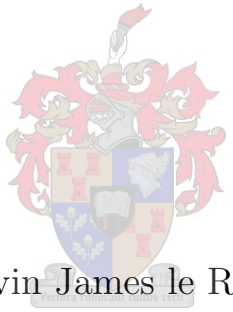
Metaheuristics for the selection and scheduling of jobs with time-dependent durations



Gavin James le Roux

Thesis presented in partial fulfilment of the requirements for the degree of
Master of Commerce (Operations Research)
in the Faculty of Economic and Management Sciences at Stellenbosch University

Supervisor: Prof SE Visagie

December 2017

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: December 2017

Abstract

A scheduling problem that determines an optimal sequence of jobs to be performed on a single machine is considered. A set of available jobs is presented to management of which a subset should be selected to maximise the total weighted profit. This is because of the schedule horizon that is limited to a maximum makespan, resulting in a pool of rejected jobs that may possibly be processed at a later stage. Each job is associated with a list of possible fractions that it requires of the machine during its processing. As a result, jobs can be processed simultaneously on the machine. Furthermore, once a job has completed its processing, there is a likelihood that the machine needs to be cleaned up and prepared for the following jobs to be processed. Setup times and setup costs are therefore associated with each ordered pair of jobs. Additionally, the processing time of a job depends on the time at which it starts processing. Lastly, certain jobs are prohibited from starting its processing directly after the processing of another given set of jobs at a specified set of time periods. The five characteristics of the considered scheduling problem are therefore batch scheduling, time-dependent processing times, sequence dependent setup times, job selection and precedence constraints. Applications of the problem may include the testing of aircraft and missile flights on a multipurpose test range, the scheduling of resources to underprivileged communities using multiple vehicles as well as the scheduling of jobs in a busy vehicle maintenance and repair facility.

A variety of metaheuristics were developed to solve the scheduling problem when jobs in a batch need to start at the same time. Three metaheuristics that use moves and/or neighbourhoods were developed, namely variable neighbourhood search (VNS), tabu search (TS) and a genetic algorithm (GA-II) in which a chromosome is defined as a sequence of batches. Moreover, three metaheuristics that utilise decoding schemes were implemented. These algorithms are particle swarm optimisation (PSO), cuckoo search (CS) and a genetic algorithm (GA-I) in which a chromosome is defined as a string of floating point numbers. The non-identical start time batch (NSTB) algorithm is executed after a set of solutions were obtained by one of the metaheuristics. This is an algorithm that was designed to address the problem that jobs in a batch are allowed to start at different times. The set of solutions in the NSTB algorithm is evolved into a population of schedules that is presented to management to decide which trade-offs should be made when selecting a schedule to implement. There are four different steps in each iteration, namely the moving of single jobs within a batch, the merging of batches, the moving of multiple jobs within a batch and the insertion of a job into a batch.

A total of 150 test cases were generated for the purpose of comparing the algorithms with one another, because real-world data are not available. Statistical hypothesis testing was performed using omnibus tests to test whether there is an overall significant difference between the profits of the algorithms for each test case. The Friedman test was used as a non-parametric statistical test with post hoc tests that included the Wilcoxon signed-rank test and the sign test. Furthermore, the Welch analysis of variance (ANOVA) test was used as a parametric alternative with post hoc

tests that included the Tamhane's T2, Dunnett's T3 and Games-Howell tests. Upon completion of these tests, it is recommended to use either VNS or TS for test cases in the elementary difficulty group and TS for test cases of an intermediate or challenging nature. There is a mere 0.3% increase in the average profit when VNS is used compared to TS for elementary test cases. For intermediate test cases, TS performs about 8.7% better than VNS as well as 9.8% and 12.4% better than GA-I and GA-II, respectively. For challenging test cases, TS most notably performs 16.6% and 16.2% better than GA-I and GA-II, respectively, and 19% better than PSO. CS is the worst performing metaheuristic in all difficulty groups.

Opsomming

In hierdie tesis word 'n skeduleringsprobleem beskou wat 'n optimale volgorde van take op 'n enkele masjien bepaal. 'n Groep van beskikbare take word gegee en die bestuur moet dan 'n subversameling van take selekteer om wins te maksimeer. Dit is nodig omdat die skedule oor 'n eindige tyshorison opgelos moet word en daar dus 'n poel van oorskottake ontstaan wat in 'n latere skedule ingesluit kan word. Elke taak het 'n versameling moontlike breuke van die masjien wat dit in beslag kan neem. Take kan dus gelyktydig (in parallel) op die masjien geprosesseer word. Wanneer 'n taak afgehandel is, is daar 'n moontlikheid dat die masjien voorberei moet word vir 'n volgende taak. Daar is dus 'n opstelkoste vir elke onderlinge paar van die take. Die uitvoertyd van 'n taak is afhanklik van sy begintyd. Laastens het sekere take ook beperkings in terme van wanneer hulle relatief tot ander take uitgevoer kan word. Die vyf eienskappe van die skeduleringsprobleem wat beskou word, is dus trosskedulering, tyd afhanklike opsteltye, volgorde afhanklike opsteltye, taakseleksie en voorgangerbeperkings. Toepassings van hierdie tipe skeduleringsprobleme is die skedulering van vliegtuig- en missieltoetse op 'n toetsbaan, die skedulering van hulpbronne aan behoeftige gemeenskappe met verskeie voertuie asook die skedulering van take by 'n besige werkswinkel wat voertuie versien.

'n Verskeidenheid van metaheuristieke is ontwikkel om die take te skeduleer in trosse waarin die take gelyktydig moet begin. Drie metaheuristieke wat skuiwe en/of omgewings gebruik is ontwikkel, naamlik 'n veranderbare-omgewing-soektog (VNS), 'n taboe-soektog (TS), en 'n genetiese algoritme (GA-II), waarin 'n chromosoom gedefinieer word as die volgorde van die trosse. Verder is drie metaheuristieke ontwikkel wat 'n dekoderingskema gebruik. Hierdie algoritmes is deeltjie-swerm-optimering (PSO), koekoe-soektog (CS) en 'n genetiese algoritme (GA-I) waarin 'n chromosoom gedefinieer word as 'n string reële getalle. Die nie-identiese-begintyd-tros-algoritme (NSTB) word gebruik nadat 'n versameling oplossing bereken is om take op verskillende tye in die trosse te laat begin. Die versameling oplossings wat die NSTB verskaf, word gebruik om 'n populasie van skedules te skep waaruit bestuur kan kies tussen die afruilings om 'n mees gewensde skedule te vind. NSTB gebruik vier stappe, naamlik die skuif van 'n enkele taak, samevoeging van trosse, die skuif van 'n groep take in 'n tros en die invoeging van 'n taak in 'n tros.

'n Totaal van 150 probleme is gegenereer (omdat daar nie regte wêreld data bestaan nie) om algoritmes met mekaar te vergelyk. Statistiese hipotesetoetsing is uitgevoer met behulp van omnibus toetse om te bepaal of daar 'n statisties beduidende verskil tussen die wins van die verskillende algoritmes is. Die Friedmantoets is gebruik as 'n nie-parametriese statistiese toets saam met post hoc toetse wat insluit die Wilcoxon-teken-rang-toets en die teken toets. Die Welch analise van variansie (ANOVA) toets is gebruik as 'n parametriese alternatief met die post hoc toetse wat insluit Tamhane's T2, Dunnett's T3 en Games-Howell-toetse. Die aanbeveling, na afloop van hierdie toetse, is om VNS en TS te gebruik vir die klein probleme en TS vir die middel en moeiliker probleemklasse. Daar is 'n geringe 0.3% toename in die gemiddelde wins indien

VNS gebruik word teenoor TS gedurende die maklike probleme. Vir die middelprobleme vaar TS ongeveer 8.7% beter as VNS sowel as 9.8% en 12.4% beter as GA-I en GA-II, onderskeidelik. Vir die moeilike probleme vaar TS die beste en doen 16.6% en 16.2% beter as GA-I en GA-II, onderskeidelik. TS vaar ook 19% beter as PSO. CS vaar die swakste oor al die probleemklasse.

Acknowledgements

Many people played a significant role in the work leading up to and during the writing of this thesis. The author hereby wishes to express his deepest gratitude towards:

- Prof SE Visagie, my supervisor, for his valuable insights, guidance and most importantly his loyal friendship during the compilation of this thesis.
- Ms L Venter and Dr L Potgieter for providing me the opportunity to be a part-time lecturer to third year students.
- Dr I Nieuwoudt for our interesting conversations when I needed a distraction from the stress involved in completing this project.
- My parents for their constant support, endless motivation and standing by my decisions during the six years of my academic career.
- My fellow Master's students Kurt Marais, Chesme Messina, Bryce Senekal, Flora Hofmann and Emma Gibson (whom I look up to) for all the good times in the lab.
- The Department of Logistics at Stellenbosch University for their computer facilities and Mrs J Thiart for always being available when a technical problem in the lab needed to be solved.
- The Harry Crossley Foundation for their generous financial support and Stellenbosch University for their financial assistance in the form of two Postgraduate Merit Bursaries.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

ACO	Ant colony optimisation
ACS	Ant colony system
ANOVA	Analysis of variance
API	adjacent pairwise interchange
ATC	Apparent tardiness cost
B&B	Branch-and-bound
B&C	Branch-and-cut
BFF	Batch first-fit
BHMOGA	Batch based hybrid multi-objective genetic algorithm
BTS	Binary tournament selection
BU	Batching update
CDS	Climbing discrepancy search
CP	Critical path
DM	Decision maker
DP	Dynamic programming
d-RFSB	Dynamic release first-sequence best
EDD	Earliest due date
ERD	Early release date
EST	Earliest start time
FPS	Fitness proportionate selection
FPTAS	Fully polynomial-time approximation scheme
GA	Genetic algorithm
GRASP	Greedy randomised adaptive search procedure
GSB	Given sequence batch procedure
HACOA	Hybrid ant colony optimisation algorithm
HEA	Hybrid evolutionary algorithm
HGA	Hybrid genetic algorithm
IMI	Inter-machine insertion
IP	Integer programming model
ISFAN	Iterative sequence first-accept next
JIT	Just in time
KS	Kolmogorov-Smirnov
LOX	Linear order crossover operator
MA	Memetic algorithm
m-ATCS	Modified apparent tardiness cost with setups
MI	Multiple insertion
MILP	Mixed integer linear programming
MMAS	Max-min ant system

MOEA	Multi-objective evolutionary algorithm
MOGA	Multi-objective genetic algorithm
MOOP	Multi-objective optimisation problem
MOVNS	Multi-objective variable neighbourhood search
MPGA	Multi-population genetic algorithm
MPS	Minimal part set
NAJ	Not all jobs
NAM	Not all machines
NSGA-II	Nondominated sorting genetic algorithm II
NSTB	Non-identical start time batch algorithm
OPOX	One-point order crossover operator
OPX	One-point crossover operator
PACS	Pareto-based ant colony system
PRHA	Priority rule-based heuristic algorithm
PSO	Particle swarm optimisation
PTS	Probabilistic tournament selection
PUOX	Parameterised uniform order crossover operator
PUX	Parameterised uniform crossover operator
RM	Rate modifying activity
RBP	Random batches procedure
RDU	Release date update
SA	Simulated annealing
SEA	Self-evolution algorithm
SMOGA	Sequence based multi-objective genetic algorithm
SPT	Shortest processing time first rule
SS	Serial schedule
SSPRT	Shortest sum of setup, processing and removal times
SSRT	Shortest sum of setup and removal times
SUS	Stochastic universal sampling
SW	Shapiro-Wilk
TPSPGA	Two-phase sub-population genetic algorithm
TS	Tabu search
TSP	Travelling salesman problem
VND	Variable neighbourhood descent
VNS	Variable neighbourhood search
WSPT	Weighted shortest processing time first rule

List of Reserved Symbols

Symbols in this thesis conform to the following font conventions:		
a	Symbol denoting a **variable**	(Roman lower case letters)
$\mathbf{a}$	Symbol denoting a **vector**	(Roman boldface lower case letters)
A	Symbol denoting a **matrix**	(Roman capitals)
$\mathcal{A}$	Symbol denoting a **set**	(Calligraphic capitals)

Symbol	Meaning
Indices	
i	An index spanning jobs in the exact formulations and an index of the solution in a set of solutions in the algorithms.
j	An index spanning jobs that another job follows in the exact formulations and an index spanning the objective functions in the algorithms.
k	An index spanning the time periods of a schedule.
l	An index spanning the available machine (for future work)
Sets	
$\mathcal{H}_i$	The set of derived jobs for job i.
$\bar{\mathcal{J}}_i$	The set of derived jobs in batch i for all the batches in the sequence.
$\bar{\mathcal{H}}_i$	The set of derived jobs for the unscheduled job i.
$\tilde{\mathcal{H}}_i$	The set of derived jobs for the scheduled job i.
$\bar{\mathcal{P}}$	The initial set of solutions for an algorithm.
$\mathcal{A}_l$	The local archive containing the set of nondominated solutions found during the algorithm. The solutions in this set influence future solutions.
$\mathcal{M}$	The set of random moves that can be made around a solution.
$\mathcal{N}$	The set of neighbourhoods around a solution.
$\mathcal{X}_i$	The set of solutions in the neighbourhood around solution Φ'_i.
$\mathcal{X}(t)$	The set of the particles' positions at time t.
$\mathcal{V}(t)$	The set of the particles' velocities at time t.
$\mathcal{P}(t)$	The set of the particles' best found positions at time t.
$\mathcal{P}$	The set of particles/nests/individuals in the population.
$\mathcal{S}$	The set of solutions that are generated for different $\mathbf{h}_i$ vectors.
$\mathcal{A}_e$	The external archive containing the set of nondominated solutions found during the algorithm. The solutions in this set do not influence future solutions as it is used to keep a historical record of all the nondominated solutions.
$\mathcal{P}_j$	The set of eggs in nest j or individuals in generation j.
$\mathcal{P}_o$	The offspring population of the current generation.
$\mathcal{P}_s$	The set of individuals in $\mathcal{P}_k \cup \mathcal{P}_o$ sorted in descending order of their fitness value.

$\mathcal{Q}$	The set of batches containing at least two jobs.
$\mathcal{T}_j$	The set of all the jobs that are being processed during period j.
$\mathcal{M}_i$	The set of consecutive batches that have space available in them and can be merged with the batch following them with i being the first of the consecutive batches.
$\bar{\mathcal{M}}$	The set of batches that are merged with the batch following them.
$\mathcal{R}$	The set of jobs not present in a schedule.
$\mathcal{T}$	The set of jobs in $\mathcal{R}$ used for PTS.
$\mathcal{T}_{ij}$	The tabu list for individual i and neighbourhood j in TS.
$\mathcal{T}_{ij}^{iter}$	The number of iterations that each of the elements are present in $\mathcal{T}_{ij}$.
$\tilde{\mathcal{N}}$	The set of neighbourhood structures in TS.
$\mathcal{C}$	The candidate list containing all the solutions whose moves are not in the tabu list.
$\mathcal{C}'$	The list containing all the solutions whose moves are in the tabu list.
Variables	
x_{ik}	Equal to 1 if the implementation of job i starts during period k.
b_{ik}	Equal to 1 if job i is implemented during period k.
y_{ik}	Binary variable for the if-then constraints when the processing times of the jobs depend on their start time only.
x_{ijk}	Equal to 1 if the implementation of job i follows on job j and starts during period k.
b_{ijk}	Equal to 1 if job i follows on job j and is implemented during period k.
q'_{ijk}	Binary variable that is set to 1 if the running of job i does not follow on job j and start during period k.
y_{ijk}	Binary variable for the if-then constraints when the processing times of the jobs depend on their start time and the preceding job.
q''_{jk}	Binary variable that is set to 1 if project j is succeeded by other projects when it starts during period k.
q'''_{jk}	Binary variable that is set to 1 if job j precedes other jobs when it starts during period k and if it can be completed before the end of the schedule.
z_{ik}	Binary variable for the if-then constraints when the processing times of the jobs depend on their start time only and jobs are inserted into batches.
z_{ijk}	Binary variable for the if-then constraints when the processing times of the jobs depend on their start time and the preceding job for when jobs are inserted into batches.
Φ_i	Solution i in the set of solutions for each iteration in an algorithm.
$f(\Phi_i)$	Value of objective function for solution i in the set of solutions.
$f_j(\Phi_i)$	Value of function j for solution i in the set of solutions.
μ	The number of batches in the sequence.
ν	The number of unscheduled jobs.
ξ	The number of batches that can be formed that contain two or more jobs.
ρ	The number of jobs in the sequence that are in batches containing two or more jobs.
Φ'_i	Solution i in the set of solutions for each iteration in an algorithm after a random move has been made in VNS.
X_i	The sequence for solution i when it is used in the general sense.
Y_i	The sequence for solution i when compared to another solution.
$\mu(X_i)$	The number of batches in the sequence of solution X_i.
$d(X_i)$	The duration of the schedule in solution X_i.

$d(X_i, Y_i)$	The weighted distance between solutions X_i and Y_i with $X_i \neq Y_i$.
f_j^{max}	The maximum value for $f_j(\Phi_i)$ for all the solutions in the local archive.
f_j^{min}	The minimum value for $f_j(\Phi_i)$ for all the solutions in the local archive.
$d_{X_i}^j$	The distance of the j^{th} nearest neighbour of solution X_i.
$\bar{d}(X_i)$	The total crowding distance of a solution X_i.
$x_i^d(t)$	Particle i's position for dimension d at time t.
$v_i^d(t)$	Particle i's velocity for dimension d at time t.
$p_i^d(t)$	Particle i's best known position for dimension d at time t.
$g^d(t)$	The best known position of all particles for dimension d at time t.
$z(X_i)$	The fitness function to determine the fitness value for particle X_i.
S_{max}	The sequence in $\mathcal{S}$ with the highest fitness value.
$c_{ij}^d(t)$	The characteristic for egg i in nest j for dimension d at time t.
X_{ij}	The sequence for solution i in a set j when it is used in the general sense.
$z(X_{ij})$	The fitness function to determine the fitness value for the egg X_{ij}.
$f_k(\Phi_{ij})$	Value of function k for solution (i, j) in the set of solutions.
q_j	The quality of a nest j.
X_{new}	The sequence determined by the cuckoo after performing Lévy flights.
S_{cur}^i	The solution that individual i in TS is currently at.
S_{best}^i	The best found solution for individual i in TS.
Φ	The solution that is taken as input for the NSTB algorithm.
b	The index of a chosen batch from a sequence when performing one of the steps in the NSTB algorithm.
b'	The index of a chosen job from a batch when performing one of the steps in the NSTB algorithm.
B	A solution that is obtained after performing one of the steps in the NSTB algorithm.
b_i	The duration of batch i.
s_i^j	The start time of job j in batch i.
t_i^j	The duration of job j in batch i.
π_i	The number of jobs in batch i.
a_i	The available space in period i.
z_i	The fitness value for solution i.
p_j	The profit for batch j.
p_i^{jk}	The profit for job k in batch j in solution i.
μ_i	The number of batches in solution i in the NSTB algorithm.
π_{ij}	The number of jobs in batch j in solution i.
$\bar{a}_i$	The available space in batch i.
p'_{max}	The maximum profit of the jobs in a set.
t_{max}	The maximum duration of the jobs in a set.
t_{min}	The minimum duration of the jobs in a set.
t_i^{jk}	The duration for job k in batch j in solution i.
$\bar{p}_j$	The average profit of job j if it starts in any of the periods in a batch.
$\bar{t}_j$	The average duration of job j if it starts in any of the periods in a batch.
h_i^j	The fraction of the machine that job j in batch i requires for the full period of implementation.

Vectors

$\mathbf{d}_{X_i}$	The vector containing the distances of the nearest neighbours of solution X_i.
$\mathbf{y}_i$	A vector containing the indices of the jobs in the order that they are inserted into batches for particle i.

$\mathbf{x}_i(t)$	The position vector of particle i at time t or the chromosome of individual i in generation t.
$\mathbf{h}_i$	The selected h_i values for the jobs in the order of their appearance in $\mathbf{y}_i$.
$\mathbf{v}_i(t)$	The velocity vector of particle i at time t.
$\mathbf{g}(t)$	The vector representing the best known position of all the particles or eggs at time t.
$\mathbf{c}_{ij}(t)$	The characteristic of egg i in nest j at time t.
$\mathbf{y}_{ij}$	A vector containing the indices of the jobs in the order that they are inserted into batches for egg i in nest j.
$\mathbf{h}_{ij}$	The selected h_i values for the jobs in the order of their appearance in $\mathbf{y}_{ij}$ for egg i in nest j.
$\mathbf{c}_k(t)$	The position of the egg after k moves have been performed during the Lévy flights at time t.
$\mathbf{b}$	A random binary string that is generated during PUX and PUOX.
$\mathbf{s}_i$	The start times of the jobs in batch i.
$\mathbf{t}_i$	The durations of the jobs in batch i.
$\mathbf{a}$	The available space in each of the periods.
$\mathbf{s}'_i$	The start times of the jobs in batch i after a job has been moved or inserted.
$\mathbf{t}'_i$	The durations of the jobs in batch i after a job has been moved or inserted.
$\bar{\mathbf{a}}$	The available space in each of the batches.

Parameters	
n	The total number of jobs, including the derived jobs.
w	The number of time periods over which the schedule can run.
t_{ik}	The processing time of job i when its implementation starts during period k.
p_{ik}	The profit of job i when its implementation starts during period k.
h_i	The fraction of the machine that job i requires for the full period of implementation.
r_i	The relative weight representing the relative importance of job i.
t_{ijk}	The processing time of job i when it follows on job j and its implementation starts during period k.
p_{ijk}	The profit of job i when it follows on job j and its implementation starts during period k.
v_{ijk}	Equal to 1 if starting project i during period k when it follows job j will continue after the permitted last period of the schedule.
l'_{ijk}	An index that is set to the period in which job i finishes if it follows job j, starts in period k and will be completed before or on the permitted last period of the schedule. The index is set to the permitted last period if job i will not be completed before the end of the schedule.
S	The matrix containing the setup times of jobs when preceding other jobs.
C	The matrix containing the costs associated when a job precedes another job.
s_{ij}	The setup time for job i if it follows on job j.
c_{ij}	The cost associated with setting up the machine for job i if job j precedes it.
η	The computational time at which the last iteration of an algorithm is performed before being terminated.
p_{min}	The minimum number of periods that the schedule is allowed to be.
p_{max}	The maximum number of periods that the schedule is allowed to be.
a_{max}	The maximum size of the local archive.
r	The number of nearest neighbours that are used in the calculation of the crowding distance.

x_{min}	The minimum value for the dimensions of the initial position for all the particles.
x_{max}	The maximum value for the dimensions of the initial position for all the particles.
v_{max}	The maximum velocity in the positive and negative directions for all the particles.
ω	The inertia factor used to update the velocity of a particle.
φ_p	The acceleration coefficient in the direction of the best known position of each particle from the previous time step.
φ_g	The acceleration coefficient in the direction of the best known position of all the particles from the previous time step.
w_j	The weight assigned to objective function j (future work).
t_j	The target value of objective function j (future work).
b_{max}	The maximum number of jobs allowed in a batch.
s	The number of solutions that are generated containing different $\mathbf{h}_i$ vectors for a particle's position or individual's chromosome.
n'	The total number of jobs, excluding the derived jobs.
δ	The number of nests.
ε	The number of eggs in each of the nests.
c_{min}	The minimum value for the dimensions of the initial eggs.
c_{max}	The maximum value for the dimensions of the initial eggs.
ψ	The probability that the cuckoo will start its Lévy flights from the position of the best egg obtained so far.
λ	The location parameter of the Lévy distribution.
σ	The scale parameter of the Lévy distribution.
ζ_w	A fraction of the nests with the smallest value for q_j that are selected to either be abandoned to build and populate a new nest with eggs or its worst egg is thrown away after which a new one is produced.
ζ_n	The probability that a nest will be abandoned after which a new one will be built at another location and populated with new eggs. The egg with the lowest value of $z_{X_{ij}}$ will be thrown out of the nest with a probability of $1 - \zeta_n$ after which a new one will be produced.
γ	The number of Lévy flights that a cuckoo performs before laying its egg in a nest.
g_i	An independent random integer variable drawn from the uniform distribution when creating processing time and profit matrices for $i = 1$ and 2.
g'_{min}	The minimum value that the trend of the processing time or profit of the jobs in a schedule is allowed to be.
g'_{max}	The maximum value that the trend of the processing time or profit of the jobs in a schedule is allowed to be.
g_{min}	The minimum value for the trend of the processing time or profit of a job.
g_{max}	The maximum value for the trend of the processing time or profit of a job.
$g_j(x)$	The function for trend j that is used to create the processing time and profit matrices.
w'	The number of periods in the processing time and profit matrices.
$\bar{s}$	The maximum setup time for a job.
$\bar{c}$	The maximum setup cost for a job.
f	The length of the strip when selecting pairs of parents in SUS.
p	The number of points that is chosen on the strip when selecting pairs of parents in SUS.

α	The probability of mutating an individual's chromosome.
α_{max}	The maximum number of genes to change during mutation for GA-I.
α'_{max}	The maximum number of successive moves that are made on an individual during mutation for GA-II.
β	The percentage of the size of the local archive when immigration is performed.
$\lvert\bar{\mathcal{N}}\rvert$	The maximum size of the subset of neighbours that are considered for the candidate list in TS.
υ	The maximum number of iterations that a move stays in the tabu list.
τ	The tabu list size.
w'_1	The weight for the profit of a job when determining its fitness value for roulette-wheel selection in the NSTB algorithm.
w'_2	The weight for the duration of a job when determining its fitness value for roulette-wheel selection in the NSTB algorithm.
η'	The number of subiterations in each of the steps, except the merging of batches, in the NSTB algorithm.
Others	
$\lvert\bar{\mathcal{J}}_i\rvert$	The number of derived jobs in batch i for all the batches in the sequence.
$\lvert\bar{\mathcal{H}}_i\rvert$	The number of derived jobs for the unscheduled job i.
$\lvert\tilde{\mathcal{H}}_i\rvert$	The number of derived jobs for the scheduled job i.
$\lvert\mathcal{A}_l\rvert$	The size of the local archive.
$\lvert\mathcal{P}\rvert$	The number of particles/individuals in the population.
$\mathbf{f}(x, \lambda, \sigma)$	A vector consisting of n' random variable that are drawn from the Lévy distribution.
κ	A random floating point number in the range $[0, 1)$.

CHAPTER 1

Introduction

Contents

Scheduling is a decision-making process used in manufacturing and service industries [80]. It involves the use of optimisation techniques and heuristics to determine good or optimal sequences in which activities should be performed in an environment.

1.1 Overview of scheduling

Environments in scheduling may be machines in a factory, runways at an airport, instruments in a testing facility and processing units in a computer. Activities for these resources may be items that need to be manufactured, take-offs and landings at an airport, the testing of missile flights and computer programs that need to be executed, respectively. Single or multiple objectives may be optimised, such as maximising the total profit of the activities that are

processed, minimising the time it takes to complete all activities (makespan), minimising the total weighted time that activities are performed after their due date (tardiness) or maximising the total weighted time that activities are performed before their due date (earliness). This section contains an introduction to scheduling in manufacturing and services as well as a variety of scheduling examples.

1.1.1 Scheduling functions in manufacturing and services

Orders that are received in a manufacturing framework are converted into a set of jobs (activities), possibly attached with due dates, that need to be scheduled on a single or multiple machines (environment). Jobs are typically released at different times in a given period and the processing time of a job may depend on the predecessor/successor job. The sequence of the jobs is thus important when setting up a schedule. Unexpected events can happen in the environment, such as machine breakdowns, delays in the processing times of jobs or fluctuations in the number of workers who are present, that sometimes need to be included in the model. Service models, on the other hand, does not deal with goods that can be stored in inventory but instead a service that is delivered to individuals. Describing a general model for a service framework is more complicated than for a manufacturing setting. A service organisation is faced with many problems compared to its counterpart. These scenarios may include the hiring or booking of resources (such as trucks or venues), the assignment and scheduling of a service provided to the public (*e.g.* the schedule of flights in the airline industry) or the scheduling of workforce (for instance the assignment of doctors in a hospital to shifts).

1.1.2 Manufacturing models

Manufacturing models can be characterised into five different scheduling models, namely single machine models, parallel machine models, flow shop models, job shop models and supply chain models [80]. A single machine model is, as the name suggests, when only one machine is present on which jobs can be processed while a parallel machine model contains multiple identical or non-identical machines that process different jobs simultaneously. When a set of operations (machines) need to be performed to manufacture or assemble a product (job) and the same sequence of machines need to be visited by all jobs, the problem is considered a flow shop model. A job shop model is a generalisation of a flow shop model in which jobs can travel different routes to visit each of the machines. Finally, a supply chain model is a network of flow shop models and job shop models.

There are several types of constraints that may be present in a machine scheduling problem. A combination of precedence constraints, machine eligibility constraints, routing constraints, waiting time constraints and transportation constraints may be found in a model. Precedence constraints need to be included in the model if a job can start its processing after a given set of jobs has started or completed its processing or alternatively if a job is forbidden to start after another set of jobs has started or completed its processing. Precedence constraints are usually accompanied with sequence dependent setup times and costs. These times and costs are included in the model if the machine needs to be prepared for the following job to be processed once a job has completed its processing. Machine eligibility constraints are used when there are restrictions as to which machines a job can be processed on if there are multiple machines in the model. Routing constraints specify which route a job has to follow when in a flow shop, job shop or supply chain model. If there is a limit in the amount of space that is available for items present on the factory floor, called work-in-progress inventory, then waiting time constraints are implemented so that there are upper bounds for the time that a job has to wait before

being processed on the following machine. Transportation constraints are included if there are transportation costs involved when moving a job from one machine to another or if there are constraints as to which time a product can move to the following machine.

1.1.3 Service models

Service models can be divided into four different classes of models [80]. The first class consists of models involving timetables and reservation systems. In a reservation system a job has a fixed start time and completion time which is known in advance. A decision needs to be made as to which jobs to process if there are any overlapping amongst the jobs. Timetables have more freedom as to when a job can be processed. In this case, each job is associated with a time window in which it should start and finish its processing. The second class is comprised of tournament scheduling and television broadcasting. A tournament scheduling model assigns teams to a set number of games and schedules these matches into time slots. The matches are subject to several preferences as to when and where they should take place as well as restrictions set by the broadcasting network for televising. The last two classes are transportation and workforce scheduling. Transportation models involve the scheduling of flights, trains and ships in the airline, railway and shipping industries. Lastly, workforce scheduling is used when workers are assigned to shifts in a service facility (such as a call centre) or crew are assigned to vehicle trips (which may include aeroplanes, trucks, trains or ships) in a transportation setting.

1.1.4 Scheduling examples

Examples in a manufacturing setting may include the production of automobiles on an assembly line, a manufacturing facility that produces semiconductors as well as the procurement, installation and testing of large computer systems [80]. Different models of automobiles belonging to the same family of cars are generally produced in an automobile assembly line. Each car comes with a variety of colour options and accessories that can be added to the car, depending on the requests received by the vehicle local distribution (or car dealership). There are several impediments that may arise at an automobile manufacturer that affects the overall production rate of the facility. An example is the paint guns that need to be cleaned every time a new colour is applied to a vehicle, which can be very time consuming. One of the objectives in this environment may be to maximise the production rate by sequencing the jobs (cars) in such a way as to ensure that the workload is balanced across all stations (machines).

A second example is a semiconductor manufacturing facility in which four phases are typically present to produce a semiconductor for memory chips and microprocessors, namely wafer fabrication, wafer probe, testing and packaging. The most technologically complex phase is the wafer fabrication in which seven operations need to be completed repeatedly on each wafer. The wafers move through the facility in sets with some machines requiring a setup time and cost that depends on the configuration of the set that just completed and the set that is about to start. Additionally, there are release dates and due dates associated with orders. The goals may therefore be to minimise the idle time of the machines, setup time of the machines, and/or tardiness. Another example in manufacturing is a system installation project. A number of tasks need to be completed on a computer that includes the installation and maintenance of hardware, software development, system debugging, *etc.* Some tasks need to be performed before others while some tasks can be performed concurrently. This gives rise to a scheduling problem with precedence relationships and multiple machines. As a result, the objective is to finish the project in the shortest time possible, *i.e.* minimise the makespan of the schedule.

Examples in a service environment may include the scheduling of nurses in a hospital, routing and scheduling of aeroplanes as well as a reservation system. For the first example, the staffing requirements in a hospital vary from day to day, mainly depending on the day of the week, and the time of day. The number of nurses that should be on duty on a weekday is generally higher than on weekends. Also, the number of nurses necessary during a night shift is usually less than a day shift. Several other additional constraints are necessary depending on the hospital's preferences and regulations set by the government. The second example is concerned with the assigning of an aircraft to flights and the scheduling of these flights to a round-trip based on customer demand. Constraints include the minimum and maximum time an aeroplane spends on the ground (turnaround time) and the shift restrictions of the crew. The final example is a reservation system in which a fleet of various types of vehicles can be rented out by a car rental agency. The agency needs to be make decisions as to whether to accept a request by a customer to rent a vehicle. A customer's request is typically denied if the reservation period for a vehicle is too short when that same vehicle could have been rented out to another customer for a longer period. The goal in this scheduling problem is to maximise the time that the vehicles are rented out which consequently maximises the agency's profit.

1.2 Planning techniques for scheduling

Several techniques that are applied before scheduling takes place are discussed in this section. They provide a background on how a scheduling problem is formed using master production scheduling, material requirement planning and capacity planning. Moreover, the methods demonstrate where the parameter values in a scheduling problem originate by using forecasting, aggregate planning and inventory control. Values may include the profit and processing time of a job as well as the jobs that are available to be processed at a given time.

1.2.1 Forecasting and aggregate planning

A production facility's aim is to maximise profit after manufacturing and selling products. A decision needs to be made as to when the production should be started and how many items to produce during a production run to meet demand. This requires forecasting techniques to be used to estimate the expected demand in the near future [94]. There are two types of forecasting, namely qualitative forecasting and quantitative forecasting. Qualitative forecasting is used in the event that no historical data is accessible, usually the case when a new product is being introduced to the market. This subjective manner of forecasting generally makes use of expert opinions, market surveys and panel discussions. One well-known approach to use a group of experts to make accurate predictions is the Delphi technique [85]. Quantitative forecasting is used when historical data is available. This forecasting group can be classified into time series analysis and causal forecasting [102]. Time series analysis is practised when the assumption is made that the demand pattern from the past will continue into the future. Examples of techniques that can be used are simple exponential smoothing for stationary data, Holt's method when the data follows a linear trend and Winter's method when the data exhibits trend and seasonality. Causal forecasting methods are applied when the demand is strongly correlated with factors other than time. Methods used in this analysis may include linear/multiple regression, simulation techniques and econometric modelling.

Aggregate planning can be done once the demand for future periods has been forecasted [94]. It is the process of planning how the forecasted demand will be met on time so as to minimise the total cost of operation. It gives management an estimation of the number of resources

to collect to manufacture the products and at which times this collection should take place. Aggregate planning also provides an idea of when inventory should be carried, when stockouts will occur, how much overtime is required by the workers and the number of workers to hire, to name but a few. Three strategies exist in aggregate planning, that is the chase strategy, level strategy and flexible strategy. The chase strategy says that a large production capacity should be implemented and workers should be hired and fired based on when they are needed. This strategy is used when the costs associated with holding items in inventory is high and the cost of hiring and laying off workers is low. The level strategy works in an opposite way so that the capacity of the production facility is large and the number of workers who are employed is fixed. It is applied when the costs of carrying inventory and placing of backorders are low as well as when the capacity of the facility is difficult to modify. The final strategy, the flexible strategy, is utilised when both the costs of carrying inventory and adjusting the capacity of the facility are low. In this strategy the production capacity is large while the number of employed workers is fixed.

1.2.2 Master production scheduling and material requirement planning

Now that the demand has been forecasted as well as the hiring/firing policies, facility capacity, the resources that are needed and other factors to meet demand on time have been decided on, master production scheduling takes place. This method is used to determine the number of individual products that should be produced instead of the number of products as a whole [94]. The planning horizon is much shorter than in forecasting and aggregate planning spanning over a few weeks or even days as opposed to months. The production environment is the main factor that determines what is considered as a product. The three environments that can be used to manufacture products are make-to-stock (in which the number of products produced depends on the supplier's demand forecasts), assemble-to-order (where products are assembled based on the options a customer has added such as computers and vehicles) and make-to-order (in which a product is only produced once an order has been placed) [99]. The fundamental rule in master production scheduling is that sufficient but not successive stock should be available to meet demand in any given week. The planning of when inventory will be delivered from a supplier and thus orders will be released to customers should be done in an optimal manner.

Material requirement planning is concerned with the components of the demand requirements obtained from master production scheduling [94]. These components include the items that were manufactured but not yet sold, items that were purchased by a customer, the functions of assemblies and subassemblies as well as tools that are required to assemble items. The intentions of this type of planning are to determine the times at which production should commence for items, the quantity of these items that should be assembled during each production run and when the items should be delivered to keep inventory levels of the production plant to a minimum. There are several considerations to be made during planning. The minimum planning period should be at least as long as the cumulative lead times of the product. Requirements need to be collected from all levels in the product trees when there are common components amongst products when many products are being planned for simultaneously. The limitations of available manpower and production capacities need to be incorporated in the planning. Lastly, most of the departments in the company need to be included in the planning phase, for instance marketing, engineering, finance and human resources.

1.2.3 Inventory control and capacity planning

Two further critical processes in planning need to be enforced before scheduling can take place. These processes are inventory control and capacity planning. Inventory planning involves determining optimal inventory levels, when ordering should take place for restocking and factors that may influence the cost of holding inventory [94]. Inventory needs to be managed, since products that have been produced or purchased cannot leave the facility immediately. The objective of inventory models is typically to determine the economic quantity that should be ordered or produced so that the holding costs of inventory and the shortage costs (if it is assumed that shortages may occur) are minimised [96]. Factors used in the model may include the demand per unit time (normally a year or month with the demand calculated using forecasting methods), the cost of holding a unit in inventory (which depends on storage costs, taxes, spoilage and obsolescence), the cost of being short one item (such as cost of special orders, lost sale and loss of future goodwill) and the cost of placing an order or starting a production run. Various scenarios can exist when managing inventory, such as when quantity discounts are allowed, multiple products are ordered or produced simultaneously, the period review policy is implemented, using the ABC classification system or enforcing a just-in-time inventory system.

Capacity planning involves the use of the capacity of machines or production plants to determine how resources should be scheduled [94]. Production plants consist of resource centres where detailed manufacturing happens depending in which stage products are during their production. A decision needs to made on a daily basis as to which items to produce and the number of items to produce. The capacity of the machines is the main factor that limits the number of a specific item that can be produced on a given day provided that sufficient raw material is accessible. Rough cut planning is used to calculate an approximation of the capacity for each resource centre depending on the current material requirement planning schedule that is implemented and if the schedule allows additional products to be manufactured. Information on product demand, labour and/or machine standards as well as how each centre has operated in the past. A more accurate method to determine the capacity of the centres uses information in the aforementioned technique in addition to the availability in the bill of materials and information on the route a product takes in the facility while it is being produced. Once the capacity of each centre has been regulated, the number of machines that are required can be calculated using information such as the daily production time, each plant's efficiency and factors that may stop production.

1.3 Characteristics of the scheduling problem considered in this thesis

This section contains an introduction to the characteristics of the scheduling problem considered in this thesis. All scheduling problems in the literature assume that there are a finite number of jobs available to schedule and a finite number of machines to schedule them on [79]. The number of jobs is typically denoted by n while m represents the number of machines. In the considered scheduling problem the number of machines is $m = 1$. In the literature, the subscript i typically refers to a machine while the subscript j refers to a job. Any theoretical scheduling problem is described by the so-called three field notation [44] given by a triplet $\alpha|\beta|\gamma$. The first field contains a single entry describing the machine environment. The β-field supplies characteristics and constraints that are present in the scheduling problem. This field may contain no or multiple entries. The γ-field presents the objective(s) to be optimised. This field contains multiple entries for a multi-objective problem. If c_j and w_j denote the profit and weight (which is a priority factor assigned to the job depending on its importance relative to other jobs) assigned to job j, respectively, then the three field notation for the considered scheduling problem is given by

$1\,|\,\beta\,|\,\sum w_j c_j$. The entries in the β-field are discussed in §1.3.1–1.3.5.

1.3.1 Batch scheduling

In the literature, batching is defined as the grouping of jobs into batches that are processed on the same machine [10]. In its most basic form, there is a setup time for each of the batches, assumed to be the same for all batches so that setup times don't depend on the sequence within the batches. There are two types of batching problems. The first type is called s-batching in which the length of a batch is computed to be the sum of the processing times of the jobs in the batch. If the length of a batch is calculated to be the maximum of the processing times of the jobs in the batch, the problem is called p-batching. The considered problem is concerned with the latter batching type, since jobs are processed simultaneously on the machine. The length of a batch is therefore equivalent to the processing time of the longest job. The considered problem is also a bounded model, since it is assumed that the maximum number of jobs that can run simultaneously is less than the total number of jobs. The symbol "p-batch" is added to the β-field of the considered scheduling problem's classification. If the makespan is minimised, the most basic form of the problem (if the problem does not contain any other characteristics or entries in the β-field and only one objective is optimised) can be optimally solved by assigning the remaining jobs with the smallest processing times in each consequent batch until no jobs remain. However, this method can not be used to solve the considered scheduling problem, since the processing times of the jobs are time-dependent.

1.3.2 Time-dependent processing times

Seegmuller *et al.* [87] noted that problems in the literature assume that the processing time of a job exhibits a functional relationship with its start time. There are three types of relationships that have been extensively studied. The first type is when the processing time of a job is either a decreasing or increasing function in terms of its start time. This function is given by $p_i = a_i \pm b_i s_i$ where p_i is the processing time of job i and s_i is the start time for job i. The constants a_i and b_i have subscripts, since the jobs may have different functional relationships. Secondly, the processing time of a job may be either stationary and then increasing or decreasing and then constant as a function of its start time. This is given by the functions $p_i = \max\{a_i, a_i + b_i(s_i - d_i)\}$ and $p_i = a_i - b_i \min\{s_i, p_i\}$, respectively, where d_i is an additional parameter. The final type is when the processing time of a job is a step function in terms of its start time, given by $p_i = a_i$ or b_i. None of these functions describe the considered scheduling problem's time-dependence. A job's processing time can for instance be a trigonometric or quadratic function with a random component such that fluctuations are present. None of the methods in the literature are designed to solve the problem for when the duration of a job is given in vector form where each element indicates its duration if it were to start in that period. The symbol "*gtdpt*" is added to the β-field to indicate the aforementioned description of general time-dependent processing times.

1.3.3 Sequence dependent setup times

The symbol "s_{jk}" is added to the β-field if there is a setup time involved between two jobs j and k [79]. The setup time for job k if it is the first job to be processed is given by s_{0k}, assumed to be zero for all jobs in the considered scheduling problem, while s_{j0} denotes the cleanup time if job j is the last job to be processed, also assumed to be zero for all jobs. The setup times for all the jobs in a batch are added to determine the total setup time for a batch. There are several methods in the literature that attempt to solve the most basic form of a scheduling

problem with sequence dependent setup times. The most notable group of methods is when the problem is formulated as a travelling salesman problem if the objective is to minimise the sum of the setup times. The travelling salesman problem is a well-known optimisation problem in which a salesman needs to travel to each of the available cities exactly once and return to the origin city such that the distance travelled is minimised. For the scheduling problem, each city represents the job while the distance between a pair of cities refers to the required setup time. Methods used to solve the travelling salesman problem include a specially designed branch-and-bound algorithm that implements solution, elimination and partitioning routines, dynamic programming as well as the closest-unvisited-city algorithm in which the closest unvisited city to the current city is always visited next [21].

1.3.4 Job selection

Job selection is present in a scheduling problem if all jobs can not be processed due to capacity or time constraints and a subset needs to be selected for scheduling. Once the subset of jobs have been determined, they need to be sequenced in such a way that the objective is optimised. There is typically a reward involved when scheduling a job which is the main deciding factor for selecting a job [92]. Various algorithms have been designed to solve scheduling problems exhibiting the selection of jobs, namely polynomial time, branch-and-bound and branch-and-cut algorithms which are discussed in §2.4. The considered scheduling problem is concerned with schedules that have a maximum makespan. This results in schedules not containing all of the available jobs causing some jobs to be rejected. Each job is associated with a reward in the form of a profit that depends on the period in which the job starts its execution. The objective is thus to maximise the total profit of the schedule that depends on the jobs that were selected. The selection and scheduling of the jobs should thus happen simultaneously, since different subsets of jobs have different optimal schedules. The symbol "$\mathcal{J}_s \subset \mathcal{J}$" is added to the β-field for job selection, where $\mathcal{J}_s$ is the set of scheduled jobs and $\mathcal{J}$ is the set of all jobs.

1.3.5 Precedence constraints

Precedence constraints may be found in a scheduling problem if a given job is allowed to start only if a set of jobs have started or completed their processing before the given job. This is indicated by the "*prec*" symbol in the β-field. Precedence constraints are usually in the form of a precedence constraints graph with each node representing a job and each arc (j, k) representing job j that needs to have started or be completed before job k can start its processing [80]. There are typically two types of graphs found in the literature, namely chains and trees (consisting of intrees and outtrees). In a chain, each job has at most one successor and at most one predecessor. An intree consists of jobs having at most one successor while an outtree only has jobs that have at most one predecessor [79]. The considered scheduling problem's precedence constraints are not confined to chains and trees. The graph can take on any form based on restrictions that a given job (1) is not allowed to follow a given set of other jobs and (2) start in a specified set of periods. The number of predecessors and successors of a job is not limited to one each. In concluding this section, the three field classification for the scheduling problem considered in this thesis is therefore given by $1 \,|\, p\text{-batch}, gtdpt, s_{jk}, \mathcal{J}_s \subset \mathcal{J}, prec \,|\, \sum w_j c_j$.

1.4 Problem description and applications

A problem description of the considered scheduling problem is given in §1.4.1 followed by a few of its applications to real-world problems provided in §1.4.2.

1.4.1 Problem background

The $1 \,|\, p\text{-batch}, gtdpt, s_{jk}, \mathcal{J}_s \subset \mathcal{J}, prec \,|\, \sum w_j c_j$ scheduling problem is essentially a service model but uses concepts generally used in a manufacturing setting. A set of available jobs is presented to management of which a subset should be selected for the schedule. This is because of the schedule horizon that is limited to a maximum makespan, resulting in certain jobs being completely rejected or left in a pool of unscheduled jobs for a later stage. Jobs can be processed simultaneously on a single machine, thus the problem exhibits a form of batch processing in which batches may overlap. Once a job has completed its processing, there is a possibility that the machine needs to be cleaned up and prepared for the next job to be processed. Setup times and costs are therefore associated with each ordered pair of jobs. Additionally, the processing time of a job depends on the time at which it starts on the machine. Lastly, certain jobs are prohibited from starting directly after the processing of another given sets of jobs at a specified set of time periods. These restrictions are imposed by the use of precedence constraints.

1.4.2 Applications

An application of the scheduling problem considered here is on the testing of aircraft and missile flights on a multipurpose test range. Batch scheduling is considered due to multiple flight tests (referred to as jobs) that can be tested (or processed) on a single test range (referred to as a machine). Tests are allowed to be processed simultaneously on the test range which is represented by a constant that indicates the fraction of the test range that the test requires. Since the duration of a test depends on the time of year that the test is being performed, as a result of factors such as weather conditions, learning curves and technical difficulties, time dependent processing times are present. Sequence dependent setup times are considered seeing that the test range needs to be prepared for the next flight test to be performed once another flight test has been completed. The setup time for the testing of a flight depends on the condition of the test range at that time, therefore the sequence of the tests plays an important role in the duration and cost of testing the flights. Since a subset of tests must be selected out of a set of available tests to be performed, job selection is present in the model. This comes from the fact that there is not sufficient available time to test all flights, thus certain tests need to be rejected or scheduled for a later time than the time frame of the current schedule being considered. Finally, precedence constraints are used because of certain flights that are unable to be tested after the testing of another specific flight at a given time period.

Another application is the scheduling of resources to underprivileged communities. There is a team of volunteers and permanent workers in a non-profit organisation who need to distribute necessities to different areas within a cluster of communities. All the areas together may be viewed as the machine and the activities (distributing of resources to a certain area) may be seen as the jobs to be scheduled on the machine. Groups of individuals may perform activities simultaneously due to multiple available vehicles, therefore batch scheduling is present. Time-dependent processing times play a significant role, since there may be more necessities in demand during winter months (or weekdays) compared to summer months (or weekends). Also, rainy weather conditions cause the travelling time from a location to another location to increase which consequently increases the processing time of a job. Job selection may be included in the model

due to a limited number of communities that can receive necessities due to a limited number of resources or amount of time available. Sequence dependent setup times and precedence constraints may also be included in the model based on factors such as the locations of the communities and the loading/unloading of resources into/from vehicles. The objective may be to maximise the impact made on the communities or to minimise travelling time or costs.

The final example of an application (there are, however, many other real-world applications not mentioned in this section) is the scheduling of jobs in a busy vehicle maintenance and repair facility. There are multiple repairers in a workshop allowing multiple vehicles to be repaired simultaneously (batch scheduling). As repairers become more skilled, they are able to perform repairs much faster as time progresses and the day of the week or time of day may influence how long it takes a repairer to work on a vehicle (time-dependent processing times). An area may need to be cleaned up and tools need to be prepared before the following vehicle can be worked on (sequence dependent setup times). If it is assumed that there are more requests than the capacity of the workshop such that there is a waiting list, certain jobs need to be rejected (job selection). Lastly, the workshop may prefer certain groups of vehicles to be worked on one after the other (precedence constraints). The objective in this case may be to maximise the number of vehicles operated on, minimise the setup times or maximise the total profit of the schedule.

1.5 Scope and objectives of thesis

The focus in this thesis is on the development of metaheuristics to solve a scheduling problem containing five characteristics, namely batch scheduling, time-dependent processing times, sequence dependent setup times, job selection and precedence constraints. The scheduling problem is simplified for the metaheuristics so that jobs are inserted into batches that are not allowed to overlap. A variety of metaheuristics, in which solutions are represented as a sequence of batches or when solutions are defined as a string of floating point numbers that require decoding, are implemented. No formulations or algorithms could be found in the literature for when jobs can be moved around within batches. This knowledge gap is addressed in this thesis by developing an algorithm that allows jobs to start at different times within the batch that they are assigned to. Furthermore, all the metaheuristics use a local or external archive for the extension of the algorithm to incorporate multi-objectivity (a problem that frequently arises in practice), to provide a set of solutions to the aforementioned algorithm for further processing and to contribute towards diversity in the population of solutions during execution of a metaheuristic. Lastly, the comparing of algorithms using parametric and non-parametric tests is often overlooked in the literature, another research gap that has been identified. As a result, the metaheuristics are compared against one another with an emphasis on statistical hypothesis testing.

The following nine objectives are set for the thesis to develop and compare metaheuristics for the considered scheduling problem.

Objective I: Provide an introduction to scheduling in general after which explanations are given to each of the characteristics of the scheduling problem.

Objective II: Give a review on the literature for each of the characteristics found in the scheduling problem. Focus is put on metaheuristics that have been used to solve a given scheduling problem (theory and heuristics are presented when little research on metaheuristics could be found).

Objective III: Formulate integer programming models for the scheduling problem along with the assumptions that should be made for when a job's processing time depends on its start

time and when a job's processing time depends on its start time and the preceding job(s) that have been processed. In both instances there is a possibility that jobs can run in parallel.

Objective IV: Present preliminary theory necessary for the use of metaheuristics and provide a background to each of the metaheuristics developed in this thesis.

Objective V: Develop metaheuristics to solve the scheduling problem in terms of batches while maximising profit.

Objective VI: Develop an algorithm that uses a population of solutions obtained from a metaheuristic that moves single and multiple jobs within a batch, merges batches and inserts jobs into a batch so that jobs in a batch don't necessarily all start at the same time.

Objective VII: Generate test cases that mimic real-life situations to compare the algorithms against one another. Also, determine the combinations of parameters that give good solutions and implement convergence analysis to decide on stopping criteria for the metaheuristics.

Objective VIII: Compare the metaheuristics against one another for different difficulty groups of test cases and conclude the best performing metaheuristic for each of the test cases using statistical hypothesis testing.

Objective IX: Recommend which metaheuristic(s) should be used for an instance that falls within a given difficulty group and suggest further studies in a detailed format.

1.6 Thesis structure

This chapter started off by providing an overview of scheduling in manufacturing and service environments. Planning techniques were then discussed that are typically performed before scheduling takes place. A problem description, applications and the characteristics for the considered scheduling problem was provided followed by the scope and objectives of the thesis. A comprehensive survey of the literature is given in Chapter 2 for each of the five characteristics for the considered scheduling problem. Chapter 3 contains the exact formulations for the scheduling problem and preliminary theory for the metaheuristics developed in this thesis. A background as well as the pseudocode (that includes a description of the algorithm) are given for each metaheuristic. Additionally, a specially designed algorithm may be found in this chapter in which jobs in a batch may start at different times as opposed to jobs in a batch starting at the same time in the metaheuristics.

In Chapter 4 test cases are generated, parameters are calibrated for each of the algorithms to determine optimal combinations of parameters and convergence graphs are plotted to choose the stopping criteria that should be used for the metaheuristics. The algorithms are then compared to one another using different measurements after which hypothesis testing is done to conclude on the best performing metaheuristic for each test case. Finally, Chapter 5 provides a summary of the thesis, recommendations regarding the parameters that should be used for the algorithms and which metaheuristics should be utilised based on their performance. A discussion on whether the objectives set for the thesis were achieved, is provided. The thesis is concluded with possible future work.

CHAPTER 2

Literature Review

Contents

An introduction to the five different characteristics of the considered scheduling problem, namely batch scheduling, time-dependent processing times, sequence dependent setup times, job selection and precedence constraints, were given in §1.3. A review on the literature of each of these characteristics may be found in §2.1–2.5.

2.1 Batch scheduling

Batch scheduling is the simultaneous processing of several jobs on one or more appropriate machines [15]. Jobs may be categorised into different families where only jobs in the same

family are allowed to be inserted into the same batch to be assigned to a machine [74]. However, the existence of one family is also possible so that any job has the ability to be processed with any other job on the same machine at a given time. Batches may contain any number of jobs provided that the batch size[1] does not exceed the capacity of the machine. All jobs in a batch need to start at the same time. No papers were found in the literature in which jobs in a batch are allowed to start at different times. A diverse selection of variations were found in the literature which include batch scheduling problems with delivery times [15, 98], bi-criteria batch scheduling [33, 68] and machines that operate in parallel [11, 53, 62]. Sections 2.1.1 and 2.1.2 contain metaheuristics present in the literature which may be of great value for the case when jobs may run simultaneously on the same machine. Section 2.1.1 contains genetic algorithms which is extended to the memetic algorithm while particle swarm optimisation and tabu search techniques may be found in § 2.1.2.

2.1.1 Genetic algorithms

Mönch *et al.* [74] conducted a study in the manufacturing of semiconductors in the area of diffusion and oxidation of a wafer facility. This was achieved by modelling the problem as a parallel dynamic machine scheduling problem where batch processing is allowed. The total weighted tardiness is minimised which depends on the different priorities/weights, due dates and ready times of the jobs to be scheduled. Two decomposition approaches, namely the two-stage and three-stage approaches, were developed as a result of the complexity of the problem. The two-stage approach assigns the jobs to the machines in the first stage using the genetic algorithm (GA) and simultaneously forms the batches and schedules the batches on the available machines. On the other hand, the three-stage approach forms batches, assigns the batches to the machines using the GA and schedules the formed batches on the available machines in the first, second and third stage, respectively. Mönch *et al.* [74] used the concept of the apparent tardiness cost (ATC) in several places within their algorithm. The procedure chooses batches from each of the families and schedules a batch according to three different indices once a machine becomes available to process the next job. The two approaches were tested for instances of 3, 4 and 5 parallel machines. It was concluded that the three-stage approach performed better than the two-stage approach in terms of solution quality and the running time of the algorithm.

Chou *et al.* [18] extended the use of the GA, applied by Mönch *et al.* [74], to a hybrid algorithm to solve the single batch machine dynamic scheduling problem while minimising the makespan. The batch machine processes multiple jobs simultaneously as a single batch while ensuring that the capacity of the machine is not exceeded. Also, preemption is not allowed and the processing time of a batch is equal to the processing time of the job in the batch that requires the longest processing on the machine. Assumptions for their model include that all jobs are compatible with one another (there is only one job family) the sum of the job sizes in the batch should not exceed the capacity of the machine and machine breakdown is not allowed (the machine is available once a batch is done processing).

Two improvements are introduced, namely the right-shifted and swapped improvement. The right-shifted improvement shifts a job from a batch to the batch following it given that the job has the longest processing time in the batch it was moved from. On the contrary, the swapped improvement trades a given job in a batch with a subset of jobs in the batch following the given job's batch. These improvements are used as a hybrid with the GA to form a hybrid GA. Chromosomes represent the job sequence with the number of genes equal to the number of jobs to be scheduled. Mutation is enforced by swapping random genes within a chromosome. Lastly, a

[1]The batch size is defined as the sum of the sizes of the jobs in a given batch. The batch size may also be viewed as the fraction of the machine that the batch occupies when the capacity of the machine is one [87].

stopping criterion is used to stop the algorithm after 1 000 generations, if no improvement on the best solution has been found in the last 100 generations or if the lower bound has been reached. The hybrid GA produces satisfactory solutions of an average quality with the improvement procedures strengthening the solution consistency.

The multi-objective GA was first introduced by Kashan *et al.* [51] to address the single batch scheduling problem when two objectives are to be considered opposed to a single objective considered by Mönch *et al.* [74] and Chou *et al.* [18]. Two different multi-objective genetic algorithms (MOGAs) were proposed to deal with the minimisation of the makespan and tardiness. The sequence based MOGA (SMOGA) generates sequences of jobs by means of GA operators in an attempt to search the solution space. The batch first-fit (BFF) heuristic assigns the jobs into batches for each of the generated sequences. The batch based hybrid MOGA (BHMOGA) generates batches while preserving feasibility using the random batches procedure (RBP) heuristic. It then applies a local search within each generation to improve the generated solutions.

