromone values. That is, it defines $[\tau_{min}, \tau_{max}]$ limits on the value of every $\tau_{\xi\varphi}$. This is also where the name of the algorithm (Max-Min Ant System) comes from. Additional to pheromone limits, MMAS further proposes a reinitialization routine to avoid stagnation. The algorithm requires the pheromone entries to be reinitialized at any time the search process shows signs of stagnation or fails to generate improved-quality solutions for a predefined number of consecutive iterations.

In MMAS, the pheromone trails are initialized to τ_{max} to obtain a higher rate of exploration at early iterations.

3.6.1. Pheromone update

Following evaporation on all solution components, pheromone deposit in MMAS is formally performed according to (3.8), which enables only the best ant to deposit pheromone, as stated before.

$$\tau_{\xi\varphi} \longleftarrow \tau_{\xi\varphi} + \Delta\tau_{\xi\varphi}^{best}, \quad \forall(\xi,\varphi) \in \pi_{best} \tag{3.8}$$

where $\Delta\tau_{\xi\varphi}^{best} = 1/c_{best}$, for TSP, and c_{best} is the cost of the solution generated by the best ant -may be selected to be IB, GB or both in an alternating way. During initial iterations of the algorithm, it is preferable to make a wider exploration of the search space. This can be achieved by using IB solutions instead of GB in that phase of the algorithm. GB solutions can be used with an increasing rate as the algorithm proceeds, focusing more on good quality regions of the search space.

3.6.2. Pheromone limits

Limiting the pheromone trails prevents solution components from having extreme probabilities (low or high) in the probability distribution of solution construction phase, see Equation (3.3). In MMAS, τ_{max} and τ_{min} are defined as in (3.9).

$$\tau_{max} = 1/\rho c_{GB}; \qquad \tau_{min} = \tau_{max}/a \tag{3.9}$$

where a is a parameter. The definition of τ_{max} requires it to be a dynamic value: updated whenever a new GB is reached.

3.6.3. Pheromone initialization and reinitialization

To let the solution components have close probabilities at early iterations, pheromone trails are initialized to an estimate of τ_{max}. This way, no components would have a decisive role in solution constructions and a wide exploration of the search space would be allowed. Pheromone trails can be initialized again in later phases to the values they had at the start of the algorithm to enable the search process escape from a possible stagnation behavior. This reinitialization is based on the relative values of the pheromone trails. For detailed information on reinitialization of pheromone trails, reader is referred to Stützle and Hoos (2000).

3.7. Ant Colony System

ACS was proposed by Dorigo and Gambardella (1997) and it has structural differences from AS. It offers more efficient solution construction and pheromone update procedures. While constructing a solution, ACS allows increased exploitation of accumulated knowledge. Additionally, to deposit pheromone, only the global best ant is used. Pheromone evaporation is also carried out by that ant only. This is in contrast with both AS and MMAS where evaporation takes place on all entries of the pheromone matrix, regardless of the solutions produced in the iteration. A local pheromone evaporation is also present in ACS, which requires an ant to decrease the pheromone on the component it just added to its partial solution. This would allow different solutions to be generated by the ants in an iteration, hence an enhanced capability of searching the solution space.

3.7.1. Solution construction

While constructing a solution in ACS, an ant on city ξ uses (3.10) to choose the next city φ to move to. The rule given in (3.10) is called the *pseudorandom proportional rule*.

$$\varphi = \begin{cases} \arg\max_{k \in \mathcal{N}_\xi} \tau_{\xi k} \eta_{\xi k}^{\beta} & \text{if } q \leq q_0 \\ \Phi & \text{otherwise} \end{cases} \tag{3.10}$$

where q is a uniformly distributed random number and q_0 is a parameter, both of them between $[0,1]$, and Φ is a random variable having the probability distribution in (3.3) with $\alpha = 1$.

Thus, if the algorithm is needed to be more explorative q_0 is kept low, while a higher exploitation of accumulated search knowledge can be ensured with a high q_0.

3.7.2. Pheromone update

ACS brings significant structural differences to pheromone update process compared to AS. It carries out two different update routines in a single iteration. The first update takes place while an ant constructs a solution. After an ant adds a solution component $(\xi\varphi)$ to its partial

solution, it updates the pheromone entry of that solution component as in (3.11). This routine is known as the *local pheromone update*.

$$\tau_{\xi\varphi} \longleftarrow (1-\theta)\tau_{\xi\varphi} + \theta\tau_0 \tag{3.11}$$

where θ is a parameter: $0 < \theta < 1$. Thus, the pheromone trail becomes a weighted average of the initial pheromone value and the latest value before the update. This decreases the intensity of the pheromone on the solution components used by an ant, making their selection probability lower during the solution construction of the other ants. The main advantage of this kind of an update is allowing a diversification among the solutions generated in an iteration, hence an increased exploration of the search space.

Once the solution construction process is completed by all ants, the second pheromone update procedure is applied. It is called the *global pheromone update* and carried out by only the GB ant. This GB ant both evaporates and deposits pheromone on the components of the solution it generated. This way, evaporation takes place only on that components and not on all components of the system, reducing the complexity of the algorithm from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$, where n is the problem size. The global pheromone update rule is given in (3.12).

$$\tau_{\xi\varphi} \longleftarrow (1-\rho)\tau_{\xi\varphi} + \rho\Delta\tau_{\xi\varphi}^{GB}, \qquad \forall(\xi,\varphi) \in \pi_{GB} \tag{3.12}$$

Similar to previous formulations, $\Delta\tau_{\xi\varphi}^{GB} = 1/c_{GB}$. The update rule requires the multiplication of the deposited pheromone amount with the parameter ρ, differently from AS.

3.8. Hyper-Cube Framework for ACO

HCF (Blum & Dorigo, 2004) proposed for ACO enables a more effective handling of pheromone values and a more robust update of the pheromone entries. It makes the pheromone trails to take value in the interval [0,1].

The HCF idea is based on the following. A solution for a combinatorial optimization can be defined as an υ-dimensional binary vector $\vec{s}$, where υ is the number of all solution components in the system. If an element of the vector is part of the solution then it takes value 1, or it takes

value 0 otherwise. That enables regarding a solution as a corner of the υ-dimensional hypercube. A subset of the corners of the hypercube would be the set of feasible solutions. In that context, the set of pheromone values can be defined as an υ-dimensional vector. HCF proposes to make the pheromone update in AS as

$$\vec{\tau} \leftarrow (1-\rho) \cdot \vec{\tau} + \rho \cdot \vec{b} \tag{3.13}$$

where $\vec{b}$ is a υ-dimensional vector with

$$\vec{b} = \sum_{h \in \mathcal{A}} \psi_h \cdot \vec{s}_h \tag{3.14}$$

where $\vec{s}_h$ is the solution constructed by ant h, $\mathcal{A}$ is the set of all ants in the algorithm and ψ_h is given as

$$\psi_h = \frac{c_h}{\sum_{h' \in \mathcal{A}} c_{h'}} \tag{3.15}$$

The pheromone update rule of (3.13) can be shown in index notation as the following:

$$\tau_{\xi\varphi} \leftarrow (1-\rho)\tau_{\xi\varphi} + \rho \cdot \sum_{\{h \in \mathcal{A} \mid (\xi,\varphi) \in \pi_h\}} \frac{c_h}{\sum_{h' \in \mathcal{A}} c_{h'}} \qquad \forall (\xi, \varphi) \in \mathrm{T} \tag{3.16}$$

The difference with the pheromone update rule of HCF with AS is 1) the multiplication of the deposited pheromone with a factor ρ, and 2) the normalization of the amount of pheromone added by each ant. This makes pheromone trail values not to be dependent on the cost value, neither on the problem instance.

In MMAS, which uses a single solution to update the trails, HCF is applied as follows:

$$\vec{\tau} \leftarrow (1-\rho)\vec{\tau} + \rho \vec{s}_{best} \tag{3.17}$$

The pheromone deposit part of (3.17) is shown in index notation as

$$\tau_{\xi\varphi} \leftarrow \tau_{\xi\varphi} + \rho, \qquad \forall (\xi, \varphi) \in \pi_{best} \tag{3.18}$$

HCF also simplifies to define the maximum and minimum pheromone limits in MMAS. It already makes $\tau_{\xi\varphi}$ to take values between [0,1] which enables to assign the limits as 0.999 for the upper bound and 0.001 for the lower bound (Blum & Dorigo, 2004).

Lastly, global pheromone update of ACS, which uses the GB to both evaporate and deposit pheromone, is shown in HCF as follows

$$\tau_{\xi\varphi} \longleftarrow (1-\rho)\tau_{\xi\varphi} + \rho, \qquad \forall(\xi,\varphi) \in \pi_{GB} \tag{3.19}$$

3.9. Pheromone Summation Rule

When ACO algorithm is adapted to scheduling problems, it is common to define $\tau_{\xi\varphi}$ as the desirability to assign job φ to position ξ in the permutation. In this type of problems, the exact or at least the approximate position of a job in the permutation comes to be important and scheduling the job to a further away position results in degraded solution quality. However, while an ant constructs a solution, it is possible for the ant to schedule a job other than job φ, albeit job φ had a higher $\tau_{\xi\varphi}$ value, to that specific position ξ because of the stochastic nature of the algorithm. Moreover, the algorithm may not even assign the job until the end sections of the permutation if its pheromone intensity is low for the positions after ξ. This is quite possible since there might be no previous iterations that assign job φ to positions further than ξ, resulting in low pheromone values for that positions, hence low probability of selection. Such a scenario would result in inferior solution quality because of not assigning a job to its most desirable place but rather assigning it to a further away position. To overcome that problem, Merkle and Middendorf (2000) proposed a useful rule in using pheromone information, called the pheromone summation rule. This rule offers use of $\tau_{\xi\varphi}$ given in (3.20) *while constructing a solution*.

$$\tau_{\xi\varphi} := \sum_{k=1}^{\xi} \tau_{k\varphi} \tag{3.20}$$

This enables job φ not assigned to a desirable position ξ to be assigned to a close position, since $\tau_{\xi\varphi}$ would be still effective in pheromone considerations of the later places. Thus, the next solution component is chosen in ACS according to

$$\varphi = \begin{cases} \arg\max_{k \in \mathcal{N}_\xi} \left(\sum_{r=1}^{\xi} \tau_{rk} \right)^{\alpha} \eta_{\xi k}^{\beta} & \text{if } q \leq q_0 \\ \Phi & \text{otherwise} \end{cases} \tag{3.21}$$

where Φ has the following probability distribution

$$p_{\xi\varphi} = \frac{\left(\sum_{k=1}^{\xi} \tau_{k\varphi} \right)^{\alpha} \eta_{\xi\varphi}^{\beta}}{\sum_{k \in \mathcal{N}_\xi} \left(\sum_{r=1}^{\xi} \tau_{rk} \right)^{\alpha} \eta_{\xi k}^{\beta}}, \quad \text{if } \varphi \in \mathcal{N}_\xi \tag{3.22}$$

Additionally, a weighted pheromone summation rule is followed by Merkle et al. (2000) that gives different influences to each pheromone entry up to position ξ:

$$\tau_{\xi\varphi} := \sum_{k=1}^{\xi} \gamma^{\xi - k} \tau_{k\varphi} \tag{3.23}$$

where γ is a parameter with $\gamma > 0$. If $\gamma < 1$, then pheromone values close to position ξ would have more influence, and inversely, if $\gamma > 1$ earlier pheromone values become more effective. If $\gamma = 1$, then basically no weights are assigned, and the rule is the one in (3.20) where every place in the permutation up to position ξ would have the same influence on the resulting pheromone evaluation $\tau_{\xi\varphi}$.

4. ACO ALGORITHM FOR PROPORTIONATE MULTIPROCESSOR OPEN SHOP PROBLEM

In this chapter, an ACO algorithm for the proportionate MPOS problem is proposed. Pheromone information, solution construction and pheromone update procedures together with the central actions proposed in the algorithm are fully described. Experimental test results are given in Chapter 5.

Before going in details of the algorithm, the specific MPOS environment studied in this thesis and the assumptions of the model are stated in the following subsection.

4.1. Problem Statement

A proportionate MPOS is considered in this study. Proportionate property refers that the processing times are dependent on stages but not on jobs (see Section 2.2). Other assumptions about the shop are listed as follows:

1. All jobs are required to be processed in all stages.
2. There are infinite capacity buffers between all stages.
3. A machine can process a single job, and a job can be processed on a single machine at a time.
4. Preemption is not allowed.
5. All jobs are ready for processing at the beginning; release times are zero.
6. The parallel machines in a stage are identical.
7. All machines are always available.
8. Machines of a stage can perform the task of only that stage and not of other stages.
9. Due dates for completion of jobs are not present.
10. All jobs are equally important.
11. Precedence relations do not exist between neither jobs nor operations.
12. All types of setup and travel times are ignored.

The problem is about constructing a schedule for proportionate MPOS that minimizes the makespan C_{max}, the time all operations are completed. The problem is shown in 3-field

notation as $O(P)|prop|C_{max}$. The schedule that would be offered as a solution for the problem should incorporate three elements: 1) Route of each job to visit the stages, 2) For any job, the machine that will be used to process it in a stage, 3) Processing sequence of relevant jobs in any machine in a stage.

4.2. Solution Representation

One of the most important part of developing a solution approach for a combinatorial problem is the use of an efficient way to represent a feasible solution of the problem. It is especially important in metaheuristic algorithms where the sub-routines are based on the particular solution representation. Neighborhood structures, for example, are defined on the solution representation at hand, and they are crucial in finding high-quality solutions if chosen appropriately. The importance of the representation is also apparent in memory-based approaches. That type of algorithms keeps promising solution characteristics at mind to refer them later again. These favorable characteristics are defined over the representation of the solution. Thus, to correctly encode the feature of the solution which makes it good, an effective representation of the solution should be used.

For some problems, such as the TSP and single machine problems, there is a natural, obvious way to represent a feasible solution for the problem. This is due to the number of dimensions the solutions of these problems require. That is, for TSP the solution should supply only the information of where to go after visiting a city and, thus, the representation is simply a permutation of cities. Again, for single machine problems, it is important only to know at what order the jobs would be processed, making a sequence of the jobs sufficient for the representation. However, in case of MPOS, a solution representation should supply three different information: 1) at what order each job would visit the stages, 2) at which machine each job would be processed in each stage, and 3) at what order each machine in each stage would process the relevant jobs. Thus, defining an efficient way of representing a solution of MPOS is relatively hard.

4.2.1. Operation-permutation representation

In the literature, operation-permutation is the commonly used representation among MPOS researches (Azadeh et al., 2014; Bai et al., 2016; Goldansaz et al., 2013; Matta, 2009; Naderi et al., 2011). It was proposed by Liaw (2000) for the classical open shop and adapted to MPOS by Matta (2009). It is a permutation of operations O_{ji}. Permutation is read from left to right and the respective operation is assigned to the first available machine in the respective stage. For an example, consider the following operation permutation for Problem 1 given in Table 4.1.

$$O_{11}O_{41}O_{42}O_{22}O_{12}O_{32}O_{31}O_{21}O_{51}O_{52} \tag{4.1}$$

For the sake of simplicity, it is common to number the operations from 1 to N, where N is the total number of operations, hence $N = n \times s$. Thus, the operations of Problem 1 can be represented by the numbers shown in Table 4.2. This makes the permutation in (4.1) to be re-written as

$$1 - 4 - 9 - 7 - 6 - 8 - 3 - 2 - 5 - 10 \tag{4.2}$$

Table 4.1. Sample proportionate MPOS problems

Problem 1	Problem 2	Problem 3
s = 2; n = 5	s = 2; n = 33	s = 3; n = 5
$m_1 = 3$; $m_2 = 2$	$m_1 = 23$; $m_2 = 10$	$m_1 = 2$; $m_2 = 2$; $m_3 = 1$
$p_1 = 7$; $p_2 = 5$	$p_1 = 10$; $p_2 = 4$	$p_1 = 7$; $p_2 = 6$; $p_3 = 4$

Table 4.2. Numbers to represent operations of Problem 1

O_{11}	O_{21}	O_{31}	O_{41}	O_{51}	O_{12}	O_{22}	O_{32}	O_{42}	O_{52}
1	2	3	4	5	6	7	8	9	10

Encoding the permutation from left to right creates the schedule in Figure 4.1. It is a semi-active schedule. Naderi et al. (2011) tested a different decoding which schedules the operation

as early as possible even if it is in a later position in the permutation. Hence a non-delay schedule resulted. However, the solution quality became degraded in average.

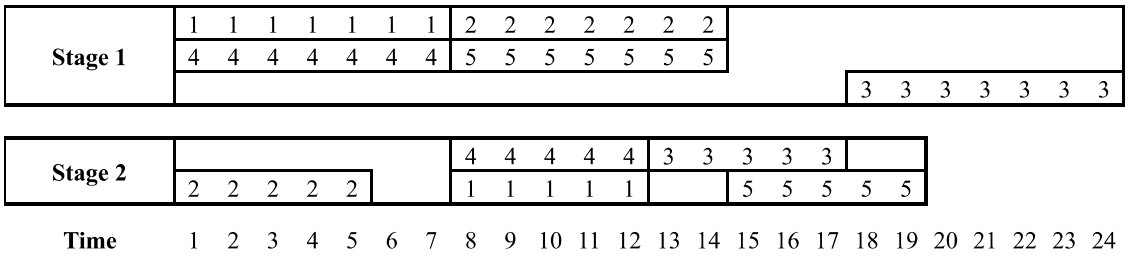

Figure 4.1. Schedule of the sample permutation

4.2.2. Inefficiencies of operation permutation in proportionate MPOS

In this subsection, it is explained why operation permutation is not an efficient way of representing a solution of proportionate MPOS. It is presented in three main topics: 1) Excessive idle time in machines, 2) Extra time for transforming the schedule into an active one, 3) Problems due to including the job identities in the permutation.

Consider the schedule given in Figure 4.1 for the operation permutation in (4.2). One can easily notice unnecessary idle times in both stages that cause increased makespan value. It is a result of the ordering in the permutation which requires job 3 first to be scheduled in stage 2 after jobs 4, 2 and 1, and then be scheduled in stage 1. Two solutions can be considered here: 1) Creating a non-delay schedule: An operation can be scheduled to earlier available machines even if it appears in a later position in the permutation, 2) Postprocessing: Converting the resulting schedule (Figure 4.1) into an active one. The first solution has been mentioned earlier in this section to lead to a degraded solution quality. The second option is necessary and detailed next.

To get rid of the meaningless idle times in the schedule, it must be converted to an active one. An active schedule can be formed by moving a job to an earlier position in the stage without delaying the current start time of any operation. Thus, job 3 is moved to be processed between times 0-7 in the same machine in stage 1, and job 5 is moved to times 0-5 to machine 1 in stage 2. The resulting improved schedule is shown in Figure 4.2, which is an optimal one in this case. However, post-processing the schedule to make it active and repeating this action

for every single solution generated requires excessive computational time in an algorithm. The time required to compute the objective function is already a headache in this type of scheduling problems. Thus, such an additional post-processing requirement is not favorable. Another problem with post-processing is that the ultimate schedule reached is different from what the permutation encodes. This poses a challenge particularly in memory-based algorithms, such as the ACO proposed in this study, which inherit good solution characteristics in future generations.