The random key representation technique is used to initialise a population of chromosomes in the SMOGA by generating a random number confined within the interval $[0, 1]$ for each of the jobs. The numbers are then arranged in descending order to determine the order in which the jobs are processed. In the BHMOGA, the RBP is used to assign jobs to batches where each gene represents the batch that the job is in. A truncated geometric distribution is used to produce the number of initial batches to avoid the algorithm from being trapped in a local minimum. Crowded tournament selection is used to select two parents for breeding to deal with the multiple objectives. If two competing individuals are both nondominated solutions, the one having the larger crowding distance is selected as a parent. It was found that the BHMOGA return solutions closer to the Pareto frontier compared to the SMOGA in all of the tested cases. This is due to the method that was used to generate the initial population, using the RBP during crossover and mutation and utilising a local search heuristic to further improve solutions.

Chiang *et al.* [16] applied a memetic algorithm (MA) to batch scheduling which takes parallel machines into account, expanding on the single machine problem solved by Kashan *et al.* [51]. Jobs are classified into different families meaning that only jobs that are in the same family may be placed in the same batch. They proposed that the encoding of the batch formation and batch sequencing should happen simultaneously. A chromosome consists of a sequence of batches, each containing at least one job with the number of jobs in each batch not exceeding the maximum batch size. Encoding takes place by inserting the batches into a chromosome from left to right in increasing order of the completion time of the previous job that was processed on the same machine, given the current schedule. The completion time of the previous batch is taken as zero for the first batch, since no batch has been processed yet. Batches are assigned to the machine with the earliest available time during the decoding process.

The fitness of an individual is determined by the reciprocal of the total weighted tardiness so that the individual with a higher fitness value is more suitable to reproduce. After the two offspring have been produced, the two individuals (amongst the two parents and two children) with the highest fitness value will proceed to the following generation. A local search procedure is started only when the best individual in the current generation has a higher fitness value than the best individual in the previous generation. Only a certain number of the best individuals are used for the local search to save computational time. Neighbouring solutions are chosen by using the batch formation mutation operator and the batch sequence mutation operator according to a predetermined probability. They compared the MA with the algorithm developed by Mönch *et al.* [74] as a benchmark. It was concluded that there is an improvement by more than 10% on average and a decrease in the running time of at least 50%.

2.1.2 Particle swarm optimisation and tabu search

Chang *et al.* [14] studied the scheduling of parallel batch processing machines by using a hybrid particle swarm optimisation (PSO) algorithm. The maximum lateness was minimised with dynamic job arrivals and several job families. The PSO algorithm simulates the flocking of birds and swarm intelligence to find improving solutions in each iteration. The solution quality is improved by combining several heuristics, including the earliest due date (EDD), release date update (RDU) and batching update (BU) methods, to produce a hybrid algorithm. The initial population is generated by applying EDD to the first particle, RDU to the second particle and BU to the third particle. The remaining particles are generated randomly.

A particle, representing a solution, is characterised by its position and velocity whose dimension is equal to the number of jobs to be scheduled. The velocity of each particle is calculated using two inertia factors, two acceleration coefficients and the best position to date of the specific particle at hand. The new position of each particle is then updated by adding its previous position to its calculated velocity. This process is repeated depending on the stopping criterion of the algorithm. In the decoding process, the sequence of jobs is determined by sorting the elements of the position vector in descending order. The rank of each of the elements determines the position of the jobs in the sequence. The proposed Given Sequence Batch (GSB) procedure is then used to place jobs in batches and assign these batches to the parallel identical batch processing machines. Computational results indicate that hybrid PSO may be used in practical production fields as it can solve large problems in a reasonable amount of time.

Meng & Tang [72] investigated the application of a tabu search (TS) heuristic to solve the scheduling problem for a single batch processing machine with non-identical job sizes. A greedy heuristic is used to generate initial solutions which is based on dynamic programming to group the jobs into batches in an attempt to minimise the number of batches. Each batch is obtained by solving a one-dimensional knapsack problem. The solution with the minimum number of batches does not necessarily represent the optimal solution, therefore a tailored TS is then used to seek an improving solution.

The TS method employs two different neighbourhoods, namely the one-to-one swap neighbourhood which swaps a job from a batch with a job from another, and the one-to-two swap neighbourhood that swaps a job from a batch with two jobs from another batch. However, it was discovered that two more complex neighbourhoods are required for further and faster improvements of a solution. The swap chain neighbourhood applies a one-to-one swap between two consecutive batches for all of the n batches creating a series of $n - 1$ swaps. The second neighbourhood, the insert chain neighbourhood, addresses the fact that the number of batches can change. It moves a job from each batch to the batch following it, forming a string of moves, which gives it the ability to add and remove batches. The TS heuristic is effective for all instances ranging from 10 to 100 jobs. It obtains optimal results for cases consisting of 10 jobs, optimal solutions for nearly all instances of 20 jobs and near-optimal schedules for 50 and 100 jobs.

2.2 Time-dependent processing times

Time-dependent processing times arise in various fields in scheduling, especially in production and service environments. The scheduling of emergency medical response teams, firefighters, resources to control epidemics, the shaping of metals in metallurgical processes and learning activities requiring associated skill levels that are reduced with time are one of the few examples [95]. Two different models exist in the literature, that is when the processing time of a job

is an increasing or a decreasing function of its starting time [57]. An example of an increasing function is when products (jobs) require processing by a cutting tool (machine). The tool quality, *e.g.* the sharpness of its blade which depends on its age and the number of products it has processed, concludes the time required to process a product. As a result, there will be an increase in the processing time of a product as time passes. On the contrary, the use of a decreasing function for the processing time of a job is also known as the "learning effect". A well-known example involves a worker who assembles a large number of similar products. After a certain period the worker will be better skilled, would have learned how to produce at a faster rate and will be better organised in his workplace. The time required to assemble one product thus decreases through the worker's learning. Heuristics for solving time-dependent problems when minimising the total weighted completion time, when learning effects are present and when machines run in parallel follow in § 2.2.1–2.2.3.

2.2.1 Minimisation of the total weighted completion time

Sundararaghavan & Kunnathur [95] defined a new type of scheduling problem in which the processing time of each job on a single machine is a binary function of a common due date for the start time. A processing time penalty is incurred for jobs that start their processing on the machine later than their due date. The objective is therefore to minimise the sum of the weighted completion times. Weights are used to allow more important jobs to have a higher priority of being processed first. Three cases were considered, namely the general case, the case when there are only two distinct weights and the case when two different penalties may be incurred. An optimal heuristic was introduced for each of the cases. The initialisation process for the general case involves the arranging of the jobs in non-increasing order of their weights after which the maximum number of jobs that can be scheduled without incurring the processing time penalty is scheduled on the machine. The remaining jobs are then placed in a set of delayed jobs. The last step performs an exchange between any early and late jobs until no exchange causes an improvement in the objective function.

The algorithm for the instance when there are two distinct weights starts by assigning early jobs with a higher weight in non-increasing order of their penalty followed by the assignment of early jobs with a lower weight in the same manner until the maximum number of jobs that can be scheduled without incurring the processing time penalty is reached. The following step interchanges early jobs having a higher weight with late jobs having a smaller weight until no further improvements on the objective function can be made. The last step interchanges early jobs having a lower weight with late jobs having a higher weight until the objective function can no longer be improved. Finally, the algorithm for the two-penalty case assigns early jobs with a higher penalty in non-increasing order of their weights followed by the assignment of early jobs with a lower penalty in the same way until the maximum number of jobs that can be scheduled without incurring the processing time penalty is reached. Early jobs having a higher penalty are exchanged with late jobs having a lower penalty until no further improvements can be made. Early jobs with a lower penalty are then exchanged with late jobs having a higher penalty until the objective function can no longer be improved.

Bachman *et al.* [4] extended the scheduling problem by Sundararaghavan & Kunnathur [95] to a problem when the processing time of a job is defined by a decreasing linear function while still minimising the total weighted completion time. Preemption is not allowed and all jobs are available for processing since the beginning of the schedule. Two heuristic algorithms were constructed to find a near-optimal solution to the presented NP-hard problem. The first algorithm schedules the jobs in non-decreasing order according to the ratio $a_i/w_i(1-b_i)$ where a_i, w_i and b_i denote the normal processing time, the weight and the decreasing rate of the

processing time of the job, respectively. The second algorithm constructs an even-odd V-shaped schedule[2].

Both algorithms generate a solution in linearithmic time, or in $\mathcal{O}(n \log n)$ steps. The algorithms were tested for when 10 jobs and 100 jobs are present. It was found that for 10 jobs the first algorithm demonstrated superior results compared to the second algorithm returning solutions very close to the optimum. The first algorithm performs better for large values of a_i and small values of b_i. For the case when 100 jobs are to be scheduled, the first algorithm again performs significantly better than the second algorithm for all the considered cases. There is therefore sufficient evidence to suggest that the jobs be scheduled in non-decreasing order according to the ratio $a_i/w_i(1 - b_i)$.

2.2.2 Time-dependent learning effects

Gao *et al.* [38] considered single machine scheduling problems when learning effects is present while minimising the total weighted completion time. This is similar to the work of Bachman *et al.* [4]. However, the problem is now extended to include precedence constraints demonstrated by relations between jobs in the form of parallel chains and series-parallel graph precedence constraints. Another difference is that the special case is considered where the decreasing rate of the processing time of a job is directly proportional to the normal processing time of the same job. A further assumption is made that a job's processing time stops decreasing as soon as its processing commences on the machine.

Two algorithms were presented to solve the problem. The algorithm for parallel chains precedence constraints selects the chain with the highest ρ^*-factor[3] from the set of remaining chains as soon as the machine is available for processing. This chain is processed until the job that determines its ρ^*-factor is done processing. The remaining jobs in the chain form a new smaller chain to be processed in the future. This process is repeated until all jobs have been scheduled. The algorithm for series-parallel graph precedence constraints proceeds by moving from the bottom of the decomposition tree upwards to find an optimal sequence of jobs by means of series composition and parallel composition. Both algorithms determine the optimal solution in polynomial time.

Lu *et al.* [70] broadened the scope of Bachman *et al.* [4] by introducing release dates, opposed to all jobs being available since the beginning of the schedule. In addition to release dates, jobs are arranged into groups with each job in a group having the same setup time and job processing time which are both decreasing linear functions of their starting time. Group technology occurs frequently in real-life situations, assists in the efficiency of operations and decreases the requirement of different facilities. The scheduling problem may be formulated as follows: There are a total of n jobs grouped into m groups that need to be processed on a single machine. A positive setup time is required if the machine has to switch from one group to another, otherwise the setup time is taken as zero. If the processing of a job may not be interrupted, find an optimal schedule that minimises the makespan.

The authors first considered the problem with decreasing time-dependent processing times and ready times of jobs without group technology to prove the lemma that an optimal schedule for the problem may be obtained by sequencing the jobs in non-decreasing order of their ready times. They then regarded the situation when group technology is present to use the lemma

[2] A V-shaped schedule sequences the early and on-time jobs in descending order of their processing time whilst the tardy jobs are sequenced in ascending order of their processing time [37].

[3] The ρ-factor of a chain was introduced by Pinedo [79] to determine which of the available chains to process in a time-dependent schedule.

to prove the following theorem: For the problem with group technology, an optimal schedule can be constructed by sequencing the jobs in each group in non-decreasing order of their ready times followed by arranging the groups in non-decreasing order of a variable that depends on the completion time of each group. This theorem was then converted into an algorithm with a complexity of at most $\mathcal{O}(n \log n)$ that determines the optimal solution with the existence of group technology.

Another variation of machine scheduling with the presence of time-dependent learning effects is when the actual processing time of a job is also a function of the job's scheduled position in the sequence as formulated by Yin *et al.* [109]. Each job is associated with a due date and a weight that is used when minimising the weighted sum of completion times. The actual processing time of a job is defined by the product of

1. the job's normal processing time,
2. a differentiable non-increasing function with an independent variable equating to the sum of the normal processing times of the jobs that were already processed, and
3. a non-increasing function that depends on the index in which the job is positioned in the sequence.

It is evident from the model that the learning effect is stronger on jobs that still need to be processed for later positions in the sequence. The authors introduced theorems and corollaries that contain methods for finding an optimal schedule for various modifications of the problem. These different methods for a single machine with learning effects are listed below.

1. Jobs are ordered according to the Shortest Processing Time first (SPT) rule when minimising the makespan.
2. When minimising the sum of the completion times to the k-th power, jobs are ordered according to the SPT rule. The use of a power is due to polynomial cost functions that need to be considered in certain scheduling circumstances.
3. Jobs are ordered according to the SPT rule when minimising the total completion time.
4. If all of the jobs have reversely agreeable weights[4] when minimising the total weighted completion time, then the jobs are ordered in non-decreasing order of the ratio of the processing time to the weight of the jobs. This is known as the Weighted Shortest Processing Time first (WSPT) rule.
5. If the normal processing time of all jobs are equal while minimising the total weighted completion time, then the jobs are ordered in non-increasing order of their weights.
6. Jobs are ordered in non-decreasing order of their due dates if the due dates are agreeable[5] when minimising the maximum lateness.
7. For the problem when the normal processing times of all jobs are equal while minimising the maximum lateness, jobs are ordered according to the EDD rule, similar to that employed by Chang *et al.* [14].

[4]Reversely agreeable weights is when the job with the smaller processing time should have the larger weight amongst two jobs for all jobs.

[5]Agreeable due dates is when the job with the smaller due date should have the smaller processing time amongst two jobs for all jobs.

8. In the situation when the due date is the same for all jobs while minimising the maximum lateness, the jobs are ordered according to the SPT rule.

The authors were able to show that the previously mentioned methods polynomially solve the problems when minimising the makespan and the sum of the completion times to the k-th power. Further, problems when minimising the total weighted completion time and the maximum lateness was showed to be solved polynomially only subject to certain conditions.

2.2.3 Parallel machine scheduling

Kuo & Yang [57] studied the scheduling of time-dependent jobs in an identical parallel machine environment. They considered both increasing and decreasing linear functions for the processing time of a job. The objectives were to minimise the total completion time of all the jobs and the total load[6] on all the machines. As a result, four different scheduling problems were analysed. It was deduced that the jobs should be sorted in non-decreasing order of their normal processing time and then one by one the jobs are assigned to a machine as soon as it becomes available for the cases when

1. minimising the total completion time for an increasing linear function,
2. minimising the total completion time for a decreasing linear function, and
3. minimising the total load on all the machines for an increasing linear function

to determine an optimal schedule. Also, it was found that jobs should be assigned to a single machine, arranged in non-increasing order of their normal processing time and split up into the total number of machines while maintaining the arranged sequence for the case when minimising the total load on all the machines for a decreasing linear function.

In contrast to only considering identical machines, Kuo *et al.* [56] focused on unrelated parallel machines with time-dependent processing times. The actual processing time of a job is a linear increasing or decreasing function, but what makes this problem different from the one considered by Kuo & Yang [57] is that a job may have different normal processing times on different machines. Also, instead of having the same increasing/decreasing rate of the processing time of a job for all the machines, each machine may have different increasing/decreasing rates. The increasing/decreasing rates are identical for all jobs on a specific machine. This assumption may be attributed by the fact that a machine is usually the cause of the deterioration or learning rate of the actual processing time of jobs. It was shown that the problems for both the increasing and decreasing function are solved in polynomial time.

Lastly, Zou *et al.* [111] considered several uniform parallel machine scheduling problems in which jobs have time-dependent processing times represented by a linear increasing function. Each machine has a speed factor that defines how fast a machine is able to process a job relative to other machines. The objectives to minimise are the same as that of Kuo & Yang [57]. They also considered the case when a job has a delivery time, an additional time that is required for the job to be delivered to an entity. This additional time is added to the completion time to give a delivery completion time.

Two polynomial algorithms were developed to solve the scheduling problems when minimising the total completion time and the total load as well as a fully polynomial-time approximation

[6]The total load is defined as the sum of the largest completion time on each of the machines or simply as the total time that all the machines are busy processing jobs.

scheme (FPTAS)[7] when minimising the total load. Furthermore, an FPTAS was derived for solving the problem when minimising the maximum delivery completion time. This was achieved by implementing the rounding-the-input-data technique and modifying the deteriorating rates by using a geometric rounding technique. Using the theorem that "for any $0 < \epsilon \leq 1$, the optimal objective function value under the modified deteriorating rates is at most $(1 + \epsilon)$ times the optimal value for the same problem under the original deteriorating rates" and the corollary that "there exists an optimal job sequence such that on each machine the jobs are sequenced in non-increasing order of their delivery times", a dynamic programming algorithm was developed to solve the problem.

2.3 Sequence dependent setup times

In several cases, industries utilise a single processing environment to run similar operations one after the other. The changeover from one operation to the next typically requires a setup time before processing may continue [75]. This setup enables the environment (machine) to be in a state so as to process the operation (job) according to its requirements and regulations. In the literature, setup times are usually defined by s_{jk} when job j is immediately preceded by job k on the same machine while minimising the makespan is one of the objectives to be considered. A trajectory metaheuristic, the tabu search method, is considered in § 2.3.1 followed by a variety of evolutionary algorithms in § 2.3.2 which include the genetic, self-evolution and hybrid evolutionary algorithms. Lastly, approaches for when the problem is transformed into a time-dependent travelling salesman problem may be found in § 2.3.3.

2.3.1 Tabu search

Eren & Güner [32] executed a study on the bicriteria scheduling of jobs with sequence dependent setup times. The two objectives were to minimise the weighted sum of the total completion time and the total tardiness. They formulated an integer programming (IP) model after which it was discovered that the model is able to efficiently solve problems of only up to 12 jobs. As a consequence, two heuristic methods were developed, namely the so-called E-G heuristic and a tabu search method. The average solution quality[8] for the large sized problems were 98.24% and 99.97% for the E-G heuristic and the tabu search method, respectively, indicating that the tabu search method is a better choice when solving the considered scheduling problem. This is due to the tabu search method using solutions obtained by the E-G heuristic as initial solutions and improving them.

The E-G heuristic starts off by sequencing the jobs according to the SPT rule as used by Yin *et al.* [109]. The first two jobs from this rearranged list are then selected and sequenced such that the weighted sum of the two objective functions is minimised. The next job in the rearranged list is selected and inserted into each of the remaining slots of the schedule to generate a k number of candidate sequences where k is the number of jobs in the sequence after inserting the job. The sequence that provides the smallest weighted sum of the two objective functions is then selected. This process is repeated until all jobs are present in the sequence. The tabu search method uses this final solution as its initial solution. The tabu list length was taken as $2\sqrt{n}$ where n is the number of jobs while adjacent pairwise interchange (API) was used as the

[7] An algorithm is a ρ-approximation algorithm if a solution is obtained that is at most ρ times that of the optimal solution. An algorithm is called an FPTAS if for each $\epsilon > 0$ the algorithm is a $(1 + \epsilon)$-approximation algorithm running in $\mathcal{O}(n^c)$ time for an input size of n and $1/\epsilon$, assuming that $c > 1$ and $0 < \epsilon \leq 1$.

[8] The solution quality is calculated by the formula (Solution quality) $= 1 - \frac{\text{(Heuristic solution)} - \text{(Optimal solution)}}{\text{(Optimal solution)}}$.

neighbourhood search strategy. The procedure continues until there is no improvement after n repetitions.

The bi-objective case was extended by Choobineh *et al.* [17] for the inclusion of more than two objectives in a single machine scheduling problem with sequence dependent setup times. An m-objective tabu search was proposed that returns a solution that reflects the weights assigned to the objectives and lies close to the best observed values of the objectives. Their algorithm is unique compared to previously implemented tabu searches, since it uses m tabu lists. That is, each objective has its own tabu list which is independent of all the tabu lists from the other objectives. Its peculiarity is also explained by the tracking of the best identified value for each objective as if solving the problem for the single objective case as well as creating a bounded solution space. By using weights for each objective, it is ensured that bounds for the more critical objectives will lie closer to their best identified values while bounds that are of an insignificant nature will be present at a greater distance from their best identified values.

The authors noted that resource utilisation and due-date agreements are the two main performance measures in shop scheduling problems. Resource utilisation is measured by the makespan while the number of tardy jobs indicates whether due-dates have been met. Determining the total tardiness further contributes to the quantifying of the level of customer satisfaction. They therefore considered minimising the makespan, number of tardy jobs and total tardiness as a 3-objective optimisation problem while conforming to the assumption that the setup time of a job is only dependent of the job immediately preceding it and not its position in the sequence. This problem was used to test the tabu search for up to only 20 jobs. The small number of jobs, compared to over 100 jobs considered by Choobineh *et al.* [17], is as a result of the increase in the number of objectives, the tabu lists that are used for each objective and the pairwise position exchange moves employed in the neighbourhood generation. Nevertheless, the algorithm is able to display optimal or near-optimal solutions to the test cases in polynomial time as the number of jobs increases.

2.3.2 Evolutionary algorithms

Vallada & Ruiz [100] presented a GA to solve the unrelated parallel machine scheduling problem with job sequence dependent setup times. This is different from the problems considered in § 2.3.1, since the processing time and setup time of a job now depend on the machine to which it was assigned to (opposed to the default related parallel machine scheduling problem). The proposed GA represents a solution as an array of jobs for each machine with the order of the elements in an array depicting the sequence of the jobs on a specific machine. The initialisation process involves the generating of one individual using the Multiple Insertion (MI) heuristic and the remaining individuals randomly for the initial population. The MI heuristic is then applied to each of the random individuals to secure a good starting population. The selection scheme used in the algorithm is called n-tournament selection. It selects a given percentage of individuals in the population. Out of this subset, it selects the individual with the lowest makespan value as the first parent. A subset is selected again after which the same process follows until the desired number of parents is obtained.

The One Point Order Crossover is used as a crossover operator between two parents to generate two children. It selects a random point on each machine from the first parent. The first child will receive all the jobs before each point while the second child will obtain the jobs after each point. The jobs from the second parent that are not yet in each of the children are added using a limited local search to minimise the completion time of each machine. A shift mutation is then applied to a child according to a specified probability to ensure diversity. Furthermore, an inter-machine insertion (IMI) local search is applied to the best individual according to a given

probability which terminates once a local optimum has been reached. Finally, offspring proceed to the following generation if they are unique and better than the worst individuals in the current generation. It was concluded after extensive statistical analysis that the GA outperforms the metaheuristic for randomised priority search proposed by Rabadi *et al.* [82] and a calibrated version of the heuristic.

Sioud *et al.* [90] introduced a hybrid GA for the single machine scheduling problem with the objective of minimising the total tardiness. What makes this algorithm special is its hybridisation of the crossover operator that is used. The position domains are utilised as in constraint programming, archives are updated as found in multi-objective evolutionary algorithms (MOEAs) and a transition rule is used such as that in ACO. Domain values are used in the position domains to understand information connected to the positions of jobs in the generated children during the crossover. They are thus used to strengthen the use of the absolute position order. The domains are constructed with information from the GA. Since poor solutions in the domains may weaken the information quality, the needed information is recovered using an archive that stores the best found solutions during the execution of the algorithm.

The transition rule is a mechanism that takes the scheduling problem's properties and memory information into consideration to finalise the crossover operation. The filling section (second parent's jobs set) is completed using the transition rule suited for the considered problem after the cross section (first parent's job set) has been defined. The crossover operator may now be summarised by the following three steps: The domain positions are fixed with jobs determined by the archive, the cross section from the first parent is inserted which represents the section between the two crossover points, and the remaining jobs are inserted using the transition rule that utilises two lists that were arranged from the second parent. To test the hybrid GA, benchmark tests were generate consisting of jobs ranging from a total of 15 to 85 jobs. It takes a average of 2.1 and 3.9 minutes to solve small and large instances, respectively, exceeding the results obtained by other approaches in the literature at that time.

Joo & Kim [50] applied a self-evolution algorithm (SEA) to solve the parallel machine scheduling problem with ready times, due times and sequence dependent setup times. They represented solutions in a different manner compared to Vallada & Ruiz [100] by using a single string array for all the machines together. The array is expressed by digits ranging from 1 to n where n is the number of jobs. After a sequence of n digits, an '*' is placed in the array to indicate the end of a machine and the beginning of a new machine. There will thus be a total of $(m-1)$ of these symbols if there are m machines. The dispatching of the jobs and their sequence on the machines are therefore determined simultaneously.

The SEA is different from the GA in the sense that a solution evolves by itself instead of evolving from two parents from the previous generation by means of a crossover operator. It works by selecting a random individual from the population and executing one of five evolution operators which is randomly selected to form a new chromosome in the individual. The five evolution operators include the pull operator, insert operator, swap operator, inner random operator and outer random operator all of which two random points are selected in the chromosome to determine the sections that the operator will modify. After the operator is applied, the new chromosome is evaluated. The chromosome will remain in its cuurent state if its fitness value has improved. However, if the fitness value declined, the chromosome will return to its original state. This iteration is repeated until a predetermined number of self-productions have been completed. An advantage of the SEA over that of the GA [100] is that it doesn't require any parameters, since self-evolution and the selection of points for each operators happens at random. The SEA takes approximately 14, 33 and 62 seconds to solve problems consisting of 50, 75 and 100 jobs, respectively, for 2, 3 and 4 parallel machines indicating that it is a very effective and efficient algorithm.

Lastly, Xu *et al.* [103] compared six different combinations of three crossover operators and two population updating strategies in a hybrid evolutionary algorithm (HEA) to solve a single machine scheduling problem with sequence dependent setup times. Extensive analysis showed that the linear order crossover operator (LOX) combined with the similarity-and-quality based population updating strategy performed the best. LOX is distinctive from the order based crossover operator due to a chromosome being considered linearly instead of in a circular string. The operation is conducted by selecting two points in each of the parents, say p_1 and p_2, at random. The jobs between the two points are then carried over to the two children, say c_1 and c_2, by keeping the jobs' positions. The remaining jobs are then copied from p_1 and p_2 to c_2 and c_1, respectively, in the order that they appear in the parents.

The worst individual in the population in each generation is replaced according to a similarity-and-quality based goodness score function using the population updating strategy. This is achieved by allowing an offspring to be included in the following generation if it has a lower total weighted tardiness (the objective to be minimised) and is not too similar to another solution in the population. The authors suggested a new similarity measurement to determine the similarity degree between two solutions. The similarity degree is defined as the number of positions that contain the same job in both sequences. A total of 120 widely used benchmarks to test problems minimising the total weighted tardiness when dealing with sequence dependent setup times, were used to test the HEA. Results showed that the algorithm was able to solve 119 of the instances to optimality in under 100 CPU seconds.

2.3.3 Time-dependent travelling salesman problem

Yalaoui & Chu [106] showed that the reduced identical parallel machine scheduling problem with sequence dependent setup times can be transformed into a travelling salesman problem (TSP). The TSP can then be solved efficiently using Little's method [64]. The second part of their proposed algorithm then takes this solution of the TSP and improves it in a step by step manner, this time taking the job splitting in the original problem into account. If the length of the section of each job to be processed on each machine is specified, then the problem can be decomposed into m independent subproblems where m is the number of machines. Each of these subproblems is then equivalent to a TSP. This is due to the individual sections of jobs that can be considered as independent jobs and therefore part of their own TSP. The cities of the TSP correspond to the jobs and the distances between the cities correspond to the setup times. An additional city[9] is added that represents the beginning of the schedule to complete a Hamiltonian cycle.

The step by step improvement attempts to reduce the makespan by decreasing the length of sections on the critical machine[10] while increasing the length of sections of the same job on other machines. The perfect schedule would be to let all machines finish at the same time. It was seen from the results that the overall performance of the proposed algorithm is low when the setup times are crucial with respect to the job durations or when the setup times are scattered. An average deviation of 4.88% from the optimal solution was obtained for 4 500 test cases which indicates a moderate general performance of the algorithm.

Bigras *et al.* [7] introduced an IP formulation of the time-dependent TSP which was extended to the single machine scheduling problem with sequence dependent setup times. The time-dependent TSP is a modification of the classical TSP in which the distance between two cities depends on the time at which the edge/arc is traversed. They used two strong IP formulations

[9]The additional city has a distance of zero from and to all other cities in the TSP.

[10]The critical machine is the machine that finishes the latest and thus the one that determines the makespan of the schedule.

by Picard & Queyranne [78], of which the first one is seen as a shortest path problem on a multipartite network that forces the path to visit each node once and only once, and the second one is seen as a Dantzig and Wolfe reformulation which uses path variables instead of flow variables. Although the Dantzig and Wolfe reformulation contains a greater number of columns compared to the first IP formulation, it can be solved easier. These formulations were then applied to three scheduling problems. The problems minimise the total flow time (the sum of the completion time of the jobs) of the machines, total tardiness of the jobs and the weighted number of late jobs all of which contain jobs that have release dates, due dates and setup times.

They showed that the formulations may be strengthened by using subtours and 2-matching cuts as found in the classical TSP, clique cuts that are originally used in solving the node packing problem and k-cycle elimination which helps to find solutions for vehicle routing problems. Subsequently, it was confirmed with computational experiments that these stronger formulations are worth using to find better lower bounds, particularly for the Dantzig and Wolfe reformulation. The running time, however, becomes unreasonable when trying to solve some instances of 50 jobs or more. It may take more than 100 000 CPU seconds to find an optimal solution using the branch-and-bound (B&B) algorithm to solve the time-dependent TSP.

Iranpoor *et al.* [49] studied the use of a rate modifying activity (RM) that is performed when a machine requires maintenance in a single machine scheduling environment. As a result, this will increase the processing rate of the machine which in turn decreases the processing time of a job. The goals for the problem are to sequence the jobs in the schedule, determine the position of a common due date for all jobs and choose the time at which the RM is performed in the sequence. Three important assumptions for the model are that the processing rate of the machine is fixed during the short term schedule, the job processing times are given and fixed, and the setup times are sequence dependent with setups not being able to be performed during RM. The processing time of a job is defined as the product of its processing time in the low-rate situation (before RM is applied) and its reduction time coefficient[11]. The setup times are not affected by RM, since RM is assumed to only affect the rate at which jobs can be processed irrespective of what their setup times are.

Two metaheuristics which have shown promising results in combinatorial optimisation were used to approach solving the problem. An ACO algorithm was used that incorporates a rank-based ant system and a MMAS to find good solutions. In addition to the ant systems, a 2-swap local search is performed on the ants' partial paths before selecting a new job. The second algorithm that was used is the greedy randomised adaptive search procedure (GRASP). This metaheuristic uses a combination of a partial local search, similar to that of the ACO algorithm, and a constructive procedure called path re-linking. A local search is also done on the constructed solution, opposed to only partial solutions in the ACO, after which path re-linking is carried out between the newly found solution and all the elite solutions. It was concluded that the ACO algorithm outperforms the GRASP for all test instances of up to 100 jobs. The average running time was less than 16 and 21 seconds for the ACO algorithm and GRASP, respectively, showing an immense improvement compared to studies conducted by Yalaoui & Chu [106] and Bigras *et al.* [7].

2.4 Job selection

In various scheduling environments, only a subset of the available jobs may be selected due to constraints in the number and type of machines that are available. Therefore, the DM must

[11]The reduction time coefficient is equal to 1 in the low-rate situation and strictly less than 1 after RM has been applied

decide which jobs should be rejected after which a penalty cost may be incurred. Examples include just-in-time (JIT) scheduling [34], manufacturers using perishable raw material [13] and companies specialising in publishing and printing services [76]. Polynomial time algorithms are found in § 2.4.1 while § 2.4.2 contains branch-and-bound and branch-and-cut algorithms. Scheduling problems when the processing time of a job is not fixed for certain time periods and may vary (different from time-dependent processing times) are presented in § 2.4.3. Finally, the section is concluded with effective heuristics and metaheuristics introduced in § 2.4.4.

2.4.1 Polynomial time algorithms

De *et al.* [24] investigated a single machine environment in which a subset of all available independent jobs must be selected for scheduling. The jobs are then sequenced on the machine to maximise the expected profit. The deadline is calculated by an exponential distribution and preemption is not allowed. Each job is characterised by revenue if the job is completed and a cost that is incurred for tools and materials at time zero if the job is processed. It was shown that the problem is NP-hard in the ordinary sense. Consequently, dynamic programming (DP) algorithms and an FPTAS were developed to find optimal solutions. For both approaches, the jobs are assumed to be in decreasing order according to the value $r_j f_j/(1-f_j)$ where r_j is the revenue and f_j is the Laplace-Stieltjies transformation of processing time's distribution function for job j.

The two DP algorithms are represented by forward and backward recursive methods, respectively. The first algorithm expands a subset of jobs by successively adding jobs with higher indices to the subset and appending these jobs to the sequence. On the other hand, the second algorithm successively adds jobs with lower indices to the subset and prepends these jobs to the sequence. The FPTAS is developed based on the first DP algorithm. It was stated that if z^- and z^+ are the minimum and maximum values, respectively, that the expected profit can be, then the subinterval of length $\Delta z = \epsilon \text{LB}/n$ where ϵ is the maximum permissible error, LB is the lower bound of the expected profit and n is the number of available jobs. Given the subintervals, the rule of the FPTAS is to remove all jobs whose expected profit fall within a certain subinterval except the one with the largest value for $\prod_{j\in\mathcal{S}} f_j$ for all subsets $\mathcal{S}$.

Kyparisis & Douligeris [58] improved on the study performed by De *et al.* [24] by considering the case when the maximum number of non-tardy jobs does not exceed the number of selected jobs for the schedule as well as the case when the number of non-tardy jobs exceed the number of selected jobs for the schedule. A modification on the original B&B method is recommended to be used for the first case while the utilisation of a simple polynomial time heuristic is advised for the second case. The objective is to minimise the total flow time of the assigned jobs subject to minimising the number of tardy jobs that are assigned. The modified B&B method starts by ordering the jobs according to their due dates by using the EDD rule. The next step is to remove a non-tardy job with the largest processing time and processing it immediately after all the non-tardy jobs are done on the machine. The algorithm is terminated if at any given time there are no jobs that are late which implies that the sequence is optimal.

The polynomial time heuristic uses the solution produced by the modified B&B method and removes the tardy jobs from the end of the solution so that all the non-tardy jobs remain. Out of the remaining jobs, the job j with the largest value for $z = (E-j+1)p_j$ is removed where E is the number of non-tardy jobs, j is the position of the job in the sequence of the remaining jobs and p_j is the processing time of job j. Jobs are removed until the desired number of jobs remain in the sequence. A theorem is then used in the last step to find an optimal sequence of the remaining jobs that minimises the total flow time.

2.4.2 Branch-and-bound and branch-and-cut algorithms

Seegmuller *et al.* [87] addressed the problem when a number of jobs need to be selected out of a pool of jobs due to the limited time period that is available to process the jobs on a single machine. What makes this problem different from other problems in this section is that jobs are allowed to be processed simultaneously on the same machine which is represented by a constant that indicates the fraction of the machine that the job requires. Additionally, the time of implementation of a job affects the processing time of the job. The objectives are to maximise the profit of the chosen jobs and minimise the total duration of the jobs selected for implementation with the former having higher priority than the latter. The four categories that are considered in the study are when (1) jobs can run in parallel, (2) jobs cannot run in parallel, (3) the processing time of a job depends on the preceding job and jobs can run in parallel, and (4) the processing time of a job depends on the preceding job and jobs cannot run in parallel.

The authors formulated the problem using two different IP models each containing two objective functions. The first IP model (for the first two categories) is when a job's processing time depends only on its start time. Setup times are not required for this circumstance suggesting that there are no restrictions for jobs that cannot run in parallel for the first category. The second IP model (for the last two categories) takes the preceding job into consideration when determining the processing time of a job. Jobs are categorised into different families of similar setup requirement for the third category. This comes from the fact that not all jobs are able to run simultaneously on the machine, since jobs require specific setup requirements. The duration and profit matrices are needed to solve the first IP while the duration, profit, setup time and setup cost matrices are needed for the second IP. For the second IP, the setup time matrix is added to the duration matrix to calculate the total duration and the cost matrix is subtracted from the profit matrix to obtain the net profit. It was concluded that commercial software take several days to find an optimal solution to realistic problems consisting of approximately 15 jobs and 30 time periods, using the B&B algorithm. This suggests that alternative approaches should be considered.

Fanjul-Peyro & Ruiz [34] focused on the event when unrelated parallel machines are available and a subset of the machines are selected to process a subset of the available jobs to minimise the makespan. The problem in which a subset of machines is selected is referred to as the "Not All Machines" (NAM) problem. It is generally found in environments where a production capacity exists on each of the machines and a decision needs to be made as to which machines to exclude from operation. Since the problem consists of unrelated machines, the DM cannot simply halt slow machines. Selecting which machines to operate is therefore a more difficult task than the selection of a subset of available jobs which is known as the "Not All Jobs" (NAJ) problem. The NAJ problem occurs at companies where jobs are either accepted or rejected based on the profit that they generate. It is thus ideal to have the total profit incorporated in one of the objective functions to maximise.

The authors formulated a mixed integer linear programming (MILP) model for both the NAM and NAJ problems. The NAJ problem can be solved effectively using the branch-and-cut (B&C) method while the NAM problem requires additional algorithms to find good solutions. Three algorithms were implemented. Firstly, ranking and selection procedures are performed to select a subset of machines after which the B&C method is applied to the possibly smaller set of unrelated parallel machines. Secondly, instead of performing B&C to the resulting machines in the previously mentioned algorithm, an insertion local search and interchange local search is done iteratively until a local optimum is reached followed by an iterated greedy search method. The last algorithm applies a fast local search to provide the B&C method with a good initial solution. Once the B&C method is completed, will the two local searches mentioned in the previous method be implemented until a local optimum is reached which is then returned by

the algorithm. Solving the NAJ problem, solutions with an average deviation of less than 0.5% from the optimal solution are obtained with a time limit of 300 seconds. The second mentioned NAM algorithm gave the best results for the same time limit as the NAJ problem resulting in an average deviation of 1.47% from the optimal solution.

2.4.3 Varying processing times

Another variation of machine scheduling problems involving job selection is when the processing time of a job may vary instead of a fixed processing time as in § 2.1 and § 2.3 or fixed time-dependent processing times as in § 2.2. Cai *et al.* [13] considered a scheduling problem where a manufacturer owns perishable raw material (raw fish) that are used to produce different products (seafood). The DM needs to decide which products (jobs) to manufacture in the facility (machine), the processing time for each product to be manufactured and the sequence in which the products should be manufactured. Products that generate the highest profit rate are selected. However, as more of the same product is produced, the marginal profit will decrease due to the quantity of the raw material becoming rare relative to the time required to produce the product. Subsequently, products with a higher unit profit are selected. In terms of selecting the sequence of the products, they are ordered such that those with the largest processing time are produced first and those with the lowest processing time are produced last.

An efficient algorithm is used to determine the time allocation of a product to the facility. The algorithm starts by finding the product with the highest profit rate that is then checked whether it satisfies the identification condition. If it satisfies the condition, then the optimal time allocation has been found. Otherwise, another product with a higher unit product must be present in the optimal machine time allocation. The algorithm returns a unique optimal solution, since the objective function representing the minimisation of the total expected cost is separable and hence strictly convex with linear constraints. Two types of experiments were conducted by changing the amount of raw material and the deadline at which the facility stops producing products. It was concluded that when the deadline is kept fixed, products with medium unit products should be manufactured first. Also, as the deadline tends towards infinity, the most profitable product should be manufactured the entire time while keeping the amount of raw material fixed.

Yang & Geunes [107] extended the problem studied by Cai *et al.* [13] by taking job-specific tardiness costs into account (since each job is associated with a due date) as well as additional costs if the manufacturer or customer has the desire to process a job faster than its normal processing time. The actual processing time of a job is given by $p_j - x_j$ with a compression cost of cx_j where p_j is the normal processing time, x_j is the reduction in the processing time and c is the cost per unit processing time that is reduced. For this reason, the total profit is given by the total revenue of all the selected jobs less the tardiness and compression costs. The goal is now to select a subset of the jobs so that the total profit is maximised. Two heuristics are presented to generate good job sequences as well as an algorithm that maximises the profit for a specific job sequence.

The first heuristic is the GRASP approach that consists of a construction phase followed by a local search phase. A local optimum is produced in each iteration in the neighbourhood of the solution that was obtained in the construction phase. A priority rule is employed which is represented by a variable whose numerator is the estimate of the net revenue and denominator is the processing time of the job. This ensures that jobs with a high new revenue and low processing time are processed first. The modified two-phase algorithm is the second heuristic that is used. It makes use of an evaluation process in which the jobs are sorted in non-decreasing order of their completion time. A stack is then initialised into which potentially good assignments are inserted.

The selection phase selects jobs starting at the top of the stack and schedules them if they have not yet been scheduled and their completion times do not overlap with a previously scheduled job. When optimising a fixed job sequence after using one of these heuristics, a compress and relax algorithm is used to determine the optimal start times, completion times and compression times. The algorithm starts by compressing each job as much as possible followed by iteratively reducing the compression time of each job until the best combination of time compressions is reached while maximising the total profit.

2.4.4 Heuristics and metaheuristics

Oğuz *et al.* [76] looked at order acceptance and scheduling decisions in single machine make-to-order systems where jobs are characterised by their release dates, due dates, deadlines[12], processing times, sequence dependent setup times and revenues. An MILP formulation was presented for the problem which is used to find an optimal solution for up to 15 jobs. Four heuristic algorithms are used to solve problems with a larger number of jobs, the first of which is called the *Iterative Sequence First-Accept Next* (ISFAN) algorithm. It uses a priority rule that uses the revenue-load ratio to assist in the decision of accepting or rejecting jobs and concepts from the SA algorithm to find good sequences of the accepted jobs. The next two algorithms that were proposed are constructive heuristics, namely the *Dynamic Release First-Sequence Best* (d-RFSB) and *Modified Apparent Tardiness Cost with Setups* (m-ATCS) heuristics. These heuristics are dynamic in nature starting with no accepted job and differ in their accepting and sequencing rule. Jobs are accepted one by one by checking the availability of the jobs and the machine, while ensuring feasibility of a sequence after scheduling the job.

The last algorithm is a local improvement procedure that performs order insertion followed by order exchange. It is used after the previously mentioned algorithms in an attempt to increase the number of accepted jobs and the total revenue generated. The order insertion algorithm inserts a job in each of the positions after which the sequence with the greatest increase in revenue is selected. The order exchange swaps every pair of accepted and rejected jobs (one of them being accepted and the other rejected) and again selects the sequence that cause the greatest increase in revenue. The ISFAN algorithm solves instances of up to 50 jobs in less than 10 minutes on average while the d-RFSB and m-ATCS heuristics solve problems consisting of 300 orders in a few seconds. The ISFAN algorithm finds solutions with an average deviation of 8.6% from the upper bound and solutions found by the d-RFSB and m-ATCS heuristics have an average deviation of 10.3% and 6%, respectively. It is therefore concluded that the ISFAN algorithm be used for small and medium sized problems while the m-ATCS heuristics should be used for problems of a large size.

Thevenin *et al.* [97] presented metaheuristics for a single machine scheduling problem in which each job has a release date, deadline and a non-decreasing cost function. Jobs are categorised into different families with a setup time and cost being incurred when switching from one family to another. Also, in the event of a job being rejected, a penalty cost must be paid. The objective is to minimise the linear combination of the sum of the completion times, penalty costs and setup costs. A greedy heuristic, tabu search method and population-based algorithms are proposed. The greedy heuristic generates initial solutions for tabu search, which is used as a local search operator to intensify solutions in the population-based algorithms. The heuristic constructs an originally empty schedule by inserting jobs one by one. An unconsidered job is inserted into the position that minimises its cost giving the heuristic its greedy nature. Tabu search uses the reinsert, swap, add and drop neighbourhood structures. The reinsert structure moves a

[12] Jobs that are completed later than their due date will be penalised while jobs that finish after their deadline will generate no revenue.

job from one position to another while the swap structure swaps two jobs that are currently in the schedule. The add and drop structures simply adds and removes a job from the schedule, respectively. If a job is scheduled to end after its due date after performing the reinsert, swap or add structure, it is removed from the schedule. In each iteration, a solution is obtained for each of the four neighbourhoods after which the best one is selected to move on to the following iteration.

Two population-based adaptive memory algorithms were developed that differ only by the recombination operator that is used. Both algorithms initialise a random population of sequences followed by using tabu search as the intensification operator. A recombination operator is then used to produce offspring solutions for the following generation followed by a memory update operator to keep track of the worst solution and solution that is the most similar to other solutions. The recombination operator of the first algorithm involves randomly selecting a solution from the population and adding its first remaining job to the new solution. A random solution is chosen again after which the same process follows until jobs can no longer be added to the new solution. The second algorithm selects two parents from the population using roulette wheel selection and applies a single point crossover operator to produce two children. It is concluded that the tabu search method is competitive with other methods due to its useful diversification strategy. Furthermore, the adaptive memory algorithm using the single point crossover operator is more efficient amongst the two population-based algorithms. Lastly, tabu search showed the best results, on average, for the set of tested instances while the proposed adaptive memory algorithm improves on tabu search's solutions for instances when more jobs are rejected.

2.5 Precedence constraints

Precedence constraints typically occur in situations when products are manufactured due to technological, marketing or assembly requirements [27]. The introduction of precedence constraints may transform a problem that is solvable in polynomial time to a problem that is NP-complete [60]. That is to say, it is highly improbable for a good algorithm to exist that finds near-optimal or optimal solutions. The complexity of the scheduling problem is mainly affected by the structure of the precedence constraints and the objective function(s) to be optimised [27]. The precedence constraints are usually given in the form of a directed acyclic graph with each directed arc (j, k) implying that job j must be completed before job k may be started [22], or in some cases that job j must start before job k can start [52]. Single machine scheduling problems with time windows to minimise the maximum lateness and total weighted tardiness may be found in §2.5.1 while §2.5.2 contains four different parallel machine scheduling problems found in the literature that deal with precedence constraints. Lastly, the solving of a single machine scheduling problem using tabu search and the developing of a hybrid genetic algorithm are discussed in §2.5.3.

2.5.1 Single machine scheduling with time windows

Liu [69] examined the problem of scheduling a set of jobs nonpreemptively on a single machine while minimising the maximum lateness. Each job is associated with a release date and a set of precedence relations containing the jobs that need to be completed before the job may start. A B&B algorithm is proposed that uses four heuristics to find upper bounds at the initial branch node, namely the early release date (ERD) heuristic, Schrage's heuristic, block heuristic and a variable neighbourhood descent (VND) procedure. The ERD heuristic schedules the jobs in nondecreasing order of their release dates with a job having an earlier due date scheduled first

if there are ties. The first $k-1$ jobs are eliminated if job k is scheduled as soon as it is released and if the maximum lateness of the first $k-1$ jobs is less than or equal to the optimal value for the relaxed problem. Schrage's heuristic finds an unscheduled job with the minimum due date in each iteration, such that its release time is not greater than the maximum of the completion time of the last completed job and the minimum release time of the unscheduled jobs. The precedence constraints also have to be satisfied by selecting the job if all of the jobs that need to precede it are not present in the set of unscheduled jobs. The selected job is then scheduled at the end of the partial schedule.

The block heuristic schedules the jobs in non-decreasing order of their release date and divide the schedule into blocks. The block with the largest maximum lateness is selected and adjusted if its maximum lateness is greater than the maximum lateness of the jobs within the adjusted blocks. The adjustment procedure removes the job with the maximum due date from the block and schedules the remaining jobs in non-decreasing order of their release date to remove idle time that may exist. Lastly, the VND procedure is initialised by choosing the schedule with the smallest maximum lateness out of the schedules returned by Schrage's heuristic and the block heuristic. Six different neighbourhoods are then explored to find an improving schedule with the smallest maximum lateness. The neighbourhoods are searched until no improving schedule is found. The B&B algorithm uses these four heuristics when branching to two different nodes. The ERD, block and Schrage's heuristics are used at each branching node and the root node while the VND procedure is only used at the root node due to it not being able to find a schedule with a critical block[13]. Branch nodes are eliminated if a schedule is found for which its maximum lateness is no less than the current upper bound or if it is impossible for a schedule to satisfy the precedence constraints. It was concluded that the B&B algorithm is able to solve 14 986 out of the 15 000 tested instances for up to 1 000 jobs when all four heuristics are used at the root node and the block heuristic is used at each branching node.

Davari *et al.* [23] considered the scheduling of jobs on a single machine with time windows and precedence constraints to minimise the total weighted tardiness. Each job is characterised by a release date, a due date, a deadline and a penalty incurred for each time unit that a job is processed beyond its due date. The precedence constraints are given by a graph with each directed arc in the graph representing a single precedence constraint. Each directed arc (j, k) indicates that job j needs to be completed before job k can start processing. A B&B algorithm is developed that makes use of two different branching strategies. What makes this B&B algorithm different from the one proposed by Liu [69] is that at each branching node a decision is made on which job to schedule next opposed to having a full of schedule at each node. Thus, an empty schedule is found at the root node and a full schedule occurs at every leaf node of the B&B tree. A forward branch and a backward branch are the two types of branches in a B&B tree. If a job is scheduled after the last job in the partial schedule, then the responsible branch is called a forward branch, while the backward branch schedules a job before the first job in the partial schedule.

As a result, there are two different branching strategies. The first strategy only uses forward branching nodes so that the tree is searched in a depth-first manner. Therefore, nodes with larger out-degrees are scheduled first. In the second branching strategy, the search incorporates backward branching whenever it gets the chance to so that the B&B tree consists of both forward and backward branching nodes. In addition to the branching strategies, the search includes several dominance rules to fathom certain nodes if a partial schedule dominates another. The first dominance rule is based on a two-job interchange. The first part of the rule swaps two jobs in a forward branching node while the second part swap two jobs in a backward branching node.

[13] A block is a critical block if its last processed job's lateness is equal to the schedule's maximum lateness, assuming that no idle time exists among the jobs in the block.

The second dominance rule is similar to the previously mentioned rule, except that it inserts a job in each of the two parts instead of swapping two jobs. The last dominance rule is based on the already scheduled jobs that uses the dominance theorem of dynamic programming to fathom nodes in the B&B tree. It eliminates the node with the largest total weighted tardiness from two partial schedules having the same subset of jobs. The B&B algorithm demonstrates effectiveness when solving small and medium sized problems. It performs better than the successive sublimation dynamic programming algorithm and several other B&B algorithms in the literature, especially when the precedence graph is dense.

2.5.2 Parallel machine scheduling

Kumar *et al.* [55] derived polylogarithmic approximations to the unrelated machine scheduling problem with precedence constraints. The precedence constraints are represented by a treelike structured directed graph whose underlying graph is a forest. The general problem is defined as the scheduling of n jobs on m unrelated machines each of which can process a single job at a given time with each job having different processing times on different machines. Thus, it is crucial to decide which machine jobs should be placed on while obeying the precedence constraints. Three problems are considered, namely the minimisation of the makespan, the weighted completion time and the weighted flow time on chains. When the makespan is minimised, the proposed algorithm partitions the forest into blocks of chains that are then separately solved as a job shop problem. For the minimisation of the weighted completion time, it is shown that the problem may be reduced to the minimisation of the makespan and solved using geometric time windows and additional linear constraints. Finally, in the case when the weighted flow time is minimised, a dependent randomised rounding scheme is applied on natural LP-relaxed problem while ensuring that all precedence constraints are satisfied and the probability of starting a job at a given time is given by its fractional LP value.

Liu & Yang [66] presented a study on the scheduling of multiple jobs with precedence constraints on multiple unrelated machines. A priority rule-based heuristic serial schedule (SS) algorithm is applied to the problem in order to minimise its makespan. It uses the arithmetic mean and standard deviation of the processing times of the jobs to assign a job to a machine as early as possible to avoid idle time as much as possible. The SS algorithm selects a job according to its priority and added to the partial schedule on the earliest available machine in each of its iterations. This is done while ensuring that a job is scheduled only if its predecessors have already been scheduled and a machine is assigned a job only if it has been idle just before the processing of the job commences. If there is only one machine that is idle, then the unscheduled job with the smallest processing time is assigned to the machine. However, if at least two machines are idle, then the unscheduled job with the maximum deviation of the processing times on the idle machines is selected to be processed on the machine that processes the job at a faster rate. It was concluded that the algorithm performs exceptional in the four sets of numerical experiments that were tested, and can be used in metaheuristics to determine good initial feasible schedules.

Gacias *et al.* [36] developed different methods for solving parallel machine scheduling problems with jobs having precedence constraints and setup times. More specifically, a B&B procedure and a climbing discrepancy search (CDS) heuristic are proposed in combination with dominance conditions. The B&B procedure's tree structure is identical to that of Liu & Yang [66] while the node evaluation is performed by calculating the minimum completion time and the minimum lateness for each of the unscheduled jobs after which the upper bound is calculated using the machine with the earliest start time. A dominance rule is put into place to restrict the search space so that further branching of dominated nodes is terminated to avoid unnecessary computation time. The CDS heuristic is used when the B&B procedure is unable to solve large

problem instances. It uses an exploration strategy that applies the earliest start time (EST) rule to assign jobs to a machine. In the event of a tie, the SPT rule is used when minimising the sum of the completion times and the EDD rule when minimising the maximum lateness. A fixed heuristic is used for job selection while a dynamic heuristic is used for machine allocation. A k-discrepancy search is performed around the best found solution in each iteration. The algorithm will then search around the new solution if it is an improvement on the current best found solution. The value of k is incremented by one if a better solution can not be found. The results of computational experiments showed that both methods are efficient and competitive with commercial solvers.

Kim & Posner [52] studied the case when s-precedence constraints is present, that is when a job can only start once all of the jobs that precede it, have started. This is opposed to the precedence constraints considered in the previously mentioned studies in this subsection when a job cannot start until all of the jobs that precede it, have completed. A deterministic scheduling problem with jobs being processed on multiple identical parallel machines with the objective to minimise the makespan, is considered. A critical path (CP) heuristic is developed and tight worst-case bounds on the relative error of the heuristic is found. The CP heuristic is initialised by determining the distances of each of the jobs in the s-precedence constraint graph. The distance of a job is defined as the value of the longest path from the job's node to a terminal node (a node with no child nodes). It is thus the sum of the processing times for the jobs on the critical path. In each iteration of the heuristic, the unscheduled job with the largest distance is selected to be scheduled on the next available machine. This is repeated until all jobs have been scheduled so that a full schedule is presented. Worst-case bounds are then found for when each job in the s-precedence constraint graph has at most one job preceding it, when there are two machines available for a general graph, and when there are at least three machines available for a general graph. The outcome of several test cases indicate that the heuristic is effective and able to solve problems within a short time. It provides better solutions as the ratio of the number of jobs against the number of machines increases and the density of the s-precedence graph decreases.

2.5.3 Metaheuristic approaches

A new tabu search procedure for solving large scale scheduling problems was derived by Pedersen *et al.* [77]. These NP-complete problems involve the processing of elastic jobs[14] having precedence constraints, time windows and capacity limitations on three servers. Each job is associated with a 2-dimensional box that has dimensions duration and capacity. Each solution consists of a box size for each job that implies the duration and capacity associated with the job and the sequence of the job in the schedule giving the order of the starting times. A box size is changed, *i.e.* the duration and the capacity of the job is altered, or the sequence is changed when moving from one solution to another in the solution space. The tabu search finds feasible solutions quickly by considering the precedence constraints first. The precedence constraints are given by a precedence graph with the nodes representing the jobs and the directed arcs showing the predecessor and successor for each precedence constraint. An initial feasible solution is obtained by satisfying the three groups of constraints for the problem, namely the precedence constraints, time window constraints and capacity constraints. For the precedence constraints, the jobs are divided into layers using the precedence graph so that a job can only start once all the jobs in the previous layer have started. A sequence is obtained by taking a convex combination of the sequences when the layers are generated forward and backward. Tabu search is applied to the solution if it is infeasible with the objective to minimise the number of jobs that

[14] An elastic job is a job whose duration depends on the capacity assigned to it.

don't satisfy the time window constraints.

Two kinds of moves are used when exploring the neighbourhood of a solution. A position move changes the sequence by moving a job to another position in the sequence while keeping the box sizes constant. On the other hand, a box move changes the box size of the job while the sequence of the jobs is kept unchanged. Candidate lists are used to keep the number of position moves under a specified limit by not evaluating moves that do not belong to the candidate list. A short-term memory tabu list is employed to avoid recently made moves so that the search is unable to get stuck in a local optimum. Each box move performed on a job is added to the tabu list so that a box move cannot be performed on that job in the following specified number of iterations. If a position move is done on a job, then the tabu list ensures that the job won't be moved to a position within a specified number of positions from the position it initially moved from. However, if the aspiration criterion is accepted that the makespan improved when the forbidden box or position move was made, then the move is allowed to be made. An intensification strategy is used to store moves for a job that improved the makespan so that moves are not made on the job for a specified number of iterations. Lastly, two diversification strategies are utilised in which the first one performs an exhaustive search in the current region while the second one changes regions.

Liu [65] considered an unrelated parallel machine scheduling problem to minimise the total tardiness in which each job has a set of precedence constraints. The proposed hybrid genetic algorithm (HGA) uses a priority rule-based heuristic algorithm (PRHA) to find a good set of initial solutions so that the HGA avoids initially searching an unpromising region. In each iteration of the PRHA, the EDD rule is applied to select a job which is then inserted to a partial schedule onto a machine while respecting the precedence constraints and keeping the start times of the already scheduled jobs unchanged. The population is initialised by using the PRHA for a fraction of the population while the rest of the population is generated randomly by assigning jobs to random machines to ensure diversity in the population. A chromosome is depicted by a string of $n+m-1$ genes where n is the number of jobs and m is the number of machines. The $m-1$ genes that are used to separate the machines is represented by the symbol "$*$" such that there is a total of m subchromosomes, each for a different machine.

Three genetic operators are used, namely the patching crossover, swap mutation and roulette wheel selection. Crossover is performed by choosing two chromosomes in the population. A total of $n+m-1$ flags are randomly produced with a value of 0 or 1 for each of the genes. All of the genes in the first parent with a flag of 1 are moved to the same position in the first child. These genes are then crossed out in the second parent so that the remaining genes from the second parent are moved to the same position in the second child. The inserted genes from the first child are moved to the second child in that order to fill the remaining gene locations and vice versa so that two complete chromosomes are obtained for the first and second child. A child is accepted if its fitness value is better than the average fitness value of the parent population, otherwise it is rejected and another child is generated. This contributes to a faster convergence towards the optimal solution. The swap mutation is performed on each accepted child according to a given probability. It is executed by randomly selecting two genes from two different subchromosomes in a child's chromosome and swapping them, *i.e.* two jobs from two different machines switch machines on which they are processed. The algorithm terminates once the number of completed generations has exceeded the maximum number of generations or if the standard deviation of the fitness values of a population is greater than a given value. Computational results show that the HGA performs well for small-sized problems returning optimal solutions for most test cases, and obtains better solutions than the GA for large-sized problems.

2.6 Chapter summary

The aim of this chapter was to introduce the concepts and terminology, the theory that was derived, and the heuristics and metaheuristics that were used to solve the scheduling problems with the presence of batch scheduling, time dependent processing times, sequence dependent setup times, job selection and precedence constraints. A combination of single and parallel machine scheduling problems may be found for each of the characteristics, for the sake of completeness. Both theoretical aspects and metaheuristics are discussed for job selection, multi-objective machine scheduling and precedence constraints. Only theoretical aspects are reviewed for time dependent processing times and the use of metaheuristics is examined for batch scheduling and sequence dependent setup times.

CHAPTER 3

Methodology

Contents

This chapter provides the exact formulations for the scheduling problem, preliminary theory for the metaheuristics and the background as well as the pseudocode for each of the metaheuristics. Additionally, a specially designed algorithm may be found in this chapter in which jobs in a batch may start at different times. The exact formulations for the four different categories that the problem may fall into, may be found in §3.1. Furthermore, the simplification of the models to be used by the metaheuristics, the representation of a solution that is provided as output by the metaheuristic and the conditions in the metaheuristic that need to hold for a solution to be feasible are discussed in §3.2. The metaheuristic that is considered first is the variable neighbourhood search in §3.3. The definition of its moves and neighbourhoods also provide groundwork for tabu search in §3.7. Additionally, three metaheuristics are considered that do not use moves or neighbourhoods but rather decoding schemes. These metaheuristics are particle swarm optimisation found in §3.4, the recently developed and fairly unknown optimisation algorithm called cuckoo search in §3.5 and the genetic algorithm with decoding in §3.6. Lastly, the genetic algorithm without decoding where a solution is defined as a sequence of batches instead of a string of floating point numbers may also be found in §3.6. The chapter is concluded in §3.8 with an algorithm in which single jobs are moved within a batch, multiple jobs are moved within a batch, batches are merged and jobs are inserted into batches.

3.1 Exact formulations

This section contains two variations for the selection and scheduling of jobs with time-dependent processing times on a single machine with the possibility of jobs being able to run in parallel with other jobs. These variations include (1) the scheduling of jobs whose processing time depends only on its start time found in §3.1.1 and (2) the scheduling of jobs whose processing time depends on its start time and the job that was performed immediately before each job, except for the first job that is processed, which is presented in §3.1.2. They are formulated as an integer programming (IP) model with the objective to maximise the total weighted profit. The set of available jobs of which a subset is selected to be scheduled should be known prior to when the scheduling problem is solved. Additionally, the profit and processing time for each of the possible start times of the jobs is assumed to be known beforehand. The discreteness of the start times and processing times of the jobs do not confine the models to a limited number of applications. The model can easily be adjusted to allow for smaller intervals, *e.g.* days instead of weeks or quarter-hours instead of hours [87].

Four different categories are considered in the exact formulations to model different degrees of complexity of the scheduling problem, *i.e.* the two aforementioned variations. The following four categories represent the classifications of the scheduling problem considered in this thesis.

1. The processing time of a job is dependent on its start time and jobs are not allowed to be implemented in parallel.

2. The processing time of a job is dependent on its start time and jobs are allowed to be implemented in parallel.

3. The processing time of a job is dependent on its start time and on the preceding job, and jobs are not allowed to be implemented in parallel.

4. The processing time of a job is dependent on its start time and on the preceding job, and jobs are allowed to be implemented in parallel.

The first two categories are modelled in §3.1.1 while the last two categories may be found in §3.1.2. To differentiate whether jobs are allowed to run in parallel or not, a parameter indicating the fraction of the machine that a job occupies can be adjusted. That is, if no jobs are allowed to run in parallel, all jobs will have a value of 1 for the parameter indicating that all jobs take up 100% of the machine. On the other hand, if at least one job has a parameter of less than 1, jobs are allowed to run simultaneously with another job on the machine.

3.1.1 Job processing time is dependent on start time

The mathematical notation used for categories 1 and 2 in the exact formulation follows below. The parameters that are used as input for the model are defined by letting

t_{ik} be the processing time of job i when its implementation starts during period k,
p_{ik} be the profit of job i when its implementation starts during period k,
h_i be the fraction of the machine that job i requires for the full period of implementation, and
r_i be the relative weight representing the relative importance of job i.

The sets of variables whose values need to be determined by the model as output are denoted by

$$x_{ik} = \begin{cases} 1, & \text{if the implementation of job } i \text{ starts during period } k \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

and

$$b_{ik} = \begin{cases} 1, & \text{if job } i \text{ is processed during period } k \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

The following assumptions are made to model the case when the processing time of a job depends on its start time only.

1. The chosen jobs must be completely processed within the duration of the schedule.
2. The processing times of the jobs have positive integer values.
3. The parameter h_i has a discrete range of values, *e.g.* $h_i = [0.2, 0.4, 0.6, 0.8, 1]$.
4. A job has different processing times and profit margins for different values of h_i. This comes from the fact that when jobs run in parallel, it will take longer to process a job with a smaller value of h_i due to it taking up a smaller fraction of the machine. The profit has the possibility of increasing as the value of h_i decreases, seeing that the simultaneous processing of jobs usually decreases the total processing time of the jobs in the long run leading to a saving in the cost of expenses.
5. Any subset of jobs can run in parallel, since there are no necessary setup requirements before jobs can commence processing.

The objective is to

$$\text{maximise} \sum_{i=1}^{n} \sum_{k=1}^{w} r_i p_{ik} x_{ik} \tag{3.3}$$

subject to

$$\sum_{k=1}^{w} x_{ik} \leq 1 \qquad i = 1, \ldots, n, \tag{3.4}$$

$$\sum_{i=1}^{n} x_{i1} \geq 1 \tag{3.5}$$

$$\sum_{i=1}^{n} h_i b_{ik} \leq 1 \qquad k = 1, \ldots, w, \tag{3.6}$$

$$\sum_{l=k}^{k+t_{ik}-1} b_{il} \geq t_{ik} x_{ik} \qquad i = 1, \ldots, n,\ k = 1, \ldots, w, \tag{3.7}$$

$$(1-w)y_{ik} + x_{ik} \leq 0 \qquad i = 1, \ldots, n,\ k = 1, \ldots, w, \tag{3.8}$$

$$(w-1)y_{ik} + t_{ik} \leq 2w - k \qquad i = 1, \ldots, n,\ k = 1, \ldots, w, \tag{3.9}$$

$$x_{sk} = 0 \qquad (s,k) \in \mathcal{S}_k, \tag{3.10}$$

$$\sum_{v \in \mathcal{H}_i} \sum_{k=1}^{w} x_{vk} \leq 1 \qquad i = 1, \ldots, n, \tag{3.11}$$

$$b_{ik}, x_{ik}, y_{ik} \in \{0,1\} \qquad i = 1, \ldots, n,\ k = 1, \ldots, w, \text{ and} \tag{3.12}$$

$$t_{ik} \in \mathbb{N}^{+} \qquad i = 1, \ldots, n,\ k = 1, \ldots, w. \tag{3.13}$$

where n is the total number of jobs, including the derived jobs, and w is the number of periods in the schedule. As an example, if there are a total of v jobs that are considered to be scheduled and half of the jobs can run in parallel with another job by occupying 1/3 or 2/3 of the machine, then there will be v derived jobs bringing the total number of jobs to $2v$. Each job that is not allowed to run in parallel has a set of derived jobs $\mathcal{H}_i = \{J_{ia}\}$ with h_i taking on a value of 1 while each job that is allowed to run in parallel with another job has a set of derived jobs $\mathcal{H}_i = \{J_{ia}, J_{ib}, J_{ic}\}$ with h_i being assigned a value of 1/3, 2/3 or 1.

The objective function (3.3) maximises the total weighted profit depending on the relative importance of each job. The fact that a job is not allowed to start more than once is determined by constraint set (3.4), while constraint (3.5) ensures that at least one job needs to start in the first period. Constraint set (3.6) ensures that the capacity of the machine is not exceeded when jobs run in parallel. Constraint set (3.7) determines the value of b_{ik} by setting it equal to one for all the time periods k during which job i is being processed on the machine. Constraint sets (3.8) and (3.9) ensure that a job can only start implementation if it will be completed before the end of the schedule. These constraints were obtained by using the statement that if $k + t_{ik} - 1 > w$, then $x_{ik} = 0$. This results in the two constraint sets

$$x_{ik} \leq M y_{ik} \qquad i = 1, \ldots, n,\ k = 1, \ldots, w, \text{ and} \tag{3.14}$$

$$k + t_{ik} - 1 - w \leq M(1 - y_{ik}) \qquad i = 1, \ldots, n,\ k = 1, \ldots, w, \tag{3.15}$$

with M a large positive number and y_{ik} a binary variable for each of the constraints. Since it can be assumed that $\max_{i,k}\{t_{ik}\} \leq w$, $M = \max\{k\} + \max_{i,k}\{t_{ik}\} - 1 - w = w - 1$ is sufficiently large if $\max_{i,k}\{t_{ik}\} = w$ in the extreme case. This simplifies to the two constraint sets given in the formulation. Constraint set (3.10) is used to prohibit job s from starting during time period k for all the job-period pairs found in the set $\mathcal{S}_k$, while constraint set (3.11) guarantees that not more than one element from the set of derived jobs $\mathcal{H}_i$ can be chosen for the schedule. Finally, the variable set (3.12) contains all the binary variables and the variable set (3.13) states that the processing time of a job must be a positive integer.

3.1.2 Job processing time is dependent on start time and preceding job

In the event that the processing times of the jobs depend on their start time and the preceding job, the parameters required as input for the modelling of categories 3 and 4 are given below. Let

- t_{ijk} be the processing time of job i when it follows on job j and its implementation starts during period k,
- p_{ijk} be the profit of job i when it follows on job j and its implementation starts during period k,
- h_i be the fraction of the machine that job i requires for the full period of implementation, and
- r_i be the relative weight representing the relative importance of job i.

Also, let

$$v_{ijk} = \begin{cases} 1, & \text{if } k + t_{ijk} - 1 > w \\ 0, & \text{otherwise.} \end{cases} \tag{3.16}$$

That is, v_{ijk} is set to 1 if starting project i during period k when it follows job j will continue after the permitted last period of the schedule or set to 0 if this is not the case.

The sets of variables are given by

$$x_{ijk} = \begin{cases} 1, & \text{if the running of job } i \text{ follows on job } j \text{ and starts during period } k \\ 0, & \text{otherwise} \end{cases} \tag{3.17}$$

and

$$b_{ijk} = \begin{cases} 1, & \text{if job } i \text{ follows on job } j \text{ and is implemented during period } k \\ 0, & \text{otherwise.} \end{cases} \tag{3.18}$$

Suppose job 0, a dummy job with a profit of 0, a duration of 1 as well as no setup time and setup cost, starts during period 0. Then

1. $x_{0,0,0} = 1$ and $b_{0,0,0} = 1$,
2. $x_{0,0,k} = 0$ and $b_{0,0,k} = 0$ for $k = 1, \ldots, w$,
3. $x_{0,j,k} = 0$ and $b_{0,j,k} = 0$ for $j = 1, \ldots, n, k = 0, \ldots, w$,
4. $x_{i,0,0} = 0$ and $b_{i,0,0} = 0$ for $i = 1, \ldots, n$,
5. $x_{i,j,0} = 0$ and $b_{i,j,0} = 0$ for $i = 1, \ldots, n, j = 1, \ldots, n$,
6. $x_{i,i,k} = 0$ and $b_{i,i,k} = 0$ for $i = 1, \ldots, n, k = 0, \ldots, w$ and
7. $x_{i,j,1} = 0$ and $b_{i,j,1} = 0$ for $i = 0, \ldots, n, j = 1, \ldots, n$.

Explanations for the assignments listed in the order mentioned above are that

1. the dummy job follows itself and starts in period 0 (this forces the dummy job to start in period 0 and finish processing at the end of this period),
2. the dummy job, following itself, may not start in any of the other periods in the schedule,

3. the dummy job may not follow any of the available jobs in the dummy period and any of the periods in the schedule,

4. any of the available jobs may not follow the dummy job and start in the dummy period,

5. any of the available jobs may not follow another available job and start in the dummy period,

6. a job cannot follow itself and start in any of the periods and

7. an available job cannot follow another available job and start in the first period (this ensures that the first job(s) to be processed in the schedule will follow the dummy job).

All the assumptions mentioned in §3.1.1 need to hold in addition to the assumptions listed below.

1. The setup times of the jobs need to have non-negative integer values.[1]

2. The setup time is added to the normal processing time of a job and the setup cost is subtracted from the profit of a job.

3. The machine is in an initial setup state before the first job in the schedule is processed, *i.e.* no setup is required for the first job and therefore the setup time and setup cost for this job are both set to zero.

The objective is to

$$\text{maximise} \quad \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{k=1}^{w} r_i p_{ijk} x_{ijk} \tag{3.19}$$

subject to

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{k=1}^{w} x_{ijk} \leq 1 \qquad i = 1, \ldots, n, \tag{3.20}$$

$$\sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij1} \geq 1 \tag{3.21}$$

$$\sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} h_i b_{ijk} \leq 1 \qquad k = 1, \ldots, w, \tag{3.22}$$

$$\sum_{q=k}^{k+t_{ijk}-1} b_{ijq} \geq t_{ijk} x_{ijk} \qquad i = 1, \ldots, n,\ j = 1, \ldots, n,\ k = 1, \ldots, w, \tag{3.23}$$

$$(1 - w) y_{ijk} + x_{ijk} \leq 0 \qquad i = 1, \ldots, n,\ j = 1, \ldots, n,\ k = 1, \ldots, w, \tag{3.24}$$

$$(w - 1) y_{ijk} + t_{ijk} \leq 2w - k \qquad i = 1, \ldots, n,\ j = 1, \ldots, n,\ k = 1, \ldots, w, \tag{3.25}$$

[1] It is possible for a job not to have a setup time. In this case the time to prepare the machine before the processing of the job is zero.

$$\sum_{\substack{i=1\\i\neq j}}^{n} x_{ijk} \leq nq''_{jk} \qquad j = 1,\ldots,n,\ k = 2,\ldots,w, \tag{3.26}$$

$$\sum_{\substack{i=1\\i\neq j}}^{n}\sum_{l=1}^{k-1} x_{jil} \geq q''_{jk} \qquad j = 1,\ldots,n,\ k = 2,\ldots,w, \tag{3.27}$$

$$x_{ijk} \leq nwq'''_{j\ell_{ijk}} \qquad i = 1,\ldots,n,\ j = 1,\ldots,n,\ k = 1,\ldots,w,\ v_{ijk} = 0, \tag{3.28}$$

$$\sum_{\substack{i=1\\i\neq j}}^{n}\sum_{l=k+1}^{w} x_{ijl} \leq nw(1 - q'''_{jk}) \qquad j = 1,\ldots,n,\ k = 1,\ldots,w-1, \tag{3.29}$$

$$\sum_{i=1}^{n}\sum_{\substack{i=1\\i\neq j}}^{n}\sum_{k=1}^{w}(1 - v_{ijk})x_{ijk} = 0, \tag{3.30}$$

$$x_{sjk} = 0 \qquad (s,j,k) \in \mathcal{S}_k, \tag{3.31}$$

$$\sum_{v\in\mathcal{H}_i}\sum_{j=1}^{n}\sum_{k=1}^{w} x_{vjk} \leq 1 \qquad i = 1,\ldots,n, \tag{3.32}$$

$$b_{ijk}, x_{ijk} \in \{0,1\} \qquad i = 0,\ldots,n,\ j = 0,\ldots,n,\ k = 0,\ldots,w, \tag{3.33}$$

$$y_{ijk} \in \{0,1\} \qquad i = 1,\ldots,n,\ j = 1,\ldots,n,\ k = 1,\ldots,w, \tag{3.34}$$

$$q''_{jk} \in \{0,1\} \qquad j = 1,\ldots,n,\ k = 2,\ldots,w, \tag{3.35}$$

$$q'''_{jk} \in \{0,1\} \qquad j = 1,\ldots,n,\ k = 1,\ldots,w, \text{ and} \tag{3.36}$$

$$t_{ijk} \in \mathbb{N}^+ \qquad i = 1,\ldots,n,\ j = 1,\ldots,n,\ k = 1,\ldots,w. \tag{3.37}$$

Again, objective function (3.19) maximises the total weighted profit depending on the relative importance of each job, similar to objective function (3.3) found in §3.1.1. Constraint set (3.20) states that a job is allowed to start at most once while constraint (3.21) ensures that at least one job starts in the first period. Constraint set (3.22) keeps the processing of the jobs within the capacity of the machine. Constraint set (3.23) ensures that jobs are not interrupted during their processing.

Constraint sets (3.26) and (3.27) declare that job j should have started before time period k if job i follows on job j and starts implementation during time period k. The dummy variable q''_{jk} is binary indicating that job j is succeeded by other jobs that start during period k. Constraint sets (3.28) and (3.29) ensure that jobs succeed each other in a chronological order where

$$\ell_{ijk} = \begin{cases} k + t_{ijk} - 1, & \text{if } k + t_{ijk} - 1 \leq w \\ w, & \text{otherwise.} \end{cases} \tag{3.38}$$

The constraint sets (3.28) and (3.29) consist of if-then constraints stating that if job i follows job j and starts during period k, then job j cannot be followed by any other job after period k. Therefore, q'''_{jk} is set to 1 if job j precedes other jobs when it finishes during period k and if it can be completed before the end of the schedule. Constraint (3.30) dictates that jobs cannot continue processing after the permitted last period of the schedule, that is they can only be implemented if they can be completed before the end of the schedule. Constraint set (3.31) is used to prohibit job s from following job j and starting during time period k for all the tuples found in the set $\mathcal{S}_k$ while constraint set (3.32) guarantees that not more than one element from the set of derived jobs $\mathcal{H}_i$ can be chosen for the schedule. Finally, the variable sets (3.33)–(3.36) contain all the binary variables and the variable set (3.37) states that the processing time of

	Categories 1 & 2	Categories 3 & 4
Parameters	$2n(w+1)$	$3(n+1)^2(w+1)+2(n+1)$
Variables	$3nw$	$3(n+1)^2(w+1)+n(w-1)+nw$
Constraints	$3nw+n+w+1+\lvert\mathcal{S}_k\rvert+\lvert\mathcal{H}\rvert$	$5n^2w+3n(w-1)+n+w+2+\lvert\mathcal{S}_k\rvert+\lvert\mathcal{H}\rvert$

Table 3.1: *The order of magnitude for the number of parameters, variables and constraints for categories 1 and 2 (when a job's processing time depends only on its start time) as well as categories 3 and 4 (when a job's processing time depends on its start time and the preceding job).*

a job must be a positive integer. Table 3.1 contains the order of magnitude for the number of parameters, variables and constraints for the exact formulations, where $|\mathcal{S}_k|$ is the number of job-period pairs (or job-job-period triplets in categories 3 and 4) in $\mathcal{S}_k$ and $|\mathcal{H}|$ is the number of sets of derived jobs. It is evident from this table that the complexity of the problem increases as the number of jobs and the number of periods increase.

3.2 Preliminary theory for metaheuristics

In order to simplify the models presented in §3.1 for the use of metaheuristics, jobs that are processed simultaneously are placed into batches. Jobs that are processed in the same batch have the same start time [81] and the completion time of the batch is equal to the completion time of the job in the batch with the longest processing time. Once the job in the batch with the longest processing time is completed, only then can the following batch commence processing. When jobs are not allowed to be processed in parallel with one another, each batch contains one job so that the job with the longest processing time in the batch is the only job in the batch for all of the batches in the schedule. A "batch" refers to when only one job is running or when more than one job is being processed simultaneously. The term is therefore used for both instances when jobs are allowed and not allowed to run in parallel covering all four of the categories mentioned in §3.1.

The fact that all the jobs in a batch need to start at the same time may be restated by saying if job i started processing in period k, then all other jobs that are being processed in that period must have also started processing in period k. Thus, if $x_{ik}=1$, then

$$\sum_{q=1}^{n} x_{qk} \geq \sum_{q=1}^{n} b_{qk} \qquad i=1,\ldots,n,\ k=1,\ldots,w. \tag{3.39}$$

This results in the constraint sets

$$\sum_{q=1}^{n} b_{qk} - \sum_{q=1}^{n} x_{qk} \leq n z_{ik} \qquad i=1,\ldots,n,\ k=1,\ldots,w,\ \text{and} \tag{3.40}$$

$$x_{ik} \leq n(1-z_{ik}) \qquad i=1,\ldots,n,\ k=1,\ldots,w, \tag{3.41}$$

with z_{ik} a binary variable for each of the constraints, that need to be added to the model presented in §3.1.1. Similarly, the constraint sets

$$\sum_{q=1}^{n} b_{qjk} - \sum_{q=1}^{n} x_{qjk} \leq n z_{ijk} \qquad i=1,\ldots,n,\ j=1,\ldots,n,\ k=1,\ldots,w,\ \text{and} \tag{3.42}$$

$$x_{ijk} \leq n(1-z_{ijk}) \qquad i=1,\ldots,nj=1,\ldots,n,\ ,\ k=1,\ldots,w, \tag{3.43}$$

with z_{ijk} a binary variable for each of the constraints, need to be added to the model presented in §3.1.2. The metaheuristics presented in §3.3–3.7 thus determine the jobs that should be placed

in the same batch, given that all of the jobs in the same batch start at the same time, and the sequence in which these batches should be placed in the schedule. The best schedule provided by a given metaheuristic may, however, not be optimal for the models in §3.1 due to the addition of constraint sets (3.40)–(3.43). For that reason, the second phase in solving the model is to optimise every batch without changing the processing time of the batch. This is achieved by moving the jobs around within a batch so that the jobs in the batch do not necessarily all start at the same time, effectively ignoring constraint sets (3.40)–(3.43).

Once jobs are moved around, space may become available in the batch. The second phase thus simultaneously tries to insert unscheduled jobs, *i.e.* jobs that are in none of the batches in the schedule, in an attempt to increase the profit. It should be noted that when a critical job, that is a job that finishes last in the batch, is moved to another position while keeping all other jobs at their current position in the batch, there is a possibility of the processing time of the batch decreasing while its profit increases. This is allowed, since this new solution will dominate the previous solution due to an increase in the total profit. This will result in an idle time within the batch. Idle time within a batch cannot be removed, since removing them will result in all of the jobs starting directly after the idle time in the schedule being shifted earlier in the schedule. This can cause the processing time of certain jobs to increase and subsequently result in these jobs overlapping other jobs causing the schedule to be infeasible. Due to the considerably small probability that jobs will not overlap when idle time is removed, idle time will not be removed in order to maintain efficiency of the algorithm.

A solution for each of the algorithms presented in this chapter is defined by the matrix

$$\Phi_i = \begin{bmatrix} \varphi_{i11} & \varphi_{i12} & \cdots & \varphi_{i1w} \\ \varphi_{i21} & \varphi_{i22} & \cdots & \varphi_{i2w} \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_{in1} & \varphi_{in2} & \cdots & \varphi_{inw} \end{bmatrix} \tag{3.44}$$

with

- i the index of the solution in the set of solutions for each iteration,
- n the total number of jobs, including the derived jobs,
- w the number of time periods over which the schedule can run.

Each element φ_{ijk} in the solution Φ_i is set to 1 if job j starts implementation during time period k and 0 if job j does not start processing in time period k. The objective function is therefore given by

$$f(\Phi_i) = \sum_{j=1}^{n} \sum_{k=1}^{w} r_j p_{jk} \varphi_{ijk}. \tag{3.45}$$

To allow for a more general case so that the same algorithm can be applied to all four categories, let

$$S = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1n} \\ s_{21} & s_{22} & \cdots & s_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n1} & s_{n2} & \cdots & s_{nn} \end{bmatrix} \tag{3.46}$$

and

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} \tag{3.47}$$

be the setup time matrix and the setup cost matrix, respectively, with s_{ij} the setup time for job i if it follows on job j and c_{ij} the cost associated with setting up the machine for job i if job j precedes it. If each job's processing time depends only on its start time and not the job that precedes it, both S and C are zero matrices. If more jobs can run in parallel, more rows and columns are added to S and C for each of the derived jobs in $\mathcal{H}_i$ for each unique job i. The values in the additional rows and columns representing the derived jobs are the same as the values in the original rows and columns of its job having an h_i value of 1 [87]. Once a solution is determined in each iteration of an algorithm, the setup times in S are added to the normal processing times of the jobs in the schedule and the setup costs in C are subtracted from the profits. For categories 3 and 4, jobs are classified into different classes with the jobs in each class requiring the same setup configurations [87]. For each of the jobs in a class, there is a setup time and setup cost, possibly zero for both, involved when following any of the jobs in another class. These setup times and setup costs are the same for all pairs of jobs between the two different classes.

Initial solutions for each of the algorithms are determined by not violating any of the constraints in constraint sets (3.4)–(3.11) when the processing time of each job depends on its start time only and constraints sets (3.20)–(3.32) when the processing time of each job depends on its start time and the job that preceded it. In other words, a solution is feasible when all of the following conditions hold.

1. At most one job from the set of derived jobs is present in the sequence.
2. The sum of the h_i values of the jobs in each of the batches is less than or equal to 1. The sum will clearly be equal to 1 for all the batches when jobs are not allowed to run in parallel, since the h_i value for each of the jobs is set to 1.
3. The duration of the schedule does not exceed the maximum number of periods that the schedule runs over.
4. None of the jobs follow a job that it is forbidden to follow in the time period that it starts processing.
5. All of the jobs are not forbidden to be in the same batch as the other jobs in its batch.

For each of the iterations in the algorithms, a solution is accepted if it is feasible. Infeasible solutions are removed from the population so that chosen solutions for the following iteration is guaranteed to be a feasible solution.

3.3 Variable neighbourhood search

Variable neighbourhood search (VNS) was first suggested by Mladenović & Hansen [73] in 1997. They noted that local search methods have the disadvantage of getting stuck in local optima if a single neighbourhood is used during the search. VNS, a simple and effective metaheuristic, changes neighbourhoods randomly during each iteration to reach distant solutions that a local search method with a single neighbourhood would take too long to reach. When a single objective is to be optimised, VNS jumps from one solution to another if and only if it is an improvement. On the contrary, when multiple objectives are optimised, VNS inserts a solution into a set of nondominated solutions if and only if it is not dominated by any other solution in the set. In the next iteration a random solution is then selected from the set of nondominated solutions after which a random neighbourhood is explored of the selected solution. VNS terminates after

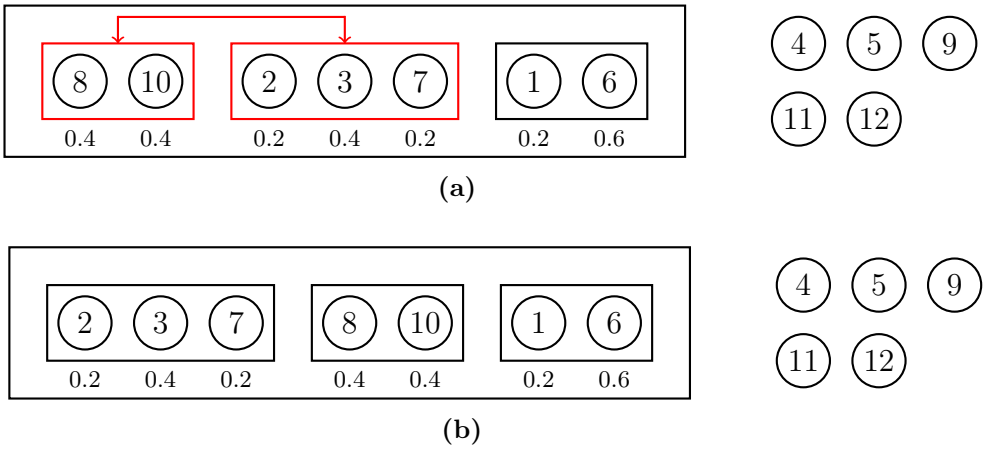


Figure 3.1: *(a) A schematic representation showing the swapping of two batches in the schedule. The schedule containing three batches with five unscheduled jobs to the right of the diagram before the move was made. The red arrow indicates the swapping of the first and second batch. (b) This is the resulting schedule after the first and second batch have been swapped.*

a specified number of iterations, if an improvement could not be found after a specified number of iterations or when a satisfactory solution has been found in terms of its objective function value(s).

3.3.1 Batch moves

Eight different moves are used to move from one solution to another within VNS for the considered scheduling problem. The first move swaps two batches, both of which are in the current schedule. It is assumed that the schedule contains at least two batches. This move is performed by removing both batches followed by inserting the first batch into the position of the second batch and inserting the second batch into the position of the first batch. The duration of the schedule may increase or decrease once the batches have been swapped due to the processing time of each job in the batches depending on their start time, including the batches that were not swapped[2]. Figure 3.1 illustrates the swapping of two batches within the schedule. It is evident from Figure 3.1(a) that the first batch containing jobs 8 and 10 is swapped with the second batch containing jobs 2, 3 and 7. The resulting schedule may be found in Figure 3.1(b) indicating the new positions of the two batches in the sequence. Also, jobs 4, 5, 9, 11 and 12 remain outside the schedule as they are still unscheduled jobs. This move only swaps two batches within a schedule.

The second move forms a batch from the unscheduled jobs, removes a batch from the schedule and inserts the formed batch into the schedule, not necessarily at the position that the batch was removed from. A batch is formed by inserting jobs with their corresponding h_i values while ensuring that $\sum h_i \leq 1$ for the jobs in the batch. This newly formed batch may possibly contain only one job. If this is the case and the job's h_i value is strictly less than 1, then its value will be switched to 1 to ensure that the job utilises the available space on the machine to decrease its processing time and increase its profit. Figure 3.2 shows the steps of this move. The first step involves the forming of a batch containing two unscheduled jobs 4 and 12 while ensuring

[2]If, for example, the first batch is swapped with the last batch and the new first batch now has a shorter processing time, then all the batches between the two batches that have been swapped will possibly be shifted to the left. Some of these batches may possibly not be shifted as a result of the complexities of time-dependent processing times.

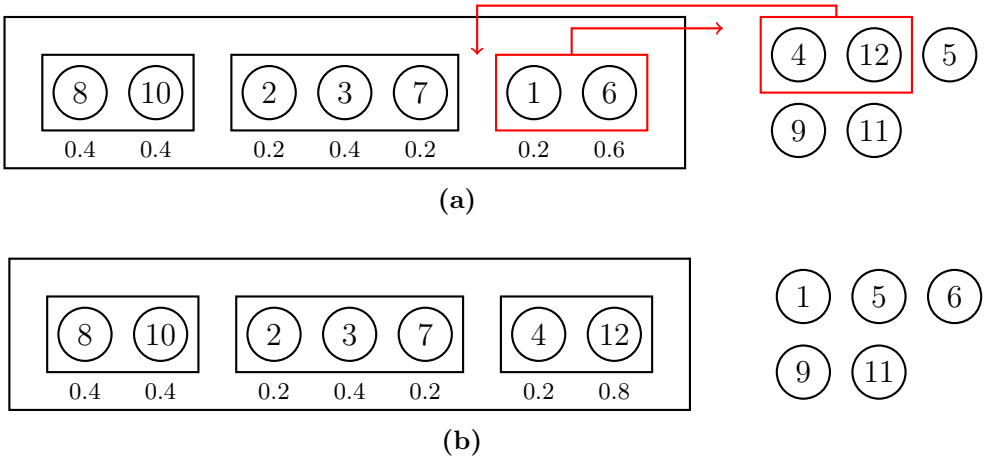


Figure 3.2: *(a) A schematic representation showing the removing of a batch from the schedule and the inserting of a batch containing unscheduled jobs. The schedule contains three batches with five unscheduled jobs to the right of the diagram before the move was made. The red arrows indicate the removal of the first batch and the insertion of the newly formed batch containing jobs 4 and 12 into the last position in the schedule. (b) This is the resulting schedule after the removal and insertion of the batches. The new five unscheduled batches may been seen to the right of the diagram.*

that the sum of their h_i values is less than or equal to 1, as seen in Figure 3.2(a). The next step removes the batch containing jobs 1 and 6 from the schedule. Finally, the newly formed batch containing jobs 4 and 12 is inserted into the third position in the schedule. Figure 3.2(b) shows the schedule after the move was done. Jobs 1 and 6 are now part of the unscheduled jobs while jobs 4 and 12 are in the schedule.

The third move removes a batch from the schedule and places it at another position in the sequence. Similar to that of the first move, it is assumed that there are at least two batches in the schedule. It should again be noted that the duration of the schedule may change due to the processing time of the jobs being dependent on when they start. Moving a batch from one position to another shifts other batches to different positions, thus causing their processing times to possibly change. Figure 3.3 demonstrates how a batch is moved from one position to another. The final batch in the sequence containing jobs 1 and 6 is moved to the front of the schedule as seen in Figure 3.3(a). This move caused the other two batches in the sequence to shift one position later in the sequence, displayed in Figure 3.3(b). Also, the set of unscheduled jobs remains the same as before the move was made.

The final move that can be made on a batch level is to form a batch from the unscheduled jobs in the same way as in the second move and inserting this batch into the schedule. Figure 3.4(a) shows how a batch is formed from the unscheduled jobs 9 and 11, and inserted into the third position in the sequence shifting the last batch one position later in the sequence. The resulting schedule is displayed in Figure 3.4(b) with the newly formed batch in its position. Jobs 4, 5 and 12 are now the only unscheduled jobs. This move is useful when there is space left in the schedule to insert a batch into the sequence without removing another batch, as opposed to the second move that also removes a batch.

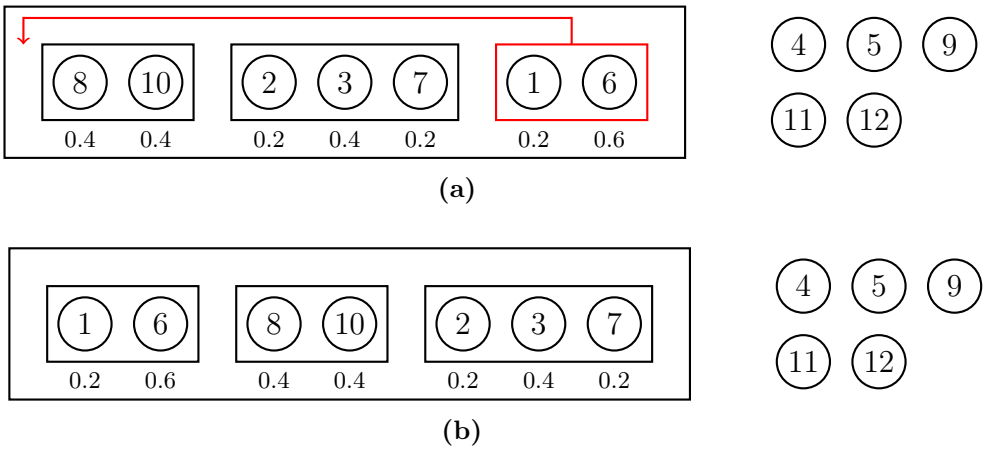


Figure 3.3: *(a) A schematic representation showing the moving of a batch from one position to another in the schedule. The schedule contains three batches with five unscheduled jobs to the right of the diagram before the move was made. The red arrow indicates the moving of the last batch in the schedule to the first position. (b) This is the resulting schedule after the last batch has been moved to the front of the sequence.*

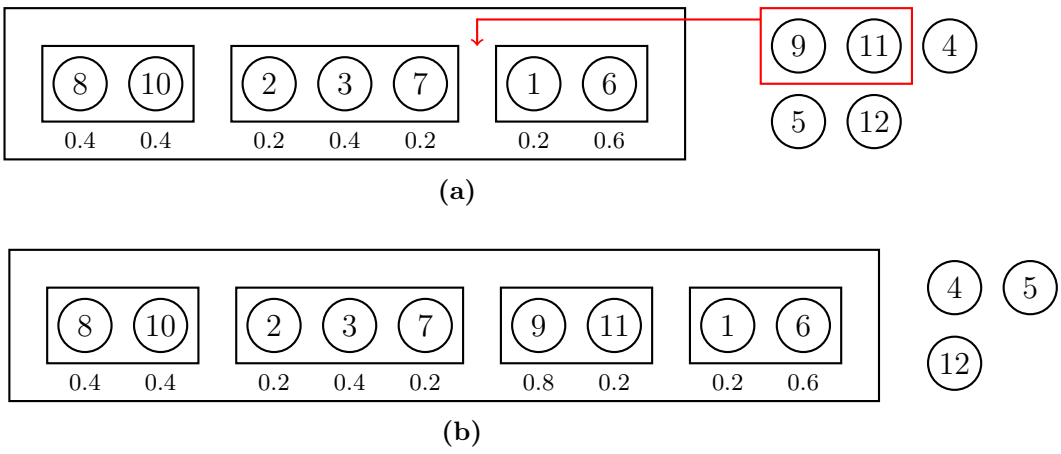


Figure 3.4: *(a) A schematic representation showing the inserting of a batch into the schedule. The schedule contains three batches with five unscheduled jobs to the right of the diagram before the move was made. The red arrow indicates the insertion of a newly formed batch, containing unscheduled jobs 9 and 11, into the third position in the sequence. (b) This is the resulting schedule after the newly formed batch has been inserted into the sequence, now containing a total of four batches. The remaining three unscheduled jobs may be seen to the right of the diagram.*

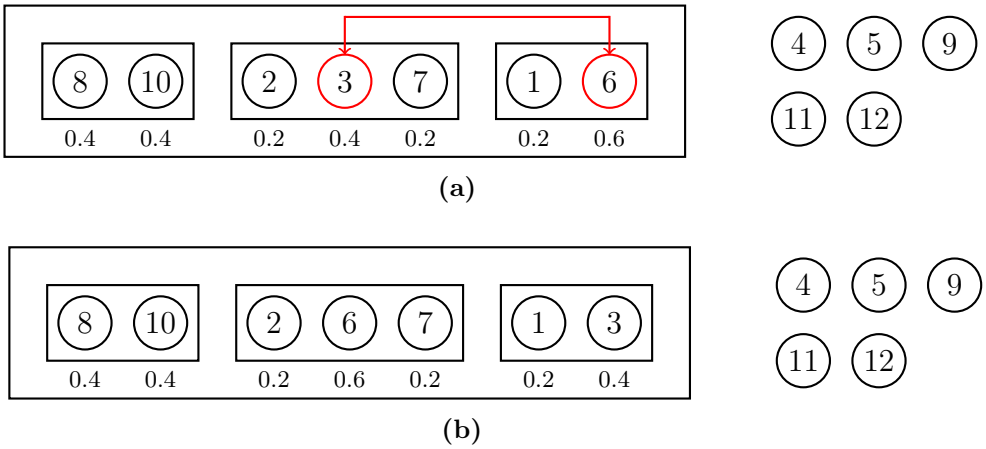


Figure 3.5: *(a) A schematic representation showing the swapping of jobs from two different batches in the schedule. The schedule contains three batches with five unscheduled jobs to the right of the diagram before the move was made. The red arrow indicates the swapping of a job in the second batch with a job in the third batch. (b) This is the resulting schedule after the jobs have been swapped.*

3.3.2 Job moves

The first of the moves on a job level swaps two jobs from two different batches in the sequence. In order to do this move, at least one of the two batches that are selected must contain at least two jobs, otherwise the move is equivalent to the first one mentioned in §3.3.1. Consequently, if all of the batches contain only one job, the move cannot be performed. This is to avoid the search being biased in favour of swapping two batches in the sequence. Furthermore, a swap can be made if $\sum h_i \leq 1$ for both of the resulting batches. The swap is performed by removing jobs from two different batches. Each of the jobs are then tested to fit in each other's batch for different h_i values. If at least one of the two jobs' derived jobs do not fit in the batch from which the other job was removed from, then the move is cancelled. Figure 3.5 illustrates how the move is performed. From Figure 3.5(a) it may be seen that jobs 3 and 6 are selected to be swapped, each of them being from different batches. They are both removed from their original batches. Job 6 is then tested to see if it fits in the second batch. The highest h_i value that its derived job can have is 0.6 to ensure that the resulting batch will have $\sum h_i \leq 1$. Its derived job having an h_i value of 0.6 is randomly selected and inserted into the third batch. The highest possible h_i value is not necessarily the best as future moves will be restricted by the jobs containing high h_i values in the batch. Job 3 is then tested for the third batch in the same way. The highest h_i value that its derived job can have is 0.8. The derived job with an h_i value of 0.6 is randomly selected to be inserted into the batch leaving space for another job with an h_i value of 0.2.

The second of the job level moves involves removing a job from a batch and inserting an unscheduled job into a batch. A job is removed from a batch containing at least two jobs, since removing a job from a batch containing only that one job is equivalent to removing the batch which may be found in the second move. An unscheduled job that can run in parallel is then selected to be inserted (if it fits) into one of the batches in the schedule. If none of the job's derived jobs fit in any of the batches in the schedule, then the batches containing one job is altered by changing their h_i values. The derived jobs are then randomly inserted into one of the altered batches. If this is still not possible, then the move is cancelled. The process may be found in Figure 3.6. Figure 3.6(a) shows how job 2 is removed from the second batch and moved to the set of unscheduled jobs. Job 5 is then randomly selected to be inserted into one of the batches (if it fits). In this case, its derived job having an h_i value of 0.2 fits and is inserted

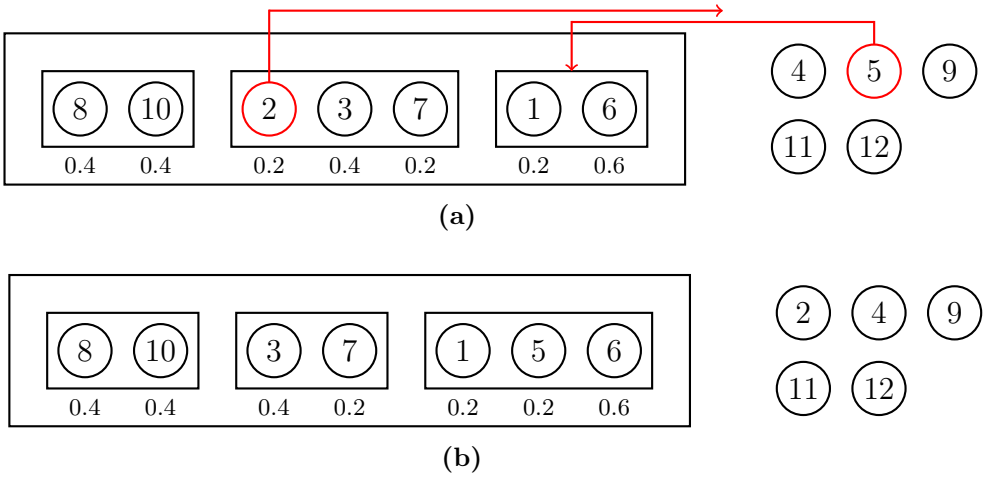


Figure 3.6: *(a) A schematic representation showing the removing of a job from the schedule and inserting an unscheduled job. The schedule contains three batches with five unscheduled jobs to the right of the diagram before the move was made. The red arrows indicate the removal of the a job in the second batch and the insertion of an unscheduled job into the last batch in the sequence. (b) This is the resulting schedule after the job has been removed and the unscheduled job was inserted into a batch in the sequence.*

into the last batch. Figure 3.6(b) displays the newly developed schedule with the new set of unscheduled jobs.

The third of the job level moves removes a job from a batch in the sequence and places it in another batch in the sequence, if possible. If the selected job is the only one in the batch, then the empty batch is removed once the job has been removed. Once again, the job can only be inserted into a batch when $\sum h_i \leq 1$ holds for the newly formed batch. If none of the job's derived jobs fit in any of the batches, then the batches containing one job are altered similar to that of the previous move. Figure 3.7 contains an illustration of how the move is executed. Job 1 is selected to be removed from the last batch, evident from Figure 3.7(a), to be inserted into the first batch. Since one of its derived jobs, the job having an h_i value of 0.2, fits in the batch, it is inserted. The resulting schedule in Figure 3.7(b) follows after the move has been made. The h_i value of job 6 in the last job is changed from 0.6 to 1 to utilise the available space in the machine.

Finally, the last move on a job level involves inserting an unscheduled job into a batch. An unscheduled job that can run in parallel is selected to be inserted into one of the batches in the schedule if it fits. If none of the job's derived jobs fit in any of the batches in the schedule, then the batches containing one job are altered by changing the h_i values, similar to that of the second move of the job level moves. The move is cancelled if the job can still not be inserted into one of the batches. This process is demonstrated in Figure 3.8. Figure 3.8(a) shows how job 5, an unscheduled job, is randomly selected and attempted to be inserted into one of the batches. In this case, its derived job having an h_i value of 0.2 is inserted into the first batch. Figure 3.8(b) exhibits the newly formed schedule with the new set of unscheduled jobs.

3.3.3 Neighbourhoods

The eight defined moves in §3.3.1 and §3.3.2 can now be used to form neighbourhoods for a given solution. The neighbourhoods that can be formed using these moves, may be found in the following list in the order that the moves were mentioned in these sections.

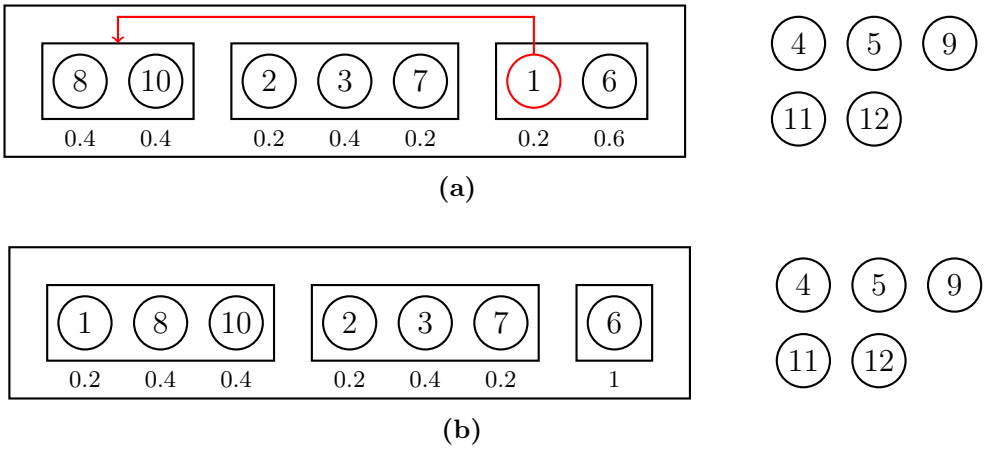


Figure 3.7: *(a) A schematic representation showing the moving of a job from one batch to another in the schedule. The schedule contains three batches with five unscheduled jobs to the right of the diagram before the move was made. The red arrow indicates the moving of a job from the third batch to the first batch in the sequence. (b) This is the resulting schedule after the job has been moved.*

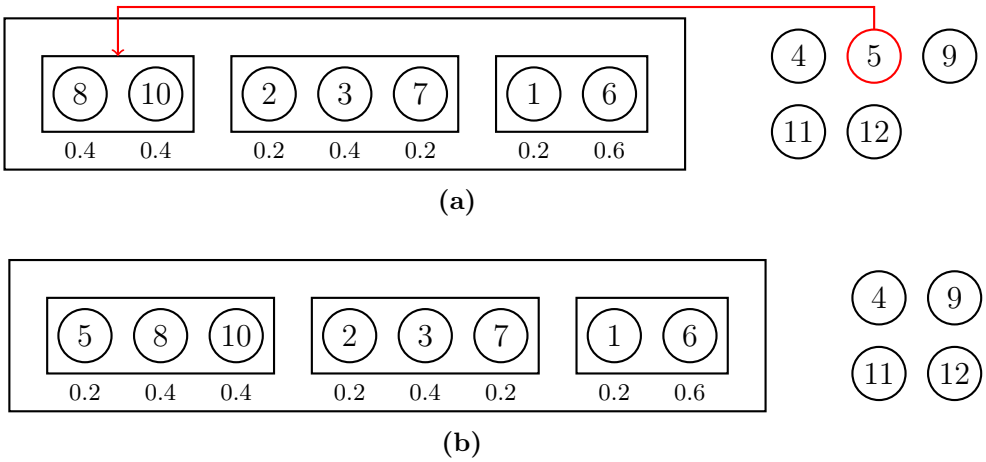


Figure 3.8: *(a) A schematic representation showing the inserting of a job into the schedule. The schedule contains three batches with five unscheduled jobs to the right of the diagram before the move was made. The red arrow indicates the inserting of the an unscheduled job into the first batch in the sequence. (b) This is the resulting schedule after the unscheduled job has been inserted into a batch in the sequence. The remaining four unscheduled jobs may be seen to the right of the diagram.*

1. Every pair of batches in the original sequence is swapped. This neighbourhood contains $\binom{\mu}{2}$ neighbours with μ the number of batches in the sequence.

2. All the pairs of batches that can be removed together with the batches that can be formed from the unscheduled jobs forms the neighbourhood. The size of this neighbourhood is $\mu^2(\nu+\xi)$ where ν is the number of unscheduled jobs and ξ is the number of batches that can be formed that contain two or more jobs. The size of the neighbourhood depends on the h_i values for each of the unscheduled jobs' derived jobs, that is the number of batches that can be formed so that $\sum h_i \leq 1$ for each of the batches.

3. Every batch is removed and inserted into another position in the sequence. There is a total of $(\mu-1)^2$ neighbours in this neighbourhood.

4. Every possible batch that can be formed from the unscheduled jobs is inserted into all the possible positions in the sequence to obtain the neighbourhood. The size of the neighbourhood is $(\mu+1)(\nu+\xi)$ as in this case there are $\mu+1$ possible positions in the sequence that the batch can be inserted into.

5. Every possible pair of jobs that can be swapped between two batches in the sequence forms this neighbourhood. The size of the neighbourhood depends on the h_i values of the jobs' derived jobs and the value of $\sum h_i$ for each of the batches in the sequence. A solution forms part of the neighbourhood only if its resulting sequence has all batches satisfying the constraint $\sum h_i \leq 1$. The size of the neighbourhood is in the range $[0,\ \sum_{i=1}^{\mu}\sum_{j>i}^{\mu}|\bar{\mathcal{J}}_i||\bar{\mathcal{J}}_j|]$ where $\bar{\mathcal{J}}_1, \bar{\mathcal{J}}_2, \ldots, \bar{\mathcal{J}}_\mu$ are the sets of derived jobs in the batches (these are the derived jobs for all the jobs in a batch).

6. Every possible job is removed from the sequence together with every possible unscheduled job that is attempted to be inserted into the batches in the sequence. The size of the neighbourhood is in the range $[0,\ \rho(\sum_{i=1}^{\nu}|\bar{\mathcal{H}}_i|-\nu)]$ where $\bar{\mathcal{H}}_1, \bar{\mathcal{H}}_2, \ldots, \bar{\mathcal{H}}_\nu$ are the sets of derived jobs for the unscheduled jobs and ρ is the number of jobs in the sequence that are in batches containing two or more jobs.

7. Every possible job is removed from a batch in the sequence and inserted into another batch in the sequence. The size of the neighbourhood is in the range $[0,\ (\mu-1)(\sum_{i=1}^{n'-\nu}|\tilde{\mathcal{H}}_i| - n'+\nu)]$ where $\tilde{\mathcal{H}}_1, \tilde{\mathcal{H}}_2, \ldots, \tilde{\mathcal{H}}_{n'-\nu}$ are the sets of derived jobs for the scheduled jobs and n' is the total number of jobs considered to be scheduled, excluding the derived jobs.

8. Every possible unscheduled job is inserted into the batches in the sequence. The size of the neighbourhood is in the range $[0,\ \mu(\sum_{i=1}^{\nu}|\bar{\mathcal{H}}_i|-\nu)]$.

One of these neighbourhoods are randomly selected and applied to a randomly selected solution in the population during each iteration of VNS.

3.3.4 Structure of the algorithm

Algorithm 1 contains the pseudocode for the implementation of VNS. It requires the initial set of solutions $\bar{\mathcal{P}}$ that is added to the local archive upon execution of the algorithm, the computational time η at which the last iteration is performed as well as p_{min} and p_{max} which are the minimum and maximum number of periods that the schedule is allowed to be, respectively, as input. The initialisation procedure may be found in lines 1 and 2 that sets the local archive $\mathcal{A}_l$ to the set of initial solutions and an index i to 1 that is used to keep track in which iterations in the algorithm solutions were found. Random solutions are generated for the initial set of solutions

$\bar{\mathcal{P}}$ such that every solution in $\bar{\mathcal{P}}$ contains at least two batches and has a duration that is at least as long as p_{min}.

A random solution, call it Φ_i, is then selected from the local archive. The set of moves $\mathcal{M}$ around Φ_i are determined in lines 5–9 that depends on whether jobs are allowed to run in parallel or not. All of the eight mentioned moves and neighbourhoods found in Figures 3.1–3.8 are used when jobs are allowed to run in parallel while only the first four moves and neighbourhoods from Figures 3.1–3.4 are applied when jobs are not allowed to run in parallel. The order of the moves in $\mathcal{M}$ is shuffled. A random move is then performed around Φ_i to determine Φ_i' in the order of the moves in $\mathcal{M}$ until a move can be made, found in lines 11–18. A similar procedure is then applied for the neighbourhoods around Φ_i'. The set of neighbourhoods in $\mathcal{N}$ is shuffled so that the neighbourhood corresponding to the move that was made around Φ_i is first in $\mathcal{N}$. The neighbourhood $\mathcal{X}_i$ around Φ_i' is then determined in the order of the neighbourhoods in $\mathcal{N}$ until a move can be made, as seen in lines 25–32.

The feasibility procedure and adding of the solutions to the local archive may be found in lines 33–42. The feasibility procedure consists of three phases that need to be completed for each solution X_i in $\mathcal{X}_i$ in the order mentioned in the pseudocode.[3] The first step is to randomly remove one of the two jobs in X_i that are not allowed to be in the same batch for all the clashes until each job is allowed to be with all the other jobs in its respective batch. The next step is then to remove a job that is forbidden to follow another job in that time period or the job that is forbidden to precede another job in that time period. This is done until all jobs are allowed to follow the job that was processed before it in the time period that they start. The final step is to remove the last batch in X_i until the duration of the schedule does not exceed p_{max}. The resulting solution X_i is then added to $\mathcal{A}_l$ if its number of batches $\mu(X_i)$ is at least 2 and its duration $d(X_i)$ is at least as long as p_{min}. Finally, the duplicate solutions as well as all the solutions that are dominated by one or more other solutions are removed from $\mathcal{A}_l$ at the end of each iteration. All the nondominated solutions in the local archive, which now only consists of feasible solutions, are returned to the decision maker upon completion of the ν iterations with their corresponding objective function values to determine which trade-offs need to be made for a schedule to be implemented.

3.3.5 Nondominated solutions in local/external archive

Two functions are used to determine whether a solution is nondominated. The first function is the objective function (3.45) to be maximised, that is the maximisation of the total weighted profit $f_1(\Phi_i) = f(\Phi_i)$. The second function is the sum of the processing times of the jobs that are scheduled, which should be minimised. This function is given by

$$f_2(\Phi_i) = \sum_{j=1}^{n} \sum_{k=1}^{w} t_{jk} x_{jk}. \tag{3.48}$$

A solution Φ_1 dominates another solution Φ_2 if $f_1(\Phi_1) > f_1(\Phi_2)$ and $f_2(\Phi_1) \leq f_2(\Phi_2)$ or $f_1(\Phi_1) \geq f_1(\Phi_2)$ and $f_2(\Phi_1) < f_2(\Phi_2)$. Therefore, a solution Φ_2 is removed from the local/external archive if it is worse than Φ_1 in at least one function. In this way, solutions that have the same value for $f_1(\Phi_i)$ and $f_2(\Phi_i)$ are all kept in the local archive, since they can be considerably different schedules. A local and external archive[4] (in subsequent metaheuristics) are used for the following

[3] The solution X_i refers to the sequence of batches in the schedule while Φ_i refers to the schedule's binary matrix defined in §3.2 representing the time periods in which the scheduled jobs start.

[4] A local archive is an archive of solutions that can take part in the generating of solutions for the following iteration. An external archive simply stores solutions that have been found throughout the search. These solutions cannot be used to generate future solutions.

reasons throughout this thesis.

1. By minimising $f_2(\Phi_i)$ and maximising $f_1(\Phi_i)$ simultaneously, ensures that there is diversity in the population. This is contradictory to only having one function in VNS which will result in only one solution in the local archive at all times. This is a greedy approach which will lead VNS to be trapped in a local optimum. VNS will consequently not be able to search a wide range of regions in the solution space.

2. Multiple objectives can easily be added to a metaheuristic. The minimisation of the sum of the processing times of the jobs that are scheduled were used throughout the thesis so that the local/external archive also contains solutions in which less jobs run in parallel. This also contributes to diversity in the population so that the population consists of a good spread of schedules containing mostly parallel jobs, mostly nonparallel jobs and an approximately equal number of parallel and nonparallel jobs. Other objective functions (which may include or exclude the the sum of the processing times of the jobs that are scheduled) can also be used in future work, depending on the preference of the decision maker.

3. The schedule with the greatest total weighted profit may in some instances not be the most desirable schedule to be implemented by the decision maker. This is due to the possibility of external factors that can not be included in the exact formulations or the metaheuristics. It is therefore ideal to present a set of solutions so that the decision maker has a wide selection of possible schedules to be implemented and is not limited to only the solution with the greatest total weighted profit. For the purpose of comparing the metaheuristics against one another, the solution with the greatest total weighted profit is always selected in Chapter 4.

3.3.6 Size of the local archive

The size of the local archive $\mathcal{A}_l$ may be kept under a specified maximum size of a_{max} to prevent the algorithm from selecting solutions that are too similar to other solutions. This process may be performed directly after all the solutions in $\mathcal{A}_l$ that are dominated by one or more other solutions in $\mathcal{A}_l$, are removed. If the number of nondominated solutions in $\mathcal{A}_l$ is more than a_{max}, solutions are removed according to the formula

$$d(X_i, Y_i) = \sqrt{\left(\frac{f_1(X_i) - f_1(Y_i)}{f_1^{max} - f_1^{min}}\right)^2 + \left(\frac{f_2(X_i) - f_2(Y_i)}{f_2^{max} - f_2^{min}}\right)^2} \tag{3.49}$$

which is defined as the weighted distance between solutions X_i and Y_i with $X_i \neq Y_i$ where

f_j^{max} is the maximum value for $f_j(X_i)$ for all the solutions in $\mathcal{A}_l$ and
f_j^{min} is the minimum value for $f_j(X_i)$ for all the solutions in $\mathcal{A}_l$.