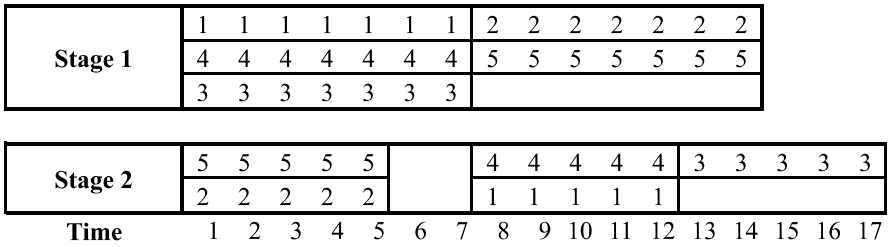

Figure 4.2. Improved schedule after post-processing

Another inefficiency in using the operation permutation in proportionate MPOS is due to the call of job identities in the permutation. That is, the permutation enforces a certain job to be scheduled. However, in proportionate MPOS the processing time of a stage is independent of the job, bringing a flexibility to the schedule. Restricting this flexibility may lead to degraded solution quality. Figure 4.3 illustrates different job assignments to have the same makespan value for Problem 1. Again, consider the dense schedule given in Figure 4.4 for Problem 2 (see Table 4.1). A dense schedule leaves no machine idle if there is an available job to be processed. An allocation of job bundles creates the schedule in the figure and no explicit call to job identities is present. By using the template and assigning different jobs for the bundles each time, numerous different schedules with the same makespan of 22 can be created.

The dense schedule of Figure 4.4 can be improved as in Figure 4.5, saving 2 time points and resulting in the optimum makespan.

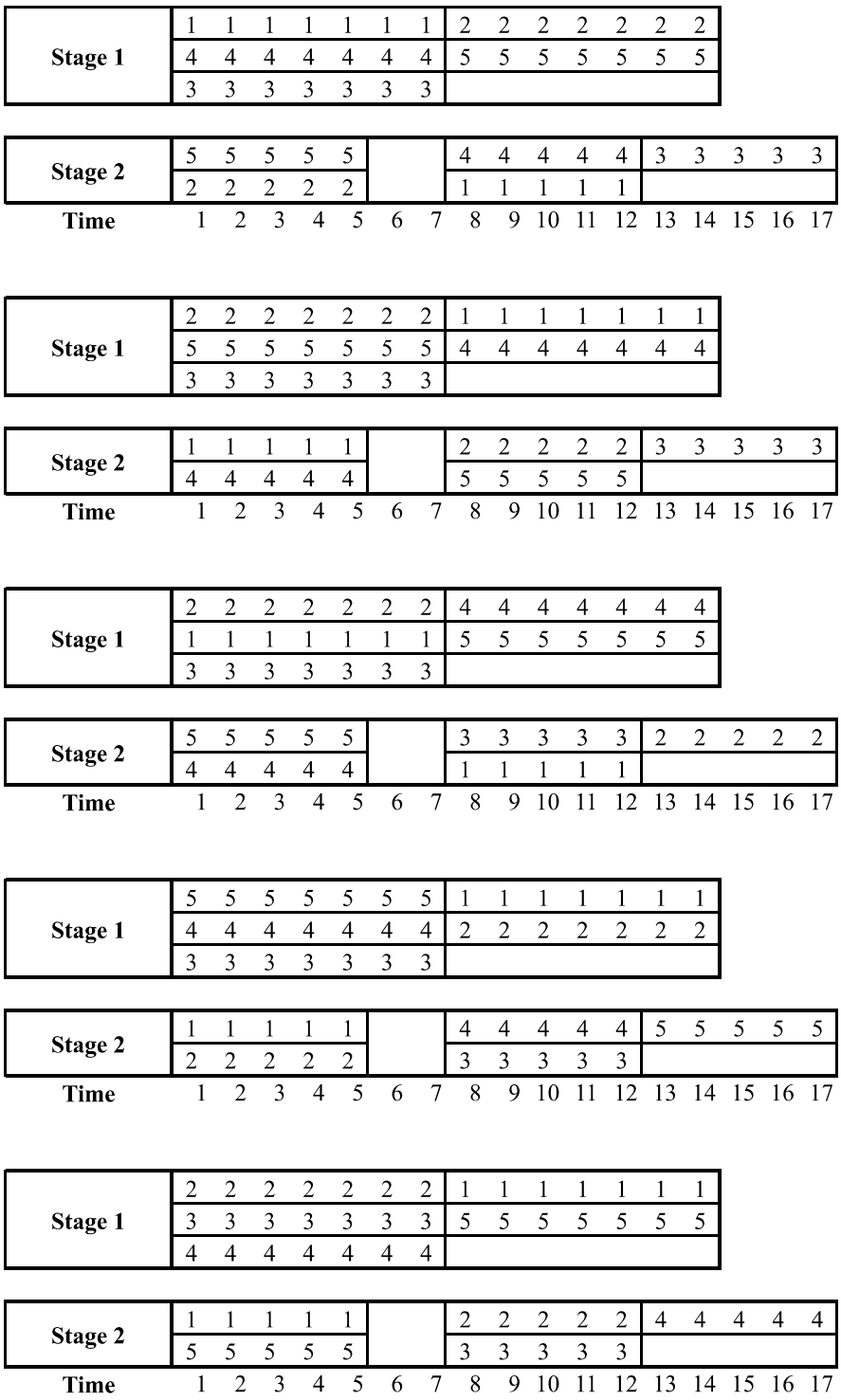

Figure 4.3. Different job assignments with same makespan value

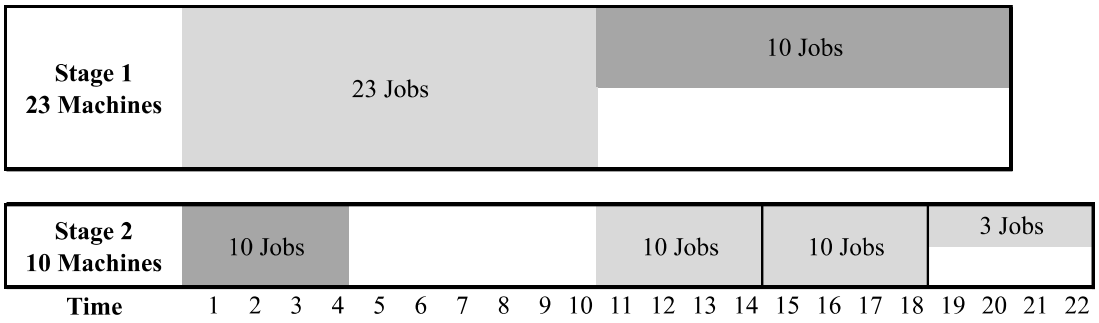

Figure 4.4. A dense schedule for Problem 2

An important implication that can be drawn out from the enhanced schedule in Figure 4.5 is that what is important in scheduling a proportionate MPOS is to determine how many jobs to allocate to each stage at distinct time points for a lower makespan. However, operation permutation does not encode this information, causing inefficiencies in extracting good solution characteristics.

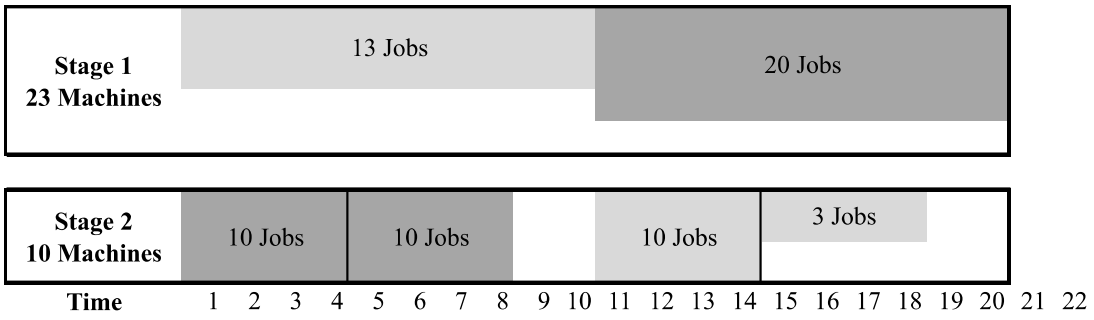

Figure 4.5. Enhanced schedule for Problem 2

4.2.3. A novel solution representation: Implicit-stage permutation

A novel way to represent a feasible solution of a proportionate MPOS is proposed in this study. The representation is named as *implicit-stage permutation*. It is a permutation of numbers that represent cumulative number of job assignments to a respective stage. It is indeed a higher-level representation of a *stage permutation*. Thus, to introduce the proposed solution representation, first *stage permutation* is defined as the following. Consider Problem 3 given in Table 4.1. It requires 5 jobs to be processed in 3 stages. Thus, a schedule would make 5 job assignments for each stage. Since, in the preceding subsection, it made clear that

job identities do not have a direct role in the resulting objective function value, it is proposed to include only calls to stages in a solution representation. n number of calls for each stage is needed, requiring a stage permutation to have $n \times s$ number of elements. Thus, a *stage permutation* is a permutation of s stages, where each stage repeats n times. The following is a sample stage permutation for Problem 3.

$$2 - 2 - 1 - 3 - 1 - 2 - 3 - 2 - 1 - 1 - 3 - 3 - 2 - 1 - 3 \tag{4.3}$$

The permutation is decoded as follows: Assign *a job* to stage 2, assign *a job* again to stage 2, then assign *a job* to stage 1, and so on. At any step, *the job* to be assigned can be chosen arbitrarily as long as it is an eligible job: still not processed in that stage and not under process in some other machine, hence available. Although arbitrary choice of a job is sufficient to construct a feasible schedule, more informed rules of job selection are proposed in this study to allow for improved solutions.

One of two rules to select a job while constructing the schedule of a permutation is making the selection based on job desirability, d_j, which is introduced as the number of stages that still needs job j. A job with lower desirability is preferred to allow non-empty future eligible job sets as much as possible. If job desirability values are equal for two jobs, then the selection is made in numerical order. Unless stated otherwise, job desirability is applied as the default rule for job selection in decoding a permutation.

The second rule to select a job is proposed as selecting a random job from an eligible job set. Although, at first glance, the rule seems to serve no purpose, it plays a significant role in decreasing the makespan in large scale instances that have many perfectly balanced stages. The notion is further explained in Section 4.3.2.

After a job is selected to assign for the respective stage, it is scheduled to earliest available machine. The final schedule of permutation (4.3) is shown in Figure 4.6.

Use of stage-permutation as the solution representation, however, has two problems. First, it includes repetition of elements, which makes it hard to extract knowledge about favorable patterns in good solutions. This knowledge would be kept in the memory of the ACO

algorithm and it is essential. Second, it does not encode how many assignments have been made to a stage up to a point, since this information is shown, in the previous subsection, to reflect an important feature of a solution. To overcome these problems, it is proposed to represent the stage permutation in a higher-level form by substituting every stage representation by a number referring to its cumulative number of appearances in the permutation. This higher-level form is named as the *implicit-stage permutation* and described next.

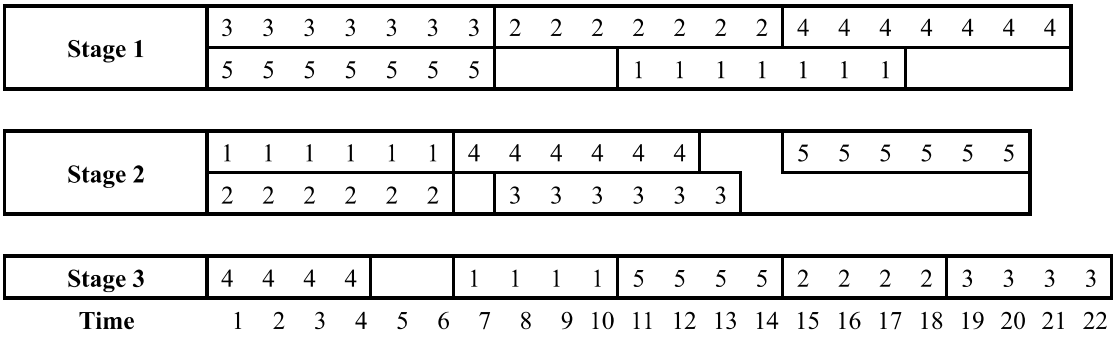

Figure 4.6. Schedule of the stage permutation in (4.3)

To construct the implicit-stage permutation from a simple stage permutation, the encoding given in Table 4.3 is used. Thus, the implicit-stage permutation representation of the permutation in (4.3) is as the following:

$$6 - 7 - 1 - 11 - 2 - 8 - 12 - 9 - 3 - 4 - 13 - 14 - 10 - 5 - 15 \qquad (4.4)$$

Table 4.3. Encoding to construct implicit-stage permutation of a stage permutation

	1st assignment	**2nd assignment**	**…**	**nth assignment**
Stage 1	1	2	…	n
Stage 2	$n+1$	$n+2$	…	$2n$
Stage 3	$2n+1$	$2n+2$	…	$3n$
⋮	⋮	⋮	…	⋮
Stage *i*	$n(i-1)+1$	$n(i-1)+2$	…	$n \times i$
⋮	⋮	⋮	…	⋮
Stage *s*	$n(s-1)+1$	$n(s-1)+2$	…	$n \times s$

The permutation in (4.4) is decoded as follows. $6 = n + 1$: from the table, $n + 1$ is read as the first assignment to stage 2, then 6 is decoded as: make a first assignment to stage 2. Similarly, 7 is decoded as: make a second assignment to stage 2, and 1 as: make a first assignment to stage 1. As can be realized, the representation encodes the cumulative number of assignments to a stage.

4.2.4. Random solution generation

In this subsection, it is presented how to generate a random solution for a proportionate MPOS problem using the implicit-stage permutation. Random solution generation is intentionally described here, since it is part of the ACO algorithm proposed in this study.

Since the representation is a permutation of numbers from 1 to $n \times s$, the first thing that comes to mind is to generate a random permutation for a random solution. However, that would result in an infeasible permutation, because the cumulative numbering for a stage may not be obeyed. That is, the numbers used to represent the total number of assignments to a distinct stage should appear in order in a permutation. The number representing a second assignment for a stage, for instance, should not appear in the permutation earlier from the number representing the first assignment. For example, considering Problem 3, encoding 10 is not allowed to precede encoding 7 in a permutation. Because, the information given by such a permutation would be read as: assign a fifth job to stage 2, then assign a second job to stage 2. This renders the representation without function and makes it meaningless.

To correctly generate a random solution, first a random stage permutation should be generated and then it should be converted to an implicit-stage permutation. Generating a random stage permutation is straightforward: it would be a random permutation of s stages, where each stage repeats n times. Once a random stage permutation is available, Table 4.3 can be used to construct the respective implicit-stage representation.

4.3. ACO Algorithm

An ACO algorithm for the proportionate MPOS is proposed based on the novel solution representation. Pseudocode of the algorithm is given in Algorithm 1.

Algorithm 1: ACO proposal for proportionate MPOS

ACO( )

// Initialization

$\alpha, \beta, \gamma, \rho, \kappa, \tau_{min}, \tau_{max}$;

$L \leftarrow n \times s$; *%permutation length*

$\tau_{\xi\varphi} \leftarrow 0.5 \quad \xi = 1, \cdots, L \quad \varphi = 1, \cdots, L$; *%pheromone information*

$\eta_{\xi\varphi} \leftarrow 1$; *%heuristic information*

while *time limit not exceeded* **do**

Solution Construction() {

output $soln_t$ %solution(s) generated in iteration t

$\tau_{\xi\varphi} \leftarrow \sum_{i=1}^{\xi} \left[\gamma^{\xi-i} \cdot \tau_{i\varphi}\right]$;

// Global exploration

for $i = 1$ **until** κ **do**

GenRandSoln(); *%generate random solutions*

ConstructSchedule(){

input RandSoln **output** Schedule, C_{max}

JobDesirabilityBasedSelectionFromEligibleJobSet();

}

return κ *random solutions*

//Exploitation

for $i = 1$ **until** L **do**

$Q_{\xi\varphi} \leftarrow \tau_{\xi\varphi}^{\alpha} \cdot \eta_{\xi\varphi}^{\beta}$; *%quality function*

$\varphi \leftarrow \arg\max Q_{\xi u} \quad u \in remaining\_implicit\_stages$; *%next solution component to assign at the ξ^{th} order in the permutation*

$\eta_{\xi\varphi} \leftarrow n_{s_v}/n$; *%Update dynamic heuristic information*

return $AntSoln$; *%one single solution*

//Local exploration

ConstructSchedule(){

input $AntSoln$ **output** Schedule, C_{max}

RandomSelectionFromEligibleJobSet();

}

$soln_t \leftarrow \arg\min f(soln) \qquad soln \in (AntSoln \cup \kappa\ random\ solutions)$;

}

if $f(soln_t) = LowerBound$ **then**

break

if $f(soln_t) \leq f(GlobalBest)$ **then**

updatePheromone();

updateGlobalBest();

return $GlobalBest$;

The subroutines of the algorithm are described next under the following topics: Solution construction, Local exploration, Pheromone information, Heuristic information, and Pheromone update.

4.3.1. Solution construction

The random proportional rule given in (3.3) is the common rule used in constructing ant solutions. It mainly aims to perform a search that is biased towards good regions of the solution space. Otherwise, a totally random search would make no help in achieving favorable solutions as it is typically unlikely to reach a good random solution in a combinatorial problem. However, in this study, it is proposed to make a complete random search in solution space instead of a pseudorandom one, in contrary to ACO algorithms in the literature. The idea is a result of the good quality random solution potential that the proposed solution representation offers.

The proposed implicit-stage permutation representation is based on stage permutation. For any randomly generated stage permutation, it is highly unlikely to have all calls to a stage placed consecutively. That is, a permutation of, for example, 2-2-2-2-2-3-3-3-3-3-1-1-1-1-1 for Problem 3 is nearly impossible to be generated randomly. Such a solution would schedule repeatedly for the same stage causing great delays in the final schedule. Instead, any random permutation would shuttle between stages generating moderate to good quality solutions. For small instances, the quality would be good and for large scale ones it would be moderate.

Since the exploration of the solution space is proposed to be carried out by complete random solutions, then the component-by-component solution construction is proposed to be performed based only on full exploitation of previous search knowledge. The stochasticity of the solution construction routine is, thus, only due to random solutions. The procedure is formally defined as follows:

$$\mathcal{U} = \kappa\ random\ solutions\ \cup Antsoln \tag{4.5}$$

where $\mathcal{U}$ is the set of solutions generated in an iteration, κ is the number of complete random solutions and $Antsoln$ is the single solution constructed component-by-component by one ant according to

$$\varphi = \arg\max_{k \in \mathcal{N}_\xi} \tau_{\xi k}^{\alpha} \eta_{\xi k}^{\beta} \tag{4.6}$$

where φ is the next implicit-stage representation to be added to the partial solution. Definitions for the pheromone and heuristic information are given in the upcoming subsections. The rule in (4.6) is part of the solution construction rule that was proposed in literature for ACS (see Section 3.7.1).

A total of $\kappa + 1$ solutions are generated in an iteration and among them, the solution(s) with minimum makespan value are returned. This is formally shown as

$$soln_t = \arg\min_{soln \in \mathcal{U}} f(soln) \tag{4.7}$$

where $soln_t$ is the solution(s) returned by the solution construction routine of iteration t. If more than one solution has the minimum makespan value, then all those solutions are returned.

4.3.2. Local exploration

Workload balance between machines is an important element in scheduling a shop in general. If there is a machine with workload much higher than the remaining machines in the shop, it becomes decisive in the final makespan of any schedule. Because it would constitute a bottleneck in the system. To avoid such a scenario, it is vital to balance the workload between machines. In case of the proportionate MPOS, stage workloads should be balanced, and this can be achieved by allocating sufficient number of parallel machines to each stage. Number of machines can be determined according to processing time requirement of the stage as follows:

$$\frac{p_i}{m_i} = \frac{p_w}{m_w} \Rightarrow m_i = \left\lceil p_i \frac{m_w}{p_w} \right\rceil \qquad \forall\, i, w \in \mathcal{S} \tag{4.8}$$

Number of jobs to be processed in a stage is not taken into account in (4.8) since it is assumed all jobs to be processed in all stages (see Section 4.1).