Let $\mathbf{d}_{X_i} = (d^1_{X_i}, d^2_{X_i}, \ldots, d^r_{X_i})$ be the vector containing the distances $d(X_i, Y_i)$ of the r nearest neighbours of solution X_i where $d^1_{X_i}$ is the distance of the nearest neighbour, $d^2_{X_i}$ is the distance of the second nearest neighbour and so on, then the total crowding distance of a solution X_i is given by

$$\bar{d}(X_i) = \sum_{j=1}^{r} d^j_{X_i}. \tag{3.50}$$

The size of the local archive is then reduced by removing the $|\mathcal{A}_l| - a_{max}$ solutions with the smallest value for $\bar{d}(X_i)$ with $|\mathcal{A}_l|$ being the size of $\mathcal{A}_l$ before any solutions are removed.

Algorithm 1: Variable neighbourhood search

Input: $\bar{\mathcal{P}}$, η, p_{min}, p_{max}
Output: A set of feasible solutions in the local archive found during the search

$\mathcal{A}_l \leftarrow \bar{\mathcal{P}}$;
$i \leftarrow 1$;
while *η seconds have not elapsed* **do**
  $\Phi_i \leftarrow$ a randomly selected solution from $\mathcal{A}_l$;
  if *jobs are allowed to run in parallel* **then**
    $\mathcal{M} \leftarrow$ the set of random moves 1–8 around Φ_i;
  else
    $\mathcal{M} \leftarrow$ the set of random moves 1–4 around Φ_i;
  end
  Shuffle the set of the random moves in $\mathcal{M}$;
  while *a random move cannot be made around Φ_i* **do**
    Select the first available random move from $\mathcal{M}$;
    if *a random move can be made around Φ_i* **then**
      $\Phi_i' \leftarrow$ the resulting solution after a random move has been made around Φ_i;
    else
      Remove the random move from $\mathcal{M}$;
    end
  end
  if *jobs are allowed to run in parallel* **then**
    $\mathcal{N} \leftarrow$ the set of neighbourhoods 1–8 around Φ_i';
  else
    $\mathcal{N} \leftarrow$ the set of neighbourhoods 1–4 around Φ_i';
  end
  Shuffle the set of the neighbourhoods in $\mathcal{N}$ such that the neighbourhood corresponding to the move that was made around Φ_i is first in the list;
  while *the size of the neighbourhood around Φ_i' is zero* **do**
    Select the first available neighbourhood from $\mathcal{N}$;
    if *the size of the neighbourhood around Φ_i' is greater than zero* **then**
      $\mathcal{X}_i \leftarrow$ the resulting set of solutions in the neighbourhood around Φ_i';
    else
      Remove the neighbourhood from $\mathcal{N}$;
    end
  end
  for *each solution X_i in $\mathcal{X}_i$* **do**
    Randomly remove one of the two jobs that are not allowed to be in the same batch for all the clashes until all jobs in X_i are allowed to be with all the other jobs in their respective batch;
    If a job is forbidden to follow another job and start in that period, then one of the jobs is removed;
    Remove the last batch in X_i until the duration of the schedule does not exceed p_{max};
    if *$\mu(X_i) \geq 2$ and $d(X_i) \geq p_{min}$* **then**
      $\mathcal{A}_l \leftarrow \mathcal{A}_l \cup X_i$;
    end
  end
  Remove the duplicate solutions from $\mathcal{A}_l$;
  Remove all solutions from $\mathcal{A}_l$ that are dominated by one or more other solutions in $\mathcal{A}_l$;
  $i \leftarrow i + 1$;
end
return $\mathcal{A}_l$;

3.4 Particle swarm optimisation

Particle swarm optimisation (PSO) was introduced by Eberhart & Kennedy [29], originally intended to simulate the movement of a flock of birds or a school of fish, that was later observed to perform optimisation when simplified. In their proceedings they applied PSO to nonlinear programming models after which they compared the results of systematic benchmark tests to other population-based algorithms, most notably the GA, as a result of its ties to evolution

strategies. They found that the use of PSO instead of the GA will improve population evolution due to the particles' collective intelligence. PSO has a memory that uses the best found positions of the particles to guide particles towards these solutions, a feature that the GA does not have.

3.4.1 General movement equations

PSO consists of a set of solutions, referred to as a population of particles, that move together in the solution space to promising regions. Each particle has a different position and velocity in each iteration that are used to determine its next position in the hyperspace by simple mathematical formulae. The algorithm is initialised by assigning random positions and velocities to the particles using a uniform distribution [19]. If we let $x_i^d(t)$ be particle i's position for dimension d at time t (or in this case iteration t), then the particle's initial position for dimension d is generated by the formula

$$x_i^d(0) = U(x_{min}, x_{max}) \tag{3.51}$$

where x_{min} and x_{max} are the minimum and maximum values for the dimensions of the initial position for all the particles, respectively. The initial velocity is then determined by the formula

$$v_i^d(0) = U(-v_{max}, v_{max}) \tag{3.52}$$

with v_{max} the maximum velocity in the positive and negative directions for all the particles [14] and $v_i^d(t)$ particle i's velocity for dimension d at time t. To determine the velocity and position for a particle during the search, the general movement equations are given by

$$v_i^d(t+1) = \omega v_i^d(t) + \varphi_p r_p(p_i^d(t) - x_i^d(t)) + \varphi_g r_g(g^d(t) - x_i^d(t)), \tag{3.53}$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1), \tag{3.54}$$

respectively, where

ω	is the inertia factor used to update the velocity of a particle,
φ_p	is the acceleration coefficient in the direction of the best known position of each particle from the previous time step,
$p_i^d(t)$	is particle i's best known position for dimension d at time t,
φ_g	is the acceleration coefficient in the direction of the best known position of all the particles from the previous time step and
$g^d(t)$	is the best known position of all particles for dimension d at time t.

Furthermore, r_p and r_g are two independent uniformly distributed random variables in the interval $[0, 1]$ that are generated for each individual dimension of a particle.

3.4.2 Decoding a particle's position

In order to apply PSO to scheduling problems, the position of a particle needs to be decoded into a sequence of batches containing jobs using an appropriate scheme. The random keys representation, suggested by Bean [5] for the use of machine scheduling, resource allocation and assignment problems, is utilised to decode the position of a particle into a batch sequence. The first step is to sort the keys of a particle's position in descending order. The number of dimensions of a particle's position and velocity is the number of jobs considered to be assigned to the schedule, excluding the derived jobs. The rank of the keys will then determine the order of the jobs' indices in a separate sequence, call it $\mathbf{y}_i$. The next step is to insert the jobs in $\mathbf{y}_i$ into batches in the order of their appearance.

A random h_i value is selected for each of the jobs out of the possible h_i values that they may have. An empty batch is initiated after which the first available job is inserted. The inserted job is then removed from $\mathbf{y}_i$. The first available job that fits into this batch, while ensuring that the condition $\sum h_i \leq 1$ holds, is then inserted. Once a job has been inserted it is removed from $\mathbf{y}_i$. More jobs are inserted into this batch until no jobs in $\mathbf{y}_i$ fits in the batch. This batch is then inserted into the sequence, call it X_i, that represents the order in which the batches will be processed on the machine in the schedule. A new batch is set up after which jobs in $\mathbf{y}_i$ are inserted into the batch. This process is repeated until there are no jobs in $\mathbf{y}_i$ and all the jobs are in batches in X_i. The feasibility procedure, consisting of three phases, mentioned in §3.3.4 is applied to the sequence X_i by removing jobs to make it feasible. Finally, the h_i value of jobs that are alone in a batch is changed to 1 to ensure that the jobs utilise the full capacity of the facility.

For example, suppose that the position of a particle is

$$\mathbf{x}_i(t) = (x_i^1(t), x_i^2(t), \ldots, x_i^{n'}(t)) \tag{3.55}$$

$$= (0.597, 0.156, 0.834, 0.966, 0.027, 0.487, 0.852, 0.313), \tag{3.56}$$

where n' is the number of dimensions of the particles, or more specifically the number of jobs considered to be assigned to the schedule which excludes the derived jobs. In this case $n' = 8$ as there are eight jobs considered to be assigned. The keys of $\mathbf{x}_i(t)$ are then sorted in descending order after which the indices of the jobs in $\mathbf{x}_i(t)$ are inserted into $\mathbf{y}_i$ to give

$$\mathbf{y}_i = (4, 7, 3, 1, 6, 8, 2, 5). \tag{3.57}$$

In other words, the highest value for the keys in $\mathbf{x}_i(t)$ is 0.966 which is at the 4^{th} position in the list, therefore the number 4 (representing the fourth job) is first inserted into $\mathbf{y}_i$. Similarly, the second highest value in $\mathbf{x}_i(t)$ is 0.852, which is at the 7^{th} position in the list, thus the number 7 appears second in $\mathbf{y}_i$. This is repeated until all the jobs are present in $\mathbf{y}_i$. Suppose the possible h_i values for the jobs are those given in Table 3.2. Jobs 1 and 7 have the freedom to occupy 25%, 50%, 75% or 100% of the facility while jobs 3, 5 and 8 are not allowed to run in parallel. Job 2 can occupy either 50% or 100%, job 4 either 25% or 100% and job 6 50%, 75% or 100% of the facility. Furthermore, suppose that the selected h_i values for the jobs in the order that they appear in $\mathbf{y}_i$ is given by the vector

$$\mathbf{h}_i = (0.25, 0.25, 1, 0.5, 0.5, 1, 0.5, 1) \tag{3.58}$$

The first empty batch is initiated after which the first job in $\mathbf{y}_i$, job 4, is inserted. Since job 4 requires 25% of the machine, 75% of the machine remains unused. Job 7, the next job that fits in the batch, is consequently inserted after which job 1 is inserted. The machine has reached full capacity and as a result, the batch is inserted into X_i. A new empty batch is then formed after which the first job of the remaining five jobs, that is job 3, is inserted. It requires the full machine, therefore the batch is inserted into X_i as the second batch to be processed. From the four remaining jobs, jobs 2 and 6 are inserted together in a batch after which jobs 8 then 5 are inserted into X_i in separate batches. The full schedule, at this point a possibly infeasible schedule, may be found in Figure 3.9. Jobs can now be removed from the schedule until a feasible solution is obtained.

The fitness value for particle X_i is then determined by the fitness function $z(\Phi_i) = f(\Phi_i)$. Multiple solutions, say a total of s solutions, are generated containing different $\mathbf{h}_i$ vectors. The sequence X_i and the fitness value $z(\Phi_i)$ is subsequently determined for each of the s solutions. The solution with the highest fitness value is then selected as the sequence corresponding to the particle's position.

Job	1	0.75	0.5	0.25
1	•	•	•	•
2	•		•	
3	•			
4	•			•
5	•			
6	•	•	•	
7	•	•	•	•
8	•			

Table 3.2: *An example of the possible h_i values for a pool of jobs that are considered to be assigned to a schedule. Jobs 1 and 7 have the freedom to occupy 25%, 50%, 75% or 100% of the facility while jobs 3, 5 and 8 are restricted by not being allowed to run in parallel. Job 2 can occupy either 50% or 100%, job 4 either 25% or 100% and job 6 50%, 75% or 100% of the facility.*

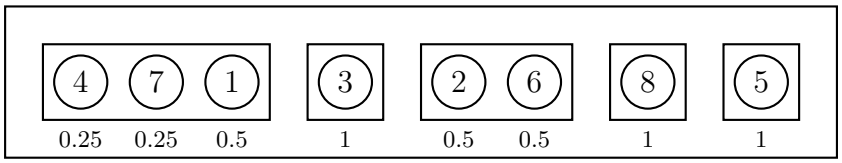


Figure 3.9: *An example of a schedule containing a sequence of batches after decoding the position of a particle. The schedule contains five batches with no unscheduled jobs. The values underneath the jobs in the batches indicate the fraction of the facility that the job occupies when its corresponding batch is being processed.*

3.4.3 Pseudocode for the algorithm

The pseudocode for PSO may be found in Algorithm 2. It requires

1. the number of particles in the population $|\mathcal{P}|$,
2. the computational time η at which the particles move for the last time,
3. the maximum number of periods p_{max} that the schedule is allowed to be,
4. the maximum initial velocity v_{max} in the positive and negative directions for all the particles which is chosen as 1 for the results in Chapter 4,
5. the inertia factor ω used to update the velocity of a particle,
6. the acceleration coefficients φ_p and φ_g in the direction of the best known position of each particle and the best known position of all particles from the previous step, respectively, and
7. the number of times s that $\mathbf{h}_i$ vectors are generated for a particle

as input to be executed. The initialisation process takes place in lines 1–20 that involves the generating of the initial best known sequences $\mathcal{P}$ of the particles as well as the best known position for each particle and the best known position of all the particles. The initial position and velocity are randomly generated for each of the particles in the set of positions $\mathcal{X}(0) = \{\mathbf{x}_1(0), \mathbf{x}_2(0), \ldots, \mathbf{x}_{|\mathcal{P}|}(0)\}$ for the particles at time 0 and the set of velocities $\mathcal{V}(0) = \{\mathbf{v}_1(0), \mathbf{v}_2(0), \ldots, \mathbf{v}_{|\mathcal{P}|}(0)\}$ for the particles at time 0, respectively, where $\mathbf{v}_i(0) = (v_i^1(0), v_i^2(0), \ldots, v_i^{n'}(0))$ is the velocity vector for particle i at time 0. The best known position

$\mathbf{p}_i(0) = (p_i^1(0), p_i^2(0), \ldots, p_i^{n'}(0))$ for particle i is then set to its initial position after which the $\mathbf{y}_i$ vector is determined for the particle. A set $\mathcal{S}$ that will contain the sequences and their corresponding value for $z(\Phi_i)$ determined by random $\mathbf{h}_i$ vectors, is initially set to an empty set. A total of s feasible sequences are then generated (as explained in §3.4.2) in lines 9–14 after which they are added to $\mathcal{S}$ with their corresponding value for $z(\Phi_i)$ in line 15. The sequence in $\mathcal{S}$ with the highest value for $z(\Phi_i)$ is added to $\mathcal{P}$ as the best known sequence for that particle. The final step in the initialisation process is to set the best known position $\mathbf{g}(0) = (g^1(0), g^2(0), \ldots, g^{n'}(0))$ to the solution in $\mathcal{P}$ with the highest value for $z(\Phi_i)$.

Lines 21–53 represent the movement of the particles with each particle moving the same number of times until η seconds have elapsed. The first step for each of the particles in the population is to update its velocity followed by its position. This is done by using random floating point numbers r_p and r_g to update the velocity for each dimension of the particle using equation (3.53) followed by the position for each dimension using equation (3.54). The $\mathbf{y}_i$ vector is determined for the particle's new position after which s sequences are generated as in lines 7–16. If the sequence in $\mathcal{S}$ with the highest value of $z(\Phi_i)$ has a higher value of $z(\Phi_i)$ than that of the best known position of the particle, then the best known position of the particle is updated to the particle's current position and the sequence replaces the particle's best found sequence in $\mathcal{P}$. The global best position found during the algorithm is updated in the same sense. That is, if the sequence in $\mathcal{S}$ with the highest value of $z(\Phi_i)$ has a higher value of $z(\Phi_i)$ than that of the best known position of all the particles, then the global best position is updated to the particle's current position. If any of these conditions don't hold, then the particle's current best position and the best found position of all the particles remain the same, as seen in lines 45 and 50. After the particles have moved the same number of times in the hyperspace, the particles' best positions are returned as output from the algorithm.

Algorithm 2: Particle swarm optimisation

Input: $|\mathcal{P}|$, η, p_{max}, v_{max}, ω, φ_p, φ_g, s
Output: A set of feasible solutions representing the particles' best found positions during the search

$\mathcal{P} \leftarrow \emptyset$;
for *each of the* $|\mathcal{P}|$ *particles* **do**
  $\mathbf{x}_i(0) \leftarrow$ randomly generated position for particle i with each dimension in the range $[0, 1]$;
  $\mathbf{v}_i(0) \leftarrow$ randomly generated velocity for particle i with each dimension in the range $[-v_{max}, v_{max}]$;
  $\mathbf{p}_i(0) \leftarrow \mathbf{x}_i(0)$;
  $\mathbf{y}_i \leftarrow$ the sequence in which the jobs are inserted into batches depending on their positions in $\mathbf{x}_i(0)$;
  $\mathcal{S} \leftarrow \emptyset$;
  for *the* s *times that random* h_i *values are determined for the jobs in* $\mathbf{y}_i$ **do**
    $\mathbf{h}_i \leftarrow$ random h_i values for the jobs in the order that they appear in $\mathbf{y}_i$;
    $X_i \leftarrow$ sequence containing the batches after all the jobs have been inserted;
    Randomly remove one of the two jobs that are not allowed to be in the same batch for all the clashes until all jobs in X_i are allowed to be with all the other jobs in their respective batch;
    If a job is forbidden to follow another job and start in that period, then one of the jobs is removed;
    Remove the last batch in X_i until the duration of the schedule does not exceed p_{max};
    $z(\Phi_i) \leftarrow f(\Phi_i)$;
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{X_i, z(\Phi_i)\}$;
  end
  $S_{max} \leftarrow$ the sequence in $\mathcal{S}$ with the highest value of $z(\Phi_i)$;
  $\mathcal{P} \leftarrow \mathcal{P} \cup S_{max}$
end
$\mathbf{g}(0) \leftarrow$ the solution in $\mathcal{P}$ with the highest value of $z(\Phi_i)$;
while η *seconds have not elapsed* **do**
  for *each of the* $|\mathcal{P}|$ *particles* **do**
    for *each dimension* d *in the particle* **do**
      $r_p \leftarrow$ a random floating point number in the range $[0, 1)$;
      $r_g \leftarrow$ a random floating point number in the range $[0, 1)$;
      $v_i^d(t+1) \leftarrow \omega v_i^d(t) + \varphi_p r_p(p_i^d(t) - x_i^d(t)) + \varphi_g r_g(g^d(t) - x_i^d(t))$;
      $x_i^d(t+1) \leftarrow x_i^d(t) + v_i^d(t+1)$;
    end
    $\mathbf{y}_i \leftarrow$ the sequence in which the jobs are inserted into batches depending on their positions in $\mathbf{x}_i(t+1)$;
    $\mathcal{S} \leftarrow \emptyset$;
    for *the* s *times that random* h_i *values are determined for the jobs in* $\mathbf{y}_i$ **do**
      $\mathbf{h}_i \leftarrow$ random h_i values for the jobs in the order that they appear in $\mathbf{y}_i$;
      $X_i \leftarrow$ sequence containing the batches after all the jobs have been inserted;
      Randomly remove one of the two jobs that are not allowed to be in the same batch for all the clashes until all jobs in X_i are allowed to be with all the other jobs in their respective batch;
      If a job is forbidden to follow another job and start in that period, then one of the jobs is removed;
      Remove the last batch in X_i until the duration of the schedule does not exceed p_{max};
      $z(\Phi_i) \leftarrow f(\Phi_i)$;
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{X_i, z(\Phi_i)\}$;
    end
    $S_{max} \leftarrow$ the sequence in $\mathcal{S}$ with the highest value of $z(\Phi_i)$;
    if $z(\Phi_i)$ *for* S_{max} *is higher than* $z(\Phi_i)$ *for* $\mathbf{p}_i(t)$ **then**
      $\mathbf{p}_i(t+1) \leftarrow \mathbf{x}_i(t+1)$;
      $\mathcal{P}_i \leftarrow S_{max}$;
    else
      $\mathbf{p}_i(t+1) \leftarrow \mathbf{p}_i(t)$;
    end
    if $z(\Phi_i)$ *for* S_{max} *is higher than* $z(\Phi_i)$ *for* $\mathbf{g}(t)$ **then**
      $\mathbf{g}(t+1) \leftarrow \mathbf{x}_i(t+1)$;
    else
      $\mathbf{g}(t+1) \leftarrow \mathbf{g}(t)$;
    end
  end
end
return $\mathcal{P}$;

3.5 Cuckoo search

Cuckoo Search (CS) was proposed by Yang & Deb [108] in 2009 to solve optimisation problems. It was inspired by the obligate brood parasitism of some cuckoo species that lay their eggs in the nests of other bird species. Furthermore, an important aspect of the algorithm is the Lévy flights that are performed by the cuckoos when searching for nests to lay their eggs in. It is characterised by the sudden movement in one direction, as opposed to doing a simple random walk. Once a cuckoo has laid its egg in a host bird's nest, there is a possibility of it being discovered. If the egg is recognised as not being from the host bird, it will either abandon its nest and build a new one at a new location or throw the detected egg out of the nest and produce a new one. Yang & Deb [108] provided an algorithm in which each nest contains only one egg. If the cuckoo's egg is better than the egg in the nest it has chosen to lay its egg in, it will replace it. If this is not the case, the host bird will discover the cuckoo's egg and get rid of it so that it keeps its own egg. A fraction of the host birds with the worst nests will then abandon their nest to build a new one. This is similar to elitism in the GA [25].

3.5.1 Overview of eggs, nests, host birds and cuckoos

Three rules are used in the basic form of CS, as described by Yang & Deb [108].

1. In each iteration, a solution is randomly determined by performing a Lévy walk after which the solution is compared to a randomly chosen solution in the set of solutions (the single eggs in the nests) in the iteration.
2. If the fitness of the randomly determined solution is higher than that of the other solution, it will replace that solution. If the randomly determined solution is, however, worse than the other solution, then it will be discarded.
3. At the end of each iteration, a predetermined fraction of the solutions with the highest fitness will proceed to the following iteration.

These ideas are now extended so that the δ nests contain more than one egg with each nest containing the same number of eggs, say ε eggs. The $\delta\varepsilon$ eggs are initialised for the δ nests the same way as for PSO in §3.4.1 and §3.4.3 with $x_{min} = 0$ and $x_{max} = 1$. That is, if an egg (i, j) is characterised by the vector

$$\mathbf{c}_{ij}(t) = (c_{ij}^1(t), c_{ij}^2(t), \ldots, c_{ij}^{n'}(t)) \tag{3.59}$$

where

i is the index for the ε eggs in the nest,
j is the index for the δ nests in the population and
t is the index for the iterations,

then the (i, j)th egg is initialised for each dimension d by the formula

$$c_{ij}^d(0) = U(c_{min}, c_{max}) \tag{3.60}$$

where c_{min} and c_{max} are the minimum and maximum values for the dimensions of the initial eggs, respectively. Once the initial eggs are determined, they need to be decoded to determine the corresponding sequence of batches that are inserted into the schedule. The same decoding scheme found in §3.4.2 is used in CS that involves determining the vectors $\mathbf{y}_{ij}$ and $\mathbf{h}_{ij}$ to insert batches into a sequence X_{ij} with the added dimension for the different nests in the population.

At the beginning of each iteration a cuckoo will perform a total of γ Lévy flights starting from the position of the best egg obtained so far, with a probability of ψ, or from the characteristics of a randomly generated position in the solution space, with a probability of $1-\psi$. Lévy flights are completed by doing a string of moves in different directions by the Markov chain

$$\mathbf{c}_{k+1}(t) = \mathbf{c}_k(t) + \mathbf{f}(x, \lambda, \sigma) \tag{3.61}$$

where

$\mathbf{c}_k(t)$	is the position of the egg after k moves have been performed during the Lévy flights at time t,
$\mathbf{f}(x, \lambda, \sigma)$	is a vector consisting of n' random variables that are drawn from the Lévy distribution,
λ	is the location parameter and
σ	is the scale parameter.

Each element in $\mathbf{f}(x, \lambda, \sigma)$ is drawn from the Lévy distribution with its probability density function defined as

$$f_i(x, \lambda, \sigma) = \sqrt{\frac{\sigma}{2\pi}} \frac{e^{-\frac{\sigma}{2(x-\lambda)}}}{(x-\lambda)^{3/2}} \tag{3.62}$$

over the domain $x \geq \lambda$. The choice of a stochastic equation for a random walk instead of the original random walk comes from the fact that Lévy flights are much more efficient in searching the solution space. The distance from the position $\mathbf{c}_0(t)$ is much further in the long run. Figures 3.10 and 3.11 contain an example of a random walk and a Lévy walk with 500 steps, respectively. It may be seen that the random walk is unable to make substantial jumps from one part in the solution space to another. On the other hand, the Lévy walk is able to search more areas of the solution space by repeatedly searching one section and moving on to another section by taking large jumps. This allows for distant solutions to be discovered much earlier than when the original random walk is performed.

The fitness value for the egg X_{ij} is determined by the fitness function $z(\Phi_{ij}) = f(\Phi_{ij})$. The quality of a nest is evaluated by giving each of the eggs in the nest equal weight. The quality of a nest j is given by

$$q_j = \sum_{i=1}^{\varepsilon} z(\Phi_{ij}). \tag{3.63}$$

A cuckoo's egg is laid in a randomly chosen nest only if it is better than the worst egg in that nest, that is if $z(X_{new}) > \min_i\{z(X_{ik})\}$ where X_{new} is the sequence determined by the cuckoo after performing Lévy flights and k is the index of the randomly chosen nest. The cuckoo will then get rid of the worst egg to keep the number of eggs in the nest the same. If the cuckoo's egg is not better than any of the eggs in the nest, it will destroy its egg and fly away. A fraction ζ_w of the nests with the smallest value for q_j are then selected. Each of these nests will either be abandoned with a probability of ζ_n after which a new nest of eggs will be randomly generated or the egg in the nest with the smallest value for $z(\Phi_{ij})$ will be removed with a probability of $1-\zeta_n$ and a new egg will be randomly generated using equation (3.60) and inserted into the nest. Finally, at the end of each iteration the new eggs that were added to the nests, including the randomly generated ones from the previous step, are added to the external archive $\mathcal{A}_e$ after which the solutions that are dominated by at least one other solution in $\mathcal{A}_e$ are removed. The algorithm returns the set of nondominated solutions (refer to §3.3.5 for the explanation of nondominated solutions) once the last iteration was completed after η seconds have elapsed.

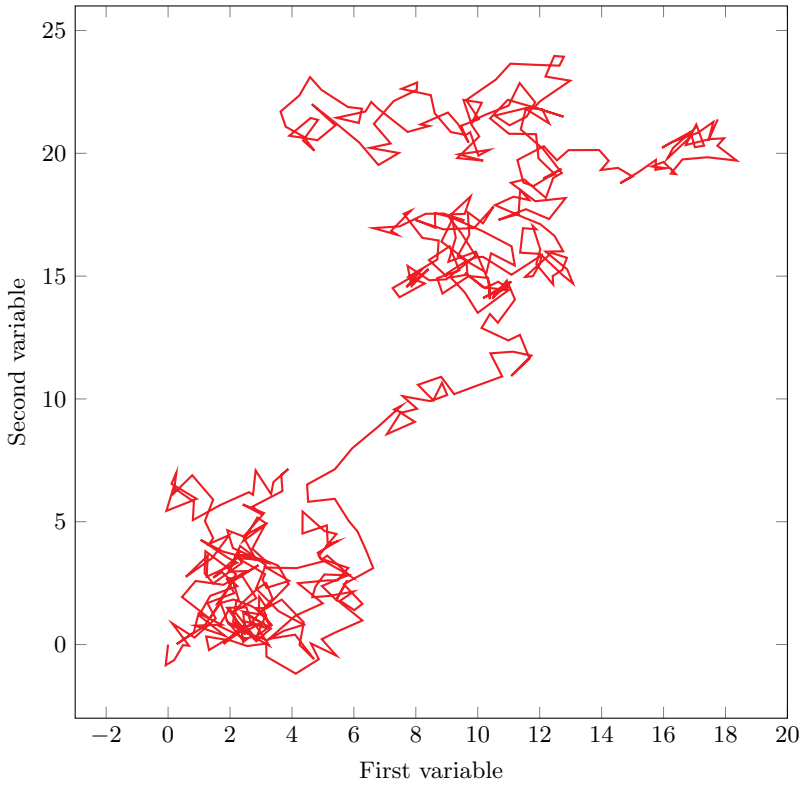


Figure 3.10: *An example of a random walk with 500 steps. A step is made in the x and y-direction, each drawn from the uniform distribution in the range* $[-1, 1]$.

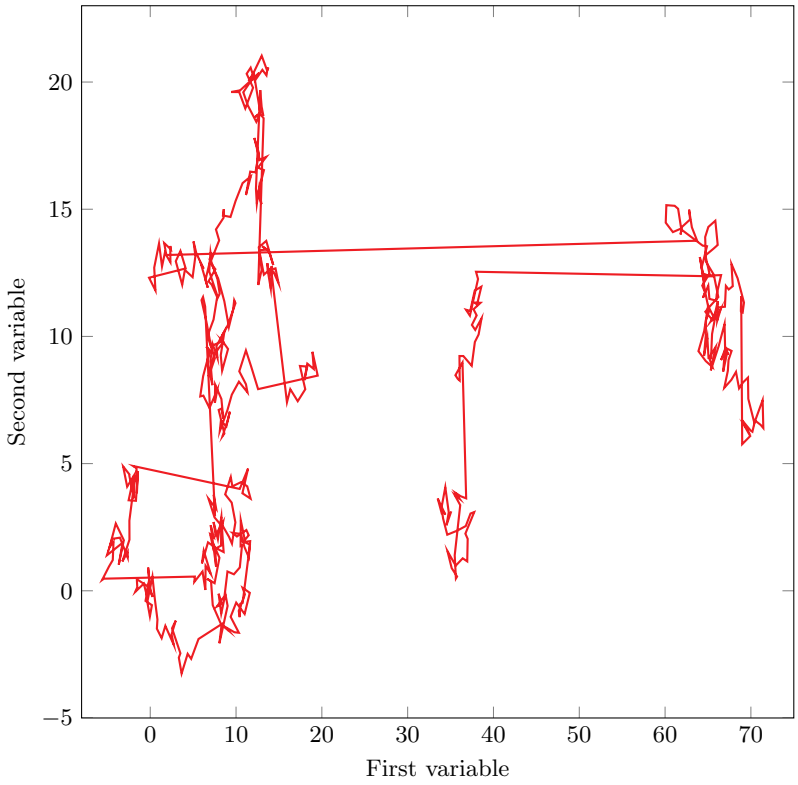


Figure 3.11: *An example of a Lévy walk with 500 steps. A step is made in the x and y-direction, each drawn from the Lévy distribution with* $\lambda = 0$ *and* $\sigma = 0.001$.

3.5.2 Layout of the algorithm

Algorithm 3 contains the pseudocode for CS. Input that are required to run the algorithm include

1. the number of nests δ in each iteration,
2. the number of eggs ε in each of the nests,
3. the computational time η at which the last time a cuckoo attempts to insert an egg in a host bird's nest,
4. the minimum and maximum number of periods p_{min} and p_{max} that the schedule is allowed to be, respectively,
5. the probability ψ that the cuckoo will start its Lévy flights from the position of the best egg obtained so far,
6. a fraction of the nests ζ_w with the smallest value for q_j that are selected to either be abandoned to build and populate a new nest with eggs or its worst egg is thrown away after which a new one is produced,
7. The probability ζ_n that a nest will be abandoned after which a new one will be built at another location and populated with new eggs (the egg with the lowest value of $z_{X_{ij}}$ will be thrown out of the nest with a probability of $1-\zeta_n$ after which a new one will be produced), and
8. the number of times s that $\mathbf{h}_i$ vectors are generated for an egg.

Lines 1–12 contain the initialisation of the algorithm. The set $\mathcal{P}$, containing sets (nests) of solutions (eggs) and the external archive $\mathcal{A}_e$ which is the set containing the nondominated solutions found during the algorithm, are initialised to an empty set. Random eggs are generated for each of the newly built nests in lines 3–11 where $\mathcal{P}_j$ is the set of eggs in nest j. Each nest $\mathcal{P}_j$ is populated with an egg which is represented by a sequence S_{max}. The sequence S_{max} is obtained in the same way as that in §3.4.2 and §3.4.3 by determining s random $\mathbf{h}_i$ vectors, displayed in Procedure 4. The sequence X_{ij} is then made feasible by following the 3-step feasibility phase after which it is inserted into the set $\mathcal{S}$ with its value for $z(\Phi_{ij})$. If the sequence in $\mathcal{S}$ with the highest value of $z(\Phi_{ij})$ contains more than two batches and has a duration of at least p_{min}, then it is added to the external archive. The procedure returns this sequence that is then added to the nest. Once all the eggs have been inserted into the nest $\mathcal{P}_j$, the nest is added to the population $\mathcal{P}$. The final step in the initialisation process is to set the best found solution $\mathbf{g}(0)$ to the solution in $\mathcal{P}$ with the highest value of $z(\Phi_{ij})$.

The attempts of a cuckoo inserting an egg in a host bird's nest may be found in lines 13–51. The cuckoo's initial position is determined in lines 14–19 by setting it to the best found position $\mathbf{g}(t)$ with a probability of ψ or to a random position with a probability of $1-\psi$. The cuckoo then performs a total of γ Lévy flights, as seen in lines 20–22, using equation (3.61). Its position $\mathbf{c}_\gamma(t)$, once the flights have been completed, is used in Procedure 4 to determine the sequence X_{new} which is inserted into $\mathcal{A}_e$. This sequence is then compared to the worst sequence in the chosen nest k. If $z(X_{new}) > \min_i\{z(X_{ik})\}$, X_{new} will replace the worst egg in the nest.

The updating of the nests by the host birds may be found in lines 29–47. The quality of the nests are evaluated by determining the value q_j for each of the nests. Each of the $[\zeta_w\delta]$ nests[5] with the lowest value for q_j is either abandoned with a probability of ζ_n after which a new nest

[5]The value $[\zeta_w\delta]$ is $\zeta_w\delta$ rounded off to the nearest integer.

is built and populated with eggs or loses its worst egg with a probability of $1 - \zeta_n$ after which a new egg is produced and inserted into the nest. In the event that a nest is abandoned, a new nest is built represented by setting $\mathcal{P}_j$ to an empty set for host bird j. The nest is then populated with randomly generated eggs that are obtained from Procedure 4 that also inserts all of the generated eggs into $\mathcal{A}_e$. If the worst egg is abandoned in the nest, then a randomly generated egg will replace that egg by generating a sequence S_{max} using Procedure 4 and replacing the sequence X_{lj} with this new sequence, where l is the index of the egg in the nest for which $z(X_{ik}) = \min_i\{z(X_{ik})\}$. Again, the new sequence S_{max} is added to $\mathcal{A}_e$. Finally, the duplicate solutions are removed from $\mathcal{A}_e$, all the solutions that are dominated by one or more solutions in $\mathcal{A}_e$ are removed from $\mathcal{A}_e$ and the best found solution $\mathbf{g}(t)$ is updated at the end of each iteration. The algorithm returns the set of nondominated solutions in $\mathcal{A}_e$ with the corresponding objective function values once the last iteration has been completed after η seconds have elapsed.

Algorithm 3: Cuckoo search

Input: δ, ε, η, p_{min}, p_{max}, ψ, ζ_w, ζ_n, γ, s
Output: A set of feasible nondominated solutions in the external archive

$\mathcal{P} \leftarrow \emptyset$;
$\mathcal{A}_e \leftarrow \emptyset$;
for *each of the* δ *nests* **do**
  $\mathcal{P}_j \leftarrow \emptyset$;
  for *each of the* ε *eggs in the nest* **do**
    $\mathbf{c}_{ij}(0) \leftarrow$ randomly generated characteristic for egg i with each dimension in the range $[0, 1]$;
    $S_{max} \leftarrow$ the sequence obtained from $\mathbf{c}_{ij}(0)$ by performing Procedure 4;
    $\mathcal{P}_j \leftarrow \mathcal{P}_j \cup S_{max}$
  end
  $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}_j$
end
$\mathbf{g}(0) \leftarrow$ the solution in $\mathcal{P}$ with the highest value of $z(\Phi_{ij})$;
while η *seconds have not elapsed* **do**
  $\kappa \leftarrow$ a random floating point number in the range $[0, 1)$;
  if $\kappa < \psi$ **then**
    $\mathbf{c}_0(t) \leftarrow \mathbf{g}(t)$;
  else
    $\mathbf{c}_0(t) \leftarrow$ randomly generated position with each dimension in the range $[0, 1]$;
  end
  for γ *iterations* **do**
    $\mathbf{c}_{k+1}(t) \leftarrow \mathbf{c}_k(t) + \mathbf{f}(x, \lambda, \sigma)$;
  end
  $X_{new} \leftarrow$ the sequence obtained from $\mathbf{c}_\gamma(t)$ by performing Procedure 4;
  $k \leftarrow$ the index of the randomly chosen nest;
  if $z(X_{new}) > \min_i\{z(X_{ik})\}$ **then**
    $l \leftarrow$ the index of the egg in the nest for which $z(X_{lk}) = \min_i\{z(X_{ik})\}$;
    $X_{lk} \leftarrow X_{new}$;
  end
  for *each of the* δ *nests* **do**
    $q_j \leftarrow \sum_{i=1}^{\varepsilon} z(\Phi_{ij})$;
  end
  for *each of the* $[\zeta_w \delta]$ *nests with the lowest value for* q_j **do**
    $\kappa \leftarrow$ a random floating point number in the range $[0, 1)$;
    if $\kappa < \zeta_n$ **then**
      $\mathcal{P}_j \leftarrow \emptyset$;
      for *each of the* ε *eggs in the new nest* **do**
        $\mathbf{c}_{ij}(t) \leftarrow$ randomly generated characteristic for egg i with each dimension in the range $[0, 1]$;
        $S_{max} \leftarrow$ the sequence obtained from $\mathbf{c}_{ij}(t)$ by performing Procedure 4;
        $\mathcal{P}_j \leftarrow \mathcal{P}_j \cup S_{max}$
      end
    else
      $l \leftarrow$ the index of the egg in the nest for which $z(X_{lk}) = \min_i\{z(X_{ik})\}$;
      $\mathbf{c}_{lj}(t) \leftarrow$ randomly generated characteristic for egg l with each dimension in the range $[0, 1]$;
      $S_{max} \leftarrow$ the sequence obtained from $\mathbf{c}_{lj}(t)$ by performing Procedure 4;
      $X_{lj} \leftarrow S_{max}$;
    end
  end
  Remove the duplicate solutions from $\mathcal{A}_e$;
  Remove all solutions from $\mathcal{A}_e$ that are dominated by one or more other solutions in $\mathcal{A}_e$;
  $\mathbf{g}(t) \leftarrow$ the solution in $\mathcal{P}$ with the highest value of $z(\Phi_{ij})$;
end
return $\mathcal{A}_e$;

Procedure 4: Determine a sequence corresponding to an egg's characteristic

Input: $\mathbf{c}_{ij}(t)$
Output: The sequence with the highest value of $z(\Phi_{ij})$

$\mathbf{y}_i \leftarrow$ the sequence in which the jobs are inserted into batches depending on their positions in $\mathbf{c}_{ij}(t)$;
$\mathcal{S} \leftarrow \emptyset$;
for *the s times that random h_i values are determined for the jobs in* $\mathbf{y}_i$ **do**
    $\mathbf{h}_i \leftarrow$ random h_i values for the jobs in the order that they appear in $\mathbf{y}_i$;
    $X_{ij} \leftarrow$ sequence containing the batches after all the jobs have been inserted;
    Randomly remove one of the two jobs that are not allowed to be in the same batch for all the clashes until all jobs in X_{ij} are allowed to be with all the other jobs in their respective batch;
    If a job is forbidden to follow another job and start in that period, then one of the jobs is removed;
    Remove the last batch in X_{ij} until the duration of the schedule does not exceed p_{max};
    $z(\Phi_{ij}) \leftarrow f(\Phi_{ij})$;
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{X_{ij}, z(\Phi_{ij})\}$;
end
$S_{max} \leftarrow$ the sequence in $\mathcal{S}$ with the highest value of $z(\Phi_{ij})$;
if $\mu(S_{max}) \geq 2$ *and* $d(S_{max}) \geq p_{min}$ **then**
    $\mathcal{A}_e \leftarrow \mathcal{A}_e \cup S_{max}$;
end
return S_{max};

3.6 Genetic algorithm

The genetic algorithm (GA) was first developed by John H. Holland in the 1970s originating from his interest in complex adaptive systems [46]. GAs have been implemented to solve a wide variety of combinatorial optimisation problems, especially scheduling problems [18]. They are biologically inspired optimisation techniques that takes a set of solutions, called a population of individuals, and evolves it into a set of improving solutions in each iteration, referred to as a generation [100]. This process imitates evolution so that individuals become more fit (smarter) in each generation as they adapt to their surroundings. The population in the final generation is then returned upon execution of the algorithm so that the fittest individuals are presented to the decision maker.

3.6.1 Synopsis of the genetic algorithm

Reproduction is the process during which new chromosomes, referred to as children, are formed by using the genes from 2 or more chromosomes, known as the parents. Each individual's chromosome is measured in order to determine the individual's fitness value according to a predefined fitness function. This fitness value determines whether an individual is selected for breeding in its generation. Various techniques exist for selecting parents to breed in each generation such as fitness proportionate selection (or roulette wheel selection), tournament selection and stochastic universal sampling. The selection process ensures that a suitable individual with a better fitness value compared to the other individuals in their generation will have a greater chance of being selected for breeding. This procedure imitates the process of natural selection. Natural selection is the process where individuals of the population with beneficial and more suitable characteristics are more likely to reproduce. It ensures that the traits of strong individuals will be present in individuals in the following generation and allows the entire population in each generation to adapt to their surroundings.

An appropriate crossover operator, to decide which genes to pass on to which children, is used during the reproduction process for which the definition is problem specific. Examples of crossover operators include single-point crossover, two-point crossover, cut-and-splice, uniform crossover and three parent crossover. These processes reflect the biological chromosomal crossover during which the combination of genetic material produces a newly formed chromosome. The population is usually kept constant in each generation. Once all children are produced, each of them may have a certain probability of being mutated using a mutation operator. Mutation enables diversity in the population to ensure that the algorithm is able to escape local optima. The probability of an individual being mutated must, however, be set low so as to avoid a random search which consequently defeats the purpose of the crossover operator.

Two well-known innovations exist in genetic algorithms which help to find good solutions quicker than if they were absent. Firstly, a repair operator can be used to repair genes in a chromosome in an attempt to move the solution closer to feasibility or provide the solution with a better fitness value. A solution may generally require a few small adjustments to significantly improve its quality. Secondly, elitism is often used to allow a percentage of the best individuals to carry over to the following generation. This guarantees that the fittest individuals will not be lost from the population. It is common when GAs are applied to multi-objective optimisation problems to maintain a local archive[6] (similar to that of VNS in §3.3) that contains nondominated feasible solutions found during the algorithm [25]. This archive is then used in conjunction with

[6] A local archive containing nondominated feasible solutions is used in the genetic algorithms for the same reasons explained in §3.3.5.

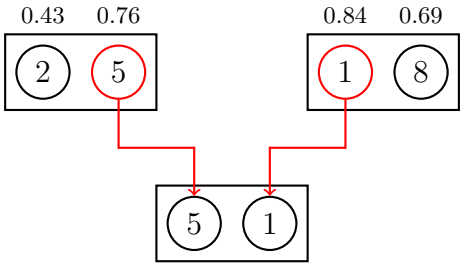


Figure 3.12: *An example of binary tournament selection in which a group of 4 distinct individuals is selected from the population and local archive to determine a pair of parents.*

the population during the selection process to produce children. This ensures that the characteristics from the best solutions found to date will be present in the population in the following generations to come.

3.6.2 Selection and crossover operators

Three techniques are used to select pairs of parents for breeding, namely binary tournament selection (BTS), fitness proportionate selection (FPS) and stochastic universal sampling (SUS). Figure 3.12 displays an example of BTS. The first step involves selecting four individuals from the population at random, regardless of their fitness value, that is each individual is selected with a probability of $1/(|\mathcal{P}| + |\mathcal{A}_l|)$ where $|\mathcal{P}|$ is the number of individuals in the population and $|\mathcal{A}_l|$ is the number of individuals in the local archive $\mathcal{A}_l$. In the example, solutions 2, 5, 1 and 8 were selected from $\mathcal{P} \cup \mathcal{A}_l$ at random in the order from left to right. The number above each solution is its fitness value $z(\Phi_i) = f(\Phi_i)$. The first pair of individuals is evaluated after which the solution with the highest fitness value is selected as the first parent. The same process is repeated for the second pair of individuals to select the second parent. In the example, individual 5 was chosen from the first pair, since $z(\Phi_5) = 0.76 \geq 0.43 = z(\Phi_2)$. Similarly, individual 1 was chosen from the second pair seeing that $z(\Phi_1) = 0.84 \geq 0.69 = z(\Phi_8)$. Groups of 4 distinct individuals are selected from the population to select $|\mathcal{P}|/2$ pairs of parents.

The second technique that is used in the selection process is called FPS (or roulette wheel selection). Individuals are allocated a slice of a strip (or a roulette wheel) where the size of the slice is proportional to the individual's fitness value relative to the other individuals' fitness values. Therefore, the larger the fitness value, the larger the size of the slice will be. As a result, individuals with a higher fitness value will have a higher chance of being selected as a parent. An example of FPS may be found in Figure 3.13. It may be seen that jobs with higher fitness values take up a larger portion of the strip and have a greater chance of being selected. The probability of an individual j being selected is thus $z(\Phi_j)/\sum_{i=1}^{|\mathcal{P}|+|\mathcal{A}_l|} z(\Phi_i)$. In order to ensure that every pair contains different parents, the first parent is removed from the strip before selecting the second parent. The first parent is then reinserted into the list before the next pair of parents is determined.

The final technique that is used to select pairs of parents is called SUS. It is an extension and improvement on FPS due to it containing no bias and having minimal spread. The same strip is used in both of the methods. Figure 3.14 contains an example of SUS. If the length of the strip is $f = \sum_{i=1}^{|\mathcal{P}|+|\mathcal{A}_l|} z(\Phi_i)$ and p points are chosen on the strip, then a random number, say q, is generated in the interval $[0, f/p)$. Points are then added at positions $q+f/p, q+2f/p, \ldots, q+(p-1)f/p$ on the strip. Two parents are then randomly selected from the individuals corresponding to the p points. Four points were determined in the example so that individuals 1, 3, 5 and 8

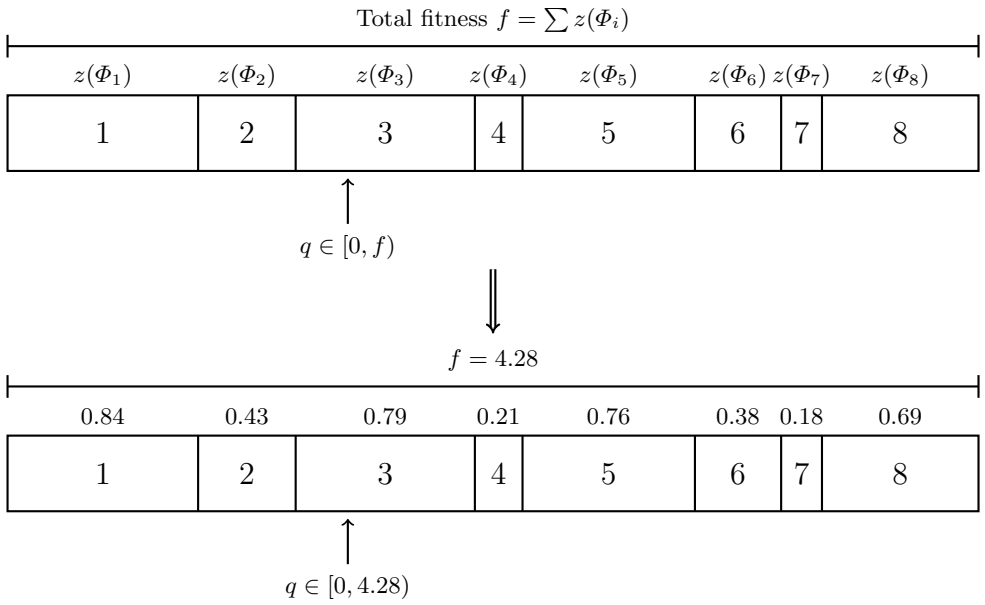


Figure 3.13: *An example of fitness proportionate selection in which all individuals in the population and local archive are allocated a slice of a strip where the size of the slice is proportional to the individual's fitness value relative to the other individuals' fitness values.*

correspond to the selected points. Two distinct parents are then selected as a pair of parents from this group of four individuals. This process is completed $|\mathcal{P}|/2$ number of times.

Two approaches of the genetic algorithm are considered in this thesis. The first approach (GA-I) is when the chromosome of a solution consists of a string of floating point numbers, similar to that considered in §3.4 (PSO) and §3.5 (CS). When determining the fitness value of an individual, its chromosome needs to be decoded into a batch sequence which needs to be made feasible (as explained in §3.4.2). The feasible batch sequence (schedule) is then associated with the chromosome until the algorithm is terminated. In the second approach (GA-II), a chromosome is defined as a batch sequence containing all the available jobs. The batch sequence is then made feasible after which the individual's fitness value can be determined. No decoding is necessary in this approach. These two approaches incorporate the same selection techniques, but different crossover operators, mutation operators and immigration schemes.

The two crossovers that are used for GA-I are explained first. The first crossover is called the one-point crossover operator (OPX). The first step is to generate a random crossover point, the same crossover point for both parents. A crossover point may not be chosen before the first gene or after the last gene. It should be chosen between the first and last gene, *i.e.* there are a possible $n' - 1$ crossover points if there are a total of n' available jobs to be scheduled. The first offspring will receive the genes to the left of the crossover point in the first parent followed by the genes to the right of the crossover point in the second parent. On the other hand, the second offspring will receive the genes to the left of the crossover point in the second parent followed by the genes to the right of the crossover point in the first parent. Figure 3.15 shows an example of OPX done on two parents to produce two offspring. The example in the figure illustrates when the crossover point is randomly chosen between the third and fourth gene for both parents.

The second crossover is called the parameterised uniform crossover operator (PUX). It is performed by generating a random binary string of length n' with each element equal to 0 or 1, each with a probability of 0.5. If the element at position i in the binary string contains a 1, the

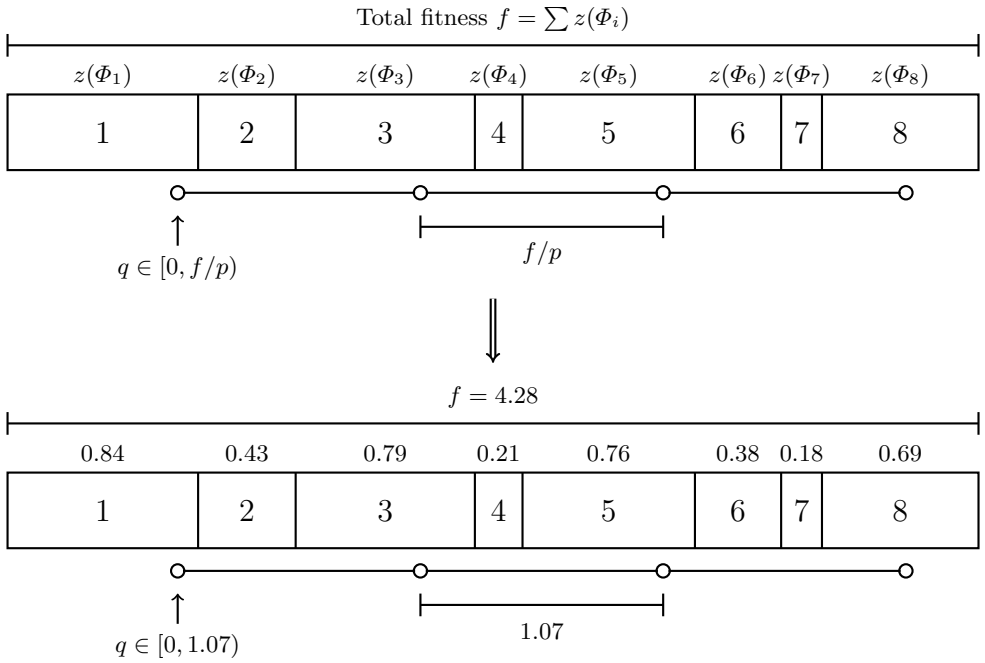


Figure 3.14: *An example of stochastic universal sampling which is an extension and improvement on fitness proportionate selection. It involves the placement of p equally spaced points of which two parents are randomly chosen from the individuals corresponding to the position of the points on the strip.*

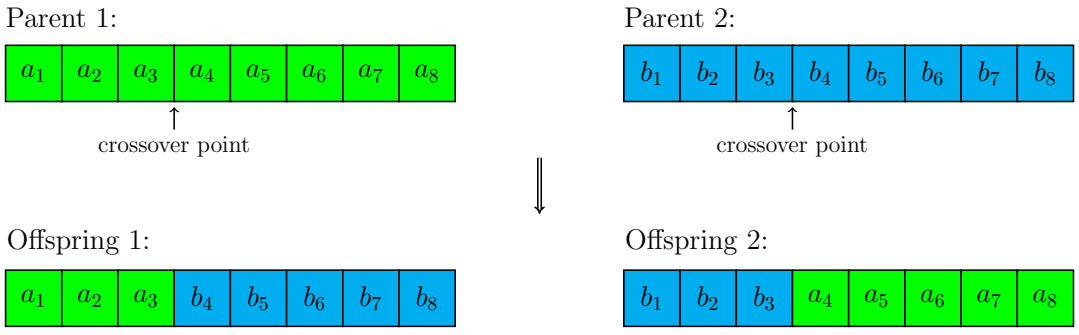


Figure 3.15: *An example of the one-point crossover operator. A random crossover point is randomly generated, the same crossover point for both parents. The first parent replicates its genes before the crossover point to the first offspring and its genes after the crossover point to the second offspring. Similarly, the second parent replicates its genes before the crossover point to the second offspring and its genes after the crossover point to the first offspring.*

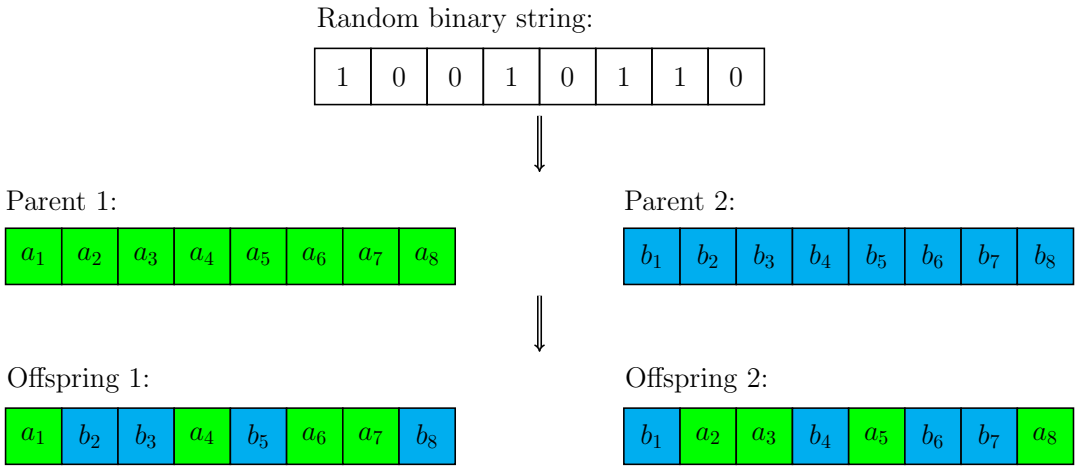


Figure 3.16: *An example of the parameterised uniform crossover operator. A random binary string is generated. For each element that is equal to 1, the first offspring will receive the gene from the first parent and the second offspring will receive the gene from the second parent at that position. On the other hand, for each element that is equal to 0, the first offspring will receive the gene from the second parent and the second offspring will receive the gene from the first parent at that position.*

first offspring will receive the gene at position i of the first parent while the second offspring will receive the gene at position i of the second parent. Otherwise, if the element at position i in the binary string contains a 0, the first offspring will receive the gene at position i of the second parent while the second offspring will receive the gene at position i of the first parent. An example of PUX may be found in Figure 3.16. A random binary string $\mathbf{b} = (1, 0, 0, 1, 0, 1, 1, 0)$ is generated after which the aforementioned process is followed to produce the two offspring in the example.

The two crossovers that are used for GA-II are more advanced variations of those used when decoding is employed. The first of these crossovers is the one-point order crossover operator (OPOX). The first step is to choose two independent crossover points for the two parents. The first offspring will receive the batches to the left of the crossover point in the first parent while the second offspring will receive the batches to the left of the crossover point in the second parent. It is then iterated through the batches of the first parent starting from the beginning of the sequence. A batch is appended to the second offspring's chromosome if at least one of the jobs in the batch is not yet present in the offspring. The same process is repeated for the second parent to complete reproduction of the first child. The final step is to remove the duplicate jobs so that all the jobs appear exactly once in each of the offspring. Figure 3.17 illustrates an example of OPOX. The crossover points are after the third batch for the first parent and after the second batch for the second parent. The first offspring contains job 1 in the second and sixth batch. One of these jobs is randomly removed to give the resulting repaired first offspring in the final part of the figure.

The final crossover is the parameterised uniform order crossover operator (PUOX). Figure 3.18 contains an example of PUOX. The first step is to generate a random binary string of length $\min\{\mu(X_1), \mu(X_2)\}$ where $\mu(X_i)$ is the number of batches in the sequence of parent i. The first $\min\{\mu(X_1), \mu(X_2)\}$ batches are connected according to the string. If a 0 in position j is followed by a 1, then the j-th batch in the first parent is connected to the $(j+1)$-th batch in the second parent. Similarly, if a 1 in position j is followed by a 0, then the j-th batch in the second parent is connected to the $(j+1)$-th batch in the first parent. The remaining

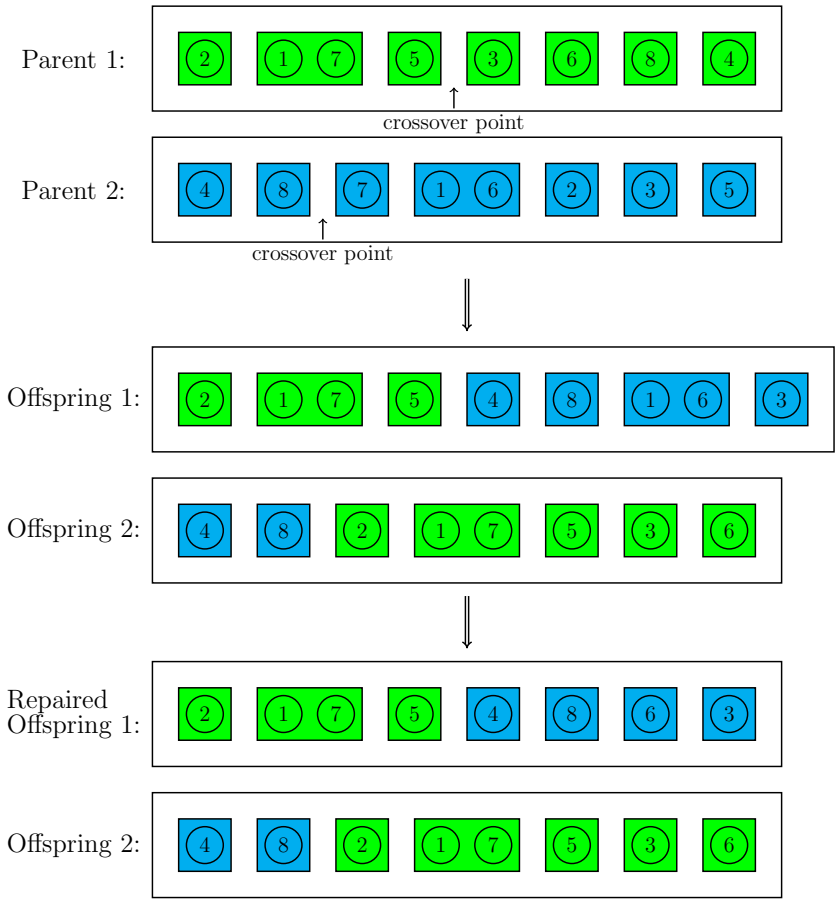


Figure 3.17: *An example of the one-point order crossover operator. Two independent crossover points are chosen for the two parents. The first parent replicates its batches before its crossover point to the first offspring while the second parent replicates its batches before its crossover point to the second offspring. Batches from the first parent are then inserted into the sequence of the second offspring while batches from the second parent are inserted into the sequence of the first offspring. Finally, the offspring are repaired by removing duplicate jobs.*

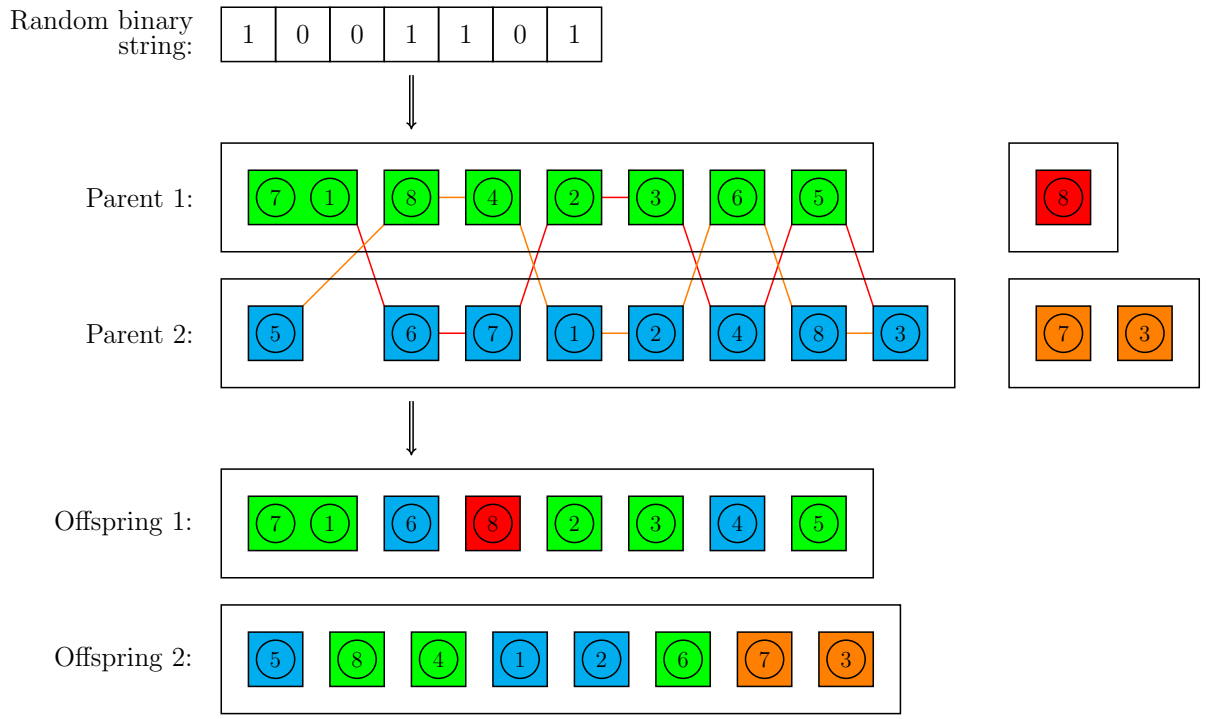


Figure 3.18: *An example of the parameterised uniform order crossover operator. A random binary string is generated after which the batches in the two parents are connected according to this string. Jobs are then inserted into the sequences of the offspring by moving along the two paths. Batches that are attempted to be inserted and whose jobs are all present in the sequence, are replaced by jobs that are not on its path. The final step is to repair the offspring by removing duplicate jobs.*

$\max\{\mu(X_1), \mu(X_2)\} - \min\{\mu(X_1), \mu(X_2)\}$ batches are connected. An additional sequence of jobs is generated for each parent that contains all the jobs that are not on the parent's path for the first $\min\{\mu(X_1), \mu(X_2)\}$ batches. The order of the jobs in this list is in the same order as that in the other parent's sequence. These lists are used when wanting to insert a job into an offspring's sequence and it's already in the sequence. The offspring are produced by inserting jobs along the two paths. Each time a batch is attempted to be inserted into an offspring's sequence and all of the jobs in the batch is already present in the sequence, then the next job in the additional list will be inserted into the offspring's sequence instead. Once the offspring have been produced, duplicate jobs are removed from the offspring's sequences in the same way as in OPOX. No duplicate jobs are present in the offspring in the example, hence this final step is not required.

3.6.3 Mutation, elitism and immigration

A mutation operator, elitism technique and immigration scheme are used to speed up convergence and to ensure that there is diversity in the population. Mutation and immigration are defined differently for GA-I and GA-II while elitism is identical for both approaches. Mutation is performed by altering a limited number of genes in the chromosome of a randomly selected individual. The parameters that are needed to perform mutation are the probability α of mutating an individual's chromosome and the maximum number of genes α_{max} to change during mutation for GA-I or the maximum number of successive moves α'_{max} that are made on an individual during mutation for GA-II. It is now evident that mutation is performed in GA-I by selecting $U(1, \alpha_{max})$ genes in an individual and changing their values to $U(0, 1)$ which is

generated separately for each selected gene. In GA-II, mutation is done by doing $U(1, \alpha'_{max})$ successive moves on an individual. These moves include the first, third, fifth and seventh moves explained in §3.3.1. That is, two batches are swapped, a batch is removed and inserted into another position in the sequence, two jobs are swapped from two different batches if possible or a job is removed from a batch and inserted into another batch in the sequence if possible. A random move is selected for each of the $U(1, \alpha'_{max})$ moves that can be made. If the fifth or seventh move cannot be made, another move will be selected.

Elitism is performed at the end of each generation to determine the population $\mathcal{P}_{k+1}$ for the following generation. If $\mathcal{P}_k$ is the population of the current generation and $\mathcal{P}_o$ is the offspring population of the current generation, then a set $\mathcal{P}_s$ is determined by sorting $\mathcal{P}_k \cup \mathcal{P}_o$ in descending order of their fitness value $z(\Phi_i)$ for each solution Φ_i in $\mathcal{P}_k \cup \mathcal{P}_o$. The first 50% of the individuals in $\mathcal{P}_s$, *i.e.* the $|\mathcal{P}|$ solutions with the highest value of $z(\Phi_i)$, will then proceed to the following generation to form $\mathcal{P}_{k+1}$. The population $\mathcal{P}_{k+1}$ will be used together with $\mathcal{A}_l$ to produce the offspring population for the following generation.

Lastly, immigration is executed to ensure diversity in the population together with the mutation operator. The process involves generating random solutions at the beginning of each generation and inserting it into the local archive so that they have a chance to become parents and assist in the production of offspring for the following generation. The parameter that is needed to perform immigration is the percentage β of the size of the local archive $|\mathcal{A}_l|$. The number of random solutions that will be generated at the beginning of each generation and inserted into $\mathcal{A}_l$ is thus $\beta|\mathcal{A}_l|$ rounded off to the nearest integer, denoted by $[\beta|\mathcal{A}_l|]$. In GA-I, the chromosome of an individual is randomly generated in the same way as in equations (3.51) and (3.55) with $x_{min} = 0$ and $x_{max} = 1$. On the other hand, a chromosome is randomly generated in GA-II by forming batches, that collectively contain all the jobs with each job appearing exactly once, which is sequenced in a random order.

3.6.4 Outline of the algorithms

Algorithm 5 contains the pseudocode for the genetic algorithm with decoding (GA-I). It requires $|\mathcal{P}|$, the computational time η at which the last generation is performed, α, α_{max}, β, p_{max} and s. The initialisation process may be found in lines 1–19. The local archive $\mathcal{A}_l$ is set to an empty set, the index k that is used for the generations is set to an initial value of 1 and the set $\mathcal{P}_k$ containing the individuals in the population for generation k is set to an empty set. The generating of the random individuals is seen in lines 4–19 where $\mathbf{x}_i(k)$ is the chromosome for individual i in generation k. This is done in the same way as in §3.4.2 and §3.4.3 to determine the sequence S_{max} in $\mathcal{S}$ with the highest fitness value. The sequence S_{max} is then added to the population for the first generation.

The evolutionary component of the algorithm may be found in lines 20–44. The immigration scheme is in lines 21–24. A total of $[\beta|\mathcal{A}_l|]$ individuals are randomly generated to determine S_{max} for each of the individuals which are added to the local archive. The $|\mathcal{P}|/2$ pairs of parents are then chosen from $\mathcal{P}_k \cup \mathcal{A}_l$ using BTS, FPS or SUS (only one selection technique is used throughout the algorithm) after which the offspring are produced in lines 26–38. The offspring population for the current generation is set to an empty set before $|\mathcal{P}|/2$ pairs of offspring are produced by each of the pairs of parents using OPX or PUX. Each of the offspring are then mutated with a probability of α after which their S_{max} is determined and added to the offspring population and the local archive. Finally, the duplicate solutions are removed from the local archive as well as all the solutions that are dominated by at least one other solution in the local archive at the end of each generation. The population $\mathcal{P}_{k+1}$ for the following generation is determined in lines 41 and 42 using the set $\mathcal{P}_s$. Once the last generation has been completed,

the local archive is returned.

The pseudocode for the genetic algorithm without decoding (GA-II) may be found in Algorithm 6. GA-II requires all the parameters as GA-I except for s. Lines 1–3 are the same as lines 1–3 in Algorithm 5. Lines 4–12 contain the generating of random individuals for the first generation. The first step is to randomly generate a batch sequence X_i for individual i containing all n' jobs. A new batch sequence Y_i is then initialised that is identical to X_i. This is done to keep track of the batch sequence containing all the jobs. The variable X_i is considered as the chromosome while Y_i is the feasible batch sequence that corresponds to the infeasible batch sequence X_i, assuming $p_{max} < d(X_i)$. The three feasibility steps are then performed on Y_i, as explained in §3.3.4, after which its fitness value is calculated. The final step in the initialisation of the individual is to add it to the population for the first generation.

The evolution of the individuals may be found in lines 13–38. The immigration scheme in lines 14–18 is similar to that in GA-I. A batch sequence X_i is randomly generated for each of the $[\beta|\mathcal{A}_l|]$ new individuals. A new batch sequence Y_i is then determined using the same procedure in lines 6–10 which is then added to the local archive together with its fitness value and X_i. The selection, crossover and mutation in lines 19–32 is similar to that in GA-I except that either OPOX or PUOX is used to perform crossover and an individual is mutated by performing $U(1, \alpha'_{max})$ moves on X_i of the individual, that is the first, third, fifth or seventh move in §3.3.1 on the batch sequence that contains all the n' available jobs. Finally, lines 33–37 are identical to lines 39–43 in GA-I that represent the removal of solutions from the local archive and determining the population for the following generation. Again, the local archive is returned to the decision maker upon completion of the last generation.

Algorithm 5: Genetic algorithm with decoding (GA-I)

Input: $|\mathcal{P}|$, η, α, α_{max}, β, p_{max}, s
Output: A set of feasible nondominated solutions in the local archive found during the search

1 $\mathcal{A}_l \leftarrow \emptyset$;
2 $k \leftarrow 1$;
3 $\mathcal{P}_k \leftarrow \emptyset$;
4 **for** *each of the* $|\mathcal{P}|$ *individuals* **do**
5 $\quad \mathbf{x}_i(k) \leftarrow$ randomly generated chromosome for individual i with each dimension in the range $[0, 1]$;
6 $\quad \mathbf{y}_i \leftarrow$ the sequence in which the jobs are inserted into batches depending on their positions in $\mathbf{x}_i(k)$;
7 $\quad \mathcal{S} \leftarrow \emptyset$;
8 $\quad$ **for** *the* s *times that random* h_i *values are determined for the jobs in* $\mathbf{y}_i$ **do**
9 $\quad\quad \mathbf{h}_i \leftarrow$ random h_i values for the jobs in the order that they appear in $\mathbf{y}_i$;
10 $\quad\quad X_i \leftarrow$ sequence containing the batches after all the jobs have been inserted;
11 $\quad\quad$ Randomly remove one of the two jobs that are not allowed to be in the same batch for all the clashes until all jobs in X_i are allowed to be with all the other jobs in their respective batch;
12 $\quad\quad$ If a job is forbidden to follow another job and start in that period, then one of the jobs is removed;
13 $\quad\quad$ Remove the last batch in X_i until the duration of the schedule does not exceed p_{max};
14 $\quad\quad z(\Phi_i) \leftarrow f(\Phi_i)$;
15 $\quad\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{\mathbf{x}_i(k), X_i, z(\Phi_i)\}$;
16 $\quad$ **end**
17 $\quad S_{max} \leftarrow$ the sequence in $\mathcal{S}$ with the highest value of $z(\Phi_i)$;
18 $\quad \mathcal{P}_k \leftarrow \mathcal{P}_k \cup S_{max}$;
19 **end**
20 **while** η *seconds have not elapsed* **do**
21 $\quad$ **for** $[\beta|\mathcal{A}_l|]$ *individuals* **do**
22 $\quad\quad S_{max} \leftarrow$ the solution obtained by repeating lines 5–17 to generate a new individual;
23 $\quad\quad \mathcal{A}_l \leftarrow \mathcal{A}_l \cup S_{max}$;
24 $\quad$ **end**
25 $\quad$ Select $|\mathcal{P}|/2$ pairs of parents from $\mathcal{P}_k \cup \mathcal{A}_l$ using BTS, FPS or SUS;
26 $\quad \mathcal{P}_o \leftarrow \emptyset$;
27 $\quad$ **for** *each of the* $|\mathcal{P}|/2$ *pairs of parents* **do**
28 $\quad\quad \mathcal{O} \leftarrow$ two offspring produced by the pair of parents using OPX or PUX;
29 $\quad\quad$ **for** *each individual in* $\mathcal{O}$ **do**
30 $\quad\quad\quad \kappa \leftarrow$ a random floating point number in the range $[0, 1)$;
31 $\quad\quad\quad$ **if** $\kappa < \alpha$ **then**
32 $\quad\quad\quad\quad$ Mutate individual by selecting $U(1, \alpha_{max})$ genes and changing their values to $U(0, 1)$;
33 $\quad\quad\quad$ **end**
34 $\quad\quad\quad \mathcal{S}_{max} \leftarrow$ the solution obtained by repeating lines 6–17 on the individual;
35 $\quad\quad\quad \mathcal{P}_o \leftarrow \mathcal{P}_o \cup S_{max}$;
36 $\quad\quad\quad \mathcal{A}_l \leftarrow \mathcal{A}_l \cup S_{max}$;
37 $\quad\quad$ **end**
38 $\quad$ **end**
39 $\quad$ Remove the duplicate solutions from $\mathcal{A}_l$;
40 $\quad$ Remove all solutions from $\mathcal{A}_l$ that are dominated by one or more other solutions in $\mathcal{A}_l$;
41 $\quad \mathcal{P}_s \leftarrow \mathcal{P}_k \cup \mathcal{P}_o$ sorted in descending order of their fitness value $z(\Phi_i)$ for each solution Φ_i in $\mathcal{P}_k \cup \mathcal{P}_o$;
42 $\quad \mathcal{P}_{k+1} \leftarrow$ the first $|\mathcal{P}|$ individuals in $\mathcal{P}_s$;
43 $\quad k \leftarrow k + 1$;
44 **end**
45 **return** $\mathcal{A}_l$;

Algorithm 6: Genetic algorithm without decoding (GA-II)

Input: $|\mathcal{P}|$, η, α, α'_{max}, β, p_{max}
Output: A set of feasible nondominated solutions in the local archive found during the search

1 $\mathcal{A}_l \leftarrow \emptyset$;
2 $k \leftarrow 1$;
3 $\mathcal{P}_k \leftarrow \emptyset$;
4 **for** *each of the* $|\mathcal{P}|$ *individuals* **do**
5 $X_i \leftarrow$ randomly generated batch sequence for individual i containing all the jobs;
6 $Y_i \leftarrow X_i$;
7 Randomly remove one of the two jobs that are not allowed to be in the same batch for all the clashes until all jobs in Y_i are allowed to be with all the other jobs in their respective batch;
8 If a job is forbidden to follow another job and start in that period in Y_i, then one of the jobs is removed;
9 Remove the last batch in Y_i until the duration of the schedule does not exceed p_{max};
10 $z(\Phi_i) \leftarrow f(\Phi_i)$;
11 $\mathcal{P}_k \leftarrow \mathcal{P}_k \cup \{X_i, Y_i, z(\Phi_i)\}$;
12 **end**
13 **while** η *seconds have not elapsed* **do**
14 **for** $[\beta|\mathcal{A}_l|]$ *individuals* **do**
15 $X_i \leftarrow$ randomly generated batch sequence for individual i containing all the jobs;
16 $Y_i \leftarrow$ the sequence obtained by repeating lines 6–10 to generate a new individual;
17 $\mathcal{A}_l \leftarrow \mathcal{A}_l \cup \{X_i, Y_i, z(\Phi_i)\}$;
18 **end**
19 Select $|\mathcal{P}|/2$ pairs of parents from $\mathcal{P}_k \cup \mathcal{A}_l$ using BTS, FPS or SUS;
20 $\mathcal{P}_o \leftarrow \emptyset$;
21 **for** *each of the* $|\mathcal{P}|/2$ *pairs of parents* **do**
22 $\mathcal{O} \leftarrow$ two offspring produced by the pair of parents using OPOX or PUOX;
23 **for** *each individual in* $\mathcal{O}$ **do**
24 $\kappa \leftarrow$ a random floating point number in the range $[0, 1)$;
25 **if** $\kappa < \alpha$ **then**
26 Mutate individual by performing $U(1, \alpha'_{max})$ moves on X_i of the individual;
27 **end**
28 $Y_i \leftarrow$ the sequence obtained by repeating lines 6–10 on the individual;
29 $\mathcal{P}_o \leftarrow \mathcal{P}_o \cup \{X_i, Y_i, z(\Phi_i)\}$;
30 $\mathcal{A}_l \leftarrow \mathcal{A}_l \cup \{X_i, Y_i, z(\Phi_i)\}$;
31 **end**
32 **end**
33 Remove the duplicate solutions from $\mathcal{A}_l$;
34 Remove all solutions from $\mathcal{A}_l$ that are dominated by one or more other solutions in $\mathcal{A}_l$;
35 $\mathcal{P}_s \leftarrow \mathcal{P}_k \cup \mathcal{P}_o$ sorted in descending order of their fitness value $z(\Phi_i)$ for each solution Φ_i in $\mathcal{P}_k \cup \mathcal{P}_o$;
36 $\mathcal{P}_{k+1} \leftarrow$ the first $|\mathcal{P}|$ individuals in $\mathcal{P}_s$;
37 $k \leftarrow k + 1$;
38 **end**
39 **return** $\mathcal{A}_l$;

3.7 Tabu search

Tabu search (TS) is an aggressive search method that was established by Glover [40, 41] in 1989. It is applied to solve a diverse group of combinatorial optimisation problems, such as the classical TSPs and graph colouring problems as well as problems in cluster analysis, space planning and scheduling [40]. The fundamental characteristic of TS that makes it unique compared to other local search methods is that it uses several intelligent techniques to find good solutions. These techniques include three types of memories that may be incorporated in the metaheuristic, aspiration criteria to avoid rejecting a solution in a neighbourhood that is better than the best found solution and accepting worse solutions to escape local optima.

The possible memory structures are short, intermediate and long-term memory [42]. Short-term memory involves a tabu list to store recently visited solutions or the reverse of recent moves that have been made, depending on the complexity of the problem to be solved. If the solution in the neighbourhood or the move that led to the solution appears in the tabu list, the search is not allowed to move to that solution. The tabu list requires one parameter before TS is executed, namely the tabu list size which is directly proportional to the size of the problem. It is therefore common practice to express the tabu list size as a percentage of, say, the number of variables in the programming model. In scheduling problems, it may be a percentage of the number of available jobs to be scheduled. Its size should be chosen cautiously as a small size causes TS to tend towards a local search while a large size leads to a random search. If an element is inserted into the list that results in the size of the list to be surpassed, the first element that was inserted is removed, *i.e.* the list follows the FIFO (first in, first out) principle.

Intermediate and long-term memory structures may be implemented for the purpose of local intensification and global diversification, respectively [40]. However, in many applications, only the use of short-term memory is sufficient to obtain good solutions [42]. Intermediate-term memory functions by storing attributes of solutions that have provided good results during a particular number of iterations. If any of the subsequent solutions contain a single, minority or majority of these attributes (depending on the level of intensification that is imposed), the search will restrict or penalise solutions that do not contain a predetermined number of these attributes. Glover [40] noted that intermediate-term memory is especially beneficial when solving large problems such as the TSP. Since the search aims its attention at finding solutions that have good value(s) for its objective function(s), it commonly considers only a subset of all the solutions that may be visited. In the TSP case, the search can eliminate edges that have not been in any or only in some of the solutions visited. The problem therefore becomes smaller and consequently more iterations can be performed to find good solutions faster.

On the other hand, long-term memory enforces concepts that are roughly the opposite than those in intermediate-term memory. The solution space is explored in regions that have not yet been extensively examined. Long-term memory is generally executed by generating biased random solutions in such a way that they are as far as possible from solutions that have already been visited. Since these randomly generated solutions are now far from recently visited solutions, the chances are small that the tabu list will contain elements restricting solutions in the near future to be visited. As a result, these new solutions will not be able to learn much from the past. Glover [40] again illustrated the utilisation of long-term memory when solving the TSP. The number of times that each edge appears in the solutions visited so far is stored in a long-term frequency based memory, contrasting to the short-term (recency based) memory that is the tabu list. Solutions can now be generated that contain the smallest value of a penalty function. The penalty function is defined in such a way as to allow most visited edges with a smaller probability and least visited edges with a great probability.

Aspiration criteria are used to allow good solutions although they may contain elements that are tabu active. The most common and straightforward aspiration criterion used in practice is to allow a solution that is better than the best found solution since the search commenced. An aspiration criterion closely related to the aforementioned one is to allow a solution if its objective function value(s) is/are within a certain range or percentage from that/those of the best found solution. A third criterion may be to give elements in the tabu list a weight each time a solution is considered based on the change in the objective function value(s) it will cause if the element is allowed. If it results in a great improvement in the objective function value(s), the solution may be allowed. Salhi [86] combined the two preceding ideas with the tabu status of a solution or move for a softer aspiration criterion indicating that it is possible to merge several aspiration concepts and that the search is not restricted to only a single aspiration criterion.

The basic structure of TS may be explained by the following steps [42]:

1. Select an initial solution in the solution space at random or obtain an initial solution from intermediate or long-term memory. Set the current solution to the initial solution.
2. Generate a subset of the neighbourhood around the current solution [43].
3. Construct a candidate list of moves/solutions from the subset and remove the candidates that contain tabu active elements and/or candidates that do not satisfy aspiration criteria.
4. Set the current solution to the best solution in the candidate list and add the solution or the move that led to the solution to the tabu list. If the chosen solution is superior to the best solution found thus far, update the best solution to be the chosen solution.
5. Remove items from the tabu list, if necessary. If a predetermined number of iterations have not been completed in total or since an improving solution to the best solution has been found, repeat steps 2–5.

A few concepts of TS should be noted from the aforementioned steps. Firstly, in some situations, it is inefficient to consider the entire neighbourhood (as in VNS) around the current solution in step 2. It may therefore be more appropriate to scan a portion of the neighbourhood to find promising regions of the solution space at a faster rate. This avoids spending too much time searching in neighbourhoods of solutions that are not yet satisfactory, especially in early iterations of the search. Secondly, the use of a tabu list avoids visiting recently discovered solutions again. It is also used to prevent cycling, *i.e.* indefinitely performing the same sequence of moves or visiting the same sequence of solutions. Cycling is also avoided by looking at a subset of the neighbourhood instead of the entire neighbourhood. Lastly, local optima are escaped by moving from a solution to a possibly worse solution in step 4. This occurs when the best solution in the candidate list is worse than the solution from which it was visited.

3.7.1 Neighbourhoods, tabu lists and aspiration criterion

The moves that are used in TS are the same as those used in VNS in §3.3.1, namely (1) swapping two batches in a schedule, (2) removing a batch from the schedule and inserting a batch containing unscheduled jobs, (3) moving a batch in the schedule from one position to another, (4) inserting a batch containing unscheduled jobs, (5) swapping two jobs from two different batches in the schedule, (6) removing a job from the schedule and inserting an unscheduled job, (7) moving a job from one batch to another in the schedule and (8) inserting an unscheduled job into the schedule. A subset $\bar{\mathcal{N}}$ of a neighbourhood is generated by performing one of the moves $|\bar{\mathcal{N}}|$ number of times and removing the duplicate solutions so that the size of the subset is at

most of size $|\bar{\mathcal{N}}|$. This method is more efficient than the time-consuming process of generating the entire neighbourhood after which a subset of solutions is taken from the neighbourhood.

Due to the large number of possible solutions that can be visited during the search, moves are made tabu instead of solutions. The chances of a move being made again is considerably higher than revisiting a solution. Furthermore, each neighbourhood structure maintains its own tabu list that is updated, if necessary, at the end of each iteration. This approach is pertinent to the considered problem, since the neighbourhood structures have their own set of rules when moves are made from one solution to another. A move that is made tabu is inserted at the end of its corresponding tabu list. As a means of diversifying the solutions that are visited, a population of $|\mathcal{P}|$ individuals search the solution space. Every individual consequently has its own set of tabu lists for the different neighbourhoods that can be visited. If, for instance, there are 8 different neighbourhood structures, a total of $8|\mathcal{P}|$ tabu lists are initiated and managed throughout the algorithm. The individuals' sets of tabu lists work independently of one another.

Tabu lists are updated by inserting the reverse (or a portion of the reverse) of the move that was made from the current solution to the chosen solution from the candidate list. Moves that are made tabu are explained using Figures 3.1–3.8 as examples.

1. If batches $\{8, 10\}$ and $\{2, 3, 7\}$ were swapped, then $\{\{8, 10\}, \{0.4, 0.4\}, \{2, 3, 7\}, \{0.2, 0.4, 0.2\}\}$ is inserted into the tabu list. This indicates that batch $\{8, 10\}$ with h_i values $\{0.4, 0.4\}$ is not allowed to be swapped with batch $\{2, 3, 7\}$ with h_i values $\{0.2, 0.4, 0.2\}$ until the move is removed from the tabu list.

2. If batch $\{1, 6\}$ is removed from the schedule, then $\{\{1, 6\}, \{0.2, 0.6\}\}$ is added to the tabu list for move 4. It shows that batch $\{1, 6\}$ with h_i values $\{0.2, 0.6\}$ is not allowed to be inserted into the sequence until the element is removed from the tabu list. Batch $\{1, 6\}$ is however allowed to be inserted with other h_i values. The insertion of batch $\{4, 12\}$ with h_i values $\{0.2, 0.8\}$ is ignored when making a move tabu, since such a restriction is considered in move 4.

3. If batch $\{1, 6\}$ is moved from the third position to the first position in the sequence, then $\{\{1, 6\}, \{0.2, 0.6\}, 3\}$ is inserted into the tabu list. It means that batch $\{1, 6\}$ with h_i values $\{0.2, 0.6\}$ is not allowed to be moved from its current position to the third position.

4. If batch $\{9, 11\}$ is inserted into the schedule, then $\{\{9, 11\}, \{0.8, 0.2\}\}$ is added to the tabu list for move 2. It shows that batch $\{9, 11\}$ with h_i values $\{0.8, 0.2\}$ is not allowed to be removed from the sequence.

5. If job 3 and 6 were swapped, then $\{3, 0.4, 6, 0.6\}$ is inserted into the tabu list. Job 3 with h_i value 0.4 is thus not allowed to be swapped with job 6 with h_i value 0.6.

6. If job 2 is removed from the schedule, then $\{1, 0.2\}$ is added to the tabu list for move 8. Job 2 with h_i value 0.2 is now not allowed to be inserted into the sequence. Similar to the second move, the job is however allowed to be inserted with another h_i value. The insertion of job 5 with h_i value 0.2 is ignored when making this move tabu, since the insertion of jobs is considered in move 8.

7. If job 1 is moved from the third batch to the first batch, then $\{1, 0.2\}$ is inserted into the tabu list. As a result, job 1 with h_i value 0.2 is not allowed to be reinserted into another batch.

8. Finally, if job 5 is inserted into the schedule, then $\{5, 0.2\}$ is added to the tabu list for move 6. This indicates that job 5 with h_i value 0.2 is not allowed to be removed from the sequence.

A move is removed from the tabu list if the tabu list is full, *i.e.* the tabu list contains $\tau + 1$ elements, and the move is the one that has been in the list the longest or after υ iterations have elapsed since the move was inserted into the tabu list, whichever comes first.

An aspiration criterion is used to allow a solution to be visited although the move that led to the solution may be tabu. If a solution in the subset of the neighbourhood of the current solution is better than the best found solution of the individual, then the best found solution is updated and the current solution is set to the best found solution. Hence, a solution that is better than the best found solution will never be rejected.

3.7.2 Framework of the algorithm

Algorithm 7 and Procedure 8 contain the pseudocode for TS. It requires the size of the population $|\mathcal{P}|$, the initial set of feasible solutions $\bar{\mathcal{P}}$ that is added to the external archive $\mathcal{A}_e$ during the initialisation of the algorithm, the computational time η at which the last iteration is performed, the maximum size $|\bar{\mathcal{N}}|$ of the subset of neighbours that are considered for the candidate list, the maximum number of iterations υ that a move stays in the tabu list and the tabu list size τ. Additionally, the metaheuristic requires p_{min} and p_{max} as input to determine the candidate solutions in Procedure 8. The fitness value of a solution is calculated using $z(\Phi_i) = f(\Phi_i)$. The justification for the additional parameter υ is because of the multiple neighbourhoods that can be visited in each iteration, opposed to the single neighbourhood structure considered in the basic form of TS. It is possible for elements to be in a neighbourhood structure's tabu list for too many iterations, leading to memory structures that are not only recency-based but also consider moves that have been made in much earlier iterations. To ensure that the memory structures remain recency-based, a restriction is put on the number of iterations that a move stays tabu.

Lines 1–9 in Algorithm 7 contains the initialisation phase of TS. The external archive $\mathcal{A}_e$ is set to the initial solutions that are generated such that every solution is feasible, contains at least two batches and has a duration that is at least as long as p_{min}. The next step in the initialisation phase is to set the current solution S^i_{cur} and the best found solution S^i_{best} for each individual i to solution i in $\bar{\mathcal{P}}$, that is $\bar{P}_i$. Concurrently, the tabu list $\mathcal{T}_{ij} = \{T_{ij1}, T_{ij2}, \ldots, T_{ij|\mathcal{T}_{ij}|}\}$ for each individual i and neighbourhood structure j is declared as an empty set. Additionally, the list $\mathcal{T}^{iter}_{ij} = \{T^{iter}_{ij1}, T^{iter}_{ij2}, \ldots, T^{iter}_{ij|\mathcal{T}^{iter}_{ij}|}\}$ containing the number of iterations that each of the elements are present in $\mathcal{T}_{ij}$ for each individual i and neighbourhood structure j, is set to an empty set.

Once the initialisation phase is complete, the iterations are performed in lines 10–48 before $\mathcal{A}_e$ is returned. In each of the iterations, a candidate list is generated for each individual. The first step to generate the candidate list is to determine the neighbourhood structures that can be used in lines 12–16. If jobs are allowed to run in parallel, neighbourhood structures 1–8 are allowed while neighbourhood structures 1–4 may be applied if jobs are not allowed to run in parallel. The neighbourhood structures in $\tilde{\mathcal{N}}$ are then shuffled after which the next current solution S^i_{cur} is determined for individual i in lines 18–30. Each of the neighbourhood structures in $\tilde{\mathcal{N}}$ are considered until a solution in the subset of the neighbourhood $\bar{\mathcal{N}}$ is found that is better than S^i_{best} or if the candidate list $\mathcal{C}$ is not empty, evident in lines 16 and 20 in Procedure 8.

The initialisation stage for each neighbourhood structure in $\tilde{\mathcal{N}}$ takes place in lines 19–21 of Algorithm 7. This includes generating $|\bar{\mathcal{N}}|$ solutions in the neighbourhood of S^i_{cur} by adding them to the set $\bar{\mathcal{N}}$, removing the duplicate solutions from $\bar{\mathcal{N}}$ and setting $\mathcal{C}$ and the set $\mathcal{C}'$ containing all the solutions whose moves are in the tabu list as empty sets. For each solution $\bar{N}_k$ in $\bar{\mathcal{N}} = \{\bar{N}_1, \bar{N}_2, \ldots, \bar{N}_{|\bar{\mathcal{N}}|}\}$, it is checked whether the move from the current solution to $\bar{N}_k$ is in the tabu list $\mathcal{T}_{ij}$ for individual i and neighbourhood structure j. If the move is not in the tabu list, the solution is added to the candidate list, otherwise the solution is added to $\mathcal{C}'$. Once

Algorithm 7: Tabu search

Input: $|\mathcal{P}|$, $\bar{\mathcal{P}}$, η, p_{min}, p_{max}, $|\bar{\mathcal{N}}|$, υ, τ
Output: A set of feasible nondominated solutions in the external archive found during the search

$\mathcal{A}_e \leftarrow \bar{\mathcal{P}}$;
for *each solution* $\bar{P}_i$ *in* $\bar{\mathcal{P}}$ **do**
  $S^i_{cur} \leftarrow \bar{P}_i$;
  $S^i_{best} \leftarrow \bar{P}_i$;
  for *each* j*th possible neighbourhood* **do**
    $\mathcal{T}_{ij} \leftarrow \emptyset$;
    $\mathcal{T}^{iter}_{ij} \leftarrow \emptyset$;
  end
end
while η *seconds have not elapsed* **do**
  for *each* i*th individual* **do**
    if *jobs are allowed to run in parallel* **then**
      $\tilde{\mathcal{N}} \leftarrow$ the set of neighbourhood structures 1–8;
    else
      $\tilde{\mathcal{N}} \leftarrow$ the set of neighbourhood structures 1–4;
    end
    Shuffle the set of neighbourhood structures $\tilde{\mathcal{N}}$;
    for *each of the neighbourhood structures in* $\tilde{\mathcal{N}}$ **do**
      $\bar{\mathcal{N}} \leftarrow$ maximum of $|\bar{\mathcal{N}}|$ solutions in the neighbourhood formed around S^i_{cur} by the neighbourhood structure j;
      $\mathcal{C} \leftarrow \emptyset$;
      $\mathcal{C}' \leftarrow \emptyset$;
      for *each solution* $\bar{N}_k$ *in* $\bar{\mathcal{N}}$ **do**
        if *the move from* S^i_{cur} *to* $\bar{N}_k$ *is not in* $\mathcal{T}_{ij}$ **then**
          $\mathcal{C} \leftarrow \mathcal{C} \cup \{\bar{N}_k\}$
        else
          $\mathcal{C}' \leftarrow \mathcal{C}' \cup \{\bar{N}_k\}$
        end
      end
      Do Procedure 8;
    end
    $j \leftarrow$ index of the neighbourhood structure for which a move from S^i_{cur} to C_{best} was made;
    if $|\mathcal{T}_{ij}| > \tau$ **then**
      Remove the first element from $\mathcal{T}_{ij}$;
      Remove the first element from $\mathcal{T}^{iter}_{ij}$;
    end
    for *each* j*th neighbourhood structure* **do**
      for *each* k*th element in* $\mathcal{T}^{iter}_{ij}$ **do**
        $T^{iter}_{ijk} \leftarrow T^{iter}_{ijk} + 1$;
        if $T^{iter}_{ijk} > \upsilon$ **then**
          Remove T_{ijk} from $\mathcal{T}_{ij}$;
          Remove T^{iter}_{ijk} from $\mathcal{T}^{iter}_{ij}$;
        end
      end
    end
  end
  Remove the duplicate solutions from $\mathcal{A}_e$;
  Remove all solutions from $\mathcal{A}_e$ that are dominated by one or more other solutions in $\mathcal{A}_e$;
end
return $\mathcal{A}_e$;

Procedure 8: Determine best candidate solution

1 **for** *each solution* C_k *in* $\mathcal{C}$ *and* $\mathcal{C}'$ **do**
2 Randomly remove one of the two jobs that are not allowed to be in the same batch for all the clashes until all jobs in C_k and C'_k are allowed to be with all the other jobs in their respective batch;
3 If a job is forbidden to follow another job and start in that period, then one of the jobs is removed;
4 Remove the last batch in C_k and C'_k until the duration of the schedules do not exceed p_{max};
5 **if** $\mu(C_k) < 2$ *or* $d(C_i) < p_{min}$ **then**
6 Remove C_k;
7 **end**
8 **end**
9 $C'_{best} \leftarrow$ null;
10 **for** *each solution* C'_k *in* $\mathcal{C}'$ **do**
11 **if** $z(C'_k) > z(C'_{best})$ **then**
12 $C'_{best} \leftarrow C'_k$;
13 **end**
14 $\mathcal{A}_e \leftarrow \mathcal{A}_e \cup C'_k$;
15 **end**
16 **if** $z(C'_{best}) > z(S^i_{best})$ **then**
17 $S^i_{cur} \leftarrow C'_{best}$;
18 $S^i_{best} \leftarrow C'_{best}$;
19 **break**;
20 **else if** $|\mathcal{C}| > 0$ **then**
21 $C_{best} \leftarrow$ null;
22 **for** *each solution* C_k *in* $\mathcal{C}$ **do**
23 **if** $z(C_k) > z(C_{best})$ **then**
24 $C_{best} \leftarrow C_k$;
25 **end**
26 $\mathcal{A}_e \leftarrow \mathcal{A}_e \cup C_k$;
27 **end**
28 **if** $z(C_{best}) > z(S^i_{best})$ **then**
29 $S^i_{best} \leftarrow C_{best}$;
30 **end**
31 $S^i_{cur} \leftarrow C_{best}$;
32 Append the reverse of the move from S^i_{cur} to C_{best} to $\mathcal{T}_{ij}$;
33 $\mathcal{T}^{iter}_{ij} \leftarrow \mathcal{T}^{iter}_{ij} \cup \{0\}$;
34 **break**;
35 **end**

each solution has been added to either $\mathcal{C}$ or $\mathcal{C}'$, Procedure 8 is performed to determine the best candidate solution.

Each solution in $\mathcal{C}$ and $\mathcal{C}'$ is made feasible in lines 1–8 of Procedure 8. The steps to obtain feasibility for the solutions are the same as those in VNS. The checking of whether the aspiration criterion holds takes place in lines 9–19. The best solution C'_{best} in $\mathcal{C}'$ is determined in lines 9–15. If the fitness value of C'_{best} is greater than the fitness value of S^i_{best}, then S^i_{cur} and S^i_{best} are updated to be C'_{best} and further neighbourhood structures in $\tilde{\mathcal{N}}$ are not considered. If the aspiration criterion is not satisfied, then the best solution C_{best} in $\mathcal{C}$ is determined in lines 21–27 if $\mathcal{C}$ contains solutions. If the fitness value of C_{best} is greater than the fitness value of S^i_{best}, then S^i_{best} is updated to be C_{best}. The current solution for individual i is updated to be C_{best} and the reverse of the move from the current solution to C_{best} is appended to tabu list $\mathcal{T}_{ij}$ according to the rules mentioned in §3.7.1. Lastly, a zero is appended to the list $\mathcal{T}^{iter}_{ij}$, indicating that the reverse of the move from S^i_{cur} to C_{best} has just been added to the tabu list, and further neighbourhood structures in $\tilde{\mathcal{N}}$ are not considered. Additionally, all solutions in $\mathcal{C}$ and $\mathcal{C}'$ are added to the external archive $\mathcal{A}_e$.

The tabu lists for individual i are now updated. If the number of solutions in the tabu list for the

neighbourhood structure j that was used to move from S^i_{cur} to C_{best} is more than the tabu list size τ, the element that has been in $\mathcal{T}_{ij}$ the longest is removed as well as its corresponding element in $\mathcal{T}^{iter}_{ij}$. Each element in the tabu lists for individual i have now been in their corresponding tabu list for an additional iteration. As a result, the number of each element in $\mathcal{T}^{iter}_{ij}$ is incremented by one for each neighbourhood structure j. If any of the elements have been in its tabu list for longer than υ iterations, it is removed from the tabu list as well as its corresponding element in $\mathcal{T}^{iter}_{ij}$. Once every individual in the population had its turn in the iteration, the external archive $\mathcal{A}_e$ is updated by removing the duplicate solutions from $\mathcal{A}_e$ in addition to the solutions that are dominated by one or more other solutions.

3.8 Non-identical start time batch algorithm

The non-identical start time batch (NSTB) algorithm was specifically designed for the problem considered in this thesis. It is executed after a population of solutions were obtained by one of the metaheuristics in § 3.3–3.6 to find solutions in which the jobs in a batch may start at different times. The algorithm takes a set of solutions $\bar{\mathcal{P}}$ with each solution given by

$$\bar{P}_i = \begin{bmatrix} \varphi_{i11} & \varphi_{i12} & \cdots & \varphi_{i1w} \\ \varphi_{i21} & \varphi_{i22} & \cdots & \varphi_{i2w} \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_{in1} & \varphi_{in2} & \cdots & \varphi_{inw} \end{bmatrix} \tag{3.64}$$

with

n the total number of jobs, including the derived jobs,
w the number of time periods over which the schedule can run,

as input to develop it into a population of solutions that is presented to the DM to decide which trade-offs should be made. There are four different steps in each iteration, namely the moving of single jobs within a batch, the merging of batches, the moving of multiple jobs within a batch and the insertion of a job into a batch, which may be found in § 3.8.1–3.8.4. The moving of single jobs within a batch is performed before each of the other three steps, bringing the total steps per iteration to six. Each of these steps, except the merging of batches, contains a predetermined number of subiterations that are carried out before proceeding to the following step. Further parameters that are needed to execute the algorithm are the computational time η at which the last iteration is performed, the number of subiterations η' in each of the steps (except for the merging of batches) and p_{max}. The pseudocode for the NSTB algorithm may be found in Algorithm 9.

The set of feasible solutions $\mathcal{P} = \{P_1, P_2, \ldots, P_{|\mathcal{P}|}\}$ is initially set to contain all the solutions in $\bar{\mathcal{P}}$, assuming that each solution is feasible and contains at least one batch that has more than one job, so that a population of solutions will evolve from the set of solutions. The set of solutions $\bar{\mathcal{P}}$ was obtained from one of the metaheuristics in § 3.3–3.7. Lines 3–16 represent η' attempts of moving a single job within a batch. In each attempt, a random index k is selected in the range $[1, |\mathcal{P}|]$ after which a random batch b in P_k containing at least two jobs is selected. A random job b' is then selected in batch b followed by the execution of Procedure 10 to obtain a set of feasible solutions $\mathcal{B}$ after a single job has been moved. In addition to the parameters P_k, b and b' required for Procedure 10, the start times $\mathbf{s}_i = [s^1_i, s^2_i, \ldots, s^{|\mathbf{s}_i|}_i]$ of the jobs in each batch i, the durations $\mathbf{t}_i = [t^1_i, t^2_i, \ldots, t^{|\mathbf{t}_i|}_i]$ of the jobs in each batch i and the duration b_i of each batch i are needed. The duration of each batch is calculated by

$$b_i = \begin{cases} \min_j\{s^j_{i+1}\} - \min_j\{s^j_i\}, & i = 1, \ldots, \mu - 1 \\ \max_j\{s^j_i + t^j_i\} - \min_j\{s^j_i\}, & i = \mu. \end{cases} \tag{3.65}$$

Algorithm 9: Non-identical start time batch algorithm

Input: $\bar{\mathcal{P}}$, η, η', p_{max}, w_1', w_2'
Output: A set of feasible solutions after the six steps have been performed for a predetermined number of times

1 $\mathcal{P} \leftarrow \bar{\mathcal{P}}$;
2 **while** η *seconds have not elapsed* **do**
3 **for** η' *subiterations* **do**
4 $k \leftarrow$ random integer in the range $[1, |\mathcal{P}|]$;
5 $\mathcal{Q} \leftarrow$ the set of batches in P_k containing at least two jobs;
6 $b \leftarrow$ a random batch in $\mathcal{Q}$;
7 $b' \leftarrow$ a random job in batch b;
8 $\mathcal{B} \leftarrow$ set of solutions obtained by performing Procedure 10 with all $\mathbf{s}_i$ and b_i in solution P_k;
9 **for** $i \leftarrow 1$ *to* $|\mathcal{B}|$ **do**
10 $z_i \leftarrow \sum_{j=1}^{\mu_i} \sum_{k=1}^{\pi_{ij}} p_i^{jk}$;
11 **end**
12 $B \leftarrow$ the solution obtained by FPS using fitness values for all z_i;
13 **if** B *not in* $\mathcal{P}$ **then**
14 $\mathcal{P} \leftarrow \mathcal{P} \cup \{B\}$;
15 **end**
16 **end**
17 **for** $k \leftarrow 1$ *to* $|\mathcal{P}|$ **do**
18 $B \leftarrow$ solution obtained by performing Procedure 11 with all $\mathbf{s}_i$, $\mathbf{t}_i$, b_i and p_i in solution P_k;
19 $P_k \leftarrow B$;
20 **end**
21 Repeat lines 3–20;
22 **for** η' *subiterations* **do**
23 $k \leftarrow$ random integer in the range $[1, |\mathcal{P}|]$;
24 $\mathcal{Q} \leftarrow$ the set of batches in P_k containing at least two jobs;
25 $b \leftarrow$ a random batch in $\mathcal{Q}$;
26 $B \leftarrow$ solution obtained by performing Procedure 12 with all $\mathbf{s}_i$ and b_i in solution P_k;
27 **if** B *not in* $\mathcal{P}$ **then**
28 $\mathcal{P} \leftarrow \mathcal{P} \cup \{B\}$;
29 **end**
30 **end**
31 Repeat lines 3–20;
32 **for** η' *subiterations* **do**
33 $k \leftarrow$ random integer in the range $[1, |\mathcal{P}|]$;
34 $\bar{\mathbf{a}} \leftarrow$ the available space in each of the batches;
35 **if** $\sum_{i=1}^{\mu_k} \bar{a}_i > 0.0$ **then**
36 $b \leftarrow$ the batch obtained by FPS using all fitness values $\bar{a}_i$;
37 $\mathcal{R} \leftarrow$ the set of jobs not present in P_k;
38 $\mathcal{T} \leftarrow$ the set of $\min\{4, |\mathcal{R}|\}$ random jobs in $\mathcal{R}$;
39 **for** *each job* j *in* $\mathcal{T}$ **do**
40 $\bar{p}_j \leftarrow \sum_{l=\min_j\{s_b^j\}}^{\min\{\min_j\{s_b^j\}+b_b-1,w\}} p_k^{bj} \Big/ \left(\min\{\min_j\{s_b^j\}+b_b-1, w\} - \min_j\{s_b^j\}+1\right)$;
41 $z_j \leftarrow \bar{p}_j$;
42 **end**
43 $b'' \leftarrow$ a job obtained by PTS using all fitness values z_j;
44 $\mathcal{B} \leftarrow$ set of solutions obtained by performing Procedure 14 with all $\mathbf{s}_i$ and b_i in solution P_k;
45 Repeat lines 9–15;
46 **end**
47 **end**
48 **end**
49 $\mathcal{P}_f \leftarrow$ the set of feasible solutions in $\mathcal{P}$ with duration at most p_{max};
50 **return** $\mathcal{P}_f$;

The duration of each batch, except the last one, is calculated by taking the minimum of the start times of the following batch's jobs and subtracting the minimum of the start times of the current batch's jobs. The duration of the last batch μ is calculated by taking the maximum of the finish times of its jobs, subtracting the minimum of the start times of its jobs and adding 1, given by

$$b_\mu = \max_j\{s_\mu^j + t_\mu^j - 1\} - \min_j\{s_\mu^j\} + 1$$

which simplifies to

$$b_\mu = \max_j\{s_\mu^j + t_\mu^j\} - \min_j\{s_\mu^j\}.$$

Once $\mathcal{B}$ is determined, the fitness value z_i of each solution i in $\mathcal{B}$ is calculated in lines 9–11 by

$$z_i = \sum_{j=1}^{\mu_i}\sum_{k=1}^{\pi_{ij}} p_i^{jk} \tag{3.66}$$

where

μ_i	is the number of batches in solution i,
π_{ij}	is the number of jobs in batch j in solution i and,
p_i^{jk}	is the profit for job k in batch j in solution i.

A solution B is then selected from $\mathcal{B}$ by FPS[7] using all fitness values z_i. It is then added to the population $\mathcal{P}$ if it is not yet present in it. Line 18 represents the merging of consecutive batches that have space available in them using Procedure 11. The solutions before the batches were merged are not lost from the population, since all the jobs' start times and durations remain the same after batches are merged. Single jobs are then attempted to be moved around again now that there is possibly more freedom for jobs to be moved around within batches if batches were merged.

The moving of multiple jobs within a batch may be found in lines 22–30. Jobs are removed and reinserted into a selected batch η' number of subiterations. Similar to when a single job was moved, the steps in each subiteration involves determining a random integer in the range $[1, |\mathcal{P}|]$ representing the solution that is chosen in the population $\mathcal{P}$, finding the set of batches $\mathcal{Q}$ in P_k containing at least two jobs and determining a random batch b in $\mathcal{Q}$ in which the jobs will be moved around. Once k and b have been determined, Procedure 12 is performed by providing solution P_k's start times $\mathbf{s}_i$ of the jobs, durations $\mathbf{t}_i$ of the jobs, durations b_i and profits p_i for each batch i. The obtained solution B is then added to the population $\mathcal{P}$ if it is not yet present in it. Single jobs are attempted to be moved around once again, as seen in line 31, now that there are new solutions in the population.

Lines 32–47 represent the η' attempts of inserting a job in a batch. For each of these subiterations, a random solution is chosen in the population after which the available space $\bar{\mathbf{a}} = [\bar{a}_1, \bar{a}_2, \ldots, \bar{a}_{\mu_k}]$ in each of the batches in the chosen solution is calculated by

$$\bar{a}_i = b_i - \sum_{j=1}^{\pi_i} h_i^j t_i^j \tag{3.67}$$

where h_i^j is the fraction of the machine that job j in batch i requires for the full period of implementation. If there is space available in any of the batches, a job will be inserted. A batch b that has space available is selected by FPS using all fitness values $\bar{a}_i$. Probabilistic tournament selection (PTS) is performed in lines 38–43 to select a job b'' to attempt to insert in batch b. This is done by selecting a set of $\min\{4, |\mathcal{R}|\}$ random jobs out of the set of jobs $\mathcal{R}$ that are

[7] FPS is explained in Figure 3.13 in §3.6.2 by substituting $z(\Phi_i)$ with z_i.

not present in P_k. For each job j in the set $\mathcal{T}$ of the selected jobs that will partake in the tournament, its average profit if it starts in any of the periods in a batch, is calculated by

$$\bar{p}_j = \sum_{l=\min_j\{s_b^j\}}^{\min\{\min_j\{s_b^j\}+b_b-1,w\}} p_k^{bj} \Bigg/ \left(\min\{\min_j\{s_b^j\}+b_b-1,w\} - \min_j\{s_b^j\}+1\right). \tag{3.68}$$

The fitness value of each of the jobs in $\mathcal{T}$ is now calculated by $z_j = \bar{p}_j$. Job b'' can now be determined by PTS using all fitness values z_j. PTS works by selecting the job with the highest value for z_j with probability $\tilde{p}$, the job with the second highest value for z_j with probability $\tilde{p}(1-\tilde{p})$, the job with the third highest value for z_j with probability $\tilde{p}(1-\tilde{p})^2$, and so on. Once job b'' has been selected, the set $\mathcal{B}$ of solutions is obtained by attempting to insert job b'' in batch b. Once $\mathcal{B}$ is determined, a solution B is selected from $\mathcal{B}$ (in the same way and for the same reason as when a single job is moved within a batch) and added to the population $\mathcal{P}$ if it is not yet present in it.

Once the iterations are completed in the algorithm, the set $\mathcal{P}_f$ of feasible solutions is determined from $\mathcal{P}$. Firstly, for each of the solutions in $\mathcal{P}$, the batches that start processing after p_{max} are removed from the schedule, starting from the last batch. The following four steps are then repeatedly performed until the duration of the schedule is at most p_{max}.

1. Remove the job that finishes last (of the jobs that do not start in the first period of the batch and finish after p_{max}).
2. Remove the jobs that are unable to follow another job as a result of the removal of the job in the first step[8]. Store these jobs in the set $\mathcal{E}$. Multiple jobs can be removed.
3. Attempt to insert all of the jobs in $\mathcal{E}$ one after the other, starting with the one that started the earliest when it was removed, until no more jobs can be inserted. Multiple jobs can be inserted and the jobs need to start in the period in which they started before they were removed. Remove the inserted jobs from $\mathcal{E}$.
4. Determine the durations of the jobs in the schedule due to the possible change in precedence. Jobs can follow other jobs, since jobs were removed and inserted that might have affected the setup times of the jobs in the schedule. Go to the first step.

If the final batch still finishes after the maximum allowable period once the aforementioned steps were completed, the batch is removed. This can happen when a job that starts processing first in the batch finishes after the maximum allowable periods. The final phase in the procedure is to check whether the capacity of the machine is exceeded in any of the periods or if there are jobs running in parallel that are not allowed to run in parallel. If any of these statements are true, the batch is discarded. The solution is appended to $\mathcal{P}_f$. Algorithm 9 then returns the set of feasible solutions.

3.8.1 Moving of a single job within a batch

The moving of a single job in Algorithm 9 may be found in Procedure 10. A set $\mathcal{C}$ is initially set to an empty set that represents the list of feasible solutions that are found by moving job b' in batch b to every possible position within batch b. The possible positions in which any job can start in batch b is in the domain $[\min_j\{s_b^j\}, \min\{\min_j\{s_b^j\}+b_i-1,w\}]$. That is to say, a job

[8]Note that if a job finishes last, it does not imply that it started last. Other jobs could have followed the job that was removed. Now that the job has been removed, other jobs are no longer able to follow this job.

may be moved between the first period of the batch and the last period of the batch, inclusive. However, if the batch finishes after the period w up until which the duration and profit of jobs can be calculated, jobs are not attempted to be started after period w in the remainder $\min_j\{s_b^j\} + b_i - w - 1$ periods of the batch.

At each start time i of the $\min\{\min_j\{s_b^j\} + b_i - 1, w\} - \min_j\{s_b^j\} + 1$ possible start times, the job is started at time period i, as seen in lines 3–6, where $\mathbf{s}'_j = [s_i'^1, s_i'^2, \ldots, s_i'^{|\mathbf{s}_i|}]$ is the new start times of the jobs in batch j. The checking of feasibility for the new solution is in lines 7–53.

1. Firstly, it is checked whether a job is being processed in the first period of the batch, by definition and a requirement to determine the durations of the jobs and the batches. If the minimum of the start times of the jobs in the new batch is different to that of the original batch, then a job is not being processed in the first period of the batch. If this is the case, feasibility has been violated and the job is started in the following position in the batch.

2. Secondly, the new durations of all the jobs in the schedule are calculated, given by $\mathbf{t}'_j = [t_i'^1, t_i'^2, \ldots, t_i'^{|\mathbf{t}_i|}]$. The duration of a job that doesn't start in the first period of the schedule is determined by following these four steps.

   (a) Go to the period in which the job starts and move backwards in time until a period that is not idle, say period υ, is found.

   (b) List all the jobs that are running during period υ and move to the period representing the minimum of their starting times, say period τ.

   (c) List all the jobs that are running from period τ to period υ. These are the possible jobs that the job may follow.

   (d) The job will then follow a job that causes the smallest setup time. If there is more than one job with the smallest setup time, then the job will follow the job that causes the smallest setup cost as a result of the maximisation of objective (3.3). The setup time is added to the normal duration of the job while the setup cost is subtracted from the job's normal profit.

   If any of the durations could not be calculated, the job is started in the following position in the batch. This will happen if there exists a job in the schedule that is not allowed to follow at least one job that is preceding it. This needs to be validated before the other feasibility checks, since the next check requires the durations of the jobs. The duration of a job cannot be calculated if it cannot follow a job in the schedule.

3. The next question that should be answered is whether all the jobs will be completed before the end of their batch. The length of the batch must stay the same so that jobs that start processing after the batch has been completed, is not shifted earlier or later in time. If jobs shift in time, their durations may be affected due to their time-dependent processing times. This will in turn cause a greater chance of the machine's capacity to be exceeded, jobs that are not allowed to follow any of the jobs preceding it, jobs not to be completed before the end of their batch and jobs running in parallel that are not allowed to run in parallel. The Boolean variable a is initially set to being false. It is false if all jobs finish before the end of their batch and true if at least one job finishes after the end of its batch. In lines 17–24, it is checked for each job in batches $b, b+1, \ldots, \mu$ whether it will be completed before the end of its batch. If there exists a job k in batch j for which its finish time $s_j'^k + t_j'^k - 1$ is greater than the finish time $\min_l\{s_j^l\} + b_i - 1$ of its batch, a is set to true and the job is started in the following position in the batch.

4. The following check involves testing whether the capacity of the machine will not be exceeded. The variable a is again initially set to false. In this case, it is true if the capacity of the machine is exceeded and false if it is not. The available space in each of the periods from the beginning of batch b to the end of the schedule is calculated, given by the vector $\mathbf{a} = [a_{\min_k\{s_j^k\}}, a_{\min_k\{s_j^k\}+1}, \ldots, a_{\max_k\{s_\mu'^k+t_\mu'^k\}-1}]$. The available space for all of the periods after batch b needs to be checked, since the durations of the jobs may change as a result of them possibly following another job that affects its setup time. The durations of the jobs that start before the commencement of batch b are not affected. The available space in period i is calculated by taking 1 (representing 100% of the period being available) and subtracting the percentages that the jobs occupy in that period. If any of the $\max_k\{s_\mu'^k + t_\mu'^k\} - \min_k\{s_b^k\}$ periods in the batch has negative space available, *i.e.* if the capacity of the machine is exceeded for any of the periods, a is set to true and the job is started in the following position in the batch.