Due to the integrality requirement of m_i, the balance may or may not be a perfect balance. When number of perfectly balanced stages in a proportionate MPOS increases, the problem becomes increasingly non-trivial to solve. The reason is as the following: Perfectly balanced stages complete their processing at close time points and require new jobs to be loaded almost simultaneously. After a while, through the end sections of a schedule, one stage waits for the other to process the remaining few jobs. This idle waiting times cause great delays in the final schedule. One solution to this problem is to distribute the jobs among those stages in such a way that they would require the same job at different time points. Thus, through the end of the schedule, one would not wait for the other since, although they need new jobs at close time points, this need would be for different jobs. To ensure this solution to take place in a schedule, an approach named as local exploration is proposed.

As explained in Section 4.2.3, the schedule of a permutation is constructed by selecting jobs according to job desirability. Once a job is assigned to a stage its desirability decreases by one, making it less desirable in the next assignment. Thus, indirectly, an ordering between jobs arises in selecting them. This order causes the perfectly balanced stages require approximately the same set of jobs when they become available at close time points. To create a distributed job assignment, as proposed above, it is proposed to select jobs randomly from an eligible job set instead of the default job desirability-based selection. The random selection rule leads to a family of schedules that a single permutation encodes. These schedules can be viewed as the neighboring schedules, and the resulting effect of such an approach is similar to a local search. Thus, the approach is named as *local exploration*.

Local exploration is applied only to the permutation constructed by the single ant, and not on randomly generated permutations. This is analogous to applying local search to the solutions constructed by the ants, common in the literature. However, different from a local search application, generating schedule based on local exploration is not repeated in an iteration. Instead, the favorable effect of this procedure is gained over several iterations. The reasoning behind this approach is as follows: The ACO algorithm proposed here performs random

exploration to search the solution space, and the search knowledge is kept in the memory to be used to build a single permutation that fully exploits this knowledge. Thus, the component-by-component permutation construction of the single ant is a deterministic process (see Equation (4.6)). If the random exploration phase does not produce better or at least same quality solutions for several iterations, the memory would not be updated and the algorithm would end up with the same solution each time, hence get stagnated. However, since it is proposed to apply local exploration to the permutation constructed by the ant, the algorithm continues to explore the neighborhood of the constructed permutation and continue to generate different schedules at those times. This is why local exploration is not repeated in the same iteration.

One consideration while applying local exploration to a permutation is that the schedule to be created would change for different random numbers and the procedure is not deterministic, cannot be repeated to get the same solution. Thus, instead of returning a permutation as the best solution at the end of the algorithm, the schedule which gives this best solution is returned.

4.3.3. Pheromone information

Pheromone information, $\tau_{\xi\varphi}$, is defined in line with the definition of the solution representation. Thus, $\tau_{\xi\varphi}$ is the desirability to place implicit-stage representation φ at the ξ^{th} position in the permutation.

Because of the nature of the proportionate MPOS, many different schedules can be constructed with the same makespan value. This is due to the processing times of stages being independent of the jobs. This feature of the problem can be observed in sample permutations given in Figure 4.7 for Problem 3. The permutations are different but their makespan equals to each other. This characteristic of the problem allows to make inferences about the position of a number (an implicit stage representation) in the permutation. It can be easily recognized that rather than the exact position, the approximate position of a number is important in a permutation. This leads to use of weighted pheromone summation rule given in (3.23) to evaluate the pheromone information (see Section 3.9). It is a useful rule if the positions of numbers are similar in permutations of good schedules (Merkle & Middendorf, 2002). The

rule is only for evaluating the pheromone information; thus it is only used in solution construction and not in pheromone update.

Makespan	Implicit stage permutation														
23	11	1	6	12	13	2	7	3	8	9	14	10	15	4	5
23	11	1	6	12	2	13	3	7	14	15	8	4	5	9	10
23	1	11	2	3	6	12	7	13	14	8	15	9	4	5	10

Figure 4.7. Different permutations with same makespan

4.3.4. Heuristic information

Heuristic information, $\eta_{\xi\varphi}$, is problem-specific knowledge that is supplied to algorithm to help it converge to better quality solutions. $\eta_{\xi\varphi}$ is defined as the heuristic desirability to place implicit-stage representation φ at the ξ^{th} position in the permutation.

For the proportionate MPOS problem, a permutation that requires to schedule for the same stage successively, having many consecutive calls to a stage, may lead to idle times in other stages and cause a delay in the final schedule. Example of such a stage permutation for Problem 3 is 2-2-2-2-2-3-3-3-3-3-1-1-1-1-1. Based on this knowledge, a heuristic information called "Most Work Remaining Heuristic (MWRH)" is proposed for the problem. The heuristic can be found in other scheduling applications in the literature. But it is adapted for the proportionate MPOS for the first time and in the context of the new solution representation in this study.

MWRH prioritize stages that have greater number of jobs still to be processed compared to other stages. Then, $\eta_{\xi\varphi}$ is defined formally as follows

$$\eta_{\xi\varphi} = \frac{n_{s_\varphi}}{n} \tag{4.9}$$

where s_φ is the corresponding stage referred by the implicit-stage representation φ, and n_i is the number of remaining jobs still to be processed in stage i.

4.3.5. Pheromone update

Only the global-best solution(s) (GB) is used to make an update in pheromone trails. Among $\kappa + 1$ solutions constructed in an iteration, iteration-best solution(s) (IB) is compared with the current GB, and update is performed if IB equals to GB or constitutes a new GB. In an iteration, it is possible to have more than one solution with minimum IB value, coming from the exploration part especially in early phases of the algorithm. However, the permutations of such solutions are different. Thus, all those solutions are used to make the update.

Before adding pheromone to entries corresponding to GB solutions, pheromone evaporation is carried out in all entries of the pheromone matrix. Pheromone update is performed in HCF of Blum and Dorigo (2004) -explained in Section 3.8. Since more than one solution can be used in the update, the procedure given in (3.16) should be used. However, here the solutions have the same makespan value. Thus, the following adapted rule is applied in the update

$$\tau_{\xi\varphi} \longleftarrow (1-\rho)\tau_{\xi\varphi} + \rho \sum_{\pi \in GB} u_{\pi} \qquad \text{where } u_{\pi} = \begin{cases} 1 & \text{if } (\xi, \varphi) \in \pi \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

Minimum and maximum limits, τ_{min} and τ_{max}, are imposed on pheromone values as proposed by Stützle and Hoos (1997). After the update is completed, any $\tau_{\xi\varphi}$ greater than τ_{max} is made equal to τ_{max}, and similarly, any $\tau_{\xi\varphi}$ smaller than τ_{min} is made equal to τ_{min}.

5. COMPUTATIONAL EXPERIMENTS

Computational tests are carried out to measure the performance of proposed ACO algorithm. This chapter presents the test results, comparison with literature and finally analysis and discussion of the results. Parameter estimation procedures and the testbed are explained first.

5.1. Parameter Estimation

Before performing computational tests, algorithm-specific parameters are estimated. To determine the parameters, commonly used values in literature are referred or initial pilot experiments are carried out. $\alpha = 1$ and $\beta = 2$, since these values are widely accepted favorable values (den Besten et al., 2000; Dorigo & Gambardella, 1997; Merkle & Middendorf, 2002; Stützle & Hoos, 2000).

Initial pilot experiments revealed $\gamma = 0.70$ and $\rho = 0.30$. The values are reasonable since γ, the weight parameter in pheromone summation rule, is preferred to be less than 1 to allow positions close to position ξ to have higher effect in value of $\tau_{\xi\varphi}$, see Equation (3.23). Moreover, it is preferred to be greater than 0.50 to avoid further positions having a negligible effect, but less than 0.90 to efficiently discriminate between the effects of close and far positions. The reasoning is made clear with the numerical example in Table 5.1 that gives the weight values for each position up to position 7 for which the pheromone value is evaluated. Position 7 is chosen arbitrarily and only for representative purposes.

Table 5.1. Weight of the positions for different γ values

Position in permutation	1	2	3	4	5	6	7
Weight multiplier	γ^6	γ^5	γ^4	γ^3	γ^2	γ^1	γ^0
$\gamma = 0.5$	0.016	0.031	0.063	0.125	0.25	0.5	1
$\gamma = 0.7$	0.118	0.168	0.240	0.343	0.49	0.7	1
$\gamma = 0.9$	0.531	0.590	0.656	0.729	0.81	0.9	1

The estimated 0.30 value for ρ is also reasonable considering the structure of the algorithm. A low rate of evaporation allows the algorithm to accumulate the search knowledge. A higher evaporation rate would be preferable in the classical solution construction approach, where

evaporation allows the algorithm to search for the solution region and prevents it to converge to an early suboptimal solution. However, in the ACO algorithm proposed here, it is not the evaporation rate that enables the search of the solution space, since the algorithm uses a random exploration routine. Thus, a low evaporation rate is favorable to increase the exploitation of accumulated experience. On the other hand, a too low evaporation would make the solution construction process to concentrate more on initial solutions, and new knowledge gained during the later phases of the algorithm would have no decisive effect (see Equation (4.10)).

κ, number of random solutions in an iteration, is taken to be 10 analogous to the commonly used 10 number of ants to construct solutions in an iteration.

Lastly, τ_{min} and τ_{max} are the remaining parameters to be estimated. Since HCF is used in pheromone update, $\tau_{\xi\varphi}$ entries can take values between [0,1]. This allows fixing the minimum and maximum limits to 0.001 and 0.999, respectively, as proposed by Blum and Dorigo (2004).

5.2. Experimental Testbed

Only one benchmark testbed exists in the literature for the proportionate MPOS problem. This testbed is used in this study to carry out the experimental tests. The testbed was created by Matta (2009) and given in Appendix A. It includes 100 proportionate MPOS instances. The instances were grouped according to the number of stages they have: 2-stage, 4-stage, 8-stage and 16-stage are the four different stage numbers considered. Among them 2-stage problems are regarded as small instances, 4-stage medium and 8 and 16-stage problems are large instances. Each group have 25 instances in it. A single problem instance was constructed as follows by assigning 3 features of the problem: 1) number of machines in each stage, 2) processing times of each stage, 3) number of jobs in the shop. Number of machines in a stage was randomly chosen from the set $\{2, \dots, 25\}$ for 2, 4 and 8-stage problems, and from $\{2, \dots, 10\}$ for 16-stage problems. The stages of a problem were then sorted in descending order of the number of machines they included. That is, stage 1 always has the greatest number

of machines, then comes stage 2, and so on. Processing times of stages were assigned so as to allow a balanced shop (see Section 2.2). This was achieved by applying the approach given in Equation (4.8). However, since the number of machines were determined first, revised form of the equation was used to assign processing times to stages as in (5.1).

$$p_i = \left\lfloor m_i \frac{p_1}{m_1} \right\rfloor \qquad \forall\, i \in \mathcal{S} \tag{5.1}$$

where p_1 is the integer processing time for stage 1 chosen randomly from the set $\{5, \ldots, 15\}$. Number of jobs in the shop was taken to be equal to total number of machines across the stages, $n = \sum_i m_i$. This allowed a "square" shop and, along with the balanced property, created difficult instances. Otherwise, if the number of jobs were too small, then the abundant number of resources (machines) would make the problem trivial to solve. Number of jobs ranges from 10 to 40 in 2-stage instances, from 28 to 90 in 4-stage instances, from 72 to 161 in 8-stage instances, and from 78 to 112 in 16-stage instances. Matta (2009) also assigned a relative deadline for each problem instance, which indicated the time to complete all work. This time is not taken into account in this study, since it may not be feasible to complete all the work by this deadline and it is allowed, both here and in Matta (2009), for a shop to run past time the deadline. Also, it was not referred in the performance analysis of neither Matta (2009) nor the other researchers.

5.3. Lower Bounds

The solution quality is evaluated using a lower bound on makespan value for each problem instance. Lower bound computations are based on the minimal requirement that must be established whatever the schedule is. In the current problem, every stage is required to process all jobs, which creates a lower bound of $\lceil n/m_i \rceil \times p_i$ on the time for a stage to complete its workload. Considering all stages, the following overall lower bound is computed for each problem instance.

$$\max_{i \in \mathcal{S}} \lceil n/m_i \rceil \times p_i \tag{5.2}$$

The computed lower bounds for the problem instances are given in LB columns in Table 5.2 - Table 5.5 in the next subsection.

5.4. Test Results and Comparison

ACO algorithm is run 10 times for each instance and statistics are collected over these runs. These statistics include, but not restricted to, average C_{max}, best C_{max}, average deviation from lower bound and average execution time, given in Table 5.2 - Table 5.5 as *Avg.* C_{max}, *Best* C_{max}, *Mean dev. (%)* and *Avg. Time*, respectively. Time limit is used as the termination criterion in a run, and it is set to 60 seconds for 2-stage and 4-stage problems, to 130 seconds for 8-stage and 180 seconds for 16-stage ones.

The same testbed was used in several other research papers on scheduling MPOS, as mentioned earlier in Section 1.2. TS algorithm proposed by Abdelmaguid et al. (2014) greatly improved the results of Matta (2009) both in solution quality and computational time. This TS algorithm results are referred for comparison in this study. Besides, a recent study by Abdelmaguid (2020) proposed an SS/PR algorithm and improved the results further, not in computational time but in solution quality in most instances. Thus, this SS/PR algorithm is also used in performance evaluations of the current ACO algorithm.

The results of the experiments are given in Table 5.2 - Table 5.5 together with the results of the TS and SS/PR for performance comparisons. The tables show *Avg.* C_{max}, *Best* C_{max} and *Mean dev. (%)* as makespan related statistics and *Avg. Time* for time performance of the algorithms. Number of jobs (n) and lower bound (LB) values are also given for each instance. The asterisk symbol (*) is used to represent a provably optimal solution which is a makespan value that equals the LB. A bold face is used to emphasis the best result among the compared algorithms for the related instance.

Table 5.2 gives the results for 2-stage instances. SS/PR results are not included, since they were not provided in the original paper and they were claimed to be same as the TS results. ACO also produces exactly the same makespan value as the TS for each instance within similar computational time. Same results had been also reported by Matta (2009). Indeed, it

is claimed in this study that the makespan values reported in Table 5.2 for 2-stage instances are the optimum ones. The reasoning of this claim is elaborated in Section 5.6.

Table 5.2. Comparative results for makespan and computational time (sec.) for 2-stage problem set

Problem	n	LB	TS				ACO			
			Avg. C_{max}	Best C_{max}	Mean dev. %	Avg. Time	Avg. C_{max}	Best C_{max}	Mean dev. %	Avg. Time
S2-P1	22	30	30	30*	0.0	0.02	30	30*	0.0	0.03
S2-P2	32	10	11	11	10.0	0.05	11	11	10.0	0.03
S2-P3	24	18	18	18*	0.0	0.00	18	18*	0.0	0.03
S2-P4	20	16	18	18	12.5	0.02	18	18	12.5	0.02
S2-P5	29	27	31	31	14.8	0.04	31	31	14.8	0.02
S2-P6	30	12	12	12*	0.0	0.00	12	12*	0.0	0.04
S2-P7	28	27	30	30	11.1	0.02	30	30	11.1	0.03
S2-P8	10	16	16	16*	0.0	0.00	16	16*	0.0	0.01
S2-P9	30	22	22	22*	0.0	0.00	22	22*	0.0	0.03
S2-P10	16	28	28	28*	0.0	0.02	28	28*	0.0	0.02
S2-P11	40	14	14	14*	0.0	0.00	14	14*	0.0	0.04
S2-P12	34	36	39	39	8.3	0.04	39	39	8.3	0.03
S2-P13	12	26	26	26*	0.0	0.00	26	26*	0.0	0.02
S2-P14	14	24	24	24*	0.0	0.00	24	24*	0.0	0.02
S2-P15	32	30	33	33	10.0	0.05	33	33	10.0	0.03
S2-P16	34	30	33	33	10.0	0.04	33	33	10.0	0.03
S2-P17	15	18	18	18*	0.0	0.00	18	18*	0.0	0.02
S2-P18	13	22	22	22*	0.0	0.02	22	22*	0.0	0.02
S2-P19	16	26	26	26*	0.0	0.02	26	26*	0.0	0.02
S2-P20	12	12	12	12*	0.0	0.00	12	12*	0.0	0.02
S2-P21	22	28	32	32	14.3	0.05	32	32	14.3	0.03
S2-P22	25	22	22	22*	0.0	0.03	22	22*	0.0	0.03
S2-P23	22	18	21	21	16.7	0.04	21	21	16.7	0.02
S2-P24	12	18	18	18*	18.0	0.00	18	18*	0.0	0.02
S2-P25	21	10	10	10*	0.0	0.02	10	10*	0.0	0.03

Performance of the ACO algorithm in 4-stage instances is illustrated in Table 5.3, together with the performances of the TS and SS/PR. Execution time statistics are not present under SS/PR in the table because they were not supplied in the paper as instance-based. Rather the overall average computational time was supplied, which is referred later in Table 5.6 for comparison of overall performance of the algorithms. Instead, results of the TS algorithm are

included in this table to compare the execution time of the algorithms. It is reasonable to use the instance-based time statistics of TS as a replacement for SS/PR, since the average computational time for 4-stage instances is higher in SS/PR than TS (Table 5.6). As can be observed from Table 5.3, ACO algorithm manages to reach the best makespan value in all instances. It reaches 3 new upper bounds, where 2 of them are provably optimal solutions. Average C_{max} values are also the lowest in ACO, except for S4-P1, where the difference with SS/PR is minimal and can be ignored. ACO is able to find up to 10% better makespan values (S4-P6) in average. The algorithm produces these favorable results in much less time than SS/PR. In 22 instances, it takes less than a second for ACO to reach the solution. The exceptionally high computational time in instance S4-P1 is due to problem structure. This issue is discussed further in Chapter 6.

Comparative results for 8-stage instances are given in Table 5.4. ACO reaches 4 new upper bounds, where 3 of them are optimum solutions. There is a single instance, S8-P3, where SS/PR had a lower makespan, also an optimum one. ACO produces lower makespan values in average. The three instances where average C_{max} is lower in SS/PR; S8-P3, S8-P8, S8-P24 had insignificantly small differences. Computational times required by ACO are much lower than the TS (also SS/PR, since it reported higher average values, see Table 5.6), as much as a 98% decrease can be observed. ACO reaches the solution in less than a second in 10 instances, and only in 5 instances the time is higher than 10 seconds.

Performance of the algorithm is particularly remarkable in the large-scale 16-stage instances, as given in Table 5.5. It reaches 9 new upper bounds, where 6 of them are provably optimal solutions. There are only 2 instances where the upper bounds provided by SS/PR are not reached. ACO produces lower makespan values in average. Again, the time performance of the algorithm is much better than TS. A decrease in computational time requirement is especially important in these large-scale instances. A 98% decrease is observable in average time.