5. The final question that needs to be answered is whether each of the jobs is allowed to run in parallel with all the jobs that are being processed during all the periods that the job is running for. The variable a is once again initially set to false. In this case, it is false if all jobs are allowed to run in parallel with one another for all the $\max_k\{s_\mu'^k + t_\mu'^k\} - \min_k\{s_b^k\}$ periods and true if there is at least one pair of jobs in the schedule running in parallel that are not allowed to run in parallel. All the sets $\mathcal{T}_j$ are initially set to empty sets, where $\mathcal{T}_j$ is the set of all the jobs that are being processed during period j, after which each of the jobs that are being processed during period j is appended to the set. If any of the $\binom{|\mathcal{T}_j|}{2}$ pairs of jobs in the sets containing at least two jobs are not allowed to run in parallel, a is set to true and the job is started in the following position in the batch.

If the algorithm reaches line 54 in iteration i, then it means that the solution is feasible if job b' in batch b starts at period i. The final step in the iteration is then to append the solution with all the new start times $\mathbf{s}_j'$ and new durations $\mathbf{t}_j'$ to the set of feasible solutions $\mathcal{C}$. After job b' is attempted to start in all of the positions in batch b, the set $\mathcal{C}$ is returned to Algorithm 9. The set $\mathcal{C}$ will be a non-empty set upon completion of Procedure 10, since the original solution with all start times $\mathbf{s}_j$ and durations $\mathbf{t}_j$ is feasible.

3.8.2 Merging of batches

The merging of batches in Algorithm 9 may be seen in Procedure 11. A set $\mathcal{C}$ is initially set to an empty set that represents the indices of the possible batches that can be merged with the batch after them. The available space in each of the batches are then determined. This is done by taking the batch's duration b_i and subtracting each job's fraction h_k of the facility that it occupies multiplied by its duration t_i^j. For each of the batches in the schedule, except the last one, it is checked whether there is space available in it and in the batch it precedes. If the condition holds, then the two consecutive batches are allowed to be merged and the index of the former batch is appended to the set $\mathcal{C}$. If no batches can be merged, the procedure is stopped so that the original solution is returned to Algorithm 9.

Batches that can be merged with more than one batch are identified in lines 12–19. The set $\mathcal{M}_i$ contains consecutive batches that have space available in them and can be merged with the batch following them with i being the first of the consecutive batches. If for example $\mathcal{C} = \{4, 5, 8, 10, 13, 14, 15, 16, 17\}$, then $\mathcal{M}_4 = \{4, 5\}$, $\mathcal{M}_8 = \{8\}$, $\mathcal{M}_{10} = \{10\}$ and $\mathcal{M}_{13} = \{13, 14, 15, 16, 17\}$. This means that batches 4, 8, 10 and 13 can merge with the batches following them, however, batch 5 can merge with batch 4 or batch 6, batch 14 can merge with batch 13 or batch 15, batch 15 can merge with batch 14 or batch 16, and so on. It needs to be decided with

Procedure 10: Moving of a single job within a batch

Input: P_k, b, b', all $\mathbf{s}_j$, all b_j
Output: A set of feasible solutions after a job has been moved within a batch

$\mathcal{C} \leftarrow \emptyset$ which is the list of feasible solutions returned in this procedure;
for $i \leftarrow \min_j\{s_b^j\}$ *to* $\min\{\min_j\{s_b^j\} + b_i - 1, w\}$ **do**
  for $j \leftarrow$ *1 to* μ **do**
    $\mathbf{s}'_j \leftarrow \mathbf{s}_j$;
  end
  $s_b'^{b'} \leftarrow i$;
  if $\min_j\{s_b^j\} \neq \min_j\{s_b'^j\}$ **then**
    continue;
  end
  for $j \leftarrow$ *1 to* μ **do**
    $\mathbf{t}'_j \leftarrow$ the new durations of the jobs in batch j in P_k;
  end
  if *any of the durations in* $\mathbf{t}'_j$ *for all batches could not be calculated* **then**
    continue;
  end
  $a \leftarrow$ false;
  for $j \leftarrow b$ *to* μ **do**
    for $k \leftarrow$ *1 to* π_j **do**
      if $s_j'^k + t_j'^k > \min_l\{s_j^l\} + b_i$ **then**
        $a \leftarrow$ true;
        break;
      end
    end
  end
  if $a =$ *true* **then**
    continue;
  end
  $a \leftarrow$ false;
  $\mathbf{a} \leftarrow$ the available space in each of the periods;
  for $j \leftarrow \min_k\{s_b^k\}$ *to* $\max_k\{s_\mu'^k + t_\mu'^k\} - 1$ **do**
    if $a_j < 0.0$ **then**
      $a \leftarrow$ true;
      break;
    end
  end
  if $a =$ *true* **then**
    continue;
  end
  $a \leftarrow$ false;
  for $j \leftarrow \min_k\{s_b^k\}$ *to* $\max_k\{s_\mu'^k + t_\mu'^k\} - 1$ **do**
    $\mathcal{T}_j \leftarrow \emptyset$;
    for *all the jobs* k *that is processed in period* j **do**
      $\mathcal{T}_j \leftarrow \mathcal{T}_j \cup \{k\}$;
    end
    for *each pair of jobs* k *and* l *that are not allowed to run in parallel* **do**
      if $k \in \mathcal{T}_j$ *and* $l \in \mathcal{T}_j$ **then**
        $a \leftarrow$ true;
      end
    end
  end
  if $a =$ *true* **then**
    continue;
  end
  $C \leftarrow$ the solution with all start times $\mathbf{s}'_j$ and durations $\mathbf{t}'_j$;
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\}$;
end
return $\mathcal{C}$;

which batch each of these batches will merge. This is determined in lines 20–37. It is possible that at least one of these batches will not merge with any batch during this procedure.

The set $\bar{\mathcal{M}}$ containing batches that are merged with the batch following them, is initially set to an empty set. For each of the sets $\mathcal{M}_i$, if the set contains only one batch (that is to say, the batch in the set can only be merged with the batch following it), then the batch is added to the set $\bar{\mathcal{M}}$ indicating that the batch will most definitely be merged with the batch following it. If the set $\mathcal{M}_i$ contains more than one batch, then there exists batches that can be merged with more than one batch. The batch j that provides the greatest profit if it would be merged with the batch following it, is added to the set $\bar{\mathcal{M}}$ meaning that batch j and $j+1$ will most definitely be merged. Since batch $j-1$ is now unable to merge with batch j due to batch j merging with batch $j+1$, it is removed from the set $\mathcal{M}_i$ if it exists in the set. Similarly, since batch $j+1$ is now unable to merge with batch $j+2$ due to batch $j+1$ merging with batch j, it is also removed from the set $\mathcal{M}_i$ if it exists in the set. Batch j is now removed from the set $\mathcal{M}_i$. This process of inserting batches into $\bar{\mathcal{M}}$ and removing batches from $\mathcal{M}_i$ is repeated until there are no more batches left to merge in the set $\mathcal{M}_i$. In the example in this section, if the profits p'_i of the batches in $\mathcal{C}$ are given by 23, 63, 82, 21, 36, 40, 31, 38 and 53, then batches 5 and 6, 8 and 9, 10 and 11, 14 and 15 as well as 17 and 18 will merge. Batches 13 and 16 do not merge with any batch, although they were listed in $\mathcal{M}_{13}$.

Lines 38–47 involve merging the batches by determining the start times and durations for the jobs in the batches that are merged. If batch j is merged with batch $j+1$, then batch j's start times and durations will now include batch $(j+1)$'s start times and durations as seen in lines 43–46. Once the solution's new start times $\mathbf{s}'_i$ and durations $\mathbf{t}'_i$ have been determined for all batches, they are returned to Algorithm 9.

3.8.3 Moving of multiple jobs within a batch

The moving of multiple jobs within a batch ensures diversity in the population, since all jobs are removed from a selected batch before being reinserted in the batch they were removed from. The pseudocode for the moving of jobs may be found in Procedure 12. A set $\mathcal{C}$ is initially set to contain all the jobs in the selected batch b with their h_i values (the fractions of the machine they occupy). A copy of all the start times $\mathbf{s}_i$ is put into all $\mathbf{s}'_i$ after which all the jobs in batch b are removed so that there are no start times present in batch b. The index i is now initialised to the first period in batch b after which all the jobs that were removed are attempted to be inserted into the batch, evident in lines 7–31, until all the jobs have been inserted, the last position of the batch has been reached or the last position in which a job may start has been reached. If i is not outside the bounds in which a job may be started, Procedure 13 is performed to determine the set of jobs that can start in period i.

In the first line of Procedure 13, a set $\mathcal{D}$ containing all the jobs that can be started in period i, is initially set to an empty set. A copy of all the start times $\mathbf{s}'_k$ is put into all $\mathbf{s}''_k$. The difference between all $\mathbf{s}'_k$ and all $\mathbf{s}''_k$ is that all $\mathbf{s}'_k$ are the final start times that will be returned in Procedure 12 if the solution that is formed is feasible and all $\mathbf{s}''_k$ are the start times that are tested for feasibility while jobs are attempted to be inserted in Procedure 13. Period i is now included with the start times $\mathbf{s}'_b$ symbolising the commencement of a job from $\mathcal{C}$ at period i in batch b. Each job j that was removed but not yet inserted back into the batch, is attempted to start in period i, as seen in lines 6–37. It is checked whether the solution produced by inserting job j, is feasible. Jobs are inserted in chronological order, starting at the first period in the batch. If jobs are not inserted in chronological order, the durations of the jobs will be influenced as a result of setup times. This will greatly reduce the chances of finding a feasible solution at the end of the procedure.

Procedure 11: Merging of batches

Input: P_k, all $\mathbf{s}_i$, all $\mathbf{t}_i$, all p_i
Output: A feasible solution after batches have been merged

$\mathcal{C} \leftarrow \emptyset$ which contains the indices of the possible batches that can be merged with the batch after them;
$\bar{\mathbf{a}} \leftarrow$ the available space in each of the batches;
for $i \leftarrow$ *1 to* $\mu - 1$ **do**
  if $\bar{a}_i > 0$ *and* $\bar{a}_{i+1} > 0$ **then**
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{i\}$;
  end
  $p'_i \leftarrow p_i + p_{i+1}$;
end
if $\mathcal{C} = \emptyset$ **then**
  return P_k;
end
$j \leftarrow$ the first element of $\mathcal{C}$;
for *each element* i *in* $\mathcal{C}$ **do**
  if $i - 1 \notin \mathcal{C}$ **then**
    $\mathcal{M}_i \leftarrow \emptyset$;
    $j \leftarrow i$;
  end
  $\mathcal{M}_j \leftarrow \mathcal{M}_j \cup \{i\}$;
end
$\bar{\mathcal{M}} \leftarrow \emptyset$;
for *each* $\mathcal{M}_i$ **do**
  if $|\mathcal{M}_i| = 1$ **then**
    $\bar{\mathcal{M}} \leftarrow \bar{\mathcal{M}} \cup \mathcal{M}_i$
  else
    while $|\mathcal{M}_i| \neq 0$ **do**
      $j \leftarrow$ the batch in $\mathcal{M}_i$ with the highest profit p'_k;
      $\bar{\mathcal{M}} \leftarrow \bar{\mathcal{M}} \cup \{j\}$;
      if $j - 1 \in \mathcal{M}_i$ **then**
        Remove $j - 1$ from $\mathcal{M}_i$;
      end
      if $j + 1 \in \mathcal{M}_i$ **then**
        Remove $j + 1$ from $\mathcal{M}_i$;
      end
      Remove j from $\mathcal{M}_i$;
    end
  end
end
for $i \leftarrow$ *1 to* μ **do**
  $\mathbf{s}'_i \leftarrow \mathbf{s}_i$;
  $\mathbf{t}'_i \leftarrow \mathbf{t}_i$;
end
for *each element* i *in* $\bar{\mathcal{M}}$ *starting with the last element* **do**
  $\mathbf{s}'_i \leftarrow \mathbf{s}'_i \frown \mathbf{s}'_{i+1}$;
  $\mathbf{t}'_i \leftarrow \mathbf{t}'_i \frown \mathbf{t}'_{i+1}$;
  Remove $\mathbf{s}'_{i+1}$;
  Remove $\mathbf{t}'_{i+1}$;
end
$C \leftarrow$ the solution with all start times $\mathbf{s}'_i$ and durations $\mathbf{t}'_i$;
return C;

For each job, the Boolean variable a is initially set to being true. It is true if the resulting solution is feasible and false if it is not. The new durations $\mathbf{t}''_k$ of the jobs in batches $k = 1, 2, \ldots, b$ are calculated. The durations for the jobs in batches $b+1, b+2, \ldots, \mu$ can not yet be calculated, since batch b is not yet completed. If any of these durations could not be calculated (line 11), it means that job j is not allowed to follow at least one job that is preceding it. If this is the case, the next job in $\mathcal{C}$ will be attempted to start in period i. Otherwise, the following three feasibility steps will be performed.

1. It is checked whether job j will be completed before the end of batch b (line 12), *i.e.* if its completion time $i + t_b^{''|\mathbf{t}''_b|} - 1$ is no later than the batch's completion time $\min_k\{s_b^k\} + b_i - 1$. If the job finishes after the batch, indicating that the solution is now infeasible, the next job in $\mathcal{C}$ is attempted to start in period i. Otherwise, the next feasibility step is performed.

2. In this step, it is checked whether the capacity of the machine will not be exceeded for the periods during which the job is running (lines 13–18). The available space in each of the periods $i, i+1, \ldots, i + t_b^{''|\mathbf{t}''_b|} - 1$ are calculated. If any of these periods have negative space available, *i.e.* the capacity of the machine is exceeded for that period, then a is set to false and the next job in $\mathcal{C}$ is attempted to start in period i. Otherwise, the last feasibility step is performed.

3. The final step checks whether the job is allowed to run in parallel with all the jobs that are being processed during all the periods that the job is running for. Again, $\mathcal{T}_k$ is the set of all the jobs that are being processed during period k. If any of the $\binom{|\mathcal{T}_k|}{2}$ pairs of jobs (assuming the set contains at least two jobs) are not allowed to run in parallel, a is set to false and the next job in $\mathcal{C}$ is attempted to start in period i. Otherwise, the solution is feasible and job j is added to $\mathcal{D}$.

The possibly empty set $\mathcal{D}$ is returned to Procedure 12 by Procedure 13 once all the jobs in $\mathcal{C}$ were attempted to start in position i.

Once the set $\mathcal{D}$ is returned to Procedure 12, it is checked whether it is empty. If it is indeed empty, none of the jobs in $\mathcal{C}$ will be able to start in period i. The algorithm then moves on to the next period in the schedule by incrementing i by one. If there is only one job in $\mathcal{D}$, that job is selected by setting $\bar{b}'$ to that job. If there is more than one job in $\mathcal{D}$, FPS is used to determine which job $\bar{b}'$ will be started in period i. The fitness value z_j of each job j in $\mathcal{D}$ is calculated in line 16 by $z_j = p_j$ where p_j is the profit of job j if it starts in period i.

Once job $\bar{b}'$ is chosen, its start time is added to $\mathbf{s}'_b$ after which it is removed from $\mathcal{C}$ as it can only be inserted into batch b once. The procedure remains in period i until no more jobs from $\mathcal{C}$ can be started in period i. The Boolean variable c is initially set to true. It is true if the solution is feasible and false if it is not. The durations of the jobs in each of the batches in the schedule are determined after which the last four feasibility steps in Procedure 10 are performed on the solution. The first feasibility step in Procedure 10 is unnecessary in this case as a job will always start in the first period in the batch. If any of the feasibility steps fail, c is set to false indicating that the solution is infeasible. Once the feasibility steps are completed, the generated solution is returned to Algorithm 9 if it is feasible. The solution that was used as input for Procedure 12 is returned back to Algorithm 9 if the generated solution is infeasible.

3.8.4 Insertion of a job into a batch

The procedure for the insertion of a job is similar to the moving of a single job within a batch. The pseudocode for this process may be found in Procedure 14. In this case the set $\mathcal{C}$ is a set

Procedure 12: Moving of multiple jobs within a batch

Input: P_k, b, all $\mathbf{s}_i$, all b_i
Output: A feasible solution after multiple jobs within a batch have been moved

1 $\mathcal{C} \leftarrow$ the set of jobs in batch b;
2 **for** $i \leftarrow 1$ *to* μ **do**
3 $\quad \mathbf{s}'_i \leftarrow \mathbf{s}_i$;
4 **end**
5 $\mathbf{s}'_b \leftarrow [\,]$;
6 $i \leftarrow \min_j\{s_b^j\}$;
7 **while** *true* **do**
8 $\quad$ **if** $|\mathcal{C}| = 0$ *or* $i \geq \min_k\{s_b^k\} + b_i$ *or* $i > w$ **then**
9 $\quad\quad$ **break**;
10 $\quad$ **else**
11 $\quad\quad$ $\mathcal{D} \leftarrow$ the set which contains all the jobs that can be started in period i obtained by performing Procedure 13;
12 $\quad\quad$ **if** $|\mathcal{D}| > 0$ **then**
13 $\quad\quad\quad$ **if** $|\mathcal{D}| = 1$ **then**
14 $\quad\quad\quad\quad$ $\bar{b}' \leftarrow$ the only job in $\mathcal{D}$;
15 $\quad\quad\quad$ **else**
16 $\quad\quad\quad\quad$ **for** *each job* j *in* $\mathcal{D}$ **do**
17 $\quad\quad\quad\quad\quad$ $z_j \leftarrow p_j$;
18 $\quad\quad\quad\quad$ **end**
19 $\quad\quad\quad\quad$ $\bar{b}' \leftarrow$ the job obtained by FPS using all fitness values z_j;
20 $\quad\quad\quad$ **end**
21 $\quad\quad\quad$ $\mathbf{s}'_b \leftarrow \mathbf{s}'^{\frown}_b[i]$;
22 $\quad\quad\quad$ Remove job $\bar{b}'$ from $\mathcal{C}$;
23 $\quad\quad$ **else**
24 $\quad\quad\quad$ $i \leftarrow i + 1$;
25 $\quad\quad$ **end**
26 $\quad$ **end**
27 **end**
28 $c \leftarrow$ true;
29 **for** $i \leftarrow 1$ *to* μ **do**
30 $\quad$ $\mathbf{t}'_i \leftarrow$ the new durations of the jobs in batch i;
31 **end**
32 Execute lines 13–53 in Procedure 10 replacing the "**continue**" statements with "$c \leftarrow$ false";
33 **if** $c =$ *true* **then**
34 $\quad$ $C \leftarrow$ the solution with all start times $\mathbf{s}'_i$ and durations $\mathbf{t}'_i$;
35 $\quad$ **return** C;
36 **else**
37 $\quad$ **return** P_k;
38 **end**

Procedure 13: Jobs that can be started in a given period

Input: $\mathcal{C}$, i, b, all $\mathbf{s}'_k$, all b_k
Output: A set of jobs that can be started in period i

$\mathcal{D} \leftarrow \emptyset$ which contains all the jobs that can be started in period i;
for $k \leftarrow$ *1 to* b **do**
  $\mathbf{s}''_k \leftarrow \mathbf{s}'_k$;
end
$\mathbf{s}''_b \leftarrow \mathbf{s}''_b{}^{\frown}[i]$;
for *each job* j *in* $\mathcal{C}$ **do**
  $a \leftarrow$ true;
  for $k \leftarrow$ *1 to* b **do**
    $\mathbf{t}''_k \leftarrow$ the durations of the jobs in batch k after job j in $\mathcal{C}$ has been inserted;
  end
  if *all of the durations in* $\mathbf{t}''_k$ *for* $k = 1, 2, \ldots, b$ *could be calculated* **then**
    if $i + t_b^{''|\mathbf{t}''_b|} \leq \min_k\{s_b^k\} + b_i$ **then**
      $\mathbf{a} \leftarrow$ the available space in each of the periods using all $\mathbf{s}''_l$, $\mathbf{t}''_l$ and b_l;
      for $k \leftarrow i$ *to* $i + t_b^{''|\mathbf{t}''_b|} - 1$ **do**
        if $a_k < 0.0$ **then**
          $a \leftarrow$ false;
        end
      end
      if $a =$ *true* **then**
        for $k \leftarrow i$ *to* $i + t_b^{''|\mathbf{t}''_b|} - 1$ **do**
          $\mathcal{T}_k \leftarrow \emptyset$;
          for *each job* l *that is processed in period* k **do**
            $\mathcal{T}_k \leftarrow \mathcal{T}_k \cup \{l\}$;
          end
          for *each pair of jobs* l *and* m *that are not allowed to run in parallel* **do**
            if $l \in \mathcal{T}_k$ *and* $m \in \mathcal{T}_k$ **then**
              $a \leftarrow$ false;
            end
          end
        end
        if $a =$ *true* **then**
          $\mathcal{D} \leftarrow \mathcal{D} \cup \{j\}$;
        end
      end
    end
  end
end
return $\mathcal{D}$;

containing the feasible solutions that was found after attempting to start job b in every available period in batch b''. It is initially set to an empty set, as seen in line 1. Job b is now attempted to be started in each of the periods in batch b'', evident in lines 3–6. Once the start times of the jobs in the batch have been determined, the durations of the jobs in all of the batches in the schedule are calculated. The last four feasibility steps mentioned in §3.8.1 are now performed on the resulting solution to determine whether it is feasible or not. The pseudocode for this process is exactly as in lines 13–53 in Procedure 10. If the solution is feasible, lines 11 and 12 will be executed so that the resulting solution is appended to $\mathcal{C}$. The set $\mathcal{C}$ can in this case be an empty set upon completion of Procedure 14. This will happen if all of the resulting solutions were infeasible after attempting to insert the job in the batch. If $\mathcal{C}$ is an empty set, the original solution P_k will be returned to Algorithm 9 in a set. If $\mathcal{C}$ contains at least one solution, it will be returned to Algorithm 9.

Two methods that are used throughout the algorithm to select solutions, batches and jobs are FPS and PTS. FPS is used when the number of solutions/batches/jobs of which one should be selected, is relatively small. This occurs at four different instances in the algorithm.

1. A solution in $\mathcal{B}$ must be selected after moving a single job within a batch. The argument is that the chances are great that the job can only start at a limited number of places in the batch that will result in a feasible solution. The size $|\mathcal{B}|$ will therefore be small so that it won't be inefficient to determine the fitness value for each of the solutions in $\mathcal{B}$.

2. A job must be selected from $\mathcal{D}$ to start in a period when removing jobs from a batch and inserting them again in Procedure 12. Every time a job is inserted, $|\mathcal{D}|$ decreases by one so that it is acceptable to use FPS instead of PTS.

3. When inserting a job into a batch in Procedure 14, a batch needs to be selected in which the job will be attempted to be inserted. Since there is a limited number of batches that a schedule can contain and the number of batches in a schedule will decrease after batches have been merged in Procedure 11, it is acceptable to use FPS.

4. The last occurrence of FPS is when a solution in $\mathcal{B}$ must be selected after inserting a job into a batch. FPS is used for the same reasons mentioned when a solution must be selected after moving a single job within a batch.

On the other hand, PTS is used when the number of objects of which one should be selected, is relatively large. There is only one instance in which PTS is necessary to ensure that the algorithm is efficient. This is when a job needs to be selected from the set of unscheduled jobs $\mathcal{R}$ in Procedure 14. The size of the set $|\mathcal{R}|$ can grow substantially large over the implementation of several schedules if the acceptance rate is low and a large number of unscheduled jobs are still considered in the schedule after the current schedule.

Procedure 14: Insertion of a job into a batch

Input: P_k, b, b'', all $\mathbf{s}_i$, all b_i
Output: A set of feasible solutions after a job has been inserted into a batch

```
1  C ← ∅ which is the list of feasible solutions returned in this procedure;
2  for i ← min_j{s_b^j} to min{min_j{s_b^j} + b_i − 1, w} do
3  |   for j ← 1 to μ do
4  |   |   s'_j ← s_j;
5  |   end
6  |   s_b'^{b''} ← s_b'^{b''}⌢[i];
7  |   for j ← 1 to μ do
8  |   |   t'_j ← the new durations of the jobs in batch j in P_k;
9  |   end
10 |   Execute lines 13–53 in Procedure 10;
11 |   C ← the solution with all start times s'_j and durations t'_j;
12 |   C ← C ∪ {C};
13 end
14 if |C| = ∅ then
15 |   return {P_k};
16 else
17 |   return C;
18 end
```

3.9 Chapter summary

A variety of metaheuristics were introduced in this chapter to solve the considered scheduling problem. Integer programming models were given in §3.1 to model two variations of the scheduling problem, that is when a job's processing time depends only on its start time and when a job's processing time depends on its start time and the preceding job. Preliminary theory necessary for the developing of the metaheuristics were derived in §3.2. Three metaheuristics that use moves and/or neighbourhoods were developed, namely variable neighbourhood search, tabu search and a genetic algorithm in which a chromosome is defined as a sequence of batches. Furthermore, three metaheuristics that utilise decoding schemes were established. These algorithms are particle swarm optimisation, cuckoo search and a genetic algorithm where a chromosome is defined as a string of floating point numbers. A local or external archive was used in each metaheuristic so that a set of solutions is provided to the non-identical start time batch algorithm. This is an algorithm that was designed to address the problem that jobs in a batch are allowed to start at different times. A job has the freedom to move around in its batch provided that the resulting solution is feasible. Its freedom is, however, limited due to the many feasibility steps that need to be performed as a result of the complexity of the problem.

CHAPTER 4

Results

Contents

This chapter starts off by explaining the method that was used to generate test cases. The parameters for the metaheuristics in Chapter 3 are then calibrated to determine the set of parameters that provide the best solutions. Once the parameters have been chosen for each metaheuristic, convergence graphs are plotted and analysed in §4.3 to determine the time limit η for the metaheuristics as well as the time limit and number of subiterations η' for the NSTB algorithm. Algorithms are compared to one another in §4.4 using different measurements. This section includes results obtained by CPLEX using the exact formulations. Finally, hypothesis testing is performed in §4.5 to find the algorithm that performs the best in each of the 150 generated test cases from §4.1. Both non-parametric and parametric statistics (including testing for normality and equal variances) are included in this section.

4.1 Test cases

Test cases are generated to test all the algorithms developed in Chapter 3 against one another due to the unavailability of real-world data. Several trigonometric, linear, quadratic and logistic functions are used to construct processing time and profit matrices. In order to generate the matrices, two independent random integer variables g_1 and g_2 are drawn from the uniform distribution in the interval $[g'_{min}, g'_{max}]$ where g'_{min} and g'_{max} are the minimum and maximum values, respectively, that the processing time or profit of the jobs in a schedule is allowed to be for the trend. The variables g_{min} and g_{max} are then determined for each of the jobs by the equations

$$g_{min} = \min\{g_1, g_2\} \text{ and} \tag{4.1}$$
$$g_{max} = \max\{g_1, g_2\} \tag{4.2}$$

indicating the minimum and maximum values for the trend of the processing time or profit. All the jobs in the schedule will then use one of the trends given by equations

$$g_1(x) = \frac{g_{max} - g_{min}}{2} \sin\left(\frac{2\pi}{w'}x\right) + \frac{g_{max} + g_{min}}{2}, \tag{4.3}$$
$$g_2(x) = \frac{g_{max} - g_{min}}{2} \cos\left(\frac{2\pi}{w'}x\right) + \frac{g_{max} + g_{min}}{2}, \tag{4.4}$$
$$g_3(x) = \frac{g_{min} - g_{max}}{2} \sin\left(\frac{2\pi}{w'}x\right) + \frac{g_{max} + g_{min}}{2}, \tag{4.5}$$
$$g_4(x) = \frac{g_{min} - g_{max}}{2} \cos\left(\frac{2\pi}{w'}x\right) + \frac{g_{max} + g_{min}}{2}, \tag{4.6}$$
$$g_5(x) = \frac{g_{max} - g_{min}}{w'}x + g_{min}, \tag{4.7}$$
$$g_6(x) = \frac{g_{min} - g_{max}}{w'}x + g_{min}, \tag{4.8}$$

$$g_7(x) = \frac{g_{max} - g_{min}}{w'^2}x^2 + g_{min}, \tag{4.9}$$
$$g_8(x) = \frac{g_{min} - g_{max}}{w'^2}x^2 + \frac{2(g_{min} - g_{max})}{w'}x + g_{min}, \tag{4.10}$$
$$g_9(x) = \frac{g_{min} - g_{max}}{w'^2}x^2 + g_{max}, \tag{4.11}$$
$$g_{10}(x) = \frac{g_{max} - g_{min}}{w'^2}x^2 + \frac{2(g_{min} - g_{max})}{w'}x + g_{max}, \tag{4.12}$$
$$g_{11}(x) = \frac{g_{max} - g_{min}}{1 + e^{-(x - \frac{w'}{2})}} + g_{min}, \tag{4.13}$$
$$g_{12}(x) = \frac{g_{min} - g_{max}}{1 + e^{-(x - \frac{w'}{2})}} + g_{max}, \tag{4.14}$$

for both the processing time and profit matrices where x is the time period and w' is the number of time periods for the matrices. The functions $g_1(x), \ldots, g_{12}(x)$ represent a sine wave, cosine wave, negative sine wave, negative cosine wave, positive linear function, negative linear function, convex increasing quadratic function, concave increasing quadratic function, concave decreasing quadratic function, convex decreasing quadratic function, increasing logistic function and a decreasing logistic function in the order listed. Two independent random integer variables v_t and v_p representing the variation in the processing time and profit, respectively, is then drawn

from the uniform distribution in the interval $[-\bar{v_t}, \bar{v_t}]$ and $[-\bar{v_p}, \bar{v_p}]$ for a total of w' times each and added to each of the data points. The parameters $\bar{v_t}$ and $\bar{v_p}$ are the maximum deviation from the actual value of $g_j(x)$ for the processing time and profit matrices, respectively. Figures 4.1–4.12 show examples for each of the trends where $g_{min} = 10$, $g_{max} = 30$ and $v_t = v_p = 2$.

The parameters needed to determine the h_i values for the jobs are the number of jobs that can run in parallel out of the total n' number of jobs, the maximum number of derived jobs per job and the increment of the fractions that the derived jobs occupy of the facility[1]. All the possible h_i values are determined for the jobs using the last mentioned parameter. For each of the selected jobs that can run in parallel, a nonempty subset of the h_i values are independently assigned to those jobs. If, for example, the increment of the fractions that the derived jobs occupy of the facility is 0.3, then the possible h_i values for the jobs are 0.3, 0.6 and 0.9 if we exclude $h_i = 1$. Consequently, the possible nonempty subsets of the h_i values for the jobs are $\{0.3\}$, $\{0.6\}$, $\{0.9\}$, $\{0.3, 0.6\}$, $\{0.3, 0.9\}$, $\{0.6, 0.9\}$ and $\{0.3, 0.6, 0.9\}$. One of these 7 subsets is then selected for each of the jobs that can run in parallel. Jobs that run in parallel will take longer to complete due to them only taking up a fraction of the facility, but may be more profitable in the long run due to more jobs that are able to be processed. To take these factors into account, the processing time and profit of the derived jobs are updated by incrementally increasing their processing time and profit as the h_i value decreases. This integer increment is randomly determined using the uniform distribution with a lower bound of 1 and an upper bound of a specified maximum increment for the processing time and profit.

The parameters needed for the setup times and setup costs are the number of different configuration setup subsets (which is equal to 1 if the processing time of a job does not depend on the previous job), the maximum setup time $\bar{s}$ and the maximum setup cost $\bar{c}$. The n' jobs are placed into the different subsets after which the subsets are randomly sequenced. For each of the two consecutive subsets, the job in the second subset of all the pairs of jobs get the same setup time and setup cost that are generated from the uniform distribution in the intervals $[0, \bar{s}]$ and $[0, \bar{c}]$, respectively. If, for example, there are 8 jobs and the sequence of the configuration setup subsets is given by $\{\{1, 3\}, \{2, 5, 7\}, \{4, 6, 8\}\}$ then the 6 pairs of jobs for the first two consecutive subsets will have the same setup time and setup cost assigned to the jobs in the second subset and the 9 pairs of jobs for the last two consecutive subsets will have its own setup time and setup cost for the jobs in the third subset, also the same for all 9 pairs. The final step in generating a test case is to generate a list containing pairs of jobs with a time period in which the second job is forbidden to follow the first job in that time period. As an example, the list $\{\{2, 1, 3\}, \{6, 7, 9\}\}$ says that job 1 is prohibited from following job 2 in time period 3 and job 7 is prohibited from following job 6 in time period 9. Secondly, a list is generated that contains all the pairs of jobs that are not allowed to be in the same batch or run in parallel, *e.g.* the list $\{\{1, 7\}, \{3, 4\}\}$ says that jobs 1 and 7 as well as jobs 3 and 4 are not allowed to run in parallel.

A total of 150 test cases were generated for the purpose to test the algorithms against one another. The test cases are divided into three groups of difficulties, namely elementary, intermediate and challenging test cases. The number of available jobs n' ranges from 10 to 60 jobs while the maximum number of periods $w = p_{max}$ that the schedule is allowed to be ranges from 15 to 210 periods. The number of jobs that can run in parallel with other jobs is either 0 (to test the performance of the algorithms when jobs are not allowed to run in parallel, *i.e.* when the h_i values of all the jobs is 1) or 60% of the total number of available jobs. The number of different configuration setup subsets ranges from 1 to 5. The maximum setup time $\bar{s}$ is varied from 1 to 5 while the maximum setup cost $\bar{c}$ is varied from R3 to R26. The fractions h_i that the derived jobs occupy of the facility ranges from 1/3 to 1 for easy problems and ranges from 1/6 to 1 for difficult problems. Consequently, $n' \leq n \leq n' + 0.6(b_{max} - 1)n' = 2.2n'$ for easy

[1]This value must be less than or equal to 0.5 to allow for the possibility of more than one derived job per job.

problems ($b_{max} = 3$ for the easiest generated problems in which jobs can run in parallel) and $n' \leq n \leq n' + 0.6(b_{max} - 1)n' = 4n'$ for difficult problems ($b_{max} = 6$ for the most difficult generated problems in which jobs can run in parallel) where b_{max} is the maximum number of jobs that can run simultaneously at a given time (which is the reciprocal of the minimum h_i value). The number of available jobs n (that includes the derived jobs) therefore ranges from 10 to 240 for the 150 generated test cases. The trend functions $g_1(x), g_2(x), \ldots, g_{12}(x)$ are used one after the other so that each trend function is in approximately the same number of test cases in each of the three groups of difficulties. Lastly, the number of pairs of jobs in which a job is forbidden to follow another job as well as the number of pairs of jobs that are not allowed to be in the same batch (or run in parallel) ranges from 0 to 44.

The exact formulation of the 150 test cases were entered into AIMMS 4.28.3 [2] and solved using the CPLEX 12.6.3 Optimiser (short for IBM ILOG CPLEX Optimisation Studio) [48]. It was able to find the optimal solution in less than an hour for only 23 of the test cases and a solution (mostly far from being near-optimal) after an hour has elapsed for 52 of the test cases before running out of memory. This confirms that the use of metaheuristics is appropriate for the considered type of scheduling problem, since using commercial software to solve the exact formulations gives unsatisfactory results. The information and results of the 75 test cases that CPLEX could find a solution for may be found in Appendix A.

The 150 test cases were then relaxed to determine an upper bound for the profit. This provides an idea of the best possible objective function value that can be acquired, although it may be impossible to obtain the value due to the relaxation. The relaxation of the problem were achieved by converting all Category 3 and 4 problems (formulated in §3.1.2) to Category 1 and 2 problems (formulated in §3.1.1) so that the processing time of a job depends only on its start time and not the jobs that preceded it. As a result, there are no setup costs or times involved. Additionally, the relaxation contains no restrictions as to certain jobs that are not allowed to run in parallel with other specified jobs and no restrictions as to certain jobs that are not allowed to follow other specified jobs and start in a given period. The information and results of the 150 relaxed test cases that were solved in CPLEX may be found in Appendix B. A total of 53 of the relaxed problems could be solved to optimality while a solution could be obtained for the other 97 relaxed test cases. Due to a substantial reduction in the number of constraints and variables (and consequently the number of nonzeros), CPLEX was able to avoid running out of memory so that a solution could be obtained for all the relaxations.

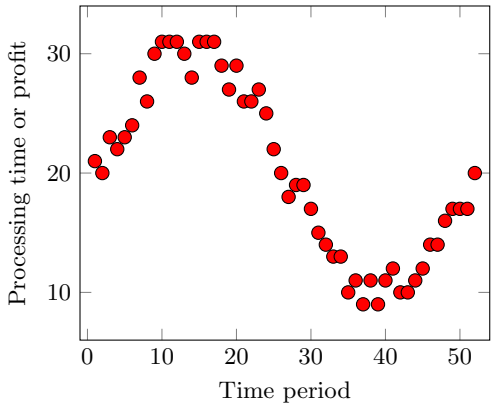


Figure 4.1: *An example of a sine wave for the processing time or profit of a job with a minimum of 10, a maximum of 30 and a variation of 2.*

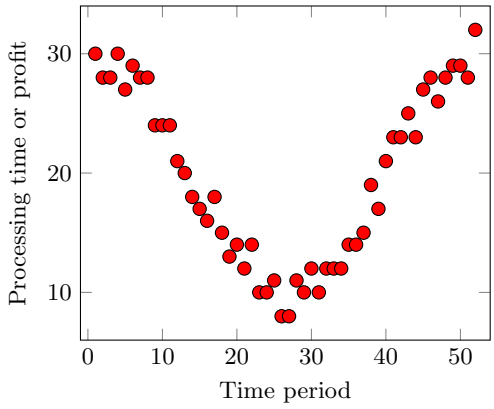


Figure 4.2: *An example of a cosine wave for the processing time or profit of a job with a minimum of 10, a maximum of 30 and a variation of 2.*

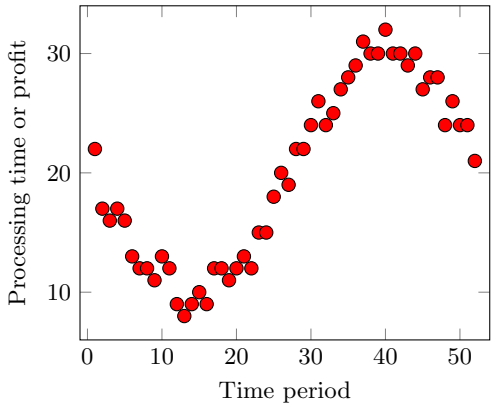


Figure 4.3: *An example of a negative sine wave for the processing time or profit of a job with a minimum of 10, a maximum of 30 and a variation of 2.*

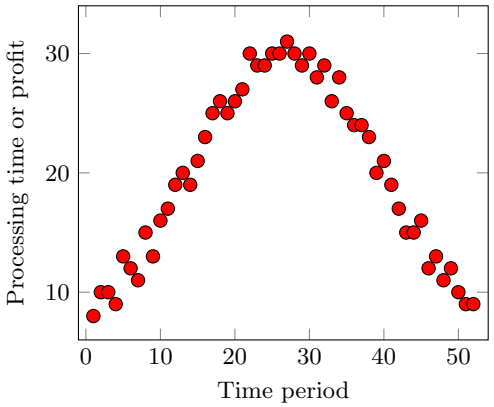


Figure 4.4: *An example of a negative cosine wave for the processing time or profit of a job with a minimum of 10, a maximum of 30 and a variation of 2.*

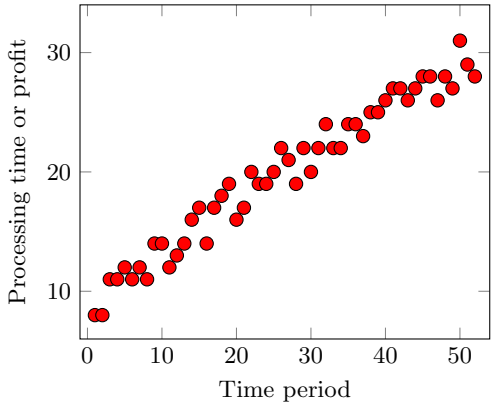


Figure 4.5: *An example of a positive linear function for the processing time or profit of a job with a minimum of 10, a maximum of 30 and a variation of 2.*

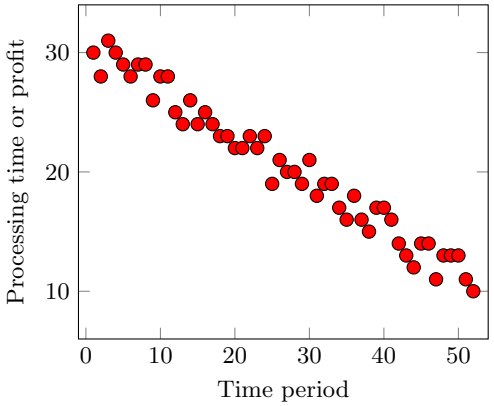


Figure 4.6: *An example of a negative linear function for the processing time or profit of a job with a minimum of 10, a maximum of 30 and a variation of 2.*

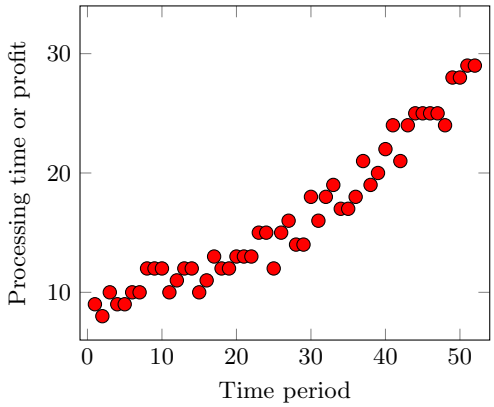


Figure 4.7: *An example of a convex increasing quadratic function for the processing time or profit of a job with a minimum of 10, a maximum of 30 and a variation of 2.*

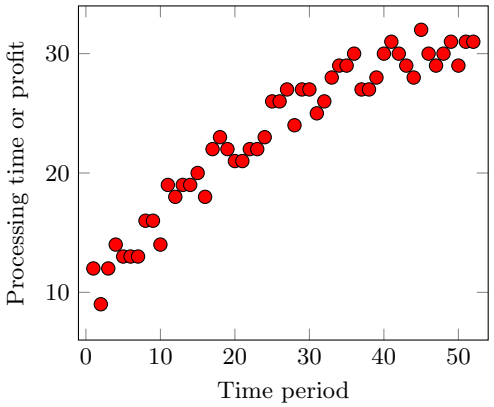


Figure 4.8: *An example of a concave increasing quadratic function for the processing time or profit of a job with a minimum of 10, a maximum of 30 and a variation of 2.*

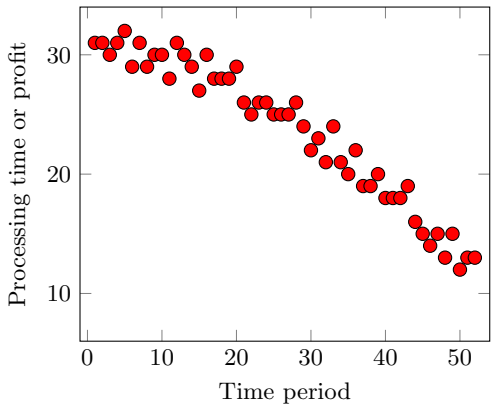


Figure 4.9: *An example of a concave decreasing quadratic function for the processing time or profit of a job with a minimum of 10, a maximum of 30 and a variation of 2.*

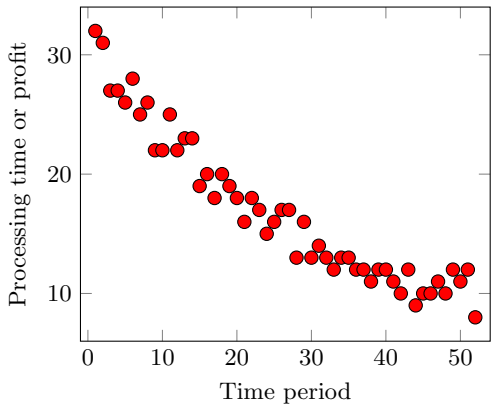


Figure 4.10: *An example of a convex decreasing quadratic function for the processing time or profit of a job with a minimum of 10, a maximum of 30 and a variation of 2.*

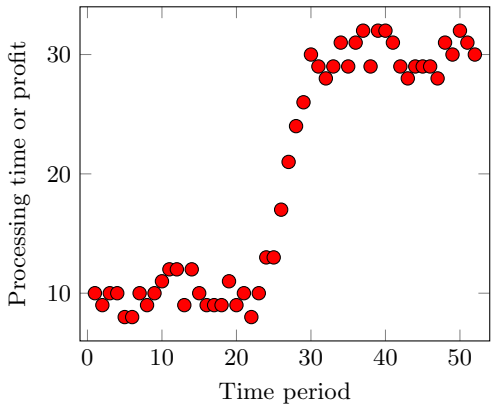


Figure 4.11: *An example of a increasing logistic function for the processing time or profit of a job with a minimum of 10, a maximum of 30 and a variation of 2.*

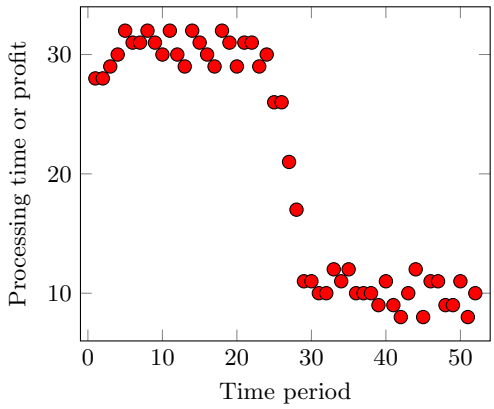


Figure 4.12: *An example of a decreasing logistic function for the processing time or profit of a job with a minimum of 10, a maximum of 30 and a variation of 2.*

4.2 Parameter calibration

All the metaheuristics are tested for different parameter settings for an intermediate instance. This section contains six tables (Tables 4.1–4.6) to report on parameter calibration. A problem was tested five times to ensure that there is consistency in the results when a combination of parameters is chosen. Each table contains the average of the profits of the solutions with the highest profit in each test run for different combinations of parameters for the corresponding metaheuristic. A cell that is highlighted indicates a combination of parameters that provides an average profit in the highest hundreds, *i.e.* when the average profit $\rho_j > 100\lfloor \max_i\{\rho_i\}/100 \rfloor$ where j is the index for all the combinations of parameters for the metaheuristic. TS is however an exception, since its objective function values are all greater than $100\lfloor \max_i\{\rho_i\}/100 \rfloor = \text{R}1\,700$. In this case a cell is highlighted if its combination provides an average profit in the highest fifties, *i.e.* when the average profit $\rho_j > 50\lfloor \max_i\{\rho_i\}/50 \rfloor$. The highlighted cells provide a visualisation for which values of a parameter give better results for the average profit. One of these highlighted cells are selected for the combination of parameters that will be used in the algorithm.

Although the average of the profits of the solutions is the most important deciding factor for selecting a set of parameters, it should not be the only one. Additionally, the average of the time it took for the solution to be found with the highest profit and the profit of the solution with the highest profit of the five times the algorithm was tested are considered. These results may be found in Appendix D. Firstly, the computational time that elapsed before the solution with the highest profit was found should be as small as possible when calibrating parameters. This is because it is more satisfactory for a good solution to be found earlier than later in the search. As seen in §4.4.4, it is not always favourable if the solution with the highest profit was found early when comparing algorithms against one another. This generally indicates that the algorithm converged too quickly and was not able to find solutions with a higher profit. However, when comparing two good solutions with different parameters for the same algorithm, the set of parameters that found its solution earlier is more preferable. Secondly, the profit of the solution with the highest profit of the five times the algorithm was tested should be as large as possible, since the profit of the schedule is maximised. A combination of parameters is selected to be used in an algorithm if its cell is the only highlighted cell, if its average profit is considerably higher than those of other highlighted cells or if it performs the best in most of the three deciding factors of three sets of parameters with the highest average profit.

4.2.1 Variable neighbourhood search

The combination of neighbourhoods that are used to form solutions after the swapping of two batches in the schedule (N_1), the moving of a batch from one position to another in the schedule (N_3), the inserting of a batch into the schedule (N_4), the swapping of jobs from two different batches in the schedule (N_5), the moving of a job from one batch to another in the schedule (N_7) and the inserting of a job into the schedule (N_8) are tested. The neighbourhoods that are used in each of the 2^6 combinations is when removing a batch from the schedule and inserting a batch containing unscheduled jobs (N_2) as well as when removing a job from the schedule and inserting an unscheduled job (N_6). These two moves ensure that there is diversity in the population. A neighbourhood is either used (indicated by the value "T" for true) or not used (indicated by the value "F" for false), as seen in Table 4.1. There is only one value in Table 4.1 that is greater than $100\lfloor \max_i\{\rho_i\}/100 \rfloor = \text{R}1\,400$, therefore this cell is selected. As a result, neighbourhoods N_1, N_2, N_3, N_4 and N_6 are used in VNS for the remainder of this chapter.

			$N_1 = T$				$N_1 = F$			
			$N_3 = T$		$N_3 = F$		$N_3 = T$		$N_3 = F$	
N_5	N_7	N_8	$N_4 = T$	$N_4 = F$	$N_4 = T$	$N_4 = F$	$N_4 = T$	$N_4 = F$	$N_4 = T$	$N_4 = F$
T	T	T	1294.6	1353.4	1278.0	1346.4	1296.8	1306.6	1378.4	1310.6
T	T	F	1299.6	1305.8	1349.6	1359.0	1334.8	1364.8	1317.6	1289.6
T	F	T	1365.4	1352.6	1302.2	1318.2	1264.2	1308.0	1324.2	1333.6
T	F	F	1303.0	1349.4	1384.2	1316.8	1353.8	1311.8	1302.4	1274.8
F	T	T	1352.6	1322.8	1361.2	1261.2	1303.2	1321.2	1354.2	1352.2
F	T	F	1283.2	1333.4	1345.8	1374.0	1349.6	1289.2	1308.2	1373.6
F	F	T	1329.0	1315.6	1313.2	1326.8	1351.0	1313.2	1264.2	1331.8
F	**F**	**F**	**1422.0**	1299.0	1360.4	1360.0	1337.6	1398.4	1285.0	1224.8

Table 4.1: *Average profit (in R) for VNS over different parameter settings.*

4.2.2 Particle swarm optimisation

The size of the population $|\mathcal{P}|$ is varied over 25, 50 and 75 while the inertia factor ω used to update the velocity of a particle is varied over 1/3, 2/3 and 1. Also, the acceleration coefficient φ_p in the direction of the best known position of each particle from the previous time step and acceleration coefficient φ_g in the direction of the best known position of all the particles from the previous time step are both varied over 0.25, 0.5, 0.75 and 1. There are a total of 57 combinations of parameters that result in an average profit that is larger than $100\lfloor \max_i\{\rho_i\}/100 \rfloor =$ R1 500. There are three combinations of parameters that result in a relatively high average profit compared to the other highlighted cells.

1. The parameters $|\mathcal{P}| = 75$, $\omega = 1/3$, $\varphi_p = 0.75$ and $\varphi_g = 0.5$ give an average profit of R1 585.20.

2. The parameters $|\mathcal{P}| = 50$, $\omega = 1$, $\varphi_p = 1$ and $\varphi_g = 0.75$ give an average profit of R1 590.40.

3. The parameters $|\mathcal{P}| = 75$, $\omega = 2/3$, $\varphi_p = 1$ and $\varphi_g = 0.75$ give an average profit of R1 587.20.

The second combination performs the best in Table 4.2 while the third combination performs the best in Tables D.3 and D.4 out of the three combinations. Since the third combination performs the best in most of the instances, the parameters $|\mathcal{P}| = 75$, $\omega = 2/3$, $\varphi_p = 1$ and $\varphi_g = 0.75$ are used in PSO for the remainder of the results.

4.2.3 Cuckoo search

The number of nests δ is varied over 25, 50 and 75 while the number of eggs in each of the nests ε is varied over 1, 3 and 5. Also, the probability ψ that the cuckoo will start its Lévy flights from the position of the best egg obtained so far is varied over 0.05 and 0.1, the fraction ζ_w of the nests with the smallest value for q_j that are selected to either be abandoned to build and populate a new nest with eggs or its worst egg is thrown away after which a new one is produced is varied over 0.1 and 0.2, the probability ζ_n that a nest will be abandoned after which a new one will be built at another location and populated with new eggs is varied over 0.15 and 0.3, and the number of Lévy flights γ that a cuckoo performs before laying its egg in a nest is varied over 5, 10 and 15. There are a total of 25 combinations of parameters that result in an average profit that is larger than $100\lfloor \max_i\{\rho_i\}/100 \rfloor =$ R1 500. There are three combinations of parameters that result in a relatively high average profit compared to the other highlighted cells.

		$\lvert\mathcal{P}\rvert = 25$			$\lvert\mathcal{P}\rvert = 50$			$\lvert\mathcal{P}\rvert = 75$		
φ_p	φ_g	$\omega = \frac{1}{3}$	$\omega = \frac{2}{3}$	$\omega = 1$	$\omega = \frac{1}{3}$	$\omega = \frac{2}{3}$	$\omega = 1$	$\omega = \frac{1}{3}$	$\boldsymbol{\omega = \frac{2}{3}}$	$\omega = 1$
0.25	0.25	1544.6	1394.8	1417.0	1498.4	1558.6	1522.4	1513.6	1500.6	1461.8
0.25	0.5	1421.6	1512.6	1511.4	1522.6	1535.2	1461.4	1535.0	1493.6	1461.6
0.25	0.75	1474.2	1518.6	1482.8	1561.6	1571.8	1457.8	1467.4	1498.4	1450.8
0.25	1.0	1375.0	1361.8	1404.4	1468.0	1511.6	1512.0	1525.6	1495.6	1526.8
0.5	0.25	1447.4	1473.4	1405.8	1511.0	1494.4	1479.4	1537.0	1491.8	1549.6
0.5	0.5	1451.2	1484.2	1398.6	1478.6	1463.8	1478.6	1529.6	1467.4	1486.2
0.5	0.75	1415.4	1374.6	1491.0	1460.4	1532.0	1492.2	1493.4	1478.0	1517.2
0.5	1.0	1523.0	1481.6	1419.8	1509.4	1453.4	1484.6	1541.2	1476.8	1526.2
0.75	0.25	1526.2	1437.8	1472.2	1467.4	1496.4	1541.0	1530.2	1507.2	1438.8
0.75	0.5	1383.8	1436.4	1442.4	1472.0	1550.2	1538.6	1585.2	1501.4	1530.8
0.75	0.75	1474.4	1463.6	1430.8	1454.8	1449.2	1451.2	1521.4	1445.8	1466.4
0.75	1.0	1491.0	1494.8	1499.4	1495.2	1465.8	1536.4	1494.0	1558.6	1482.6
1.0	0.25	1487.4	1411.4	1495.4	1505.8	1493.2	1539.8	1523.0	1560.6	1536.8
1.0	0.5	1425.2	1475.0	1428.6	1511.2	1447.0	1510.6	1542.2	1557.0	1542.8
1.0	**0.75**	1508.4	1432.6	1443.0	1486.0	1521.8	1590.4	1560.0	**1587.2**	1444.2
1.0	1.0	1511.4	1392.4	1506.6	1485.0	1413.0	1453.4	1543.8	1523.2	1452.4

Table 4.2: *Average profit (in R) for PSO over different parameter settings.*

1. The parameters $\delta = 75$, $\varepsilon = 5$, $\psi = 0.05$, $\zeta_w = 0.1$, $\zeta_n = 0.3$ and $\gamma = 10$ give an average profit of R1 534.80.
2. The parameters $\delta = 25$, $\varepsilon = 3$, $\psi = 0.1$, $\zeta_w = 0.2$, $\zeta_n = 0.3$ and $\gamma = 10$ give an average profit of R1 537.40.
3. The parameters $\delta = 75$, $\varepsilon = 5$, $\psi = 0.1$, $\zeta_w = 0.2$, $\zeta_n = 0.3$ and $\gamma = 5$ give an average profit of R1 529.60.

The first combination performs the best in Tables D.5 and D.6, the second combination performs the best in Table 4.3 while the third combination performs the best in none of the tables out of the three combinations. As a result, the parameters $\delta = 75$, $\varepsilon = 5$, $\psi = 0.05$, $\zeta_w = 0.1$, $\zeta_n = 0.3$ and $\gamma = 10$ are used in CS for the remainder of this chapter.

4.2.4 Genetic algorithms

For both GA-I and GA-II, the population size $|\mathcal{P}|$ is set at 25, 50 and 75, the probability α of mutating an individual's chromosome is set at 0.1 and 0.2, the percentage β of the size of the local archive when immigration is performed is set at 0.1 and 0.2, and the selection techniques that are used are BTS ("t"), FPS ("r") and SUS ("s"). The maximum number of genes α_{max} to change during mutation is set at 3, 6 and 9 while the crossover operators that are used are OPX ("o") and PUX ("u") for GA-I. On the other hand, the maximum number of successive moves α'_{max} that are made on an individual during mutation is set at 3, 6 and 9 while the crossover operators that are used are OPOX ("o") and PUOX ("u") for GA-II.

There are a total of 20 combinations of parameters that result in an average profit that is larger than $100\lfloor \max_i\{\rho_i\}/100 \rfloor$ = R1 600 for GA-I. It is evident that the parameters $|\mathcal{P}| = 25$, $\alpha_{max} = 6$, $\alpha = 0.2$ and $\beta = 0.1$ with FPS as selection technique and PUX as crossover operator give an average profit of R1 675. This is on average higher than the other highlighted cells. Although there is a combination of parameters that give a higher maximum profit, as seen in Table D.8, consistency in finding good solutions is the most important aspect when evaluating results. Therefore, it should be clear that the aforementioned parameters will be used in GA-I for the remainder of the results.

				$\delta = 25$			$\delta = 50$			$\boldsymbol{\delta = 75}$		
ψ	ζ_w	ζ_n	γ	$\varepsilon = 1$	$\varepsilon = 3$	$\varepsilon = 5$	$\varepsilon = 1$	$\varepsilon = 3$	$\varepsilon = 5$	$\varepsilon = 1$	$\varepsilon = 3$	$\boldsymbol{\varepsilon = 5}$
0.05	0.1	0.15	5	1470.6	1465.6	1466.2	1488.6	1482.2	1488.2	1490.4	1494.8	1493.4
0.05	0.1	0.15	10	1490.0	1446.4	1478.6	1493.8	1499.2	1445.6	1482.6	1466.0	1457.2
0.05	0.1	0.15	15	1478.2	1468.6	1475.6	1509.0	1477.2	1476.4	1456.6	1456.8	1481.8
0.05	0.1	0.3	5	1455.8	1486.8	1469.0	1457.2	1485.6	1457.4	1485.2	1504.2	1480.4
0.05	**0.1**	**0.3**	**10**	1463.4	1470.6	1480.0	1467.2	1473.2	1444.8	1464.0	1496.0	**1534.8**
0.05	0.1	0.3	15	1490.2	1469.2	1459.4	1475.8	1457.4	1493.0	1479.0	1484.8	1449.8
0.05	0.2	0.15	5	1464.6	1471.0	1488.4	1453.6	1467.6	1455.8	1490.6	1463.4	1489.8
0.05	0.2	0.15	10	1481.4	1485.8	1486.0	1504.2	1480.2	1460.2	1449.6	1487.0	1467.0
0.05	0.2	0.15	15	1477.8	1461.0	1462.8	1500.8	1486.6	1471.0	1512.0	1464.0	1475.6
0.05	0.2	0.3	5	1487.4	1490.8	1458.6	1466.6	1497.6	1483.4	1473.4	1485.2	1456.4
0.05	0.2	0.3	10	1455.4	1460.8	1480.2	1490.2	1475.0	1453.4	1471.8	1484.0	1464.4
0.05	0.2	0.3	15	1467.8	1492.6	1460.2	1473.6	1497.8	1490.0	1461.8	1492.6	1477.8
0.1	0.1	0.15	5	1440.4	1479.6	1462.4	1493.8	1473.4	1488.6	1493.8	1473.2	1458.6
0.1	0.1	0.15	10	1465.6	1502.0	1447.0	1497.0	1481.6	1469.2	1472.4	1463.2	1458.6
0.1	0.1	0.15	15	1488.2	1486.2	1520.8	1485.2	1488.0	1500.4	1472.0	1489.4	1447.6
0.1	0.1	0.3	5	1480.8	1494.2	1489.6	1479.0	1452.2	1481.6	1486.0	1474.8	1467.8
0.1	0.1	0.3	10	1488.8	1485.2	1492.6	1472.0	1510.8	1506.2	1452.0	1479.0	1454.4
0.1	0.1	0.3	15	1513.2	1453.2	1464.4	1477.6	1456.8	1480.8	1465.2	1482.6	1489.8
0.1	0.2	0.15	5	1495.2	1510.0	1497.4	1471.4	1484.6	1480.2	1452.2	1484.0	1466.2
0.1	0.2	0.15	10	1446.8	1470.0	1474.0	1476.8	1478.4	1475.0	1501.4	1495.2	1497.6
0.1	0.2	0.15	15	1467.0	1497.0	1475.2	1476.4	1485.8	1509.0	1516.8	1486.0	1467.6
0.1	0.2	0.3	5	1482.2	1474.8	1507.8	1487.6	1466.0	1468.8	1456.2	1485.0	1529.6
0.1	0.2	0.3	10	1511.0	1537.4	1486.6	1517.4	1466.4	1488.0	1483.2	1503.4	1465.2
0.1	0.2	0.3	15	1506.8	1503.2	1467.6	1447.8	1501.2	1477.4	1444.6	1486.4	1494.4

Table 4.3: *Average profit (in R) for CS over different parameter settings.*

Lastly, there are a total of 6 combinations of parameters that result in an average profit that is larger than $100\lfloor \max_i\{\rho_i\}/100 \rfloor$ = R1 600 for GA-II.

1. The parameters $|\mathcal{P}| = 25$, $\alpha'_{max} = 6$, $\alpha = 0.1$ and $\beta = 0.2$ with FPS as selection technique and PUOX as crossover operator give an average profit of R1 638. This combination of parameters performs the best in Table 4.5.

2. The parameters $|\mathcal{P}| = 25$, $\alpha'_{max} = 9$, $\alpha = 0.1$ and $\beta = 0.2$ with FPS as selection technique and PUOX as crossover operator give an average profit of R1 622.40. This combination of parameters performs the best in Tables D.9 and D.10.

3. The parameters $|\mathcal{P}| = 50$, $\alpha'_{max} = 3$, $\alpha = 0.1$ and $\beta = 0.2$ with FPS as selection technique and PUOX as crossover operator give an average profit of R1 614.60. This combination of parameters performs the best in none of the tables.

Since the second combination performs the best in most of the instances, out of the 6 highlighted cells, the parameters $|\mathcal{P}| = 25$, $\alpha'_{max} = 9$, $\alpha = 0.1$ and $\beta = 0.2$ with FPS as selection technique and PUOX as crossover operator are used in GA-II for the remainder of this chapter.

4.2.5 Tabu search

The population size $|\mathcal{P}|$ is set at 3, 6 and 9 while the maximum size $|\bar{\mathcal{N}}|$ of the subset of neighbours that are considered for the candidate list is set at 25, 50 and 75. Furthermore, the tabu list size τ is set at 20%, 40% and 60% of the total number of jobs (excluding the derived jobs) and the maximum number of iterations υ that a move stays in the tabu list is set at 0%, 20% and 40% of the total number of jobs (excluding the derived jobs) greater than the tabu list

				$\|\mathcal{P}\| = 25$			$\|\mathcal{P}\| = 50$			$\|\mathcal{P}\| = 75$		
α	β	selection	crossover	$\alpha_{max} = 3$	$\alpha_{max} = 6$	$\alpha_{max} = 9$	$\alpha_{max} = 3$	$\alpha_{max} = 6$	$\alpha_{max} = 9$	$\alpha_{max} = 3$	$\alpha_{max} = 6$	$\alpha_{max} = 9$
0.1	0.1	t	o	1423.6	1380.4	1393.2	1439.4	1397.0	1395.2	1373.4	1408.8	1393.2
0.1	0.1	t	u	1409.8	1431.6	1393.6	1456.8	1392.8	1414.6	1399.4	1420.2	1388.2
0.1	0.1	r	o	1599.2	1612.6	1604.2	1563.4	1578.8	1524.8	1545.2	1502.8	1516.0
0.1	0.1	r	u	1616.8	1649.8	1594.6	1520.2	1492.2	1499.4	1509.4	1483.2	1525.4
0.1	0.1	s	o	1569.4	1561.8	1602.8	1544.6	1530.4	1558.6	1491.4	1501.2	1511.8
0.1	0.1	s	u	1566.0	1577.4	1577.2	1545.2	1498.4	1485.4	1477.2	1531.4	1441.2
0.1	0.2	t	o	1451.2	1412.8	1428.2	1418.4	1425.8	1430.2	1395.0	1385.8	1416.4
0.1	0.2	t	u	1424.8	1436.2	1437.8	1418.8	1391.4	1428.2	1415.0	1385.0	1425.6
0.1	0.2	r	o	1605.6	1581.4	1626.0	1518.8	1548.2	1503.6	1561.8	1503.2	1540.6
0.1	0.2	r	u	1598.4	1581.0	1622.2	1543.2	1564.8	1558.8	1446.8	1481.4	1497.6
0.1	0.2	s	o	1621.2	1570.8	1570.4	1508.2	1579.4	1524.8	1543.6	1507.4	1581.2
0.1	0.2	s	u	1602.0	1564.4	1614.2	1547.2	1478.2	1505.2	1498.6	1482.2	1486.8
0.2	0.1	t	o	1395.2	1394.2	1421.6	1397.8	1399.6	1367.8	1383.2	1426.0	1388.0
0.2	0.1	t	u	1374.8	1404.8	1397.0	1381.6	1371.2	1360.0	1382.4	1399.2	1401.2
0.2	0.1	r	o	1585.6	1614.8	1586.6	1596.0	1514.2	1540.0	1524.6	1481.2	1484.2
0.2	**0.1**	**r**	**u**	1597.8	**1675.0**	1633.8	1525.4	1597.0	1515.6	1497.8	1463.6	1514.2
0.2	0.1	s	o	1589.6	1616.4	1588.4	1552.0	1579.4	1555.8	1516.4	1529.8	1512.8
0.2	0.1	s	u	1595.4	1611.6	1596.2	1500.0	1502.6	1517.2	1498.6	1465.4	1462.6
0.2	0.2	t	o	1439.8	1393.4	1415.2	1425.4	1411.8	1434.4	1443.4	1432.0	1421.8
0.2	0.2	t	u	1378.2	1444.8	1403.2	1401.8	1437.0	1411.2	1455.0	1451.0	1413.4
0.2	0.2	r	o	1641.8	1612.6	1580.2	1565.0	1555.0	1554.2	1522.2	1507.0	1577.6
0.2	0.2	r	u	1580.8	1588.2	1556.0	1478.2	1507.2	1579.6	1497.0	1490.0	1480.8
0.2	0.2	s	o	1559.2	1577.8	1601.6	1540.2	1567.4	1608.6	1524.4	1519.6	1506.2
0.2	0.2	s	u	1565.8	1571.4	1572.4	1500.0	1514.6	1488.0	1470.0	1476.8	1506.6

Table 4.4: *Average profit (in R) for GA-I over different parameter settings.*

				$\|\mathcal{P}\| = 25$			$\|\mathcal{P}\| = 50$			$\|\mathcal{P}\| = 75$		
α	β	selection	crossover	$\alpha'_{max} = 3$	$\alpha'_{max} = 6$	$\alpha'_{max} = 9$	$\alpha'_{max} = 3$	$\alpha'_{max} = 6$	$\alpha'_{max} = 9$	$\alpha'_{max} = 3$	$\alpha'_{max} = 6$	$\alpha'_{max} = 9$
0.1	0.1	t	o	1408.4	1398.8	1387.8	1393.8	1423.6	1411.0	1416.8	1375.6	1403.0
0.1	0.1	t	u	1409.4	1410.2	1412.0	1422.0	1457.2	1387.8	1404.8	1421.0	1370.4
0.1	0.1	r	o	1404.4	1509.4	1461.2	1431.6	1450.4	1454.4	1466.4	1475.0	1472.8
0.1	0.1	r	u	1576.2	1523.0	1548.4	1542.6	1550.0	1564.8	1540.4	1545.8	1544.8
0.1	0.1	s	o	1405.8	1430.0	1455.4	1431.6	1449.6	1430.6	1442.8	1441.8	1480.6
0.1	0.1	s	u	1553.4	1625.6	1598.6	1548.0	1488.2	1505.6	1491.0	1515.6	1592.2
0.1	0.2	t	o	1406.2	1446.8	1420.8	1426.0	1423.2	1435.2	1426.6	1388.2	1407.6
0.1	0.2	t	u	1420.0	1421.8	1452.0	1420.8	1430.6	1440.2	1443.0	1435.6	1402.0
0.1	0.2	r	o	1499.0	1491.2	1482.8	1472.0	1499.4	1429.8	1455.2	1510.6	1510.8
0.1	**0.2**	**r**	**u**	1537.0	1638.0	**1622.4**	1614.6	1597.0	1565.2	1554.2	1566.4	1519.8
0.1	0.2	s	o	1464.6	1471.0	1445.2	1442.8	1433.2	1438.2	1460.2	1447.4	1468.0
0.1	0.2	s	u	1626.4	1559.0	1579.8	1538.4	1568.4	1561.4	1499.0	1530.6	1538.0
0.2	0.1	t	o	1389.4	1411.6	1376.2	1416.8	1415.8	1379.6	1400.6	1421.4	1405.0
0.2	0.1	t	u	1386.2	1385.2	1385.8	1400.6	1419.2	1432.8	1378.4	1408.8	1382.0
0.2	0.1	r	o	1449.2	1385.0	1487.0	1416.0	1490.0	1472.8	1494.0	1461.0	1516.8
0.2	0.1	r	u	1545.0	1548.4	1589.0	1576.0	1549.4	1588.8	1512.6	1505.2	1570.8
0.2	0.1	s	o	1462.8	1450.2	1443.0	1441.4	1473.4	1443.6	1462.8	1451.2	1453.6
0.2	0.1	s	u	1562.8	1591.8	1539.0	1556.4	1520.6	1531.4	1579.2	1487.6	1540.0
0.2	0.2	t	o	1430.0	1459.2	1411.2	1427.0	1444.6	1427.8	1395.6	1399.8	1411.4
0.2	0.2	t	u	1422.4	1396.2	1422.2	1415.4	1424.2	1411.4	1421.6	1426.8	1415.4
0.2	0.2	r	o	1477.4	1510.8	1480.6	1451.6	1470.6	1456.4	1452.4	1420.4	1493.0
0.2	0.2	r	u	1604.2	1572.2	1593.6	1590.8	1578.4	1579.6	1566.0	1510.6	1548.4
0.2	0.2	s	o	1471.6	1479.6	1473.8	1455.2	1439.8	1493.4	1455.8	1471.8	1474.8
0.2	0.2	s	u	1544.2	1532.0	1556.4	1542.6	1548.6	1575.2	1528.2	1479.4	1541.0

Table 4.5: *Average profit (in R) for GA-II over different parameter settings.*

		$\lvert\mathcal{P}\rvert = 3$			$\lvert\boldsymbol{\mathcal{P}}\rvert = \mathbf{6}$			$\lvert\mathcal{P}\rvert = 9$		
υ	τ	$\lvert\bar{\mathcal{N}}\rvert = 25$	$\lvert\bar{\mathcal{N}}\rvert = 50$	$\lvert\bar{\mathcal{N}}\rvert = 75$	$\lvert\bar{\mathcal{N}}\rvert = 25$	$\lvert\bar{\mathcal{N}}\rvert = 50$	$\lvert\bar{\mathcal{N}}\rvert = \mathbf{75}$	$\lvert\bar{\mathcal{N}}\rvert = 25$	$\lvert\bar{\mathcal{N}}\rvert = 50$	$\lvert\bar{\mathcal{N}}\rvert = 75$
τ	$0.2n'$	1759.2	1745.8	1747.6	1753.2	1723.0	1753.4	1731.4	1733.4	1727.0
τ	$0.4n'$	1733.8	1756.6	1761.4	1744.0	1735.8	1738.6	1717.8	1742.4	1710.6
τ	$0.6n'$	1734.4	1729.4	1705.4	1719.4	1705.0	1733.4	1715.4	1720.0	1742.6
$\tau + 0.2n'$	$0.2n'$	1741.8	1748.6	1757.0	1735.2	1761.0	1747.2	1764.8	1737.6	1769.6
$\tau + 0.2n'$	$0.4n'$	1722.2	1760.8	1734.4	1754.0	1708.4	1720.8	1707.2	1726.0	1746.4
$\tau + 0.2n'$	$0.6n'$	1702.4	1726.8	1745.4	1730.0	1715.8	1700.2	1724.2	1701.4	1734.6
$\boldsymbol{\tau + 0.4n'}$	$\mathbf{0.2n'}$	1747.6	1761.6	1769.2	1732.8	1737.2	**1774.4**	1755.8	1755.4	1754.6
$\tau + 0.4n'$	$0.4n'$	1757.8	1752.2	1755.0	1717.0	1727.4	1720.0	1742.8	1739.2	1765.6
$\tau + 0.4n'$	$0.6n'$	1713.8	1743.8	1724.2	1705.4	1756.4	1707.0	1713.8	1727.0	1731.0

Table 4.6: *Average profit (in R) for TS over different parameter settings.*

size. There are a total of 22 combinations of parameters for which the average profit is greater than $50\lfloor \max_i\{\rho_i\}/50 \rfloor$ = R1 750. Three combinations of parameters with the highest average profit are compared to determine the values of the parameters that will be used.

1. The parameters $|\mathcal{P}| = 9$, $|\bar{\mathcal{N}}| = 75$, $\tau = 0.2n'$ and $\upsilon = \tau + 0.2n'$ give an average profit of R1 769.60.

2. The parameters $|\mathcal{P}| = 3$, $|\bar{\mathcal{N}}| = 75$, $\tau = 0.2n'$ and $\upsilon = \tau + 0.4n'$ give an average profit of R1 769.20.

3. The parameters $|\mathcal{P}| = 6$, $|\bar{\mathcal{N}}| = 75$, $\tau = 0.2n'$ and $\upsilon = \tau + 0.4n'$ give an average profit of R1 774.40.

The first combination performs the best in none of the tables, the second combination performs the best in Table D.11 while the third combination performs the best in Tables 4.6 and D.12 out of the three combinations. Thus, the parameters $|\mathcal{P}| = 6$, $|\bar{\mathcal{N}}| = 75$, $\tau = 0.2n'$ and $\upsilon = \tau + 0.4n'$ are used in TS for the remainder of this chapter.

4.3 Convergence of the profit

This section contains convergence graphs in §4.3.1 for all the metaheuristics considered in this thesis to determine the time limit at which the metaheuristics are terminated. The time limit depends on the size of the problem that is being solved. If a specific metaheuristic reaches the chosen time limit, its current iteration is the last iteration that will be completed after which solutions are returned to the NSTB algorithm for further processing. The running time of the metaheuristic will therefore be at least as large as the time limit, but not substantially longer as iterations can be quite short. Furthermore, §4.3.2 contains convergence graphs for the NSTB algorithm after solutions have been obtained from the metaheuristics. These convergence graphs are used to determine the computational time at which the last iteration of the NSTB algorithm is performed before solutions are returned to the decision maker as well as the number of subiterations η' in each of the steps (except for the merging of batches).

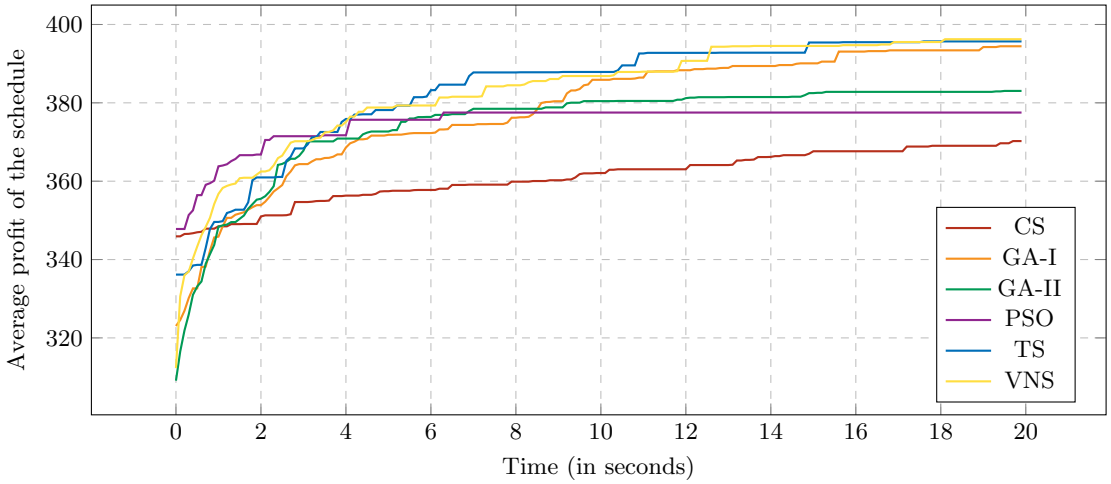


Figure 4.13: *Convergence graphs for the average profit (in R) of the metaheuristics' schedules for the first 25 elementary test cases.*

4.3.1 Time limit for metaheuristics

Figures 4.13–4.18 contain the convergence graphs of the metaheuristics for different sizes of the scheduling problems to be solved. Each of the three difficulty groups (containing 50 test cases each) are split into two subgroups containing its first and last 25 test cases. The convergence graph of a metaheuristic is constructed so that the average of the profit for the 25 test cases in the subgroup is plotted over time.[2] The six subgroups of test cases were run for 20, 30, 45, 75, 110 and 200 seconds, respectively. These times were chosen in such a way that the graph for each of the metaheuristics are nearly flat at the chosen time. The time limit η that will be given to each of the metaheuristics before their last iteration is performed is then determined by observing the approximate time that all of the plots start approaching near flatness. This occurs at approximately 15, 25, 35, 60, 90 and 180 seconds for the six subgroups, respectively. By giving each of the metaheuristics the same time limit for a subgroup ensures fairness when comparing the algorithms against one another.

4.3.2 Time limit and number of subiterations for NSTB algorithm

Figures 4.19–4.24 contain the convergence graphs of the NSTB algorithm for the six difficulty subgroups mentioned in §4.3.1. The convergence graphs are again constructed so that the average of the profit for the 25 test cases in the subgroup is plotted over time. For each of the 25 test cases that are tested for a convergence graph, the solution with the highest profit that the metaheuristic could obtain is used for the graph. The average of the profits for these 25 solutions is then calculated to determine the starting point of the convergence graph. Changes in the average objective function value are much smaller compared to the changes in the convergence graphs in Figures 4.13–4.18. This is due to jobs not having much freedom to move around within their batches and the fact that batches cannot be moved around as this will influence the processing times of the jobs in the schedule.. As a result, the graphs appear to be flat in the figures although incremental changes were taken into consideration to determine the time limit given to the NSTB algorithm and the number of subiterations η'.

The number of subiterations was varied over 5, 10, 25, 50 and 100 for each of the subgroups. It

[2]The average of the profit is plotted, since it will be too difficult to distinguish the algorithms from one another if each metaheuristic has 25 plots on the same pair of axes.

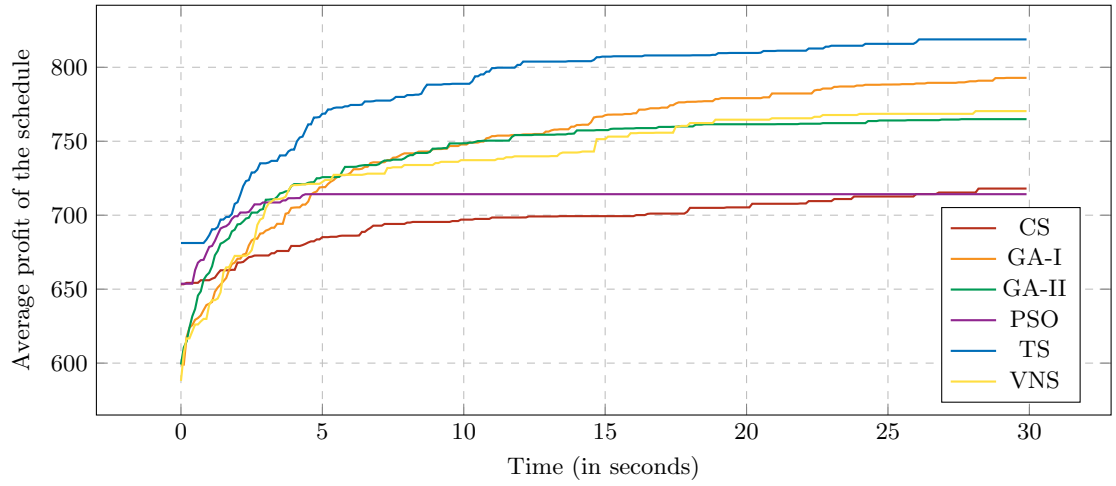


Figure 4.14: *Convergence graphs for the average profit (in R) of the metaheuristics' schedules for the last 25 elementary test cases.*

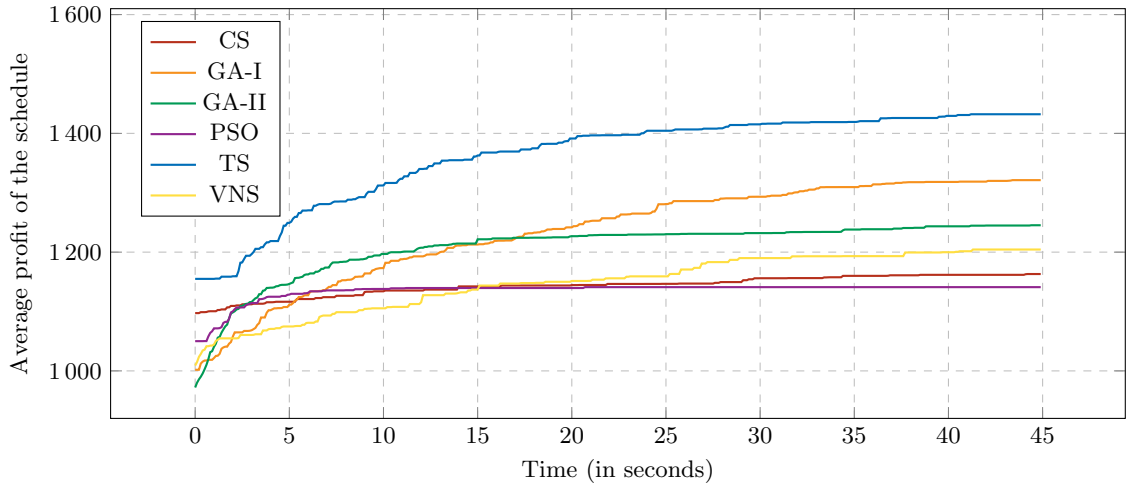


Figure 4.15: *Convergence graphs for the average profit (in R) of the metaheuristics' schedules for the first 25 intermediate test cases.*

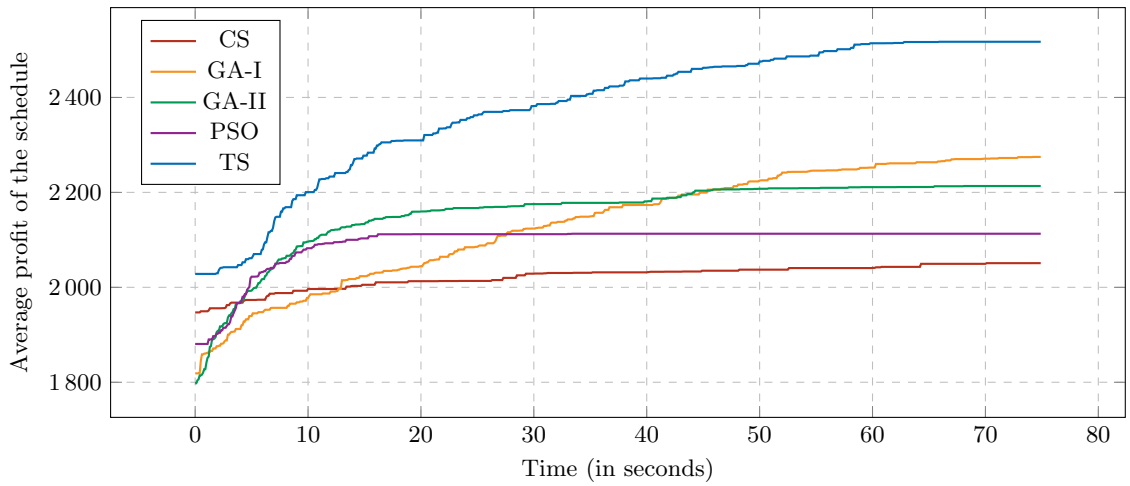


Figure 4.16: *Convergence graphs for the average profit (in R) of the metaheuristics' schedules for the last 25 intermediate test cases.*

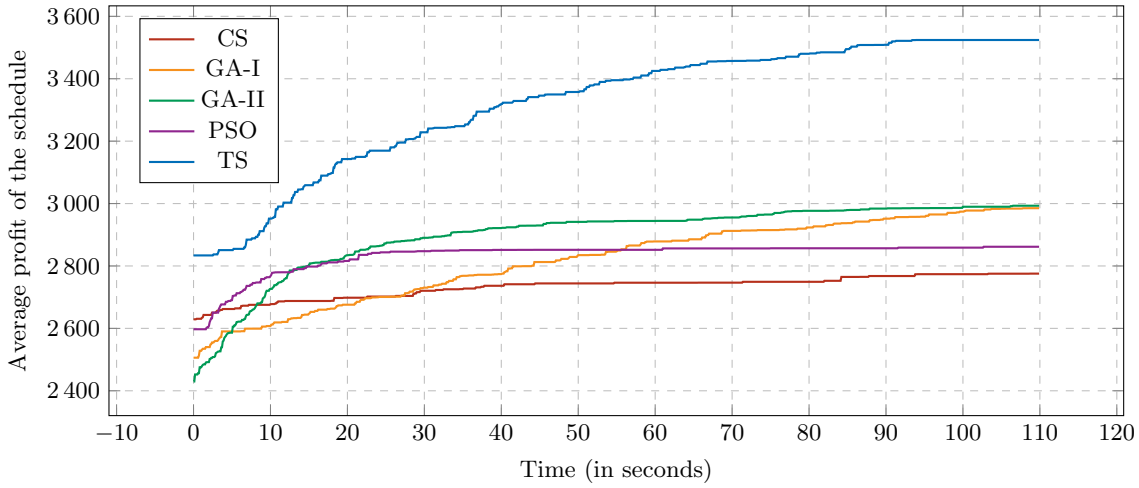


Figure 4.17: *Convergence graphs for the average profit (in R) of the metaheuristics' schedules for the first 25 challenging test cases.*

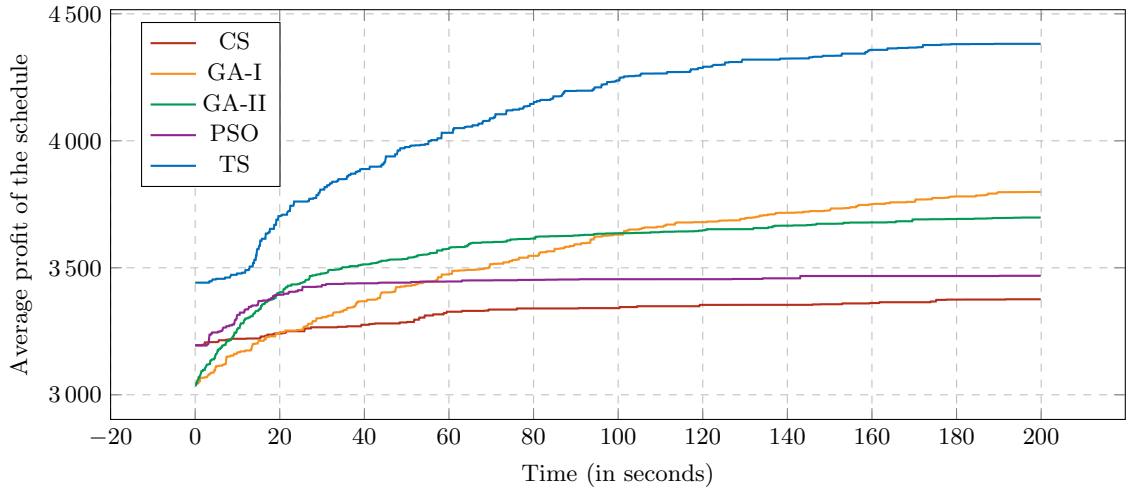


Figure 4.18: *Convergence graphs for the average profit (in R) of the metaheuristics' schedules for the last 25 challenging test cases.*

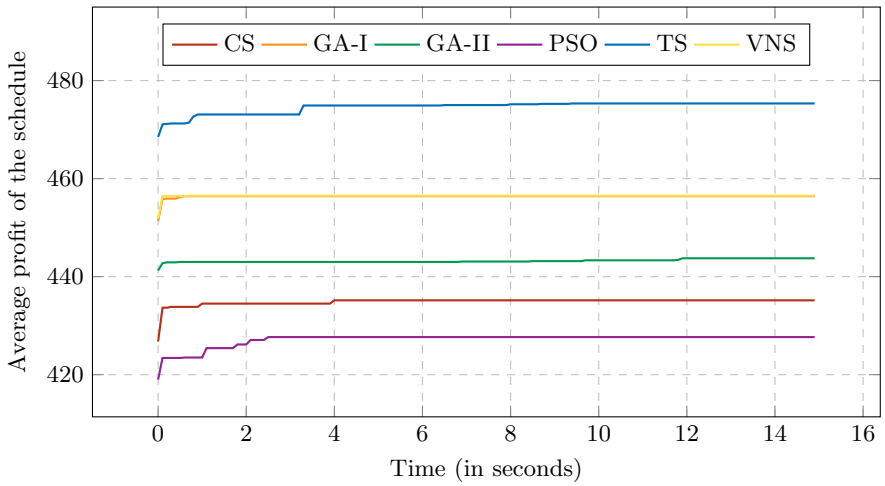


Figure 4.19: *Convergence graphs for the average profit (in R) of the NSTB algorithm's schedules for the first 25 elementary test cases.*

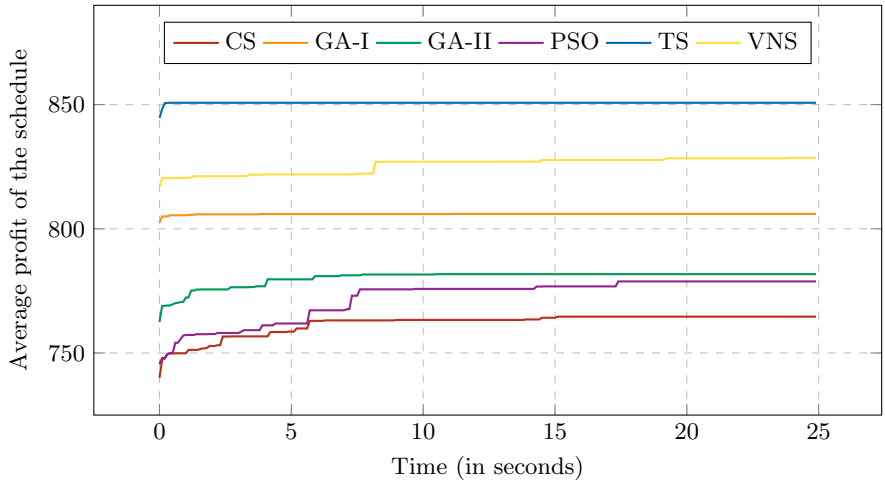


Figure 4.20: *Convergence graphs for the average profit (in R) of the NSTB algorithm's schedules for the last 25 elementary test cases.*

was found that a small number of subiterations produces the best results. The value for $\eta' = 5$ for the first, third, fourth, fifth and sixth subgroups while $\eta' = 10$ for the second subgroup. The convergence graphs in Figures 4.19–4.24 are for the aforementioned number of subiterations. The second step was to determine the time limit η for the NSTB algorithm. Similar to the method used in §4.3.1, the time limit for a subgroup was chosen in such a way that the average of the profits is approximately flat for all of the algorithms at the chosen time. This occurs at approximately 8, 15, 25, 45, 65 and 100 seconds for the six subgroups, respectively.

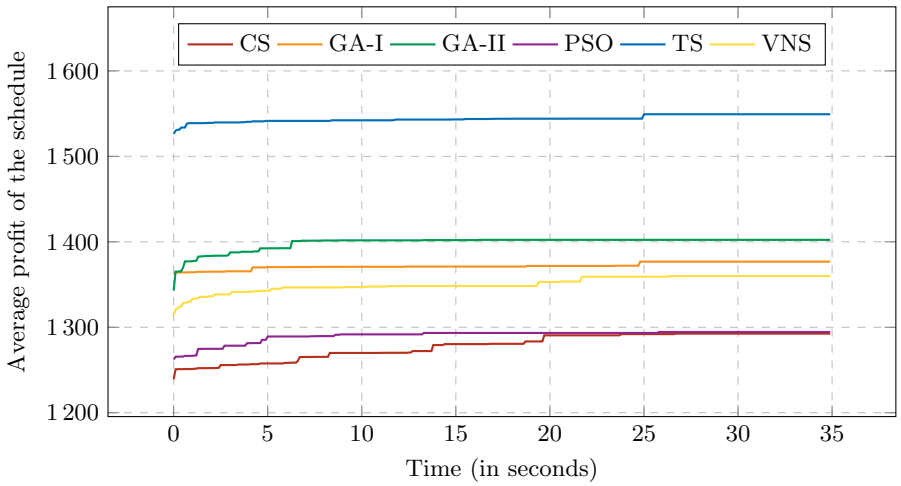


Figure 4.21: *Convergence graphs for the average profit (in R) of the NSTB algorithm's schedules for the first 25 intermediate test cases.*

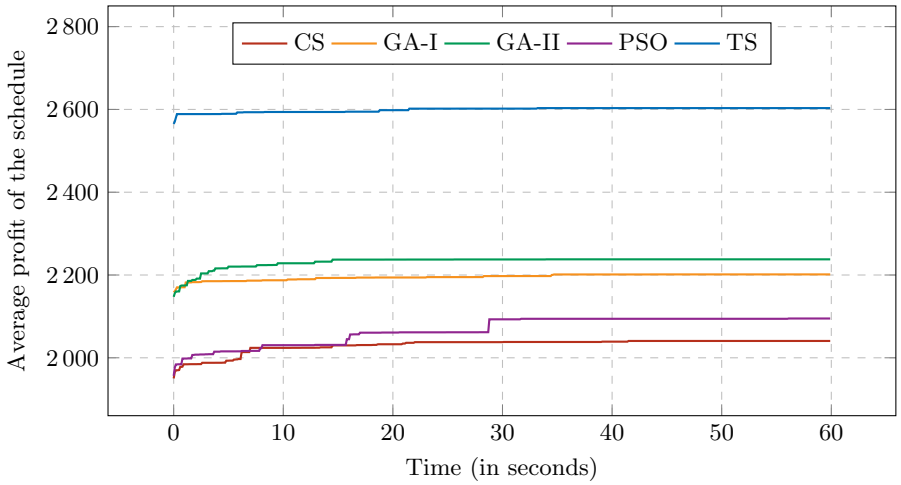


Figure 4.22: *Convergence graphs for the average profit (in R) of the NSTB algorithm's schedules for the last 25 intermediate test cases.*

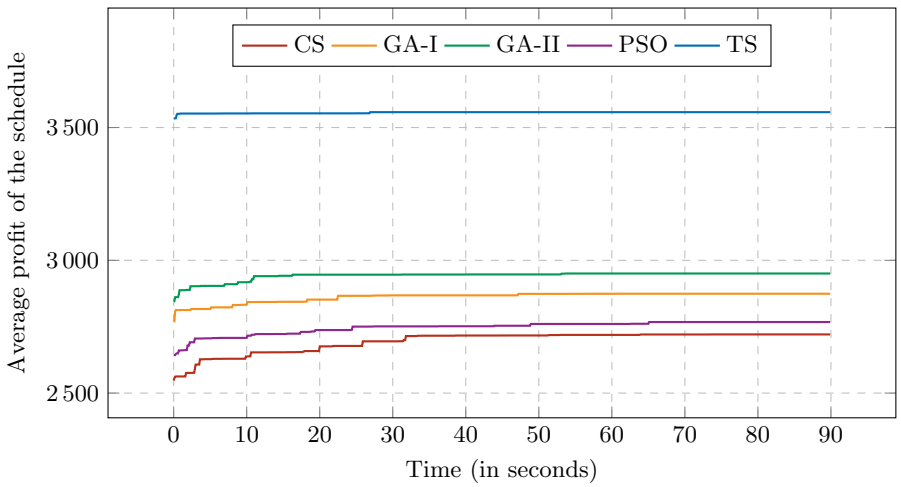


Figure 4.23: *Convergence graphs for the average profit (in R) of the NSTB algorithm's schedules for the first 25 challenging test cases.*

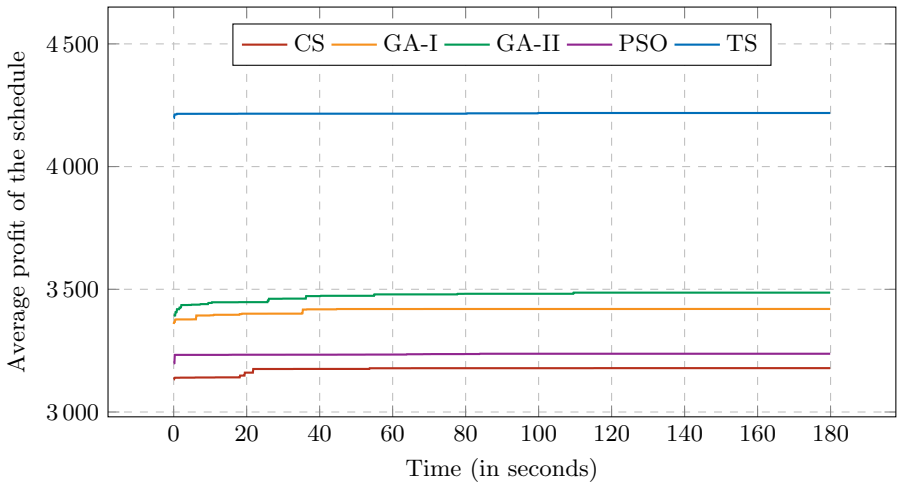


Figure 4.24: *Convergence graphs for the average profit (in R) of the NSTB algorithm's schedules for the last 25 challenging test cases.*

4.4 Algorithm comparisons

The algorithms are compared to one another in this section by utilising different measurements. A total of 30 test cases (10 test cases per difficulty group) were randomly chosen to be plotted in §4.4.1–4.4.3 while a test case per difficulty group is used in §4.4.4. The same set of test cases are used in §4.4.2 and §4.4.3. All test cases in §4.4.1–4.4.3 were run fives times per algorithm followed by the NSTB algorithm after which the results were obtained so that algorithms can be compared across the sections. The profit (the objective to be maximised) of the schedules and the sum of the durations of the scheduled jobs are looked at in §4.4.1 while §4.4.2 considers the makespan of the schedules as well as the number of time periods in which the machine is idle (due to the moving around of jobs in the NSTB algorithm). The number of scheduled jobs, the percentage of scheduled jobs running in parallel and the available space in the schedule for each of the 30 test cases may be found in §4.4.3. Appendix E contains the results of the seven measurements for all 150 test cases while Appendix C is comprised of the results of the seven measurements for the 150 original and relaxed problems that were solved in CPLEX. Lastly, this section is concluded in §4.4.4 by presenting boxplots containing the minimum, the three quartiles and the maximum of the computational times at which the solution with the highest profit was found.

4.4.1 Profit of schedules and sum of job durations

Figures 4.25–4.27 contain grouped bar charts of the profit of the schedule for 30 selected test cases. The maximum profit of the five times that the metaheuristic was run for is plotted per test case. Test cases 8, 9, 17, 22, 25, 36, 38, 42, 44 and 46 were selected to be plotted for the results of elementary test cases. Test cases 56, 62, 67, 69, 71 were selected as intermediate test cases that VNS could solve while test cases 78, 88, 90, 93 and 97 were chosen as the intermediate test cases that VNS was unable to solve. Lastly, test cases 105, 109, 114, 115, 122, 126, 133, 134, 143 and 147 were chosen for the challenging test cases.

For the elementary test cases, it can be seen that CPLEX performs better than all of the algorithms for the easiest test cases (test cases 8, 9 and 17) while it performs slightly better than some of the algorithms for test case 22 in terms of the profit. TS performed the best while VNS and GA-I performed second best for a total 4 and 2 times, respectively, for the

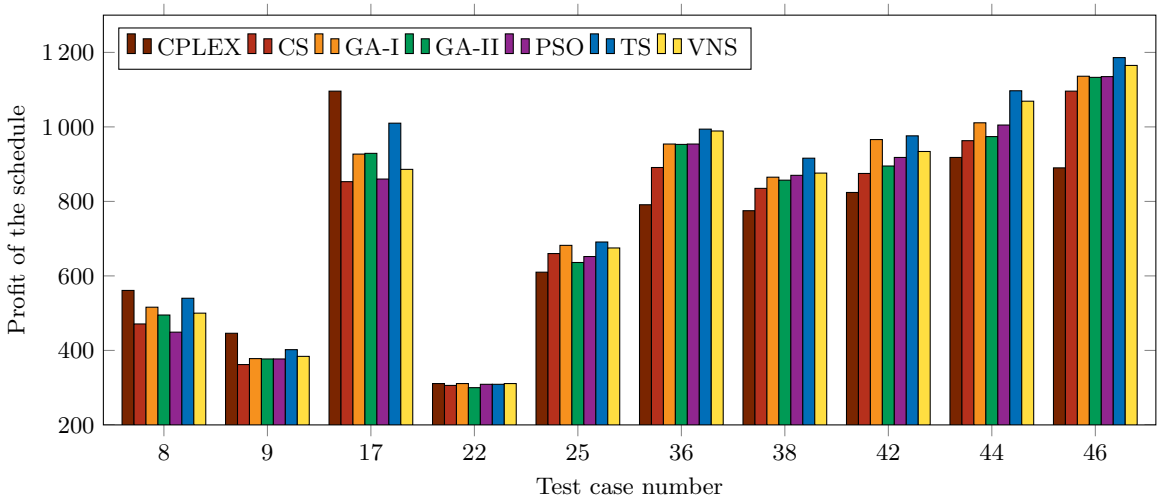


Figure 4.25: *A grouped bar chart for the profit of the schedule (in R) for elementary test cases.*

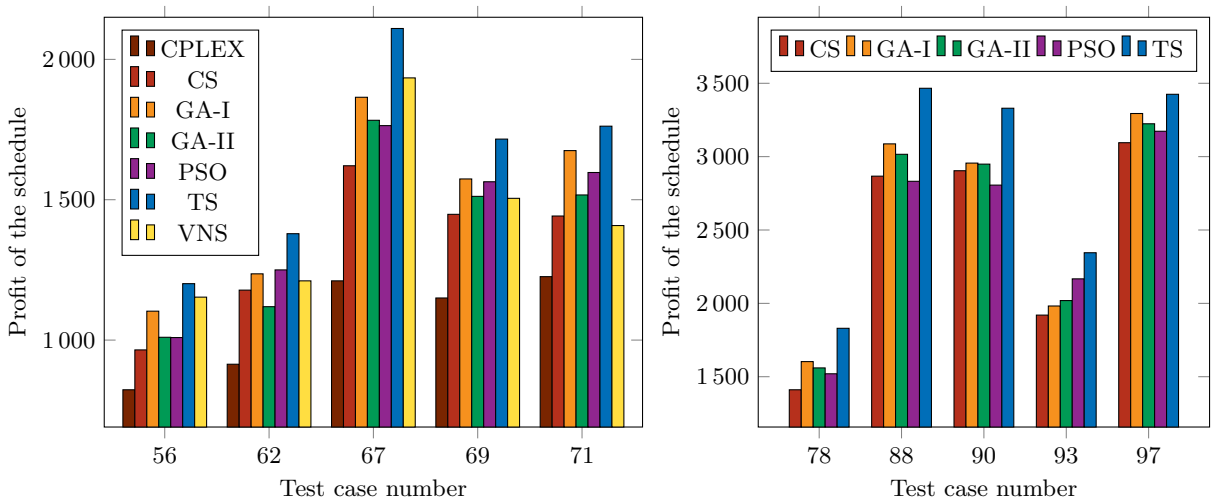
	CS	GA-I	GA-II	PSO	TS	VNS
GA-I	7.219%					
GA-II	3.837%	-3.06%				
PSO	3.803%	-3.145%	-0.011%			
TS	8.202%	1.037%	4.315%	4.463%		
VNS	8.353%	1.036%	4.322%	4.452%	0.301%	
CPLEX	2.811%	-4.064%	-1.01%	-0.924%	-4.659%	-4.829%

Table 4.7: *The average percentage increase in the profit of the schedule (in R) for elementary test cases.*

six remaining elementary test cases. TS showed the best results in terms of profit in all 20 intermediate and challenging test cases. As test cases become increasingly difficult, it appears as though VNS's performance gets weaker while GA-I retains its position as the second best algorithm, evident in Figure 4.26(a). For the intermediate test cases that VNS was unable to solve and the challenging test cases, it appears as though the two genetic algorithms perform second best for a majority of the test cases. PSO performs slightly worse than GA-I and GA-II in most of the test cases while CS is the worst performing metaheuristic.

Tables 4.7–4.9 contain the average percentage improvement of one algorithm over another in terms of profit. A positive percentage indicates an improvement of the algorithm in the first column over the algorithm in the first row (header row) while a negative percentage shows a deterioration. For the elementary test cases, the percentage improvement of all pairs of algorithms is calculated for all 50 test cases after which the average of the improvements is calculated for all pairs. These values are listed in the table. The table contains a total of $\binom{7}{2} = 21$ values for the 21 pairs of algorithms (that includes CPLEX). The same process was followed for the intermediate and challenging test cases. From the tables, it is apparent that the profit for GA-I is, on average, 3.06% and 2.1% better than GA-II for elementary and intermediate test cases, respectively. GA-II performs, on average, 0.311% better than GA-I for challenging test cases in terms of the profit. It is also clear from the tables that CPLEX performs increasingly worse for more difficult test cases compared to the metaheuristics while TS shows a greater improvement over other metaheuristics as test cases become more difficult.

Grouped bar charts of the sum of the scheduled jobs' durations for 30 selected test cases may be found in Figures 4.28–4.30. The minimum sum of job durations of the five times that the metaheuristic was run for is plotted per test case. CPLEX has a much higher value for the sum



(a) Test cases solvable by CPLEX and VNS. **(b)** Test cases unsolvable by CPLEX and VNS.

Figure 4.26: *Grouped bar charts for the profit of the schedule (in R) for intermediate test cases.*

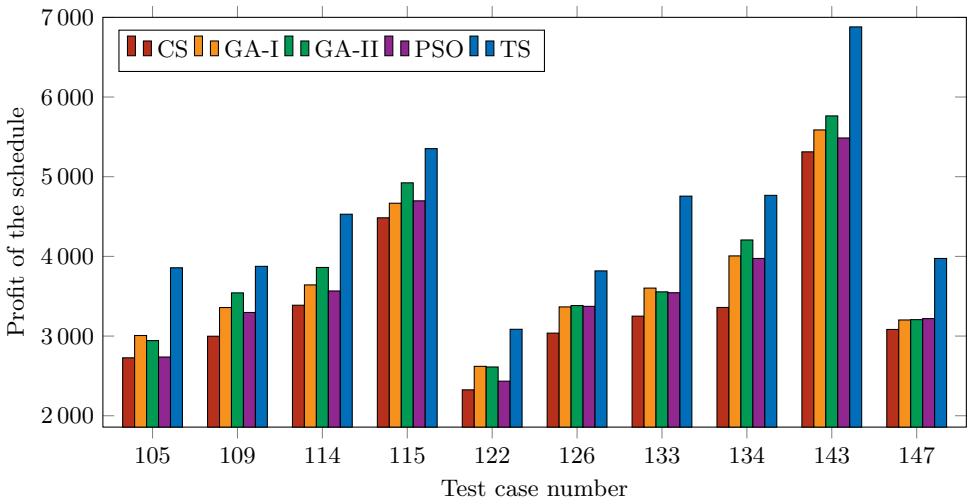


Figure 4.27: *A grouped bar chart for the profit of the schedule (in R) for challenging test cases.*

	CS	GA-I	GA-II	PSO	TS	VNS
GA-I	10.153%					
GA-II	7.695%	-2.1%				
PSO	6.289%	-3.481%	-1.182%			
TS	20.762%	9.843%	12.369%	14.1%		
VNS	9.429%	-1.654%	2.033%	2.716%	-8.658%	
CPLEX	-28.076%	-35.426%	-32.709%	-32.298%	-39.937%	-33.509%

Table 4.8: *The average percentage increase in the profit of the schedule (in R) for intermediate test cases.*

	CS	GA-I	GA-II	PSO
GA-I	10.831%			
GA-II	11.057%	0.311%		
PSO	8.621%	-1.909%	-2.019%	
TS	28.865%	16.562%	16.245%	18.967%

Table 4.9: *The average percentage increase in the profit of the schedule (in R) for challenging test cases.*

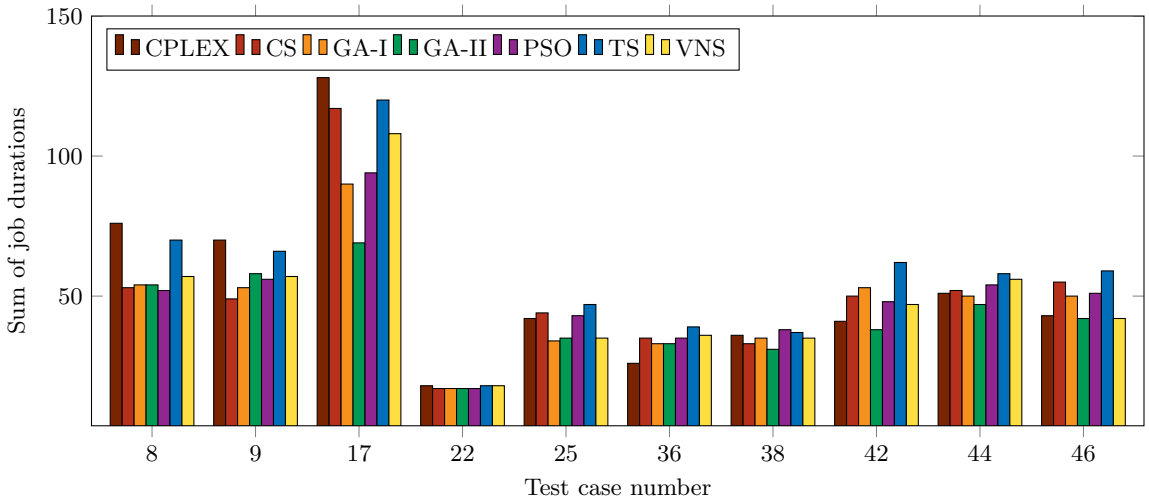


Figure 4.28: *A grouped bar chart for the sum of job durations for elementary test cases.*

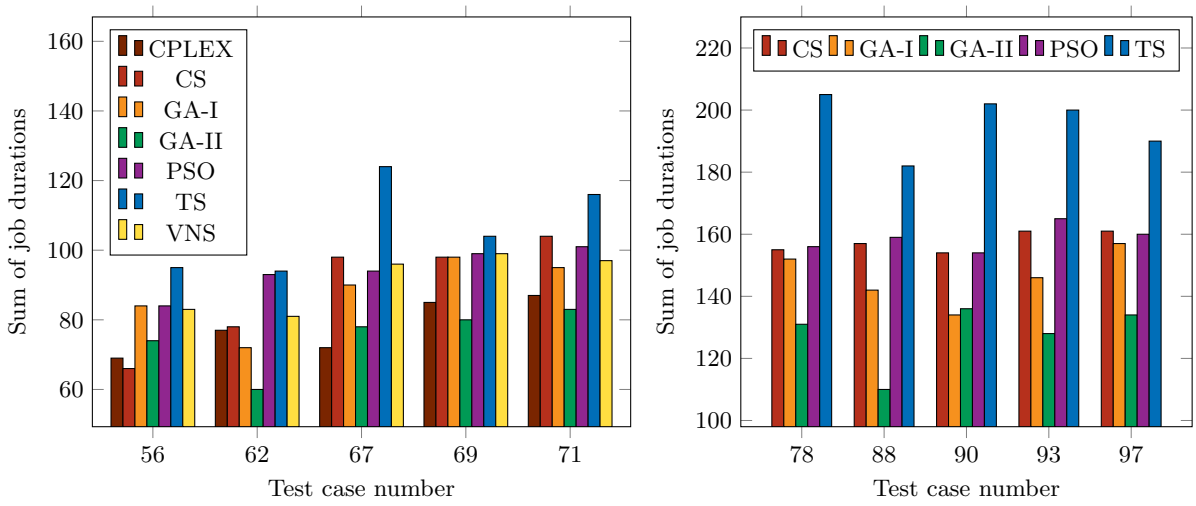
of job durations compared to most algorithms for the easiest test cases. From Figure 4.29 and Figure 4.30, it is clear that TS has a higher sum of job durations compared to all other algorithms for intermediate and challenging test cases. It is also evident that GA-I has a considerably higher sum of job durations compared to GA-II indicating that GA-I generally schedules jobs with a longer duration. Tables 4.10–4.12 contain the average percentage increase of one algorithm over another in terms of the sum of job durations. It can be seen from these tables that TS has, on average, a considerably greater sum of job durations compared to all other algorithms in all the groups. Also GA-II has, on average, a noticeable increase in the sum of job durations compared to GA-I (especially for intermediate and challenging test cases).

4.4.2 Makespan and idle time of schedules

Figures 4.31–4.33 contain grouped bar charts for the makespan of the schedule for 30 selected test cases. The average makespan of the schedule of the five times that the metaheuristic was run for is plotted per test case. It can be seen that there is no considerable difference between

	CS	GA-I	GA-II	PSO	TS	VNS
GA-I	0.758%					
GA-II	-3.444%	-4.111%				
PSO	0.096%	-0.312%	4.657%			
TS	10.069%	9.77%	15.751%	10.27%		
VNS	1.747%	1.151%	6.275%	1.839%	-6.821%	
CPLEX	2.7%	2.105%	7.554%	2.808%	-6.213%	1.107%

Table 4.10: *The average percentage increase in the sum of job durations for elementary test cases.*



(a) Test cases solvable by CPLEX and VNS. **(b)** Test cases unsolvable by CPLEX and VNS.