Table 5.3. Comparative results for makespan and computational time (sec.) for 4-stage problem set

Problem	n	LB	TS				SS/PR				ACO			
			Avg. C_{max}	Best C_{max}	Mean dev. %	Avg. Time	Avg. C_{max}	Best C_{max}	Mean dev. %	Avg. Time	Avg. C_{max}	Best C_{max}	Mean dev. %	Avg. Time
S4-P1	49	26	33.0	33	26.9	19.59	**29.4**	**29**	**13.1**	29.5	**29**	13.5	11.37
S4-P2	39	21	**21.0**	**21***	**0.0**	0.00	**21.0**	**21***	**0.0**	**21.0**	**21***	**0.0**	0.07
S4-P3	63	48	54.0	51	12.5	3.79	51.0	51	6.3	**50.4**	**48***	**5.0**	6.07
S4-P4	38	27	30.0	30	11.1	7.15	**27.0**	**27***	**0.0**	**27.0**	**27***	**0.0**	0.16
S4-P5	56	32	**32.0**	**32***	**0.0**	0.00	**32.0**	**32***	**0.0**	**32.0**	**32***	**0.0**	0.12
S4-P6	60	25	28.1	28	12.4	12.39	28.0	28	12.0	**25.0**	**25***	**0.0**	0.67
S4-P7	53	36	40.0	40	11.1	10.33	**36.0**	**36***	**0.0**	**36.0**	**36***	**0.0**	0.07
S4-P8	40	33	34.0	34	3.0	14.46	**33.0**	**33***	**0.0**	**33.0**	**33***	**0.0**	0.07
S4-P9	65	39	48.0	47	23.1	26.00	41.9	**41**	7.4	**41.0**	**41**	**5.1**	0.16
S4-P10	53	56	**56.0**	**56***	**0.0**	0.00	**56.0**	**56***	**0.0**	**56.0**	**56***	**0.0**	0.09
S4-P11	55	40	40.6	**40***	1.5	0.00	**40.0**	**40***	**0.0**	**40.0**	**40***	**0.0**	0.10
S4-P12	58	30	37.2	36	24.0	1.10	**32.0**	**32**	**6.7**	**32.0**	**32**	**6.7**	0.08
S4-P13	37	40	**40.0**	**40***	**0.0**	0.00	**40.0**	**40***	**0.0**	**40.0**	**40***	**0.0**	0.06
S4-P14	42	45	48.0	48	6.7	16.80	**45.0**	**45***	**0.0**	**45.0**	**45***	**0.0**	0.08
S4-P15	28	36	**36.0**	**36***	**0.0**	0.00	**36.0**	**36***	**0.0**	**36.0**	**36***	**0.0**	0.05
S4-P16	28	32	34.0	34	6.3	0.14	**32.0**	**32***	**0.0**	**32.0**	**32***	**0.0**	0.11
S4-P17	90	32	37.6	36	17.5	36.00	**36.0**	**36**	**12.5**	**36.0**	**36**	**12.5**	0.13
S4-P18	30	24	28.0	28	16.7	3.40	**24.0**	**24***	**0.0**	**24.0**	**24***	**0.0**	0.08
S4-P19	63	36	37.0	37	2.8	7.64	**36.0**	**36***	**0.0**	**36.0**	**36***	**0.0**	0.09
S4-P20	62	60	68.5	**63**	14.2	3.27	**63.0**	**63**	**5.0**	**63.0**	**63**	**5.0**	0.10
S4-P21	64	32	37.0	**34**	15.6	7.32	**34.0**	**34**	**6.3**	**34.0**	**34**	**6.3**	0.09
S4-P22	58	35	36.7	**35***	4.9	0.30	**35.0**	**35***	**0.0**	**35.0**	**35***	**0.0**	0.08
S4-P23	61	21	24.6	24	17.1	5.14	24.0	24	14.3	**22.0**	**22**	**4.8**	1.09
S4-P24	54	44	46.6	46	5.9	17.72	**44.0**	**44***	**0.0**	**44.0**	**44***	**0.0**	0.10
S4-P25	34	21	23.5	22	11.9	8.79	**21.0**	**21***	**0.0**	**21.0**	**21***	**0.0**	0.17

Table 5.4. Comparative results for makespan and computational time (sec.) for 8-stage problem set

Problem	n	LB	TS				SS/PR			ACO			
			Avg. C_{max}	Best C_{max}	Mean dev. %	Avg. Time	Avg. C_{max}	Best C_{max}	Mean dev. %	Avg. C_{max}	Best C_{max}	Mean dev. %	Avg. Time
S8-P1	146	48	55.9	53	16.5	116.80	**52.0**	**52**	**8.3**	**52.0**	**52**	**8.3**	1.61
S8-P2	144	32	35.0	35	9.4	115.20	34.8	**34**	8.7	**34.0**	**34**	**6.3**	4.13
S8-P3	87	32	34.9	34	9.1	69.60	**32.8**	**32***	**2.5**	33.0	33	3.1	5.04
S8-P4	161	108	120.3	120	11.4	128.80	111.3	111	3.1	**109.2**	**108***	**1.1**	18.44
S8-P5	117	78	**78.0**	**78***	**0.0**	0.00	**78.0**	**78***	**0.0**	**78.0**	**78***	**0.0**	0.56
S8-P6	99	63	64.3	64	2.1	79.20	**63.0**	**63***	**0.0**	**63.0**	**63***	**0.0**	0.48
S8-P7	84	36	36.3	**36***	0.8	26.88	**36.0**	**36***	**0.0**	**36.0**	**36***	**0.0**	0.45
S8-P8	110	55	64.3	59	16.9	88.00	**58.0**	**58**	**5.5**	58.9	**58**	7.1	3.80
S8-P9	128	42	43.6	43	3.8	61.44	42.1	**42***	0.2	**42.0**	**42***	**0.0**	9.74
S8-P10	90	32	37.3	36	16.6	72.00	**34.0**	**34**	**6.3**	**34.0**	**34**	**6.3**	4.25
S8-P11	102	45	**45.0**	**45***	**0.0**	0.00	**45.0**	**45***	**0.0**	**45.0**	**45***	**0.0**	0.47
S8-P12	92	60	62.8	61	4.7	73.60	60.2	**60***	0.3	**60.0**	**60***	**0.0**	3.86
S8-P13	101	35	38.1	36	8.9	80.80	36.0	36	2.9	**35.5**	**35***	**1.4**	28.39
S8-P14	72	84	85.1	**84***	1.3	28.80	**84.0**	**84***	**0.0**	**84.0**	**84***	**0.0**	0.34
S8-P15	100	60	61.9	61	3.2	32.00	**60.0**	**60***	**0.0**	**60.0**	**60***	**0.0**	3.04
S8-P16	81	70	76.2	76	8.9	64.80	75.0	75	7.1	**74.6**	**73**	**6.6**	14.39
S8-P17	100	60	60.2	**60***	0.3	16.00	**60.0**	**60***	**0.0**	**60.0**	**60***	**0.0**	0.42
S8-P18	106	56	58.0	**56***	3.6	25.44	**56.0**	**56***	**0.0**	**56.0**	**56***	**0.0**	0.58
S8-P19	108	75	78.0	78	4.0	86.40	77.1	**75***	2.8	**75.0**	**75***	**0.0**	7.33
S8-P20	105	49	49.9	**49***	1.8	25.20	**49.0**	**49***	**0.0**	**49.0**	**49***	**0.0**	0.41
S8-P21	152	42	42.5	**42***	1.2	36.48	**42.0**	**42***	**0.0**	**42.0**	**42***	**0.0**	1.19
S8-P22	104	30	**30.0**	**30***	**0.0**	0.00	**30.0**	**30***	**0.0**	**30.0**	**30***	**0.0**	0.41
S8-P23	97	75	**75.0**	**75***	**0.0**	0.00	**75.0**	**75***	**0.0**	**75.0**	**75***	**0.0**	0.47
S8-P24	104	35	38.0	**36**	8.6	83.20	**36.0**	**36**	**2.9**	36.1	**36**	3.1	35.10
S8-P25	101	35	37.2	36	6.3	80.80	36.0	36	2.9	**35.5**	**35***	**1.4**	34.75

Overall performance of the algorithms is compared in Table 5.6. In all four statistics compared in the table, ACO has the highest performance in all problem sizes. Its performance in reaching optimal solutions is particularly prevalent, even in the large size 16-stage instances. The time performance of the algorithm is very favorable, even a 10-fold decrease can be observed. Although, SS/PR produced higher quality results than TS, it required more time to accomplish this. However, it is quite the opposite for ACO. It manages to produce higher quality results than both TS and SS/PR in much less time. Further runtime analysis of the algorithm is given in Section 5.5.6.

5.5. Analysis of Algorithm and Results

This subsection introduces several further analyses about the proposed ACO algorithm to analyze the dynamics of the algorithm and assess the strengths and weakness of it.

How problem size can be defined in MPOS problem and are size of the test problems realistic or are they imaginary small cases not applicable in real life? These questions are dealt with in Section 5.5.1. Contributions of algorithm elements are analyzed in Section 5.5.2. The algorithm visits how many different solutions until it returns the best? Associated analysis is reported as instance-based in Section 5.5.3. The algorithm's stability across several runs on the same instance is assessed in Section 5.5.4. The succeeding Section 5.5.5 compares the configurations of the computers used in the algorithms referred in performance comparisons in the previous section. Section 5.5.6 gives a graphical runtime analysis of the algorithms. Lastly, statistical significance of algorithm results is evaluated in Section 5.5.7.

5.5.1. Problem size

Problem size is defined by the size of the input. In MPOS problem, input includes number of stages (s), number of jobs (n), number of machines (m_i) in each stage and processing times of jobs in stages (p_{ji}). Hence, the problem size is defined as in (5.3).

Table 5.5. Comparative results for makespan and computational time (sec.) for 16-stage problem set

Problem	n	LB	TS				SS/PR				ACO			
			Avg. C_{max}	Best C_{max}	Mean dev. %	Avg. Time	Avg. C_{max}	Best C_{max}	Mean dev. %		Avg. C_{max}	Best C_{max}	Mean dev. %	Avg. Time
S16-P1	88	135	144.6	144	7.1	140.80	144.0	144	6.7		**141.0**	**141**	**4.4**	58.98
S16-P2	102	99	100.0	**99***	1.0	16.32	**99.0**	**99***	**0.0**		**99.0**	**99***	**0.0**	6.33
S16-P3	99	140	146.2	144	4.4	158.40	143.0	143	2.1		**140.9**	**140***	**0.6**	43.13
S16-P4	90	104	108.2	**106**	4.0	144.00	107.3	107	3.2		**107.2**	107	**3.1**	22.65
S16-P5	96	100	108.2	**104**	8.2	153.60	**104.0**	**104**	**4.0**		**104.0**	**104**	**4.0**	3.72
S16-P6	104	143	147.1	145	2.9	166.40	144.0	144	0.7		**143.2**	**143***	**0.1**	8.57
S16-P7	106	121	124.2	123	2.6	166.90	122.9	122	1.6		**122.6**	**121***	**1.3**	56.75
S16-P8	81	63	**63.0**	**63***	**0.0**	0.00	**63.0**	**63***	**0.0**		**63.0**	**63***	**0.0**	2.48
S16-P9	101	121	124.2	**121***	2.6	129.28	121.9	**121***	0.7		**121.0**	**121***	**0.0**	3.56
S16-P10	96	120	122.2	121	1.8	153.60	121.0	121	0.8		**120.0**	**120***	**0.0**	35.81
S16-P11	93	80	**80.0**	**80***	**0.0**	0.00	**80.0**	**80***	**0.0**		**80.0**	**80***	**0.0**	5.58
S16-P12	110	154	166.7	166	8.2	176.00	164.0	163	6.5		**162.0**	**162**	**5.2**	65.34
S16-P13	112	180	188.1	183	4.5	180.00	181.8	**180***	1.0		**180.0**	**180***	**0.0**	16.94
S16-P14	97	84	88.0	88	4.8	155.20	86.0	85	2.4		**85.0**	**84***	**1.2**	66.12
S16-P15	86	126	130.3	**127**	3.4	137.60	**127.1**	**127**	0.9		127.7	**127**	1.3	28.49
S16-P16	106	56	59.2	**57**	5.7	169.60	57.3	**57**	2.3		**57.0**	**57**	**1.8**	5.79
S16-P17	94	70	71.3	**70***	1.9	45.12	70.3	**70***	0.4		**70.2**	**70***	**0.3**	48.13
S16-P18	102	110	115.0	112	4.5	163.20	110.2	**110***	0.2		**110.0**	**110***	**0.0**	6.11
S16-P19	80	112	124.6	117	11.3	128.00	**115.9**	**115**	**3.5**		117.9	117	5.3	51.56
S16-P20	84	90	94.2	**92**	4.7	134.40	**92.0**	**92**	**2.2**		**92.0**	**92**	**2.2**	1.86
S16-P21	78	88	94.3	**90**	7.2	124.80	**90.0**	**90**	**2.3**		92.3	92	4.9	33.72
S16-P22	79	60	62.2	**60***	3.7	50.56	**60.0**	**60***	**0.0**		**60.0**	**60***	**0.0**	5.42
S16-P23	97	70	74.7	73	6.7	155.20	73.0	73	4.3		**72.9**	**72**	**4.1**	49.45
S16-P24	93	60	60.5	**60***	0.8	14.88	**60.0**	**60***	**0.0**		**60.0**	**60***	**0.0**	6.08
S16-P25	96	120	122.6	121	2.2	153.60	121.0	121	0.8		**120.1**	**120***	**0.1**	33.08

Table 5.6. Summary comparative statistics for the testbed

	2-stage		4-stage			8-stage			16-stage		
	TS[ϑ]	ACO	TS[ϑ]	SS/PR[δ]	ACO	TS[ϑ]	SS/PR[δ]	ACO	TS[ϑ]	SS/PR[δ]	ACO
Average deviation of mean C_{max} from LB (%)	4.31	4.31	9.80	3.30	**2.35**	5.56	2.10	**1.79**	4.17	1.9	**1.6**
Average deviation of best C_{max} from LB (%)	4.31	4.31	7.63	3.18	**2.07**	3.41	1.79	**1.46**	2.08	1.6	**1.33**
No. of provably optimal solutions (out of 25)	16	16	7	16	**18**	10	16	**18**	7	9	**15**
Average computational time (sec.)	0.02	0.02	8.05	9.33	**0.85**	55.66	59.68	**7.19**	120.81	207.93	**26.63**

ϑ: TS proposed by Abdelmaguid et al. (2014)
δ: SS/PR proposed by Abdelmaguid (2020)

$$\text{Problem size} := s \cup n \cup m_i \cup p_{ji} \qquad j \in \mathcal{J};\ \ i \in \mathcal{S} \tag{5.3}$$

However, if total number of machines across stages exceeds n, then the problem becomes trivial to solve; if it is less than n, a bottleneck occurs in the problem. Thus, $\min(n, \sum m_i)$ can be considered as a determinant in problem size. Time-complexity of the problem -defined by an approximation algorithm- would be in order of the problem size elements.

Real-world cases of the MPOS problem at hand are exemplified in Section 1.1. It can be seen that the size of the problems dealt with in this study -up to 16 stages with up to 10 machines in a stage- is sufficiently large to enable the application of the current proposal to real-world sized problems.

5.5.2. Contributions of algorithm elements

The proposed ACO has the following structural algorithm elements that are specifically defined for the current proposal. Each element is either a completely new approach or an adopted version of an existing approach.

Algorithm elements:

- Implicit-stage permutation representation
- Generation of complete random solutions (Random exploration)
- Pheromone information
- Heuristic information
- Local exploration

To assess the contribution of an element to overall performance of the algorithm, only that element should be excluded or replaced by an alternative engine and remaining parts of the algorithm should be kept the same, then the algorithm should be re-run. However, this is not possible for any element except for the local exploration. Replacing the current solution representation with operation-permutation representation, for example, would require the pheromone and heuristic information to be re-defined for the new representation, as they are tailored approaches for implicit-stage permutation representation. That means one or more

algorithm elements would be changed concurrently, which would prevent examining the exact contribution of an element. Again, to evaluate the contribution of the random exploration routine, instead can be used the commonly applied biased stochastic exploration where the probability of a solution component depends on the value of quality function, $Q_{\xi\varphi} = \tau_{\xi\varphi}^{\alpha}\eta_{\xi\varphi}^{\beta}$. Thus, the contributing effects of the pheromone and heuristic information would again be present in the replacing exploration routine. That would again prevent examining the exact contribution of an element.

Local exploration routine contributed significantly to improve the results for large-scale instances, especially 16-stage ones. To observe the favorable contribution of the module, the algorithm is run with and without local exploration routine for 8 and 16-stage instances.

Table 5.7 gives the percent change caused by local exploration in best makespan and average makespan values of the 10 runs for 8-stage instances. Average computational times are also included in the table. LE enabled the algorithm to reach 2 new upperbounds; one of them being a provably optimal solution. However, there is an instance, S8-P10, where ACO without LE reaches a lower upperbound. Average solution quality produced by the algorithm does not necessarily improve with the inclusion of LE. Indeed, an apparent contribution of the LE module to solution quality is not present in 8-stage instances. However, there are considerable decreases in computational time caused by LE in certain instances. But the decreasing trend in average time is not common in all instances. There are cases where the inclusion of the module increases the computational time.

The changing effect of LE in solution quality is due to working principles of the module. LE serves the following purpose in the algorithm. It generates schedules where jobs have a distributed view of their respective locations in machines. If this has no role in decreasing makespan of the problem instance at hand (either perfectly balanced stages are few in number or the problem size is small and not being affected from the presence of those stages), then it may serve no purpose. However, when the problem size becomes increasingly large, searching for a better solution around a single permutation generates better schedules most of the time. Also, the presence of perfectly balanced stages causes great delay if the problem size is large, and LE routine does have a decreasing role in makespan (see Section 4.3.2).

Table 5.7. Contribution of local exploration routine in 8-stage instances

		ACO (No LE)			ACO + LE			Percent Improvement by LE	
	LB	Best C_{max}	Avg. C_{max}	Avg. Time	Best C_{max}	Avg. C_{max}	Avg. Time	Best C_{max} (%)	Avg. C_{max} (%)
S8-P1	48	52	52.5	29.60	52	**52**	**1.61**	--	**0.95**
S8-P2	32	34	34	30.99	34	34	**4.13**	--	--
S8-P3	32	33	33	**2.89**	33	33	5.04	--	--
S8-P4	108	111	111	**12.57**	**108***	**109.2**	18.44	**2.70**	**1.62**
S8-P5	78	78*	78	**0.54**	78*	78	0.56	--	--
S8-P6	63	63*	63	0.68	63*	63	**0.48**	--	--
S8-P7	36	36*	36	**0.40**	36*	36	0.45	--	--
S8-P8	55	58	59.1	50.29	58	**58.9**	**3.80**	--	**0.34**
S8-P9	42	42*	42.1	40.11	42*	42	**9.74**	--	**0.24**
S8-P10	32	**33**	**33.9**	13.90	34	34	**4.25**	-3.03	-0.29
S8-P11	45	45*	45	**0.45**	45*	45	0.47	--	--
S8-P12	60	60*	60	4.12	60*	60	**3.86**	--	--
S8-P13	35	35*	35.6	**22.60**	35*	**35.5**	28.39	--	**0.28**
S8-P14	84	84*	84	1.33	84*	84	**0.34**	--	--
S8-P15	60	60*	60	22.71	60*	60	**3.04**	--	--
S8-P16	70	74	**74.3**	36.69	**73**	74.6	**14.39**	**1.35**	-0.40
S8-P17	60	60*	60	0.44	60*	60	**0.42**	--	--
S8-P18	56	56*	56	0.78	56*	56	**0.58**	--	--
S8-P19	75	75*	75	**7.24**	75*	75	7.33	--	--
S8-P20	49	49*	49	0.45	49*	49	**0.41**	--	--
S8-P21	42	42*	42	**1.14**	42*	42	1.19	--	--
S8-P22	30	30*	30	0.44	30*	30	**0.41**	--	--
S8-P23	75	75*	75	**0.40**	75*	75	0.47	--	--
S8-P24	35	36	**36**	**17.38**	36	36.1	35.10	--	-0.28
S8-P25	35	35*	**35.3**	**34.40**	35*	35.5	34.75	--	-0.57

The favorable contribution of the routine in large problem size can be observed in the results of 16-stage instances given in Table 5.8. In 13 instances LE causes the algorithm to reach a lower makespan, where 6 of them are provably optimal solutions. In 8 of these instances, even a shorter time is required to reach the higher quality results, while the improvements in the remaining instances are achieved at the expense of a higher computational time. 68% of the time, ACO with LE produces decreased makespan values in average. There are some cases

where the same solution quality is obtained in a shorter time with LE. This means that the algorithm without LE indeed reaches the permutation which would produce that solution. However, since LE is not in action different schedules around the permutation are not searched for, hence the computational time is increased. Again, there are cases where ACO with LE reaches same quality solution but in increased time. This is explained by the problem structure. If the problem instance at hand is a relatively simple one, then incorporating additional modules in the algorithm leads to increased computational time.

Table 5.8. Contribution of local exploration routine in 16-stage instances

		ACO (No LE)			ACO + LE			Percent Improvement by LE	
	LB	Best C_{max}	Avg. C_{max}	Avg. Time	Best C_{max}	Avg. C_{max}	Avg. Time	Best C_{max} (%)	Avg. C_{max} (%)
S16-P1	135	150	150	**0.84**	**141**	**141**	58.98	**6.00**	**6.00**
S16-P2	99	99*	99	**1.80**	99*	99	6.33	--	--
S16-P3	140	145	146.8	**9.66**	**140***	**140.9**	43.13	**3.45**	**4.02**
S16-P4	104	108	109.7	69.22	**107**	**107.2**	**22.65**	**0.93**	**2.28**
S16-P5	100	108	108.2	77.21	**104**	**104**	**3.72**	**3.70**	**3.88**
S16-P6	143	143*	145.2	10.45	143*	**143.2**	**8.57**	--	**1.38**
S16-P7	121	123	123.3	75.47	**121***	**122.6**	**56.75**	**1.63**	**0.57**
S16-P8	63	63*	63	**2.14**	63*	63	2.48	--	--
S16-P9	121	121*	121	27.32	121*	121	**3.56**	--	--
S16-P10	120	120*	121	49.66	120*	**120**	**35.81**	--	**0.83**
S16-P11	80	80*	80	43.04	80*	80	**5.58**	--	--
S16-P12	154	167	167.9	**8.84**	**162**	**162**	65.34	**2.99**	**3.51**
S16-P13	180	189	189.9	18.81	**180***	**180**	**16.94**	**4.76**	**5.21**
S16-P14	84	88	88	**20.42**	**84***	**85**	66.12	**4.55**	**3.41**
S16-P15	126	**126***	**126**	**1.46**	127	127.7	28.49	-0.79	-1.35
S16-P16	56	58	59.4	45.06	**57**	**57**	**5.79**	**1.72**	**4.04**
S16-P17	70	70*	70.2	81.71	70*	70.2	**48.13**	--	--
S16-P18	110	114	114.6	74.09	**110***	**110**	**6.11**	**3.51**	**4.01**
S16-P19	112	117	117.9	**3.80**	117	117.9	51.56	--	--
S16-P20	90	94	94	48.58	**92**	**92**	**1.86**	**2.13**	**2.13**
S16-P21	88	**91**	**92.2**	59.65	92	92.3	**33.72**	-1.10	-0.11
S16-P22	60	60*	60.3	87.36	60*	**60**	**5.42**	--	**0.50**
S16-P23	70	75	75.7	**38.19**	**72**	**72.9**	49.45	**4.00**	**3.70**
S16-P24	60	60*	60.4	65.99	60*	**60**	**6.08**	--	**0.66**
S16-P25	120	121	121.8	81.16	**120***	**120.1**	**33.08**	**0.83**	**1.40**

5.5.3. Number of objective function evaluations

One of the important analysis about an algorithm is its capability to reach the solution in a reasonable number of solutions visited. It is already possible for most of the search algorithms to reach a high-quality solution if it is given a sufficiently long time. It would visit millions of solutions, even may return the optimum. There are convergence analyses of various algorithms in the literature to prove that the algorithm would yield an optimal solution if it is allowed enough time.

The strength of an algorithm is about searching the solution space -which is typically extremely large- in an *efficient* manner to reach a high-quality solution. Otherwise, it would return to a process of simple enumerating. To measure the performance of the algorithm in this regard, it is common to report the number of objective function evaluations. It is an indicator of how many solutions the algorithm has dealt with.

Table 5.9 gives the minimum, maximum and average number of objective function evaluations in 10 runs for 2-stage instances. Remind that 11 solutions are visited in each iteration of the algorithm. The table shows that at each 2-stage instance ACO finds the solution -the optimum as explained in Section 5.6- in the first iteration. This makes 2-stage problem size trivial to solve for the algorithm. Also, note that the 2-stage problem sizes of the testbed may not be regarded as very small-sized since they include up to 40 number of jobs and 20 machines in a stage.

In 12 of 25 4-stage instances, ACO manages to find the solution in the first iteration in all the 10 runs, as given in Table 5.10. 8 of those solutions are provably optimal solutions. There are only 2 instances where the maximum number of solutions visited exceeds 1000. Average number of evaluations are very favorable even in these instances. Overall, the algorithm visits minimal number of solutions to find the reported high-quality solutions. This feature of the algorithm continues to exist even in large-sized 8 and 16-stage instances as can be observed in Table 5.11 and Table 5.12, respectively. There are still 8 8-stage instances where ACO finds the solution -the optimum in all 8- in the first iteration at every run. It was reported only in 4 instances in TS (as the Avg. Time column in Table 5.4 implies). The maximum number

Table 5.9. Run statistics for 2-stage instances

Problem	LB	Best C_{max}	Avg. C_{max}	Variation		Number of objective function evaluations		
				Standard deviation	Coefficient of variation (CV)	Min	Max	Avg.
S2-P1	30	30*	30	0.00	0.00	11	11	11
S2-P2	10	11	11	0.00	0.00	11	11	11
S2-P3	18	18*	18	0.00	0.00	11	11	11
S2-P4	16	18	18	0.00	0.00	11	11	11
S2-P5	27	31	31	0.00	0.00	11	11	11
S2-P6	12	12*	12	0.00	0.00	11	11	11
S2-P7	27	30	30	0.00	0.00	11	11	11
S2-P8	16	16*	16	0.00	0.00	11	11	11
S2-P9	22	22*	22	0.00	0.00	11	11	11
S2-P10	28	28*	28	0.00	0.00	11	11	11
S2-P11	14	14*	14	0.00	0.00	11	11	11
S2-P12	36	39	39	0.00	0.00	11	11	11
S2-P13	26	26*	26	0.00	0.00	11	11	11
S2-P14	24	24*	24	0.00	0.00	11	11	11
S2-P15	30	33	33	0.00	0.00	11	11	11
S2-P16	30	33	33	0.00	0.00	11	11	11
S2-P17	18	18*	18	0.00	0.00	11	11	11
S2-P18	22	22*	22	0.00	0.00	11	11	11
S2-P19	26	26*	26	0.00	0.00	11	11	11
S2-P20	12	12*	12	0.00	0.00	11	11	11
S2-P21	28	32	32	0.00	0.00	11	11	11
S2-P22	22	22*	22	0.00	0.00	11	11	11
S2-P23	18	21	21	0.00	0.00	11	11	11
S2-P24	18	18*	18	0.00	0.00	11	11	11
S2-P25	10	10*	10	0.00	0.00	11	11	11

of objective function evaluations required in the remaining 8-stage instances are also remarkable by not exceeding 3000 in any instance, with an average of at most 500s.

In most of 16-stage problem instances, there are runs where the algorithm finds the solution at the first iteration. However, it is not the case for all runs in no instance. That is, ACO visits 15 to 610 number of solutions in average to find the solution. The numbers are quite low, particularly for a MPOS problem of that size. The high performance of the algorithm in this

regard is due to very efficient representation of solution, intense use of problem knowledge and high exploit of search knowledge.

Table 5.10. Run statistics for 4-stage instances

Problem	LB	Best C_{max}	Avg. C_{max}	Variation		Number of objective function evaluations		
				Standard deviation	Coefficient of variation (CV)	Min	Max	Avg.
S4-P1	26	29	29.5	0.53	1.79	55	6600	1786.4
S4-P2	21	21*	21.0	0.00	0.00	11	11	11
S4-P3	48	48*	50.4	1.26	2.51	11	2750	492.8
S4-P4	27	27*	27.0	0.00	0.00	11	286	67.1
S4-P5	32	32*	32.0	0.00	0.00	11	22	14.3
S4-P6	25	25*	25.0	0.00	0.00	11	209	48.4
S4-P7	36	36*	36.0	0.00	0.00	11	11	11
S4-P8	33	33*	33.0	0.00	0.00	11	11	11
S4-P9	39	41	41.0	0.00	0.00	11	121	36.3
S4-P10	56	56*	56.0	0.00	0.00	11	44	15.4
S4-P11	40	40*	40.0	0.00	0.00	11	22	13.2
S4-P12	30	32	32.0	0.00	0.00	11	11	11
S4-P13	40	40*	40.0	0.00	0.00	11	11	11
S4-P14	45	45*	45.0	0.00	0.00	11	66	18.7
S4-P15	36	36*	36.0	0.00	0.00	11	11	11
S4-P16	32	32*	32.0	0.00	0.00	11	44	20.9
S4-P17	32	36	36.0	0.00	0.00	11	11	11
S4-P18	24	24*	24.0	0.00	0.00	11	44	20.9
S4-P19	36	36*	36.0	0.00	0.00	11	11	11
S4-P20	60	63	63.0	0.00	0.00	11	11	11
S4-P21	32	34	34.0	0.00	0.00	11	11	11
S4-P22	35	35*	35.0	0.00	0.00	11	11	11
S4-P23	21	22	22.0	0.00	0.00	22	352	118.8
S4-P24	44	44*	44.0	0.00	0.00	11	11	11
S4-P25	21	21*	21.0	0.00	0.00	11	143	42.9

5.5.4. Robustness

It is vital for a solution approach to produce similar quality solutions across several runs on the same instance. This makes the method more robust and trustworthy. To measure the robustness of the algorithm, variation in the results of 10 runs are analyzed for each of 100

problem instances. Standard deviation (SD) and coefficient of variation (CV) are the two measures of variation calculated in this study.

Table 5.11. Run statistics for 8-stage instances

Problem	LB	Best C_{max}	Avg. C_{max}	Variation		Number of objective function evaluations		
				Standard deviation	Coefficient of variation (CV)	Min	Max	Avg.
S8-P1	48	52	52.0	0.00	0.00	11	242	49.5
S8-P2	32	34	34.0	0.00	0.00	11	154	52.8
S8-P3	32	33	33.0	0.00	0.00	11	374	108.9
S8-P4	108	108*	109.2	1.55	1.42	11	737	115.5
S8-P5	78	78*	78.0	0.00	0.00	11	11	11
S8-P6	63	63*	63.0	0.00	0.00	11	11	11
S8-P7	36	36*	36.0	0.00	0.00	11	33	17.6
S8-P8	55	58	58.9	0.32	0.54	44	165	90.2
S8-P9	42	42*	42.0	0.00	0.00	11	88	29.7
S8-P10	32	34	34.0	0.00	0.00	11	55	25.3
S8-P11	45	45*	45.0	0.00	0.00	11	11	11
S8-P12	60	60*	60.0	0.00	0.00	11	594	205.7
S8-P13	35	35*	35.5	0.53	1.48	11	1749	397.1
S8-P14	84	84*	84.0	0.00	0.00	11	11	11
S8-P15	60	60*	60.0	0.00	0.00	11	1287	250.8
S8-P16	70	73	74.6	0.70	0.94	11	2365	563.2
S8-P17	60	60*	60.0	0.00	0.00	11	11	11
S8-P18	56	56*	56.0	0.00	0.00	11	22	12.1
S8-P19	75	75*	75.0	0.00	0.00	77	682	226.6
S8-P20	49	49*	49.0	0.00	0.00	11	22	13.2
S8-P21	42	42*	42.0	0.00	0.00	11	11	11
S8-P22	30	30*	30.0	0.00	0.00	11	11	11
S8-P23	75	75*	75.0	0.00	0.00	11	11	11
S8-P24	35	36	36.1	0.32	0.88	11	2002	954.8
S8-P25	35	35*	35.5	0.53	1.48	11	2233	795.3

SD measures the variance among the elements of a sample by considering the distance of each element from the sample mean. It is in units of the mean and calculated by the following formula.

$$SD = \sqrt{\frac{\sum_{run}[\,(C_{max})_{run} - Avg.C_{max}]^2}{Sample\ size - 1}} \tag{5.4}$$

where the sample is the set of 10 runs carried out on every instance.

CV is the representation of the variation in a sample as a proportion, hence a unitless measure of variation. It is used to compare the deviation in different samples with different means. It is the ratio of the SD to the sample mean, expressed as a percentage. Then the formula for CV is as follows:

$$CV = \frac{SD}{sample\ mean} \times 100 \tag{5.5}$$

SD and CV calculations for the runs of 2-stage instances are given in Table 5.9. Since, the proposed ACO gives the optimum at every run of those instances, the SD and CV values are all zero. For 4-stage instances, SD and CV are given in Table 5.10. ACO algorithm shows a robust behavior and produces same results in almost all 4-stage problems, as there are only 2 instances where SD is greater than zero. However, the deviation is minimal in these 2 instances with a CV of less than 3%.

The robust behavior of the algorithm continues, even more strongly, in the larger 8 and 16-stage instances, as can be observed in Table 5.11 and Table 5.12, respectively. In only 6 of 8-stage instances SD is greater than zero: a variation of below 2% at most. In 16-stage ones, there is only a single instance where the variation goes beyond 1%, which is already minimal.

5.5.5. Comparison of computer configurations

When the computational times required by different algorithms are compared for a process, configurations of the computers should be assessed to be fair. It should be differentiated whether the speed superiority of the algorithm is due to effectiveness of subroutines or it is a benefit derived by a high-performance computer.

Table 5.12. Run statistics for 16-stage instances

Problem	LB	Best C_{max}	Avg. C_{max}	Variation		Number of objective function evaluations		
				Standard deviation	Coefficient of variation (CV)	Min	Max	Avg.
S16-P1	135	141	141.0	0.00	0.00	11	2398	610.5
S16-P2	99	99*	99.0	0.00	0.00	11	66	40.7
S16-P3	140	140*	140.9	1.29	0.91	33	704	347.6
S16-P4	104	107	107.2	0.63	0.59	55	1804	430.1
S16-P5	100	104	104.0	0.00	0.00	11	44	18.7
S16-P6	143	143*	143.2	0.63	0.44	11	55	29.7
S16-P7	121	121*	122.6	0.97	0.79	11	693	184.8
S16-P8	63	63*	63.0	0.00	0.00	11	99	40.7
S16-P9	121	121*	121.0	0.00	0.00	11	187	45.1
S16-P10	120	120*	120.0	0.00	0.00	11	891	226.6
S16-P11	80	80*	80.0	0.00	0.00	33	275	133.1
S16-P12	154	162	162.0	0.00	0.00	132	682	333.3
S16-P13	180	180*	180.0	0.00	0.00	33	198	90.2
S16-P14	84	84*	85.0	0.94	1.11	44	913	282.7
S16-P15	126	127	127.7	0.48	0.38	11	616	214.5
S16-P16	56	57	57.0	0.00	0.00	11	44	20.9
S16-P17	70	70*	70.2	0.42	0.60	55	1683	485.1
S16-P18	110	110*	110.0	0.00	0.00	11	88	45.1
S16-P19	112	117	117.9	0.32	0.27	110	825	390.5
S16-P20	90	92	92.0	0.00	0.00	11	33	15.4
S16-P21	88	92	92.3	0.48	0.52	11	1760	293.7
S16-P22	60	60*	60.0	0.00	0.00	11	1089	201.3
S16-P23	70	72	72.9	0.57	0.78	11	814	180.4
S16-P24	60	60*	60.0	0.00	0.00	11	187	69.3
S16-P25	120	120*	120.1	0.32	0.26	11	484	137.5

The solution quality produced by the proposed ACO was compared with the state-of-the-art SS/PR algorithm on the problem testbed, as reported in Section 5.4. SS/PR research reported only the overall stage-based average time statistics, not the instance-based ones. These overall results implied that SS/PR required higher computational times than the previous best algorithm, TS. ACO produced better quality results than SS/PR in much less computational time.

Table 5.13 gives the configurations of the computers used in TS and SS/PR research as well as the one used in this study for ACO. The table shows that although the computer used to program ACO in this study had a lower quality compared to the one used for SS/PR, the algorithm managed to find higher quality results in less computational time. Thus, it is not the computer qualifications that lead to rapid and high-quality results of this study but rather it is the good performance of the proposed algorithm.

Table 5.13. Computer configurations of algorithm runs

	Programming language	Processor	CPU speed	RAM / Cache size	Operating system
TS	Visual C# 2010	Intel Core 2 Duo	1.83 GHz	2 MB Cache	--
SS/PR	C++	Intel Core i7	2.7 GHz	8 GB RAM	Windows 10
ACO	MATLAB 9.3	Intel Core i5	2.40 GHz	8 GB RAM	Windows 10

5.5.6. Runtime analysis

Figure 5.1 shows a graphical interpretation of how runtime changes with increasing problem size for ACO and other algorithms referred in this study: GA, TS and SS/PR. As the figure reveals out, SS/PR shows nearly an exponential increase in computational time and reported runtimes as high as the initial GA algorithm for large instances. The graph indicates that it may pass further GA if the problem size increases beyond 16 stages. Although it produced the highest quality results in the literature, it is certain that the runtime performance of the SS/PR algorithm should be enhanced. The ACO proposed in this study, on the other hand, shows a very favorable trend in its runtime increase as the problem size gets larger, as can be observed in the figure. The increase in runtime is almost linear with a very gentle slope.

5.5.7. Statistical significance of results

To assess whether the differences between the results of ACO and TS, and ACO and SS/PR are statistically significant, statistical analysis is conducted and presented in this subsection.

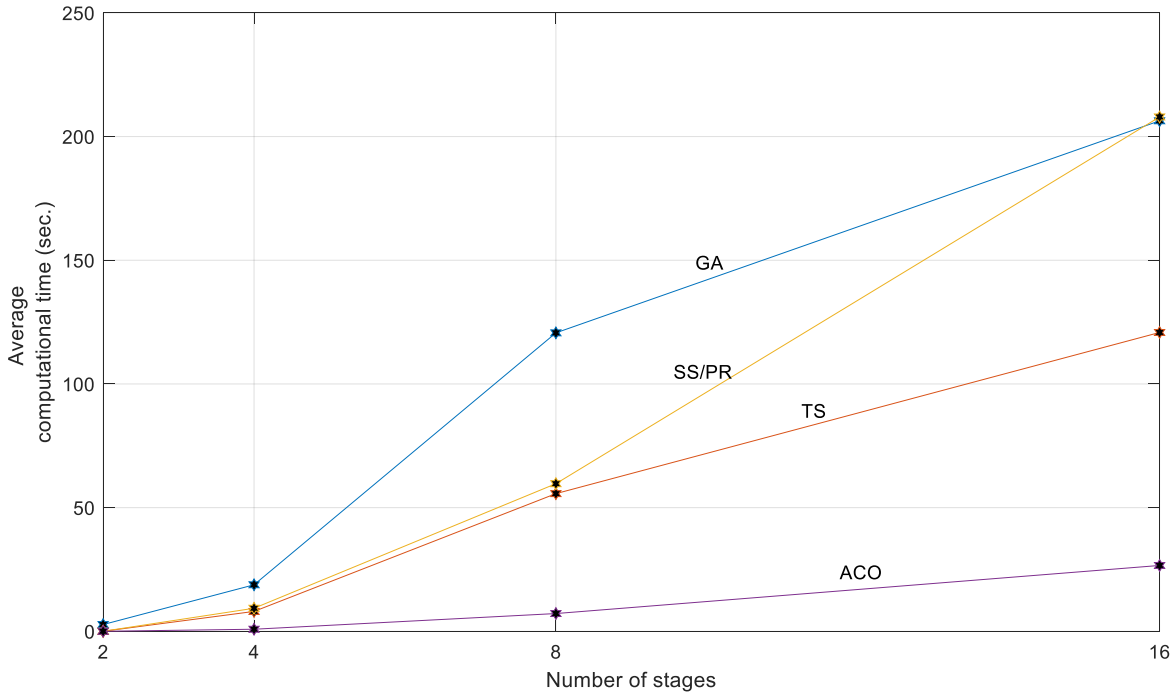

Figure 5.1. Change in computational time with increasing problem size

Paired t-test is the common statistical test used to measure whether the mean difference in paired values of two samples are significantly different. However, one important assumption of paired t-test is that the differences are required to follow a normal distribution. That is, the instance-based differences between Avg. C_{max} results of ACO and TS, and between ACO and SS/PR should follow a normal distribution to apply the parametric paired t-test. The instance-based Avg. C_{max} differences are given in Table 5.14 for ACO-TS and ACO-SS/PR. Note that, each of the 3 problem sizes (4, 8 and 16-stage problems) are considered as separate samples and the statistical tests are repeated for each, both for TS and SS/PR, which makes a total of 6 samples to carry out tests on. 2-stage problem set is not considered since all the three algorithms reported the same results for this set.

To observe whether each of the 6 samples of differences in Table 5.14 comes from a normally distributed population, normal probability plots are generated as given in Figure 5.2, Figure 5.3 and Figure 5.4. Neither of the plots suggest a normal distribution of the data points as they do not form a linear pattern. In ACO-TS plots, although the centers are more linear, tails show considerable departures from the fitted line. ACO-SS/PR plots, on the other hand, are

completely far from resembling a normal distribution. To further test the normality assumption of the populations, Kolmogorov-Smirnov test is also carried out.

Table 5.14. Differences between Avg. C_{max} results of the algorithms

	Avg. C_{max} Differences			Avg. C_{max} Differences			Avg. C_{max} Differences	
Problem	ACO − TS	ACO − SS/PR	Problem	ACO − TS	ACO − SS/PR	Problem	ACO − TS	ACO − SS/PR
S4-P1	-3.5	0.1	S8-P1	-3.9	0	S16-P1	-3.6	-3
S4-P2	0	0	S8-P2	-1	-0.8	S16-P2	-1	0
S4-P3	-3.6	-0.6	S8-P3	-1.9	0.2	S16-P3	-5.3	-2.1
S4-P4	-3	0	S8-P4	-11.1	-2.1	S16-P4	-1	-0.1
S4-P5	0	0	S8-P5	0	0	S16-P5	-4.2	0
S4-P6	-3.1	-3	S8-P6	-1.3	0	S16-P6	-3.9	-0.8
S4-P7	-4	0	S8-P7	-0.3	0	S16-P7	-1.6	-0.3
S4-P8	-1	0	S8-P8	-5.4	0.9	S16-P8	0	0
S4-P9	-7	-0.9	S8-P9	-1.6	-0.1	S16-P9	-3.2	-0.9
S4-P10	0	0	S8-P10	-3.3	0	S16-P10	-2.2	-1
S4-P11	-0.6	0	S8-P11	0	0	S16-P11	0	0
S4-P12	-5.2	0	S8-P12	-2.8	-0.2	S16-P12	-4.7	-2
S4-P13	0	0	S8-P13	-2.6	-0.5	S16-P13	-8.1	-1.8
S4-P14	-3	0	S8-P14	-1.1	0	S16-P14	-3	-1
S4-P15	0	0	S8-P15	-1.9	0	S16-P15	-2.6	0.6
S4-P16	-2	0	S8-P16	-1.6	-0.4	S16-P16	-2.2	-0.3
S4-P17	-1.6	0	S8-P17	-0.2	0	S16-P17	-1.1	-0.1
S4-P18	-4	0	S8-P18	-2	0	S16-P18	-5	-0.2
S4-P19	-1	0	S8-P19	-3	-2.1	S16-P19	-6.7	2
S4-P20	-5.5	0	S8-P20	-0.9	0	S16-P20	-2.2	0
S4-P21	-3	0	S8-P21	-0.5	0	S16-P21	-2	2.3
S4-P22	-1.7	0	S8-P22	0	0	S16-P22	-2.2	0
S4-P23	-2.6	-2	S8-P23	0	0	S16-P23	-1.8	-0.1
S4-P24	-2.6	0	S8-P24	-1.9	0.1	S16-P24	-0.5	0
S4-P25	-2.5	0	S8-P25	-1.7	-0.5	S16-P25	-2.5	-0.9

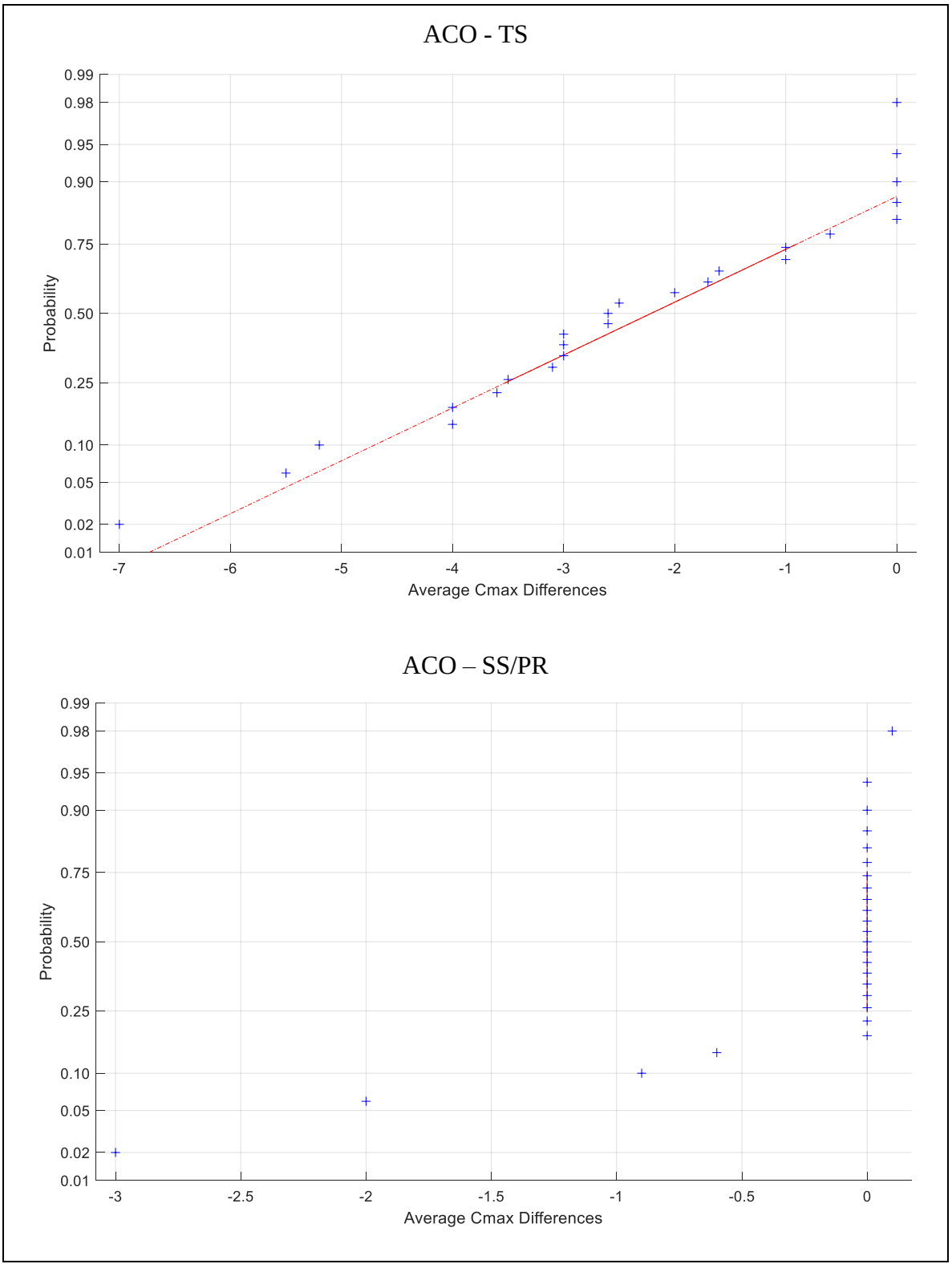

Figure 5.2. Normal probability plots of Avg. C_{max} differences for 4-stage instances

Kolmogorov-Smirnov test is a nonparametric test used to decide whether a sample is from a population with a hypothesized distribution. Here, it is hypothesized that the sample of differences is coming from a population with standard normal distribution. Thus, Kolmogorov-Smirnov tests the null hypothesis that the data follows a standard normal distribution against the alternative one. The test is applied for the 6 samples separately and the p-values are given in Table 5.15. p-values lead to strongly reject the null hypothesis at any significance level and conclude that the samples cannot be assumed to come from normally distributed populations.

Table 5.15. p-values of the samples of differences for Kolmogorov-Smirnov test

Sample	p-value
4-stage ACO – TS	1.96×10^{-9}
4-stage ACO – SS/PR	2.30×10^{-5}
8-stage ACO – TS	3.06×10^{-7}
8-stage ACO – SS/PR	0.000909
16-stage ACO – TS	2.04×10^{-12}
16-stage ACO – SS/PR	0.000937

Since normal distribution cannot be assumed for difference populations, non-parametric Wilcoxon Signed Rank test is used instead of the paired t-test to test whether the mean differences are significantly different from zero. It tests the null hypothesis that the median difference is zero against the alternative that it is positive.

p-values of the 6 samples for the Wilcoxon Signed Rank tests are given in Table 5.16. The p-values indicate that ACO produces significantly different results than TS at any significance level in all problem sizes. The difference between average C_{max} results of ACO and SS/PR is significant at 13% level for 4-stage problem size and at 1% level for 8-stage size. The levels of significance are high for these problem sizes and it may be regarded as the performance of ACO over SS/PR is insignificant. However, that is wrong because there remained little room for improvement, particularly in 4-stage instances, by the implementation of the SS/PR algorithm on the testbed. Despite, ACO manages to produce higher quality results where

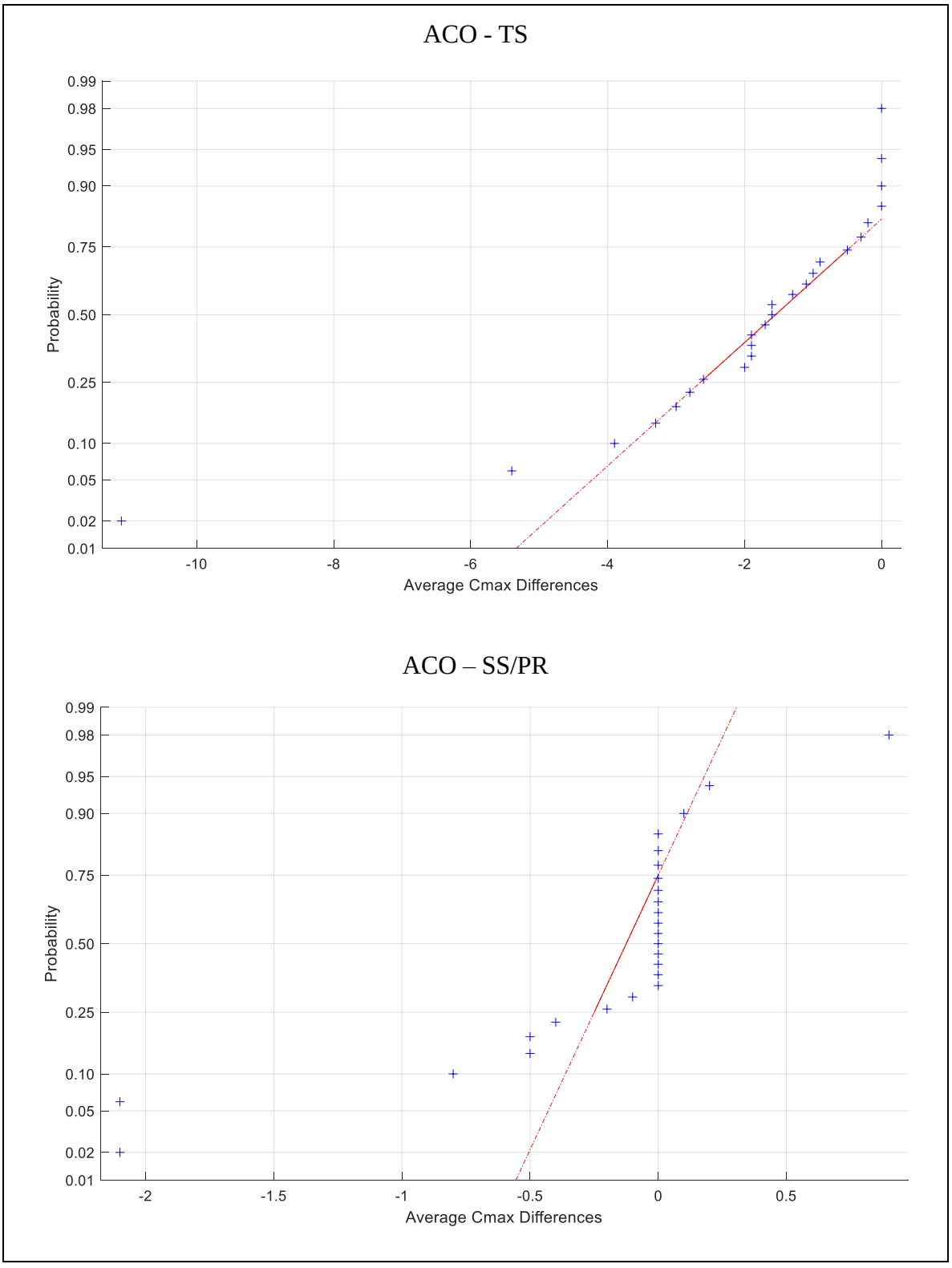

Figure 5.3. Normal probability plots of Avg. C_{max} differences for 8-stage instances

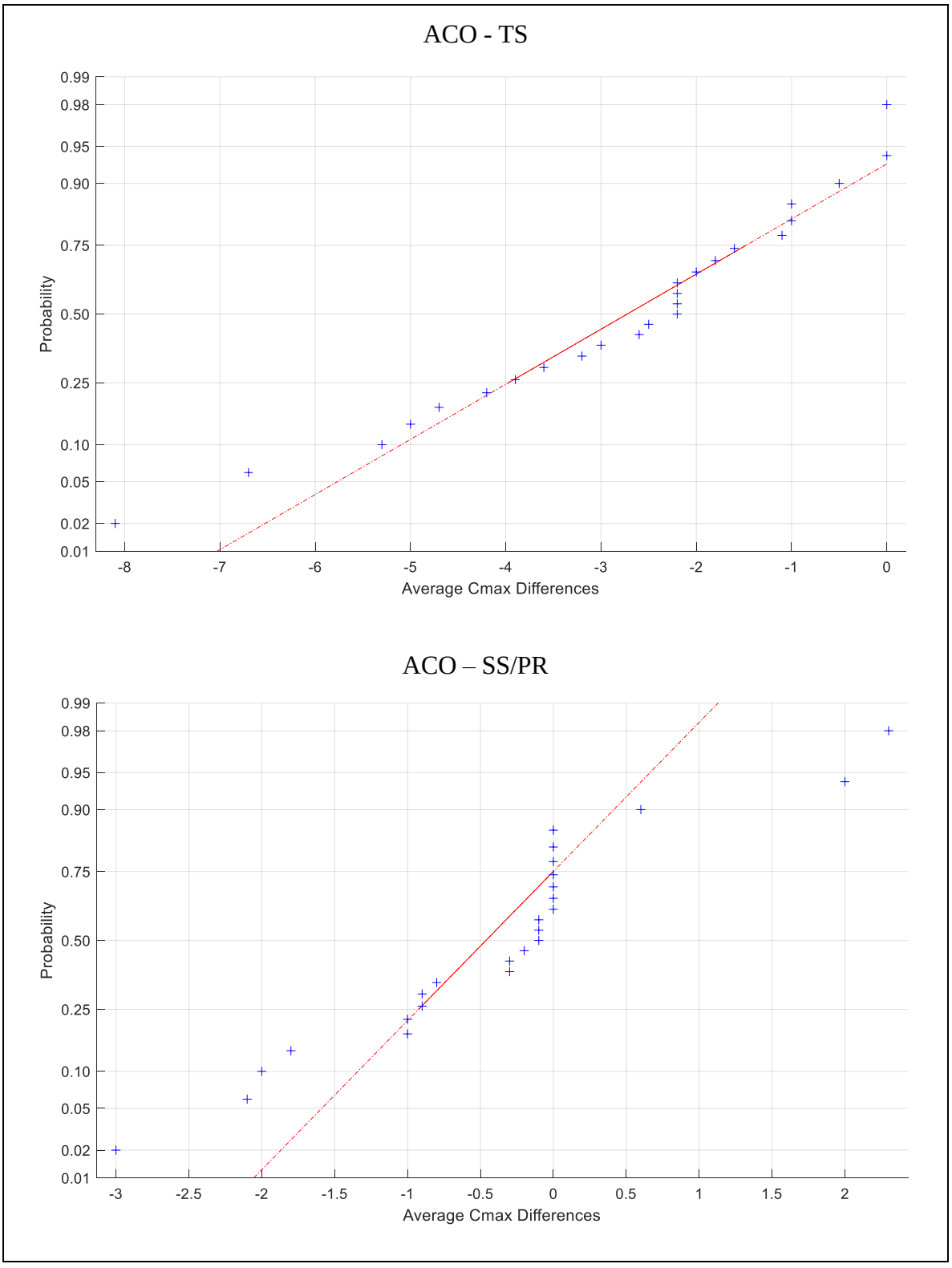

Figure 5.4. Normal probability plots of Avg. C_{max} differences for 16-stage instances

possible or reaches the previously reported provably optimal solution or the best result (may already be an optimum).

For the large-scale 16-stage instances, ACO produces significantly better average C_{max} results than SS/PR at 5% significance level.

Table 5.16. p-values of the samples of differences for Wilcoxon Signed Rank test

Sample	p-value
4-stage ACO – TS	8.77×10^{-5}
4-stage ACO – SS/PR	0.1250
8-stage ACO – TS	5.92×10^{-5}
8-stage ACO – SS/PR	0.095703
16-stage ACO – TS	2.67×10^{-5}
16-stage ACO – SS/PR	0.040481

5.6. Optimality of Results for 2-Stage Instances

Provably optimal solutions are reached in 16 of the 25 2-stage instances. It is claimed in this study that the Best C_{max} values reported for the remaining 9 instances are also the optimal solutions. The reasoning behind this claim is explained in this subsection. It is a rational explanation based on making the minimum possible deviation from the LB, with no technical proofs provided. Also, the other three algorithms in the literature, GA, TS, SS/PR, reported the same solutions for 2-stage instances, which supports the optimality claim here.

Statement is presented on a sample instance, S2-P16, while the same reasoning holds for all remaining 2-stage instances. Shop parameters for the problem is given in Table 5.17.

Table 5.17. Shop parameters for sample instance

S2-P16		
$n = 34$		
	m_i	p_i
Stage 1	19	13
Stage 2	15	10

There are 34 jobs to be processed in each of two stages. There are 19 machines in the first stage. Since the number of jobs is greater than the number of machines in the stage, at least two time-blocks are required to complete processing of all jobs in stage 1. A block is defined as a p_i-length time period in stage i. This creates a LB of 26 time units for stage 1 to complete processing of all jobs. Similarly, 3 blocks are required in stage 2, where there are 15 machines to process 34 jobs. Thus, LB for stage 2 is 30, leading to an overall LB of 30 time units for the shop. Schematic representation of the blocks and stage LBs are shown in Figure 5.5. The blocks for stage 1 are B1-1 and B1-2, and for stage 2 are B2-1, B2-2 and B2-3.

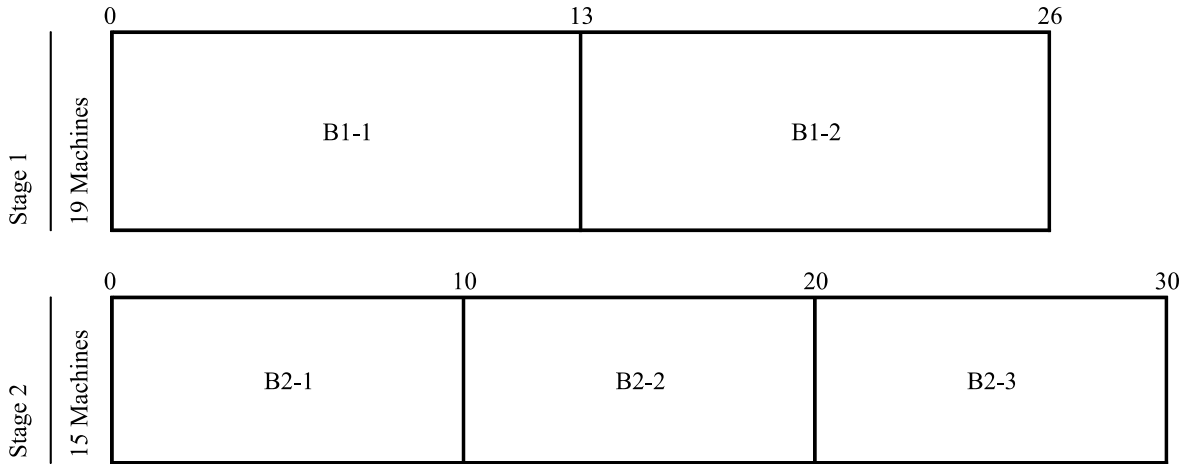

Figure 5.5. Representation of blocks and stage LBs for S2-P16

For a schedule to have an optimal makespan, it should either equal to problem LB for the makespan or, if not possible, should have a minimum deviation from the LB. Considering problem S2-P16, 34 jobs should be processed in one of the three blocks in stage 2 for the problem to have a makespan equal to LB 30. This requires processing at least 4 jobs in block B2-2. However, since a job can be processed on a single machine at a time, these 4 jobs cannot be processed in neither of the blocks in stage 1 once assigned to block B2-2 in stage 2. Thus, one of the two blocks in stage 1 should be shifted to prevent B2-2 from intersecting with both two blocks of stage 1. There is 4 units of slack time in stage 1 that allow shifting the blocks by at most 4 time units and still preserving the problem LB of 30. However, to remove the intersection of B2-2 with both of stage 1 blocks, at least 7 units shift to right in B1-2 is required. This creates a 3 time unit increase in the makespan. Alternatively, instead of B1-2,

B2-2 can be shifted to right by 3 time units. In either case, the resulting makespan is 33, which is the optimum since an increase of less than 3 units above the LB is not possible. The optimal makespan value of 33 is consistent with the algorithm results. Two optimal placements of time blocks are shown in Figure 5.6 and Figure 5.7.

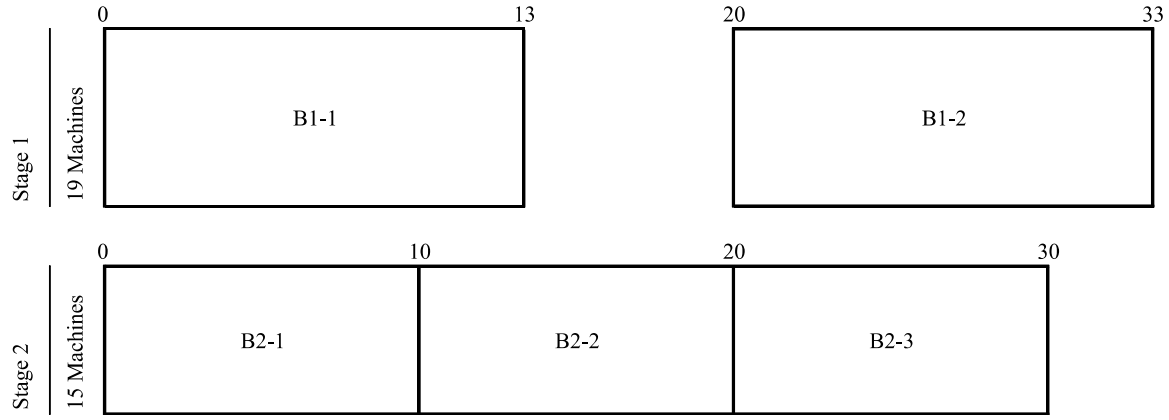

Figure 5.6. Placement of time blocks in the optimal schedule for S2-P16

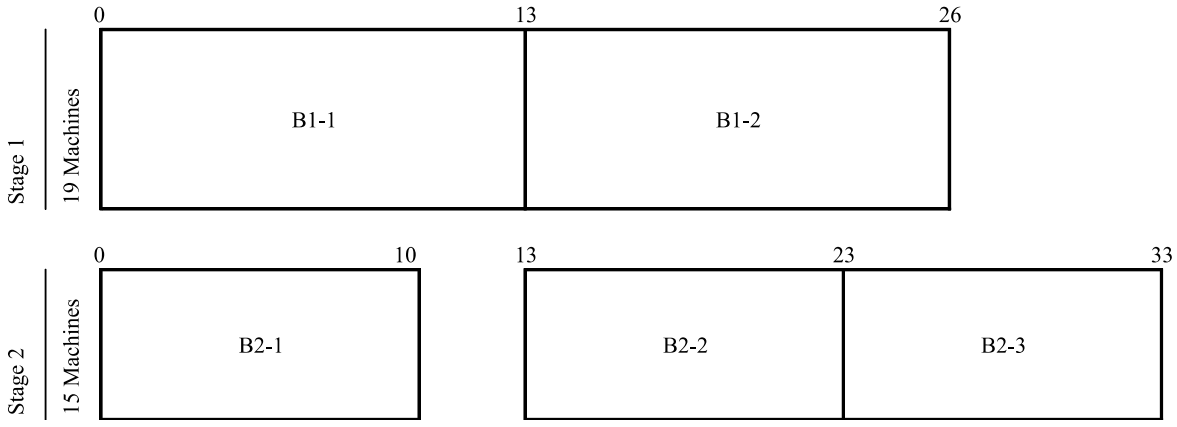

Figure 5.7. Alternative placement of time blocks in the optimal schedule for S2-P16

6. DISCUSSION

The proposed ACO algorithm showed a remarkable performance in reaching high solution quality within favorably short computational times. This performance is explained by 5 four factors in the algorithm. First, the ACO algorithm is based on a method of solution representation that encodes critical solution knowledge very efficiently. Second, it incorporates a random exploration routine which simplifies and speeds up the search in the solution space, contributing to low execution time. Third, during solution construction it exploits the accumulated search knowledge by selecting only the solution component that maximizes the quality function. Fourth, the algorithm also exploits the problem knowledge by applying Most Work Remaining Heuristic as the heuristic information. Lastly, the proposed local exploration routine allows both for searching different schedules around a permutation and for a distributed view of jobs in the final schedule to decrease the makespan.

The novel solution representation efficiently encodes schedule-specific knowledge and provides significant help in memory-based algorithms. The representation is mainly based on stage selection and job identities are not referred to. Thus, it is also adoptable to problem types other than MPOS where job identity is not a determinant of the value of the objective function. This is mostly the case in proportionate environments. Proportionate versions of the flow shop, job shop and open shop are three possible shop environments that the novel solution representation can be used. However, while making a job assignment to a stage, eligible job set should be formed by taking into account the machine route in flow shop and the job route in job shop.

Having the proportionate property in a MPOS has advantageous effects on scheduling the shop to minimize the makespan. It decreases the problem size as the number of inputs for processing times reduces significantly. Also, it enables multiple schedules with optimum makespan value. However, multiple schedules also exist for every makespan value. This may cause an algorithm to spend numerous iterations generating different schedules, with no improvement in makespan though. Thus, it also poses a challenge for an algorithm.

Deviation from LB is used as a performance evaluation point in assessing results of the tests carried out. It should be noted that these deviations underestimate the true performance of the algorithm if the optimal makespan is far from the LB. This can be exemplified by the apparent 4% average deviation in 2-stage instances, where, however, the results are shown to be optimal and the actual mean deviation is zero. Thus, the actual deviations for 4, 8 and 16-stage problem sets are very likely to be lower than what is reported.

The high performance of the algorithm in large problem size is particularly important since various solution approaches proposed in the literature showed poor performance in large-sized MPOS problems (Goldansaz et al., 2013; Matta, 2009). ACO managed to reach significantly higher solution quality in large-scale instances than the state-of-the-art SS/PR algorithm for the current testbed. It accomplished this in very favorable computational times, however SS/PR required considerable improvement in terms of computational time in large problem size.

One element that affects the performance of proposed ACO algorithm is the problem structure. Workload balance between stages is part of the structure and as the balance gets close to perfect equality the problem gets increasingly harder. A problem with many perfectly balanced stages would have little room for change in the optimal schedule. In this type of problems, the solution landscape has very narrow valleys for the optimal solution, while there are wide valleys of suboptimal solutions in the landscape. The shape of the landscape makes it hard for the algorithm to reach the narrow optimal region and easily get stuck in large suboptimal areas. This behavior explains why the algorithm manages to find a provably optimal solution for S16-P8 within 3 seconds while it finds a solution for S16-P1 with 4.4% deviation in 59 seconds.

The procedure described to explain the optimality of results for 2-stage instances is not limited to current problem set and it can be applied for every 2-stage proportionate MPOS problem to find an optimal solution. It is an easy and straightforward method based on constructing an optimal placement of time blocks in stages where the problem LB is preserved as the makespan or the minimum possible expansion is chosen.

7. CONCLUSION

Proportionate multiprocessor open shop is considered in this study. It is a shop environment appears widely in medical testing facilities, emergency units of hospitals, auto repair shops and inspection and quality control operations. Despite it is common appearance in service and production sectors, scheduling of this shop model has gained little attention in the literature.

This study proposes an Ant Colony Optimization algorithm to find a schedule for the shop to minimize the makespan. The algorithm is based on a novel very efficient way of solution representation that is also adoptable to proportionate flow shop, job shop and open shop problems. It is a permutation representation with ease of use and straightforward encoding and decoding procedures, making it useful for many heuristic and metaheuristic applications. The proposed ACO algorithm uses complete random solutions to search the solution space as part of its solution construction phase. The random search, which is untypical of an ACO algorithm, is allowed by moderate-good solution quality the representation supplies and is enabled a speed up in the algorithm. Solution construction also includes exploit of problem knowledge by implementing Most Work Remaining Heuristic and of accumulated search knowledge by selecting only the component that maximizes the quality function. As a supporting subroutine in the algorithm, a procedure named local exploration is proposed which enables assessment of different schedules around a single permutation. The routine also provides a distributed view of job placements in the final schedule which is shown to increase solution quality in instances with many perfectly balanced stages. Local exploration proposal in the algorithm is shown to be useful particularly in large-size instances.

The proposed ACO algorithm is tested on 100 benchmark instances -ranging from small to large in size- from the literature and the performance of the algorithm is compared with results of current state-of-the-art scatter search and path relinking algorithm. ACO is shown to outperform the SS/PR algorithm in terms of both solution quality and computational time. Its performance in particularly large-size instances is remarkable with significantly decreased makespan values reached in much less runtime. The relatively low difference in solution quality between ACO and SS/PR in 4 and 8-stage problem size is due to little room left for improvement in these instances. Still ACO manages to find reduced objective function values

where possible. The algorithm's performance in finding provably optimal solutions is also better than SS/PR.

ACO produced solutions with 2.51% average deviation from the lower bound in 100 instances, while the ratio was reported as 2.90% for SS/PR. Out of 100 instances, 67 provably optimal solutions were reached with ACO, while it was 57 for SS/PR. ACO required an average of 8.67 seconds computational time, which is 87% lower than the 69.24 seconds computational time in SS/PR. It should be noted that ACO managed such a significant decrease in computational time with a technically less capable computer used in this study.

Overall, the proposed ACO algorithm proves to be an efficient near-optimal solver for the proportionate multiprocessor open shop. There are several lines of research for future studies. First, the proposed solution representation can be adopted for other possible shop environments. Even can be used to improve the results of the previous research on present shop model. Its implementation in different search algorithms should also be investigated. Second, for the present benchmark testbed, optimal solutions for small and medium sized instances should be produced even if it takes extended periods of time for a MIP solver. This would enable a more correct evaluation of the algorithm performance. Third, the performance of the proposed ACO should be further assessed on different problem sets. Fourth, the algorithm should be revised for the solution of the more general multiprocessor open shop where the proportionate property is not assumed. Finally, the procedure described in this study for the optimality of 2-stage results should be re-stated as a formal polynomial-time optimum solution algorithm for 2-stage proportionate multiprocessor open shop.

REFERENCES

Abdelmaguid, T. F. (2014). A hybrid PSO-TS approach for proportionate multiprocessor open shop scheduling. 2014 IEEE International Conference on Industrial Engineering and Engineering Management, 107-111, 9-12 Dec., Bandar Sunway, Malaysia.

Abdelmaguid, T. F. (2020). Scatter search with path relinking for multiprocessor open shop scheduling. *Computers & Industrial Engineering,* 141, 106292.

Abdelmaguid, T. F., Shalaby, M. A., & Awwad, M. A. (2014). A tabu search approach for proportionate multiprocessor open shop scheduling. *Computational Optimization and Applications,* 58(1), 187-203.

Adak, Z., Arıoğlu Akan, M. Ö., & Bulkan, S. (2020). Multiprocessor open shop problem: literature review and future directions. *Journal of Combinatorial Optimization,* 40(2), 547-569.

Azadeh, A., Hosseinabadi Farahani, M., Torabzadeh, S., & Baghersad, M. (2014). Scheduling prioritized patients in emergency department laboratories. *Computer Methods and Programs in Biomedicine,* 117(2), 61-70.

Bai, D., Zhang, Z.-H., & Zhang, Q. (2016). Flexible open shop scheduling problem to minimize makespan. *Computers & Operations Research,* 67, 207-215.

Blum, C., & Dorigo, M. (2004). The hyper-cube framework for ant colony optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics),* 34(2), 1161-1172.

Chen, B., & Strusevich, V. A. (1993). Worst-case analysis of heuristics for open shops with parallel machines. *European Journal of Operational Research,* 70(3), 379-390.

den Besten, M., Stützle, T., & Dorigo, M. (2000). Ant Colony Optimization for the Total Weighted Tardiness Problem. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, & H.-P. Schwefel (eds), Parallel Problem Solving from Nature PPSN VI. PPSN 2000. Lecture Notes in Computer Science, Vol. 1917, 611-620, Springer, Berlin, Heidelberg.

Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms*. PhD Thesis, Politecnico di Milano, Milan.

Dorigo, M., & Caro, G. D. (1999). Ant colony optimization: a new meta-heuristic. Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), 1470-1477, 6-9 July, Washington, DC, USA.

Dorigo, M., Caro, G. D., & Gambardella, L. M. (1999). Ant Algorithms for Discrete Optimization. *Artificial Life,* 5(2), 137-172.

Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation,* 1(1), 53-66.

Dorigo, M., Maniezzo, V., & Colorni, A. (1991). *The Ant System: An autocatalytic optimizing process*. Technical report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics),* 26(1), 29-41.

Dorigo, M., & Stützle, T. (2004). *Ant Colony Optimization*. MIT Press, Cambridge.

Dorigo, M., & Stützle, T. (2019). Ant Colony Optimization: Overview and Recent Advances. In M. Gendreau & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (pp. 311-351). Springer International Publishing.

Goldansaz, S. M., Jolai, F., & Zahedi Anaraki, A. H. (2013). A hybrid imperialist competitive algorithm for minimizing makespan in a multi-processor open shop. *Applied Mathematical Modelling,* 37(23), 9603-9616.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. H. G. R. (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. In P. L. Hammer, E. L. Johnson, & B. H. Korte (Eds.), *Annals of Discrete Mathematics* (Vol. 5, pp. 287-326). Elsevier.

Hurink, J., Jurisch, B., & Thole, M. (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations-Research-Spektrum,* 15(4), 205-215.

Jansen, K., & Sviridenko, M. I. (2000). Polynomial Time Approximation Schemes for the Multiprocessor Open and Flow Shop Scheduling Problem. In H. Reichel & S. Tison (eds), STACS 2000. Lecture Notes in Computer Science, Vol. 1770, 455-465, Springer, Berlin, Heidelberg.

Józefowska, J., & Weglarz, J. (2006). *Perspectives in modern project scheduling* (Vol. 92). Springer US.

Kononov, A., & Sviridenko, M. (2002). A linear time approximation scheme for makespan minimization in an open shop with release dates. *Operations Research Letters*, 30(4), 276-280.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., & Shmoys, D. B. (1985). *The Traveling Salesman Problem.* John Wiley & Sons, Chichester, UK.

Lawler, E. L., Luby, M. G., & Vazirani, V. V. (1982). Scheduling open shops with parallel machines. *Operations Research Letters*, 1(4), 161-164.

Liaw, C.-F. (2000). A hybrid genetic algorithm for the open shop scheduling problem. *European Journal of Operational Research*, 124(1), 28-42.

Mao, W. (1995). Multi-operation multi-machine scheduling. In B. Hertzberger & G. Serazzi (eds), High-Performance Computing and Networking. HPCN-Europe 1995. Lecture Notes in Computer Science, Vol. 919, 33-38, Springer, Berlin, Heidelberg.

Matta, M. E. (2009). A genetic algorithm for the proportionate multiprocessor open shop. *Computers & Operations Research*, 36(9), 2601-2618.

Matta, M. E., & Elmaghraby, S. E. (2010). Polynomial time algorithms for two special classes of the proportionate multiprocessor open shop. *European Journal of Operational Research*, 201(3), 720-728.

Merkle, D., & Middendorf, M. (2000). An Ant Algorithm with a New Pheromone Evaluation Rule for Total Tardiness Problems. In S. Cagnoni (eds), Real-World Applications of Evolutionary Computing. EvoWorkshops 2000. Lecture Notes in Computer Science, Vol. 1803, 290-299, Springer, Berlin, Heidelberg.

Merkle, D., & Middendorf, M. (2002). Ant Colony Optimization with the Relative Pheromone Evaluation Method. In S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, & G. R. Raidl (eds), Applications of Evolutionary Computing. EvoWorkshops 2002. Lecture Notes in Computer Science, Vol. 2279, 325-333, Springer, Berlin, Heidelberg.

Merkle, D., Middendorf, M., & Schmeck, H. (2000). Pheromone evaluation in Ant Colony Optimization. 2000 26th Annual Conference of the IEEE Industrial Electronics Society. IECON 2000. 2000 IEEE International Conference on Industrial Electronics, Control and Instrumentation. 21st Century Technologies, Vol. 4, 2726-2731, Nagoya, Japan.

Naderi, B., Fatemi Ghomi, S. M. T., Aminnayeri, M., & Zandieh, M. (2011). Scheduling open shops with parallel machines to minimize total completion time. *Journal of Computational and Applied Mathematics,* 235(5), 1275-1287.

Queyranne, M., & Sviridenko, M. (2002). Approximation algorithms for shop scheduling problems with minsum objective. *Journal of Scheduling,* 5(4), 287-305.

Schuurman, P., & Woeginger, G. J. (1999). Approximation algorithms for the multiprocessor open shop scheduling problem. *Operations Research Letters,* 24(4), 157-163.

Sevastianov, S. V., & Woeginger, G. J. (2001). Linear time approximation scheme for the multiprocessor open shop problem. *Discrete Applied Mathematics,* 114(1), 273-288.

Stützle, T., & Hoos, H. H. (1997). MAX-MIN Ant System and local search for the traveling salesman problem. Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97), 309-314, Indianapolis, IN, USA.

Stützle, T., & Hoos, H. H. (2000). MAX–MIN Ant System. *Future Generation Computer Systems,* 16(8), 889-914.

Wang, Y. T., & Chou, F. D. (2017). A Bi-criterion Simulated Annealing Method to Solve Four-Stage Multiprocessor Open Shops with Dynamic Job Release Time. 2017 International Conference on Computing Intelligence and Information System (CIIS), 13-17, Nanjing.

Zhang, J., Wang, L., & Xing, L. (2019). Large-scale medical examination scheduling technology based on intelligent optimization. *Journal of Combinatorial Optimization*, 37(1), 385-404.

APPENDIX

APPENDIX A. Test Problems

This appendix presents test problems used in this study. The testbed is a benchmark testbed and was constructed by Matta (2009). It includes 100 problems: 25 for each of 2, 4, 8, and 16 stage cases. A problem is defined by four features: 1) Number of stages, 2) Number of jobs, 3) Number of machines in each stage, and 4) Processing time of each stage.

APPENDIX A-Table 1. 2-stage problems

Problem ID	Number of stages	Number of jobs	Number of machines		Processing time
S2-P1	2	22	Stage 1	15	15
			Stage 2	7	7
S2-P2	2	32	Stage 1	20	5
			Stage 2	12	3
S2-P3	2	24	Stage 1	18	9
			Stage 2	6	3
S2-P4	2	20	Stage 1	12	8
			Stage 2	8	5
S2-P5	2	29	Stage 1	17	13
			Stage 2	12	9
S2-P6	2	30	Stage 1	20	6
			Stage 2	10	3
S2-P7	2	28	Stage 1	16	12
			Stage 2	12	9
S2-P8	2	10	Stage 1	5	8
			Stage 2	5	8
S2-P9	2	30	Stage 1	15	11
			Stage 2	15	11
S2-P10	2	16	Stage 1	13	14
			Stage 2	3	3
S2-P11	2	40	Stage 1	20	7
			Stage 2	20	7
S2-P12	2	34	Stage 1	19	15
			Stage 2	15	12

APPENDIX A-Table 1. 2-stage problems (continued)

Problem ID	Number of stages	Number of jobs	Number of machines		Processing time
S2-P13	2	12	Stage 1	8	13
			Stage 2	4	6
S2-P14	2	14	Stage 1	10	12
			Stage 2	4	4
S2-P15	2	32	Stage 1	20	15
			Stage 2	12	9
S2-P16	2	34	Stage 1	19	13
			Stage 2	15	10
S2-P17	2	15	Stage 1	12	9
			Stage 2	3	2
S2-P18	2	13	Stage 1	10	11
			Stage 2	3	3
S2-P19	2	16	Stage 1	13	13
			Stage 2	3	3
S2-P20	2	12	Stage 1	8	6
			Stage 2	4	3
S2-P21	2	22	Stage 1	13	14
			Stage 2	9	9
S2-P22	2	25	Stage 1	18	11
			Stage 2	7	4
S2-P23	2	22	Stage 1	13	9
			Stage 2	9	6
S2-P24	2	12	Stage 1	8	9
			Stage 2	4	4
S2-P25	2	21	Stage 1	15	5
			Stage 2	6	2

APPENDIX A-Table 2. 4-stage problems

Problem ID	Number of stages	Number of jobs	Number of machines		Processing time
S4-P1	4	49	Stage 1	25	13
			Stage 2	10	5
			Stage 3	10	5
			Stage 4	4	2
S4-P2	4	39	Stage 1	19	7
			Stage 2	11	4
			Stage 3	6	2
			Stage 4	3	1
S4-P3	4	63	Stage 1	22	15
			Stage 2	18	12
			Stage 3	14	9
			Stage 4	9	6
S4-P4	4	38	Stage 1	14	9
			Stage 2	14	9
			Stage 3	5	3
			Stage 4	5	3
S4-P5	4	56	Stage 1	17	8
			Stage 2	13	6
			Stage 3	13	6
			Stage 4	13	6
S4-P6	4	60	Stage 1	20	7
			Stage 2	20	7
			Stage 3	14	5
			Stage 4	6	2
S4-P7	4	53	Stage 1	17	9
			Stage 2	17	9
			Stage 3	11	6
			Stage 4	8	4
S4-P8	4	40	Stage 1	19	11
			Stage 2	10	6
			Stage 3	7	4
			Stage 4	4	2
S4-P9	4	65	Stage 1	24	13
			Stage 2	24	13
			Stage 3	13	7
			Stage 4	4	2

APPENDIX A-Table 2. 4-stage problems (continued)

Problem ID	Number of stages	Number of jobs	Number of machines		Processing time
S4-P10	4	53	Stage 1	17	14
			Stage 2	12	10
			Stage 3	12	10
			Stage 4	12	10
S4-P11	4	55	Stage 1	22	13
			Stage 2	17	10
			Stage 3	12	7
			Stage 4	4	2
S4-P12	4	58	Stage 1	22	10
			Stage 2	22	10
			Stage 3	9	4
			Stage 4	5	2
S4-P13	4	37	Stage 1	14	12
			Stage 2	9	8
			Stage 3	9	8
			Stage 4	5	4
S4-P14	4	42	Stage 1	17	15
			Stage 2	13	11
			Stage 3	8	7
			Stage 4	4	3
S4-P15	4	28	Stage 1	12	12
			Stage 2	8	8
			Stage 3	4	4
			Stage 4	4	4
S4-P16	4	56	Stage 1	21	10
			Stage 2	17	8
			Stage 3	13	6
			Stage 4	5	2
S4-P17	4	90	Stage 1	24	8
			Stage 2	24	8
			Stage 3	24	8
			Stage 4	18	6
S4-P18	4	30	Stage 1	15	12
			Stage 2	5	4
			Stage 3	5	4
			Stage 4	5	4

APPENDIX A-Table 2. 4-stage problems (continued)

Problem ID	Number of stages	Number of jobs	Number of machines		Processing time
S4-P19	4	63	Stage 1	20	9
			Stage 2	20	9
			Stage 3	16	7
			Stage 4	7	3
S4-P20	4	62	Stage 1	18	15
			Stage 2	18	15
			Stage 3	13	11
			Stage 4	13	11
S4-P21	4	64	Stage 1	18	8
			Stage 2	18	8
			Stage 3	14	6
			Stage 4	14	6
S4-P22	4	58	Stage 1	20	10
			Stage 2	14	7
			Stage 3	14	7
			Stage 4	10	5
S4-P23	4	61	Stage 1	23	7
			Stage 2	17	5
			Stage 3	17	5
			Stage 4	4	1
S4-P24	4	54	Stage 1	17	11
			Stage 2	17	11
			Stage 3	12	8
			Stage 4	8	5
S4-P25	4	34	Stage 1	14	7
			Stage 2	8	4
			Stage 3	8	4
			Stage 4	4	2

APPENDIX A-Table 3. 8-stage problems

Problem ID	Number of stages	Number of jobs	Number of machines		Processing time
S8-P1	8	146	Stage 1	25	8
			Stage 2	25	8
			Stage 3	25	8
			Stage 4	25	8
			Stage 5	13	4
			Stage 6	13	4
			Stage 7	10	3
			Stage 8	10	3
S8-P2	8	144	Stage 1	24	5
			Stage 2	24	5
			Stage 3	24	5
			Stage 4	19	4
			Stage 5	19	4
			Stage 6	19	4
			Stage 7	10	2
			Stage 8	5	1
S8-P3	8	87	Stage 1	18	6
			Stage 2	18	6
			Stage 3	12	4
			Stage 4	12	4
			Stage 5	12	4
			Stage 6	9	3
			Stage 7	3	1
			Stage 8	3	1
S8-P4	8	161	Stage 1	24	15
			Stage 2	24	15
			Stage 3	24	15
			Stage 4	24	15
			Stage 5	20	12
			Stage 6	20	12
			Stage 7	15	9
			Stage 8	10	6
S8-P5	8	117	Stage 1	23	13
			Stage 2	23	13
			Stage 3	18	10
			Stage 4	18	10
			Stage 5	13	7
			Stage 6	9	5
			Stage 7	9	5
			Stage 8	4	2

APPENDIX A-Table 3. 8-stage problems (continued)

Problem ID	Number of stages	Number of jobs	Number of machines		Processing time
S8-P6	8	99	Stage 1	22	12
			Stage 2	16	9
			Stage 3	16	9
			Stage 4	16	9
			Stage 5	13	7
			Stage 6	8	4
			Stage 7	4	2
			Stage 8	4	2
S8-P7	8	84	Stage 1	25	9
			Stage 2	19	7
			Stage 3	19	7
			Stage 4	9	3
			Stage 5	3	1
			Stage 6	3	1
			Stage 7	3	1
			Stage 8	3	1
S8-P8	8	110	Stage 1	23	11
			Stage 2	23	11
			Stage 3	23	11
			Stage 4	13	6
			Stage 5	9	4
			Stage 6	9	4
			Stage 7	5	2
			Stage 8	5	2
S8-P9	8	128	Stage 1	24	7
			Stage 2	24	7
			Stage 3	24	7
			Stage 4	17	5
			Stage 5	14	4
			Stage 6	14	4
			Stage 7	7	2
			Stage 8	4	1
S8-P10	8	90	Stage 1	18	6
			Stage 2	18	6
			Stage 3	12	4
			Stage 4	12	4
			Stage 5	9	3
			Stage 6	9	3
			Stage 7	9	3
			Stage 8	3	1

APPENDIX A-Table 3. 8-stage problems (continued)

Problem ID	Number of stages	Number of jobs	Number of machines		Processing time
S8-P11	8	102	Stage 1	25	9
			Stage 2	19	7
			Stage 3	19	7
			Stage 4	9	3
			Stage 5	9	3
			Stage 6	9	3
			Stage 7	9	3
			Stage 8	3	1
S8-P12	8	92	Stage 1	17	10
			Stage 2	12	7
			Stage 3	12	7
			Stage 4	12	7
			Stage 5	12	7
			Stage 6	9	5
			Stage 7	9	5
			Stage 8	9	5
S8-P13	8	101	Stage 1	22	7
			Stage 2	22	7
			Stage 3	16	5
			Stage 4	13	4
			Stage 5	13	4
			Stage 6	7	2
			Stage 7	4	1
			Stage 8	4	1
S8-P14	8	72	Stage 1	14	14
			Stage 2	14	14
			Stage 3	14	14
			Stage 4	9	9
			Stage 5	9	9
			Stage 6	4	4
			Stage 7	4	4
			Stage 8	4	4
S8-P15	8	100	Stage 1	23	12
			Stage 2	23	12
			Stage 3	17	9
			Stage 4	17	9
			Stage 5	8	4
			Stage 6	4	2
			Stage 7	4	2
			Stage 8	4	2

APPENDIX A-Table 3. 8-stage problems (continued)

Problem ID	Number of stages	Number of jobs	Number of machines		Processing time
S8-P16	8	81	Stage 1	17	14
			Stage 2	17	14
			Stage 3	12	10
			Stage 4	9	7
			Stage 5	9	7
			Stage 6	9	7
			Stage 7	4	3
			Stage 8	4	3
S8-P17	8	100	Stage 1	24	12
			Stage 2	14	7
			Stage 3	14	7
			Stage 4	14	7
			Stage 5	14	7
			Stage 6	8	4
			Stage 7	8	4
			Stage 8	4	2
S8-P18	8	106	Stage 1	22	10
			Stage 2	22	10
			Stage 3	17	8
			Stage 4	13	6
			Stage 5	13	6
			Stage 6	9	4
			Stage 7	5	2
			Stage 8	5	2
S8-P19	8	108	Stage 1	24	15
			Stage 2	19	12
			Stage 3	15	9
			Stage 4	15	9
			Stage 5	10	6
			Stage 6	10	6
			Stage 7	10	6
			Stage 8	5	3
S8-P20	8	105	Stage 1	22	9
			Stage 2	17	7
			Stage 3	17	7
			Stage 4	17	7
			Stage 5	8	3
			Stage 6	8	3
			Stage 7	8	3
			Stage 8	8	3

APPENDIX A-Table 3. 8-stage problems (continued)

Problem ID	Number of stages	Number of jobs	Number of machines		Processing time
S8-P21	8	152	Stage 1	25	6
			Stage 2	25	6
			Stage 3	25	6
			Stage 4	25	6
			Stage 5	17	4
			Stage 6	13	3
			Stage 7	13	3
			Stage 8	9	2
S8-P22	8	104	Stage 1	25	6
			Stage 2	17	4
			Stage 3	13	3
			Stage 4	13	3
			Stage 5	13	3
			Stage 6	9	2
			Stage 7	9	2
			Stage 8	5	1
S8-P23	8	97	Stage 1	24	15
			Stage 2	19	12
			Stage 3	19	12
			Stage 4	10	6
			Stage 5	10	6
			Stage 6	5	3
			Stage 7	5	3
			Stage 8	5	3
S8-P24	8	104	Stage 1	22	7
			Stage 2	16	5
			Stage 3	16	5
			Stage 4	16	5
			Stage 5	13	4
			Stage 6	13	4
			Stage 7	4	1
			Stage 8	4	1
S8-P25	8	101	Stage 1	22	7
			Stage 2	22	7
			Stage 3	16	5
			Stage 4	13	4
			Stage 5	13	4
			Stage 6	7	2
			Stage 7	4	1
			Stage 8	4	1

APPENDIX A-Table 4. 16-stage problems

Problem ID:	Number of stages: 16												Number of jobs: 88			
S16-P1	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	8	8	6	6	6	6	6	6	4	4	4	4	4	4	2
	Processing Time	15	12	12	9	9	9	9	9	9	6	6	6	6	6	6	3
Problem ID:	Number of stages: 16												Number of jobs: 102			
S16-P2	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	10	10	8	6	6	6	6	4	4	4	4	2	2
	Processing Time	9	9	9	9	9	7	5	5	5	5	3	3	3	3	1	1
Problem ID:	Number of stages: 16												Number of jobs: 99			
S16-P3	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	8	8	8	8	8	5	5	5	4	4	2	2	2
	Processing Time	13	13	13	10	10	10	10	10	7	7	7	5	5	2	2	2
Problem ID:	Number of stages: 16												Number of jobs: 90			
S16-P4	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	7	7	7	7	6	6	4	4	4	2	2	2	2
	Processing Time	11	11	11	8	8	8	8	6	6	4	4	4	2	2	2	2
Problem ID:	Number of stages: 16												Number of jobs: 96			
S16-P5	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	10	8	8	8	6	4	4	4	4	4	2	2	2
	Processing Time	10	10	10	10	8	8	8	6	4	4	4	4	4	2	2	2

APPENDIX A-Table 4. 16-stage problems (continued)

Problem ID:	Number of stages: 16										Number of jobs: 104					
S16-P6	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	10	8	8	6	6	6	6	6	4	4	4	4	2
	Processing Time	13	13	13	13	10	10	7	7	7	7	7	5	5	5	5	2
Problem ID:	Number of stages: 16										Number of jobs: 106					
S16-P7	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	10	8	8	8	8	8	6	4	4	4	4	2	2
	Processing Time	11	11	11	11	8	8	8	8	8	6	4	4	4	4	2	2
Problem ID:	Number of stages: 16										Number of jobs: 81					
S16-P8	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	7	7	6	6	6	3	3	3	2	2	2	2	2
	Processing Time	7	7	7	5	5	4	4	4	2	2	2	1	1	1	1	1
Problem ID:	Number of stages: 16										Number of jobs: 101					
S16-P9	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	10	10	10	7	6	6	4	4	4	4	2	2	2
	Processing Time	11	11	11	11	11	11	8	6	6	4	4	4	4	2	2	2
Problem ID:	Number of stages: 16										Number of jobs: 96					
S16-P10	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	8	8	8	8	8	6	6	6	4	4	4	2	2	2
	Processing Time	12	12	9	9	9	9	9	7	7	7	4	4	4	2	2	2

APPENDIX A-Table 4. 16-stage problems (continued)

Problem ID:	Number of stages: 16									Number of jobs: 93						
S16-P11	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	10	8	8	8	5	5	5	4	2	2	2	2	2
	Processing Time	8	8	8	8	6	6	6	4	4	4	3	1	1	1	1	1
Problem ID:	Number of stages: 16									Number of jobs: 110						
S16-P12	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	8	8	8	8	8	8	8	8	6	4	2	2	2
	Processing Time	14	14	14	11	11	11	11	11	11	11	11	8	5	2	2	2
Problem ID:	Number of stages: 16									Number of jobs: 112						
S16-P13	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	10	10	10	8	8	8	6	6	6	4	2	2	2
	Processing Time	15	15	15	15	15	15	12	12	12	9	9	9	6	3	3	3
Problem ID:	Number of stages: 16									Number of jobs: 97						
S16-P14	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	10	10	7	7	5	5	5	4	4	4	2	2	2
	Processing Time	8	8	8	8	8	6	6	4	4	4	3	3	3	1	1	1
Problem ID:	Number of stages: 16									Number of jobs: 86						
S16-P15	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	10	8	4	4	4	4	4	4	4	4	2	2	2
	Processing Time	14	14	14	14	11	5	5	5	5	5	5	5	5	2	2	2

APPENDIX A-Table 4. 16-stage problems (continued)

Problem ID:	Number of stages: 16										Number of jobs: 106					
S16-P16	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	10	8	8	8	8	8	6	6	6	2	2	2	2
	Processing Time	5	5	5	5	4	4	4	4	4	3	3	3	1	1	1	1
Problem ID:	Number of stages: 16										Number of jobs: 94					
S16-P17	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	10	10	6	6	6	6	6	3	3	2	2	2	2
	Processing Time	7	7	7	7	7	4	4	4	4	4	2	2	1	1	1	1
Problem ID:	Number of stages: 16										Number of jobs: 102					
S16-P18	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	10	10	8	8	6	6	6	6	4	2	2	2	2
	Processing Time	10	10	10	10	10	8	8	6	6	6	6	4	2	2	2	2
Problem ID:	Number of stages: 16										Number of jobs: 80					
S16-P19	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	8	6	6	6	4	4	4	2	2	2	2	2	2
	Processing Time	14	14	14	11	8	8	8	5	5	5	2	2	2	2	2	2
Problem ID:	Number of stages: 16										Number of jobs: 84					
S16-P20	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	8	8	8	8	6	6	6	2	2	2	2	2	2	2
	Processing Time	10	10	8	8	8	8	6	6	6	2	2	2	2	2	2	2

APPENDIX A-Table 4. 16-stage problems (continued)

Problem ID:	Number of stages: 16										Number of jobs: 78					
S16-P21	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	10	8	4	4	4	4	2	2	2	2	2	2	2
	Processing Time	11	11	11	11	8	4	4	4	4	2	2	2	2	2	2	2
Problem ID:	Number of stages: 16										Number of jobs: 79					
S16-P22	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	7	7	6	6	6	6	3	3	3	3	3	2	2	2
	Processing Time	7	7	5	5	4	4	4	4	2	2	2	2	2	1	1	1
Problem ID:	Number of stages: 16										Number of jobs: 97					
S16-P23	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	7	7	7	7	7	6	6	6	3	3	3	3	2
	Processing Time	7	7	7	5	5	5	5	5	4	4	4	2	2	2	2	1
Problem ID:	Number of stages: 16										Number of jobs: 93					
S16-P24	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	10	7	7	7	7	5	5	5	4	4	4	4	2	2
	Processing Time	6	6	6	4	4	4	4	3	3	3	2	2	2	2	1	1
Problem ID:	Number of stages: 16										Number of jobs: 96					
S16-P25	Stages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Number of Machines	10	10	8	8	8	8	8	6	6	6	4	4	4	2	2	2
	Processing Time	12	12	9	9	9	9	9	7	7	7	4	4	4	2	2	2

CURRICULUM VITAE

Education

PhD	Marmara University | Industrial Engineering | 2017-2020
Master's	Boğaziçi University | Computational Science and Engineering | 2008-2011
Bachelor's	Marmara University | Industrial Engineering | 2003-2007

Employment

Research Assistant	Gebze Technical University | 2017-
Systems Analyst	Marmara University | Information Technology Dept. | 2012-2014

Scientific research

Adak, Z., Arıoğlu Akan, M.Ö. & Bulkan, S. Multiprocessor open shop problem: literature review and future directions. *Journal of Combinatorial Optimization*, 40, 547–569, 2020.

Adak, Z. & Demiriz, A. Hybridization of population-based ant colony optimization via data mining. *Intelligent Data Analysis*, 24(2), 291-307, 2020.

Borekci, O.S. & Adak, Z. Yakın kıyı dalga hesaplamaları için ağsız sayısal modelleme. 7. Kıyı Mühendisliği Sempozyumu, November, 2011, Trabzon.