Figure 4.29: *Grouped bar charts for the sum of job durations for intermediate test cases.*

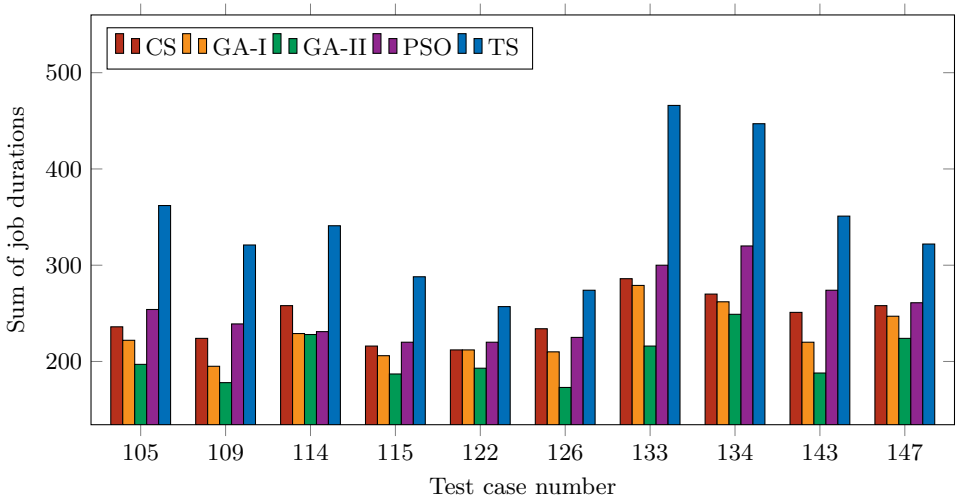


Figure 4.30: *A grouped bar chart for the sum of job durations for challenging test cases.*

	CS	GA-I	GA-II	PSO	TS	VNS
GA-I	-2.148%					
GA-II	-13.376%	-11.615%				
PSO	2.938%	5.404%	20.42%			
TS	24.109%	27.386%	46.37%	21.076%		
VNS	5.216%	6.898%	22.098%	3.812%	-13.092%	
CPLEX	-28.1%	-26.894%	-15.764%	-29.41%	-40.835%	-30.288%

Table 4.11: *The average percentage increase in the sum of job durations for intermediate test cases.*

	CS	GA-I	GA-II	PSO
GA-I	-3.718%			
GA-II	-15.326%	-12.027%		
PSO	1.733%	6.022%	21.608%	
TS	27.323%	33.27%	53.957%	25.723%

Table 4.12: *The average percentage increase in the sum of job durations for challenging test cases.*

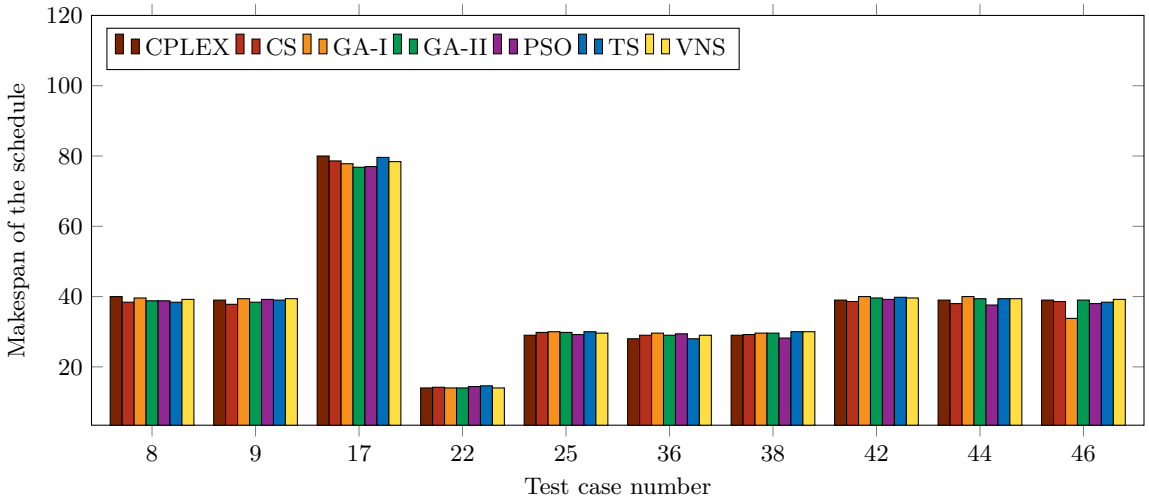
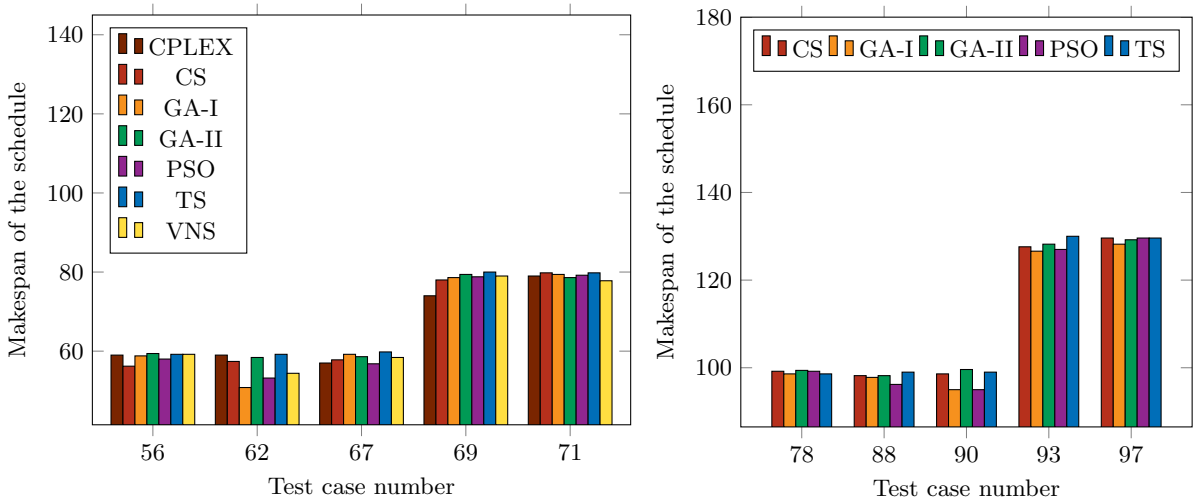


Figure 4.31: *A grouped bar chart for the makespan of the schedule for elementary test cases.*



(a) Test cases solvable by CPLEX and VNS.

(b) Test cases unsolvable by CPLEX and VNS.

Figure 4.32: *Grouped bar charts for the finish time of the schedule for intermediate test cases.*

the algorithms. This is because the algorithms attempt to insert as many jobs in the schedule to fill as much space in the schedule as possible to maximise the profit. There are, however, a few instances where an algorithm has a much shorter makespan than the maximum number of periods that the schedule is allowed to be. From the plots, it can be seen that this occurs for CPLEX in test case 69, GA-I in test cases 46, 62 and 90, PSO in test cases 62, 90 and 122 as well as VNS in test case 62. Tables 4.13–4.15 contain the average percentage increase in the makespan of one algorithm over another. There are no considerable differences between the algorithms. There are, however, subtle differences (where the percentage is greater than 2.5%) in the intermediate and challenging groups. For the intermediate test cases, this occurs for GA-II over CPLEX, TS over PSO, TS over CPLEX and VNS over CPLEX. For challenging test cases, GA-II and TS are on average somewhat better than PSO.

The idle time of a schedule is the number of time periods in which no jobs are being processed from the beginning of the schedule until the end of the schedule. Grouped bar charts of the idle time of the schedule for 30 selected test cases may be found in Figures 4.34–4.36. The average idle time of the schedule for the five times that the metaheuristic was run is plotted per test

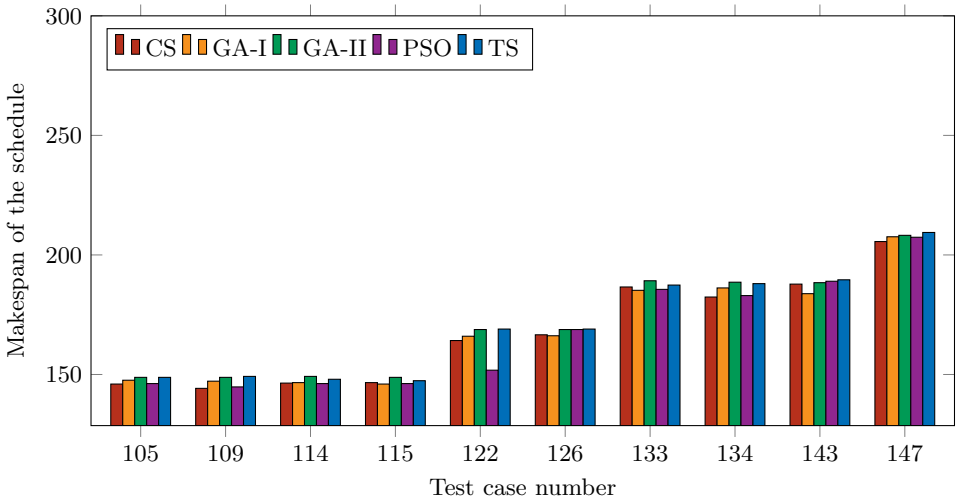


Figure 4.33: *A grouped bar chart for the makespan of the schedule for challenging test cases.*

	CS	GA-I	GA-II	PSO	TS	VNS
GA-I	1.227%					
GA-II	0.752%	-0.421%				
PSO	-0.248%	-1.413%	-0.975%			
TS	1.01%	-0.175%	0.258%	1.346%		
VNS	1.455%	0.276%	0.716%	1.794%	0.512%	
CPLEX	0.836%	-0.333%	0.108%	1.182%	-0.091%	-0.587%

Table 4.13: *The average percentage increase in the makespan of the schedule for elementary test cases.*

	CS	GA-I	GA-II	PSO	TS	VNS
GA-I	0.833%					
GA-II	1.622%	0.866%				
PSO	0.048%	-0.736%	-1.535%			
TS	2.364%	1.631%	0.78%	2.536%		
VNS	2.481%	1.45%	-0.023%	1.139%	-0.797%	
CPLEX	-0.333%	-1.29%	-2.823%	-1.629%	-3.569%	-2.751%

Table 4.14: *The average percentage increase in the makespan of the schedule for intermediate test cases.*

	CS	GA-I	GA-II	PSO
GA-I	1.065%			
GA-II	2.059%	1.006%		
PSO	-0.58%	-1.59%	-2.577%	
TS	1.981%	0.925%	-0.068%	2.807%

Table 4.15: *The average percentage increase in the makespan of the schedule for challenging test cases.*

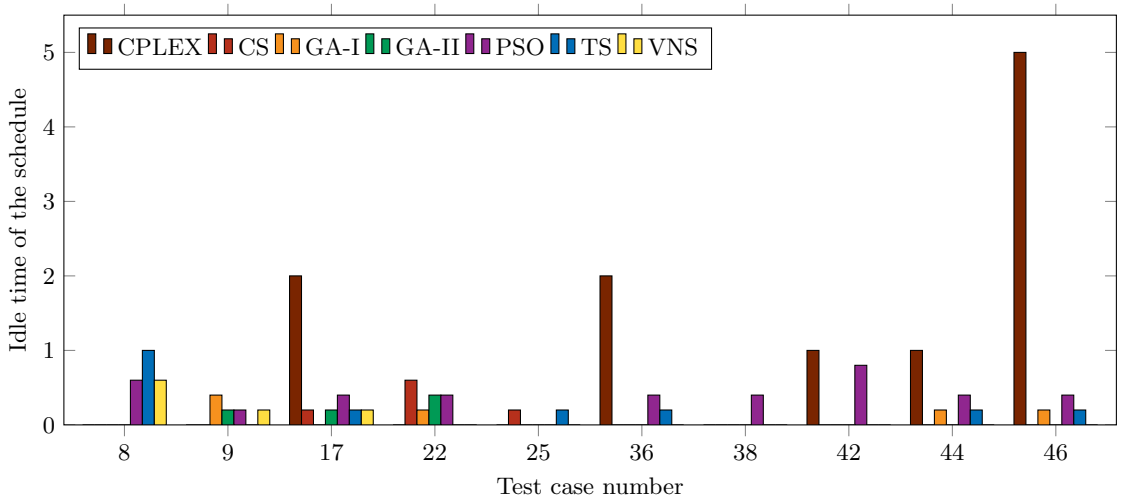
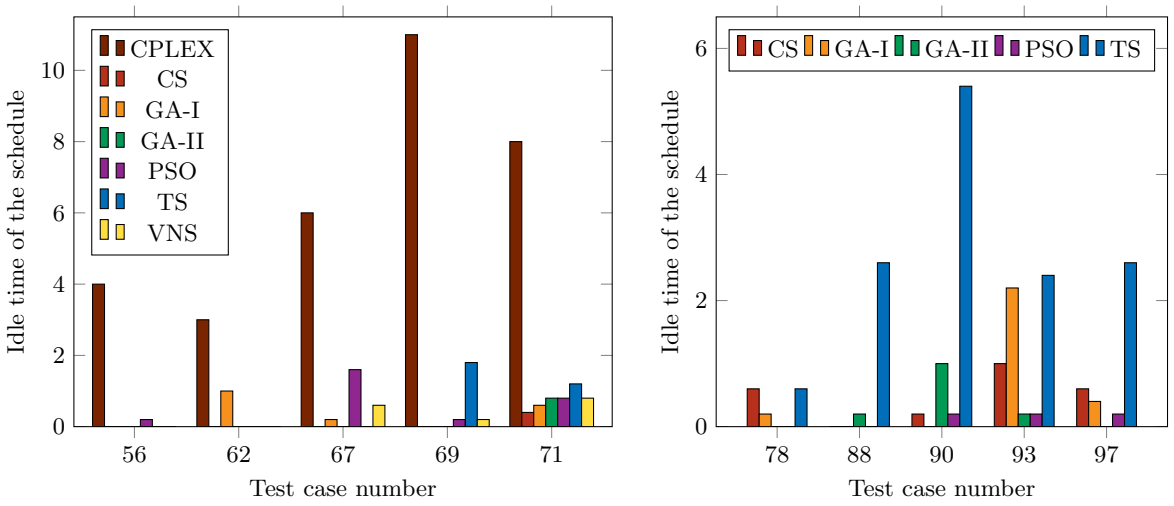


Figure 4.34: *A grouped bar chart for the idle time of the schedule for elementary test cases.*



(a) Test cases solvable by CPLEX and VNS. **(b)** Test cases unsolvable by CPLEX and VNS.

Figure 4.35: *Grouped bar charts for the idle time of the schedule for intermediate test cases.*

case. Most algorithms don't generally have a large average idle time. The two algorithms that have a large number of noticeable spikes (an average idle time greater than 3) in the plots are CPLEX and TS. This occurs at test cases 46, 56, 62, 67, 69 for CPLEX and at test cases 90, 105 and 114 for TS.

4.4.3 Jobs and available space in schedules

Figures 4.37–4.39 contain grouped bar charts for the number of scheduled jobs for 30 selected test cases. The average number of scheduled jobs of the five times that the metaheuristic was run for is plotted per test case. The number of scheduled jobs is considerably higher for CPLEX compared to the metaheuristics for the easiest test cases and gradually becomes considerably lower compared to the metaheuristics when the test cases' difficulty increases. Also, TS schedules increasingly more jobs compared to the other metaheuristics as test cases become more difficult. Tables 4.16–4.18 contain the average percentage increase in the number of scheduled jobs of one algorithm over another. These tables verify the fact that CPLEX schedules less jobs as test

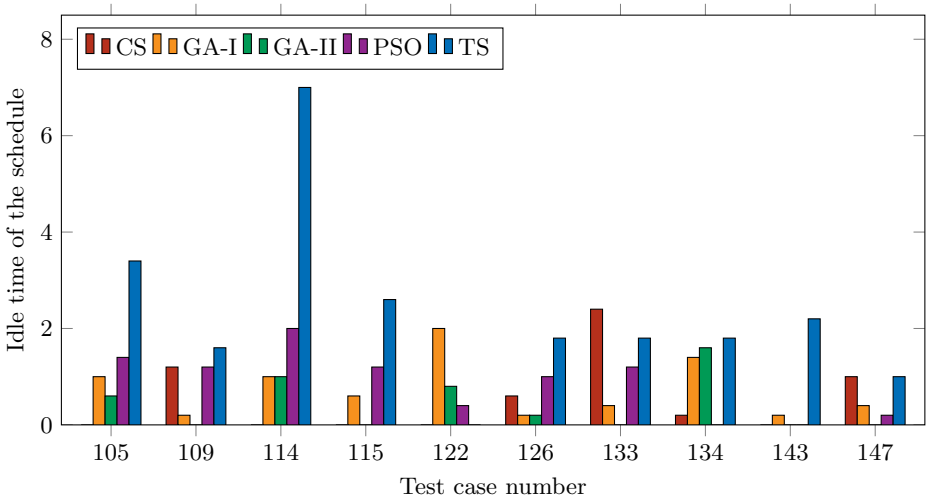


Figure 4.36: *A grouped bar chart for the idle time of the schedule for challenging test cases.*

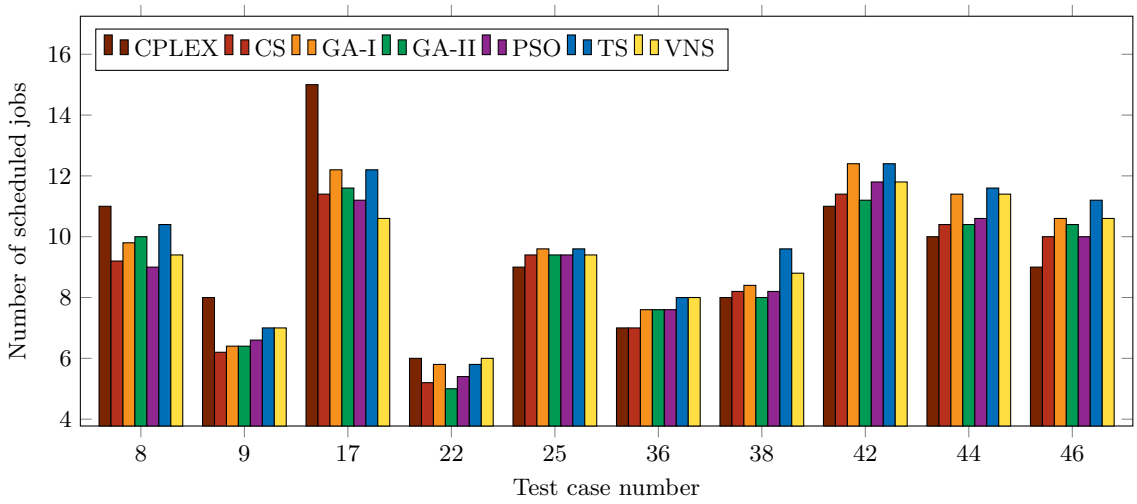
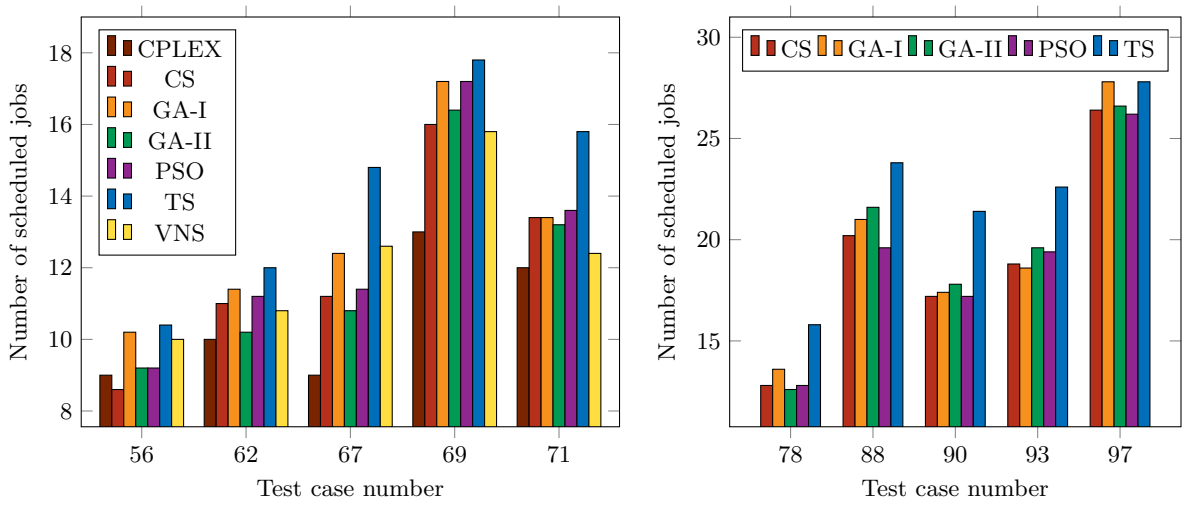


Figure 4.37: *A grouped bar chart for the number of scheduled jobs for elementary test cases.*

cases become more difficult while TS schedules more jobs as test cases become more difficult, compared to the other algorithms.

Grouped bar charts for the percentage of scheduled jobs running in parallel for 30 selected test cases may be found in Figures 4.40–4.42. The average percentage of scheduled jobs running in parallel of the five times that the metaheuristic was run for is plotted per test case. CPLEX again performs well for test cases of an easy nature being able to schedule a high percentage of jobs running in parallel compared to other algorithms. It gradually becomes weaker for more difficult test cases, most notably for test case 36 where it was unable to schedule any jobs running in parallel. TS had schedules containing the largest percentage of jobs running in parallel compared to the other metaheuristics followed by CS and PSO. The genetic algorithms performed the worst in terms of scheduling jobs that run in parallel.

The available space in a schedule is calculated by adding the available space of all the batches in the schedule. The available space for a batch is calculated using equation (3.67). Figures 4.43–4.45 contain grouped bar charts for the available space in the schedule for 30 selected test cases. The average of the available space in the schedule for the five times that the metaheuristic was run for is plotted per test case. It appears from the charts as though the available space in the



(a) Test cases solvable by CPLEX and VNS. **(b)** Test cases unsolvable by CPLEX and VNS.

Figure 4.38: *Grouped bar charts for the number of scheduled jobs for intermediate test cases.*

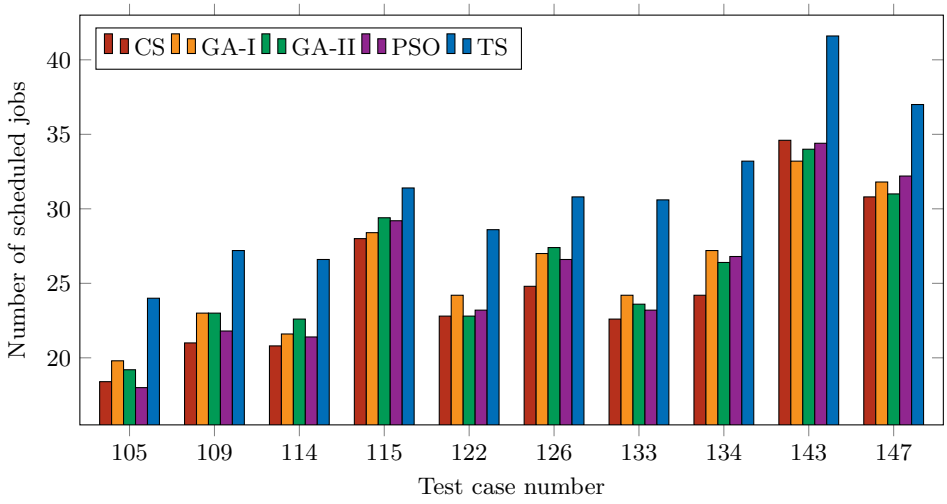


Figure 4.39: *A grouped bar chart for the number of scheduled jobs for challenging test cases.*

	CS	GA-I	GA-II	PSO	TS	VNS
GA-I	5.782%					
GA-II	1.412%	-4.068%				
PSO	-0.007%	-5.386%	-1.241%			
TS	6.529%	0.782%	5.239%	6.705%		
VNS	6.248%	0.465%	4.935%	6.37%	0.192%	
CPLEX	5.797%	0.043%	4.419%	5.802%	0.038%	-0.106%

Table 4.16: *The average percentage increase in the number of scheduled jobs for elementary test cases.*

	CS	GA-I	GA-II	PSO	TS	VNS
GA-I	6.344%					
GA-II	3.366%	-2.636%				
PSO	1.875%	-4.042%	-1.299%			
TS	17.362%	10.639%	13.723%	15.36%		
VNS	4.656%	-2.687%	1.752%	2.265%	-10.557%	
CPLEX	-22.81%	-28.043%	-24.561%	-24.015%	-33.923%	-25.405%

Table 4.17: *The average percentage increase in the number of scheduled jobs for intermediate test cases.*

	CS	GA-I	GA-II	PSO
GA-I	6.448%			
GA-II	4.341%	-1.875%		
PSO	3.867%	-2.321%	-0.365%	
TS	22.555%	15.288%	17.534%	18.145%

Table 4.18: *The average percentage increase in the number of scheduled jobs for challenging test cases.*

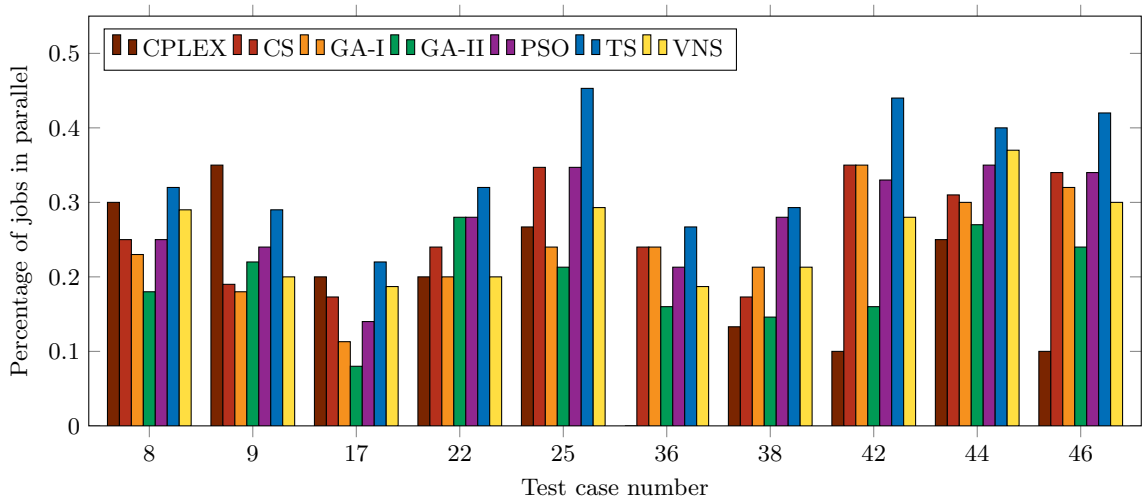
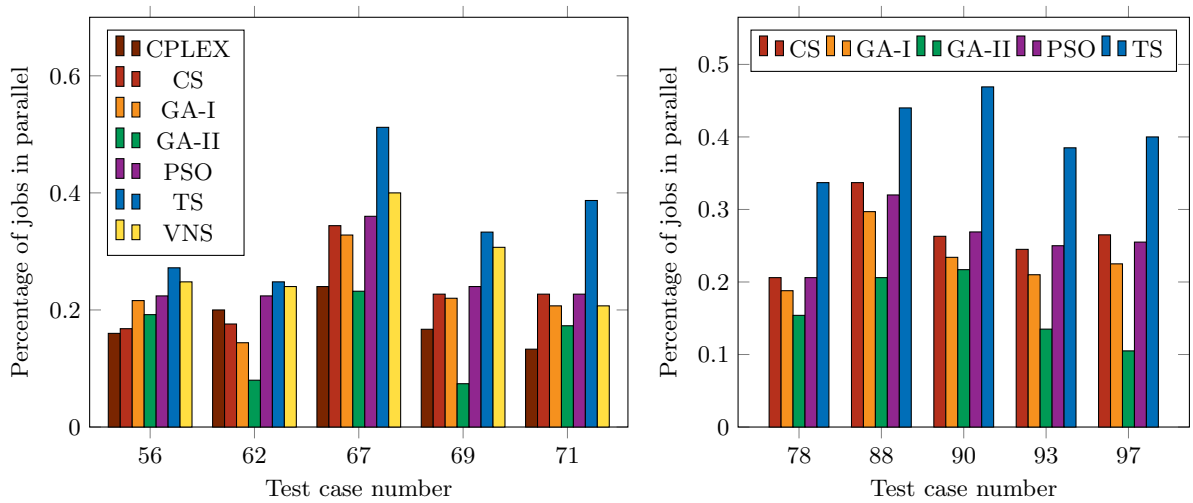


Figure 4.40: *A grouped bar chart for the percentage of parallel jobs for elementary test cases.*



(a) Test cases solvable by CPLEX and VNS. **(b)** Test cases unsolvable by CPLEX and VNS.

Figure 4.41: *Grouped bar charts for the percentage of parallel jobs for intermediate test cases.*

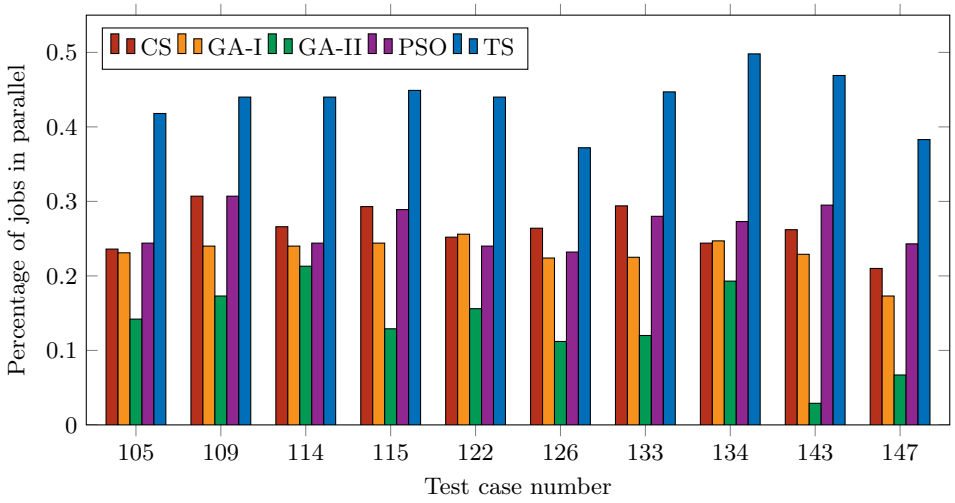


Figure 4.42: *A grouped bar chart for the percentage of parallel jobs for challenging test cases.*

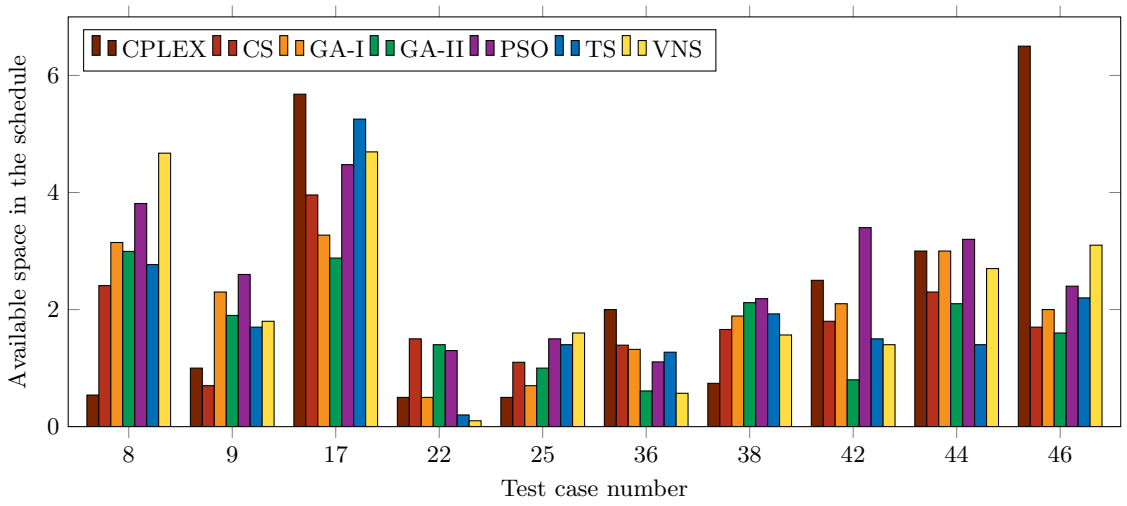
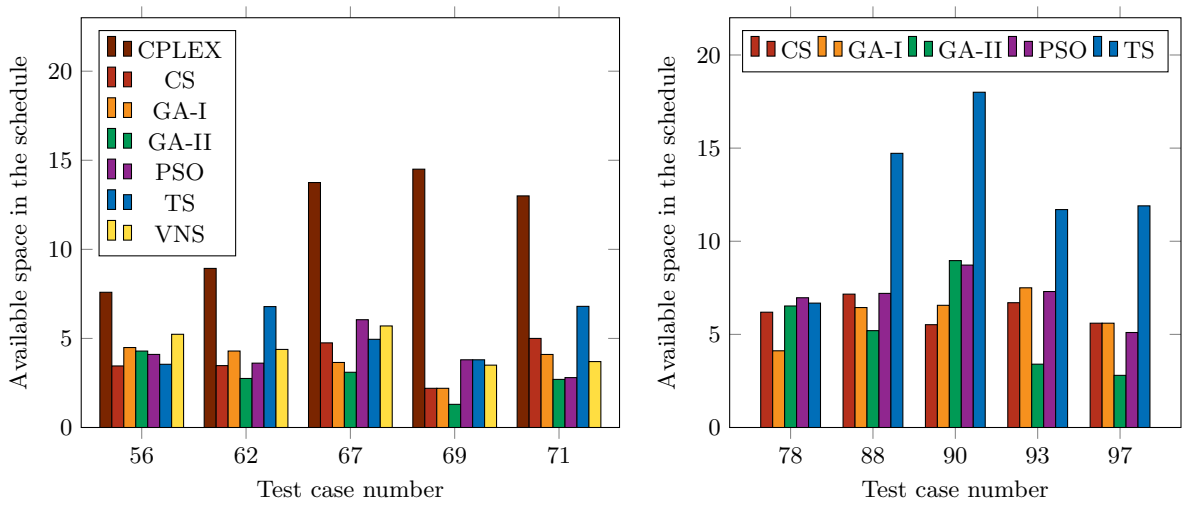


Figure 4.43: *A grouped bar chart for available space in the schedule for elementary test cases.*

schedules produced by CPLEX compared to the metaheuristics increases as test cases become more difficult. A similar observation can be made for TS. The available space in the schedules produced by TS is generally less than those of other metaheuristics for elementary test cases. It is only for the more difficult intermediate test cases and challenging test cases that more space becomes available compared to other metaheuristics. GA-II generally has the least available space in the schedule over the three groups of test cases. This is expected, since it generally schedules more jobs that do not run in parallel as seen in Figures 4.40–4.42. If a job does not run in parallel, it takes up the entire machine so that no space is available for the entire duration of the job.

4.4.4 Time to find solution

Figure 4.46 contains graphical boxplots of the time at which the solution with the highest profit throughout the search was found for elementary test case 25 and intermediate test case 75 while Figure 4.47 contains graphical boxplots for challenging test case 125. Each test case was run for a total of 30 times per metaheuristic. The boxplots for all metaheuristics are displayed.



(a) Test cases solvable by CPLEX and VNS. **(b)** Test cases unsolvable by CPLEX and VNS.

Figure 4.44: *Grouped bar charts for available space in the schedule for intermediate test cases.*

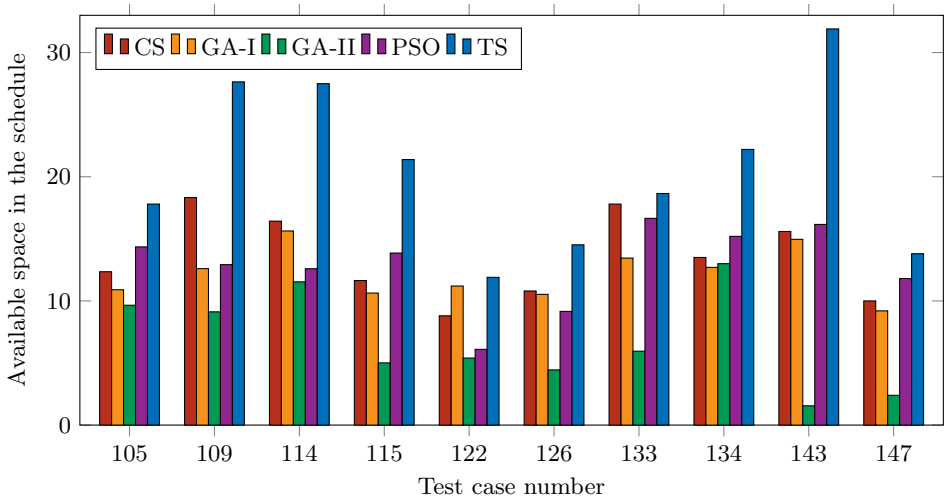
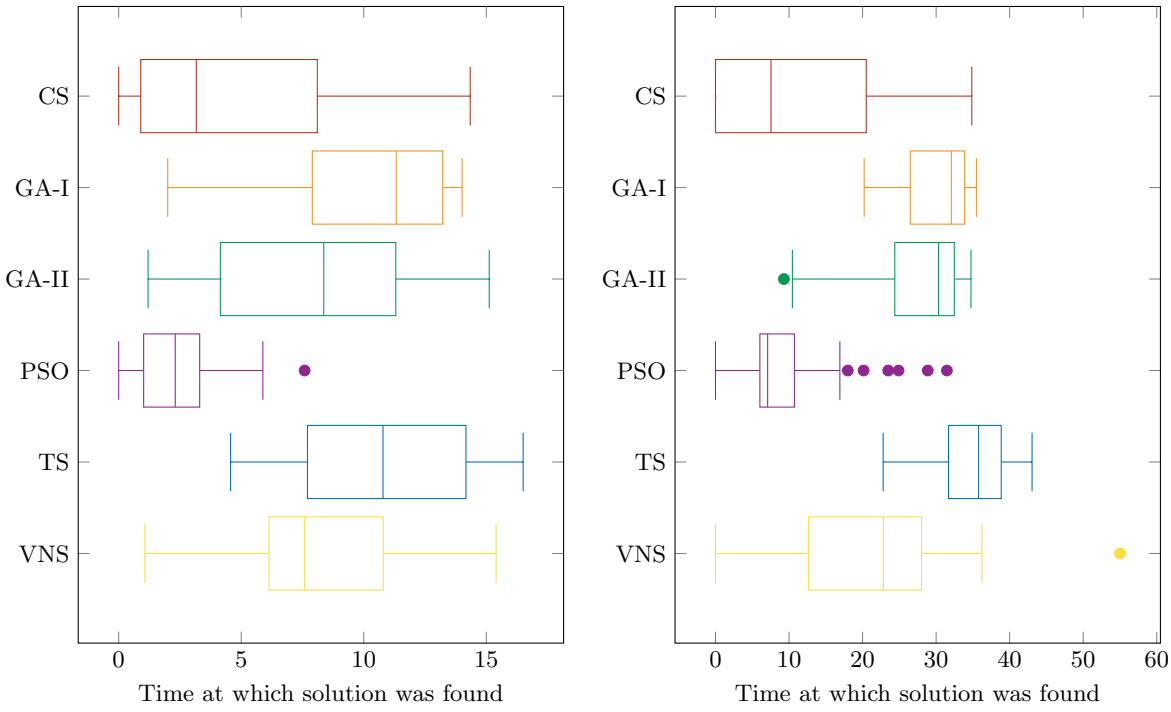


Figure 4.45: *A grouped bar chart for available space in the schedule for challenging test cases.*

The position of the boxplots on the domain gives valuable information regarding the general convergence of a metaheuristic for a difficulty group compared to other metaheuristics. On the other hand, the skewness of the boxplots indicates how condensed the times are at a section of the boxplot (there are four sections in a boxplot; the minimum time to the first quartile Q_1, Q_1 to the median Q_2, Q_2 to the third quartile Q_3 and Q_3 to the maximum time).

PSO has a low maximum value for all difficulty groups compared to the other metaheuristics. This shows that PSO has a much faster convergence compared to other metaheuristics. Despite this, it does not mean that PSO performs the best, since the boxplots say nothing in terms of the profit that PSO could achieve (refer to §4.3 for convergence graphs based on the profit). TS performs considerably better than the other algorithms in all difficulty groups in terms of convergence. The fact that its position is more to the right (its minimum and maximum times are usually much greater compared to those of other metaheuristics), shows that it does not exhibit premature convergence. CS is skewed to the right (more condensed to the left of the median than to the right) in the elementary and intermediate test cases. This indicates that CS generally convergences much quicker than its mean. On the contrary, the genetic algorithm GA-I and GA-II are skewed to the left in all three test cases. This shows that GA-I and GA-II generally take longer to converge than their mean. In conclusion, TS and the genetic algorithms converge the slowest which can be in their favour to finding schedules with a high profit.



(a) Boxplots for elementary test case 25.

(b) Boxplots for intermediate test case 75.

Figure 4.46: *Graphical boxplots of the time at which the solutions were found for elementary and intermediate test cases 25 and 75, respectively. The first quartile (Q_1) may be found at the left of the box while the right side of the box indicates the third quartile (Q_3). The vertical line inside the box represents the median (Q_2). The whiskers, the two vertical lines on either side outside the box, indicate the minimum and maximum times of the non-outliers. The individually plotted coordinates represent the outliers. They are the times that are not within the interval $[Q_1 - 1.5(Q_3 - Q_1), Q_3 + 1.5(Q_3 - Q_1)]$ where the difference between Q_1 and Q_3 is known as the interquartile range.*

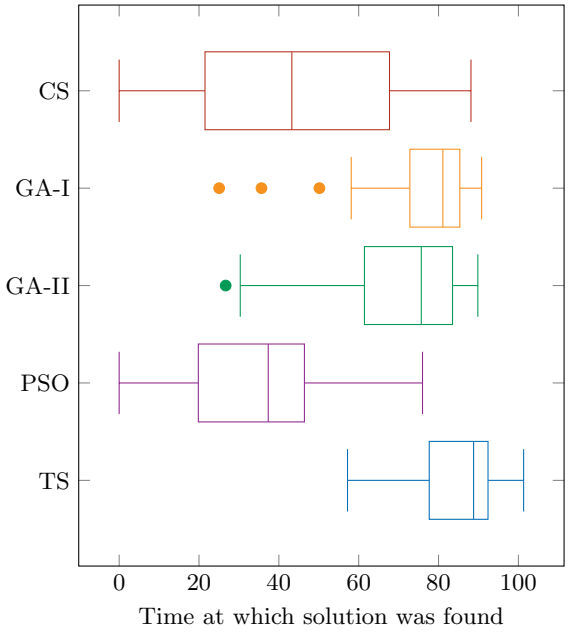


Figure 4.47: *Graphical boxplots of the time at which the solutions were found for the challenging test case 125.*

4.5 Statistical hypothesis testing

In this section, omnibus tests are used to test whether there is an overall significant difference between the profits of the algorithms for each of the generated test cases. The non-parametric omnibus test that is used is the Friedman test (in which the observations do not have to be normally distributed as seen in §4.5.1) while the ANOVA test, or more specifically the Welch ANOVA test (because of inequality in the variances as seen in §4.5.3), is used as the parametric omnibus test. If there is an overall difference in the ranks (for the Friedman test) or the means (for the Welch ANOVA test) for a specific test case, post hoc analysis is performed on the two algorithms with the highest medians (for the Friedman test) or the highest means (for the Welch ANOVA test). If the post hoc analysis indicates that there is a significant difference between the ranks/means of the pair of algorithms, the algorithm with the highest median/mean outperforms the other algorithms. It is shown in §4.5.2 that the observations are believed to be normally distributed and that their variances are heterogeneous. As a result, the Welch ANOVA test and its post hoc analyses are more powerful than the Friedman test and its post hoc analyses.

4.5.1 Non-parametric statistics

Non-parametric statistical tests are used to test the differences between two or more algorithms when the normality assumption for the data may be violated [35]. Since more than two algorithms are compared simultaneously, the Friedman test is used as opposed to tests that are designed to test two algorithms against one another when repeated measurements are conducted, such as the Wilcoxon signed-rank test and the sign test which will be discussed later in this section.

The Friedman test [26] is performed by letting r_i^j be the rank of profit i (or alternatively repeated measurement i) for algorithm j (known as group j in the literature). For each of the $m'' = 5$ times that the algorithms were run for, ranks are assigned to the algorithms using fractional ranking. The algorithm that performed the best receives a rank of 1, the second best algorithm

receives a rank of 2, and so forth. If there is a tie between algorithms, their average ordinal rank is taken. As an example, if two algorithms share 2nd place, both of them receive a rank of 2.5 (the average of 2 and 3). The next algorithm will have a rank of at least 4. The same holds for when there is a tie between three or more algorithms. If for example four algorithms share 2nd place, all of them receive a rank of 3.5 (the average of 2, 3, 4 and 5). Similarly, the next algorithm will have a rank of at least 6.

For each of the n'' algorithms,

$$\bar{r}_j = \frac{1}{m''} \sum_i r_i^j \tag{4.15}$$

is calculated which is simply the average of its ranks. The Friedman test's chi-square statistic is then calculated by

$$\chi_F^2 = \frac{12m''}{n''(n''+1)} \left[\sum_j \bar{r}_j - \frac{n''(n''+1)^2}{4} \right] \tag{4.16}$$

that is distributed according to the χ^2-distribution with $n'' - 1$ degrees of freedom under the hypotheses

H_0 : there is no significant difference between the algorithms in terms of profit
H_a : there is a significant difference between the algorithms in terms of profit

where H_0 is the null hypothesis and H_a is the alternative hypothesis. For there to be a statistically significant difference between the mean ranks of the algorithms, the null hypothesis needs to be rejected in favour of the alternative. For this to happen, χ_F^2 needs to be greater than the critical value of $\chi_{0.05}^2(5) = 11.07$ when testing 6 metaheuristics against one another or greater than $\chi_{0.05}^2(4) = 9.488$ when testing 5 metaheuristics against one another (for the test cases where VNS is omitted) at a 95% confidence interval (or a significance level of 0.05). Alternatively, the p-value needs to be less than 0.05 at a confidence interval of 95%.

The Friedman test is an omnibus test, since it can only convey whether there is a significant overall difference in the algorithms. To find the pairs of algorithms that differ from each other which in turn cause the significant overall differences in the algorithms, statistical testing methods that were developed for the purpose of comparing two groups need to be performed. According to Demšar [26], the Wilcoxon signed-rank test and the sign test may be used to detect a significant difference between two groups when the normality assumption does not necessarily hold. Although the Wilcoxon signed-rank test may be more reliable due to the normal distribution not having to be assumed and the outliers having less of an effect on the test statistics, compared to post hoc tests where the normality assumption needs to hold, the Wilcoxon signed-rank test is less powerful than these post hoc tests if the normality assumption does indeed hold. Moreover, the sign test is weaker than the Wilcoxon signed-rank test. Demšar [26] showed that for 5 repeated measurements, the one algorithm needs to outperform the other in all of the instances to reject the null hypothesis at both 90% and 95% confidence intervals.

The Wilcoxon signed-rank test, named after Frank Wilcoxon after proposing the use of his method when comparing two independent samples [101], is an alternative to the paired t-test (for which the assumption of normality needs to hold). He noted that it is more efficient than the Friedman test when two groups are being compared, since Wilcoxon's method considers both the magnitude of the differences and the signs of the differences. In order to use Wilcoxon's method, d_i is defined as the difference between the $m'' = 5$ profits of the algorithms for the ith repeated measurement for the two algorithms [26]. The differences are ranked according to their absolute values in the same way as in Friedman's method using fractional ranking. If R^+ is the sum of the ranks for all the instances where the second algorithm outperformed the

	m''															
significance level	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0.05	5	6	7	7	8	9	9	10	10	11	12	12	13	13	14	15
0.1	5	6	6	7	7	8	9	9	10	10	11	12	12	13	13	14

Table 4.19: *The critical values for the sign test where m'' ranges from 5 to 20 for significance levels of 0.05 and 0.1.*

first algorithm and R^- is the sum of the ranks for all the instances where the first algorithm outperformed the second algorithm, the sum of the ranks are given by

$$R^+ = \sum_{d_i>0} r(d_i) + \frac{1}{2}\sum_{d_i=0} r(d_i) \tag{4.17}$$

and

$$R^- = \sum_{d_i<0} r(d_i) + \frac{1}{2}\sum_{d_i=0} r(d_i) \tag{4.18}$$

where $r(d_i)$ is the rank for difference i. Ranks for which $d_i = 0$ are split evenly amongst R^+ and R^-. If there is an odd number of ranks for which $d_i = 0$, one of them is excluded in the sums. Friedman's test statistic is given by $T = \min\{R^+, R^-\}$. Finally, the z-statistic is calculated by

$$z = \frac{T - \frac{1}{4}m''(m''+1)}{\sqrt{\frac{1}{24}m''(m''+1)(2m''+1)}} \tag{4.19}$$

that is approximately normally distributed. At a significance level of 0.05, H_0 is rejected in favour of the alternative if $z < -1.96$ (or alternatively if its p-value is less than 0.05) inferring that there is a statistically significant difference between the ranks of the algorithm. This suggests that the algorithm that generally has higher profits performs significantly better than the algorithm it was compared to.

The sign test dates back to 1710 when it was used by John Arbuthnot [20] to examine the male to female birth ratio in London in the period from 1629 to 1710. He found that for each of the years the number of males born was greater than the number of females born. He is arguably the first author to use the term "statistical significance" in his research as well as writing the first paper on non-parametric tests. Nicholas Bernoulli [45] completed Arbuthnot's analysis in the years of 1710 to 1713 by fitting the binomial distribution with parameter $p = 18/35$, the first set of data to be analysed using a binomial distribution. The sign test is performed by counting the number of times p'' that an algorithm (the algorithm that performs better the most between the two algorithms) performs better than the other. The value p'' is binomially distributed. The critical value that p'' should be greater than for H_0 to be rejected, may be found in Table 4.19. The column in which $m'' = 5$ is used for the results in this section. Due to the long computational time required to execute the 150 test cases five times each for the time limits determined in §4.3, m'' was chosen to be 5. The sign test in this section is therefore performed at a significance level of 0.1 instead of 0.05 as a p-value of less than 0.05 is unattainable for $m'' = 5$.

The Friedman test alongside the post hoc tests that include the Wilcoxon signed-rank and sign test may be found in Tables 4.20–4.22. Each of the tables display the results for elementary, intermediate and challenging test cases, respectively. For each of the test cases, the first step is to determine whether there is a significant difference between the profits of the algorithms using the Friedman test. If there is a significant difference, *i.e.* if the Friedman test's p-value is less than 0.05, the algorithms with the highest and second highest median are listed. If their medians are equal, they are not listed. The median was chosen as the measure of central tendency to

determine the top performing algorithm for a test case as the mode and mean are inapplicable in this situation. The mode is not used, since the multiple occurrence of an objective function value for the algorithms does not contribute to the results, especially for more difficult test cases when results for an algorithm can vary substantially. Also, the mean is irrelevant in these tests insomuch as it is sensible to use it when both the normality assumption needs to hold for the tests to be performed and the normality assumption does indeed hold for the data. In this case, only the latter is true as will be seen in §4.5.2. The median of the profit of the schedule may be found in Tables F.1–F.3 in Appendix F.1 for all of the test cases.

The second step is to perform post hoc analyses for the test cases that have a best and second best algorithm. Post hoc analysis is carried out on only this pair of algorithms instead of all $\binom{6}{2} = 15$ pairs of algorithms for test cases 1–75 and $\binom{5}{2} = 10$ pairs of algorithms for test cases 76–150. The first post hoc test is the Wilcoxon signed-rank test followed by the sign test. Wilcoxon's more powerful test is performed at a confidence interval of 95% while the less powerful sign test is conducted at a confidence interval of 90%. If both the p-value of Wilcoxon's method is less than 0.05 and the sign test's p-value is less than 0.1, the best algorithm can be said to outperform the second best algorithm in terms of their medians and the ranking procedures used by the post hoc tests. As a result, the algorithm significantly outperforms all other algorithms based on their medians. An algorithm that outperforms all other algorithms for a specific test case is in bold, evident in Tables 4.20–4.22. VNS outperformed the algorithms in 6, and 1 of the elementary and intermediate test cases, respectively, while TS outperformed the algorithms in 11, 38 and 41 of the three groups of test cases, respectively. For test cases 51–75 (the test cases that VNS could solve), TS outperformed the algorithms a total of 18 times. CS, GA-I, GA-II and PSO were unable to outperform the algorithms in any of the test cases.

4.5.2 Testing for normality and equal variances

In order to use the one-way analysis of variance (ANOVA) technique performed on the profits of the test cases in §4.5.3, six assumptions need to hold [59].

1. The dependent variable should be continuous. This is the case for the scheduling problem as the profit of the schedule can range from 0 (if the schedule doesn't contain any jobs) to the optimal value for the profit of the schedule.

2. The second assumption is that the independent variable needs to incorporate at least two categorical independent groups. This assumption holds, since six algorithms are being compared to one another and results of one algorithm does not depend on the results of another.

3. There should be independence in the observations. The assumption is met due to the random generating of solutions for each of the algorithms. Each time a specific metaheuristic is executed, it starts with possibly a different set of initial solutions compared to the previous time it was executed. Additionally, the random components of the metaheuristics cause the observations not to have a relationship between one another.

4. Observations should not contain any significant outliers. There are indeed no significant outliers in the profits using boxplots, since each metaheuristic was executed five times for each of the test cases. Five observations does not have any outliers, since the lowest profit is assigned as the minimum value, the second lowest as the first quartile, the third lowest as the median (or second quartile), the second highest as the third quartile and the highest as the maximum value. Outliers would be possible if each of the metaheuristics were executed six or more times.

	Friedman test		Algorithm		Wilcoxon signed-rank test		Sign test
test case	chi-square statistic	p-value	first	second	z-statistic	p-value	p-value
1	8.69	0.122					
2	15.42	0.009	VNS	CS	-1.826	0.068	0.125
3	19.76	0.001					
4	8.46	0.133					
5	19.52	0.002					
6	20.43	0.001	GA-I	PSO	-0.944	0.345	0.375
7	17.64	0.003	**VNS**	GA2	-2.023	0.043	0.063
8	21.57	0.001	**TS**	GA-I	-2.023	0.043	0.063
9	16.99	0.005	**TS**	VNS	-2.023	0.043	0.063
10	4.43	0.489					
11	20.98	0.001	VNS	GA-I	-1.841	0.066	0.125
12	15.46	0.009	**VNS**	GA-I	-2.023	0.043	0.063
13	9.09	0.106					
14	14.32	0.014	VNS	GA-I	-1.633	0.102	0.25
15	8.72	0.121					
16	14.83	0.011	TS	PSO	-1.753	0.08	0.375
17	14.03	0.015	TS	GA-II	-1.753	0.08	0.375
18	18.25	0.003					
19	7.07	0.215					
20	10.37	0.065					
21	16.02	0.007	GA-I	TS	-1.473	0.141	0.625
22	12.59	0.028					
23	12.3	0.031	**VNS**	PSO	-2.023	0.043	0.063
24	11.01	0.051					
25	10.95	0.052					
26	18.62	0.002	TS	VNS	-1.826	0.068	0.125
27	15	0.01	VNS	TS	-0.405	0.686	1
28	14.14	0.015	**TS**	GA-I	-2.023	0.043	0.063
29	15.98	0.007	TS	GA-II	-1.826	0.068	0.125
30	17.46	0.004	VNS	GA-I	-1.753	0.08	0.375
31	19.4	0.002	**VNS**	GA-I	-2.023	0.043	0.063
32	21.34	0.001	**VNS**	GA-I	-2.032	0.042	0.063
33	17.23	0.004	**VNS**	TS	-2.023	0.043	0.063
34	19.51	0.002	**TS**	GA-II	-2.023	0.043	0.063
35	15.51	0.008	**TS**	GA-I	-2.023	0.043	0.063
36	15.86	0.007	**TS**	GA-I	-2.023	0.043	0.063
37	15.66	0.008	TS	GA-I	-1.604	0.109	0.25
38	15.97	0.007	**TS**	VNS	-2.032	0.042	0.063
39	16.66	0.005	TS	VNS	-0.405	0.686	1
40	12.24	0.032	**TS**	CS	-2.023	0.043	0.063
41	22.49	0	**TS**	VNS	-2.023	0.043	0.063
42	19.63	0.001	**TS**	GA-I	-2.023	0.043	0.063
43	18.1	0.003	TS	GA-I	-1.826	0.068	0.125
44	18.45	0.002	TS	VNS	-1.214	0.225	1
45	19.74	0.001	**TS**	GA-I	-2.023	0.043	0.063
46	13.34	0.02	**TS**	VNS	-2.023	0.043	0.063
47	20.43	0.001	TS	VNS	-1.753	0.08	0.375
48	13.33	0.02	VNS	TS	-1.084	0.279	1
49	19.63	0.001	TS	VNS	-0.405	0.686	1
50	9.8	0.081					

Table 4.20: *The results of the Friedman test for elementary test cases with post hoc analyses that include the Wilcoxon signed-rank and sign tests.*

	Friedman test		Algorithm		Wilcoxon signed-rank test		Sign test
test case	chi-square statistic	p-value	first	second	z-statistic	p-value	p-value
51	9.71	0.084					
52	15.58	0.008	VNS	GA-I	-1.753	0.08	0.375
53	16.09	0.007	VNS	GA-I	-0.944	0.345	0.375
54	21.11	0.001	**VNS**	GA-I	-2.023	0.043	0.063
55	19.86	0.001	TS	GA-I	-1.753	0.08	0.375
56	21.49	0.001	**TS**	GA-I	-2.023	0.043	0.063
57	18.49	0.002	**TS**	VNS	-2.023	0.043	0.063
58	15.86	0.007	TS	GA-I	-1.761	0.078	0.375
59	17.69	0.003	**TS**	GA-I	-2.023	0.043	0.063
60	19.86	0.001	**TS**	VNS	-2.023	0.043	0.063
61	20.09	0.001	**TS**	GA-I	-2.023	0.043	0.063
62	16.77	0.005	**TS**	GA-I	-2.023	0.043	0.063
63	20.77	0.001	**TS**	GA-I	-2.023	0.043	0.063
64	17.57	0.004	**TS**	GA-I	-2.023	0.043	0.063
65	14.14	0.015	**TS**	GA-II	-2.023	0.043	0.063
66	18.14	0.003	TS	GA-II	-1.753	0.08	0.375
67	16.89	0.005	**TS**	GA-I	-2.023	0.043	0.063
68	20.66	0.001	**TS**	GA-I	-2.023	0.043	0.063
69	21.67	0.001	**TS**	PSO	-2.023	0.043	0.063
70	21.23	0.001	**TS**	GA-I	-2.023	0.043	0.063
71	16.77	0.005	**TS**	GA-I	-2.023	0.043	0.063
72	19.74	0.001	**TS**	VNS	-2.023	0.043	0.063
73	19.74	0.001	**TS**	GA-I	-2.032	0.042	0.063
74	17	0.004	**TS**	VNS	-2.023	0.043	0.063
75	20.77	0.001	**TS**	GA-II	-2.023	0.043	0.063
76	4	0.406					
77	14.72	0.005	**TS**	GA-II	-2.023	0.043	0.063
78	17.12	0.002	**TS**	GA-I	-2.023	0.043	0.063
79	14.88	0.005	**TS**	GA-I	-2.023	0.043	0.063
80	14.24	0.007	**TS**	GA-I	-2.023	0.043	0.063
81	14.88	0.005	**TS**	GA-II	-2.023	0.043	0.063
82	13.92	0.008	**TS**	GA-II	-2.023	0.043	0.063
83	13.28	0.01	**TS**	GA-I	-2.023	0.043	0.063
84	8.8	0.066					
85	12.32	0.015	**TS**	GA-I	-2.023	0.043	0.063
86	17.28	0.002	**TS**	GA-II	-2.023	0.043	0.063
87	11.04	0.026	**TS**	GA-I	-2.023	0.043	0.063
88	14.72	0.005	**TS**	GA-II	-2.023	0.043	0.063
89	13.6	0.009	**TS**	GA-I	-2.023	0.043	0.063
90	17.28	0.002	**TS**	GA-I	-2.023	0.043	0.063
91	4.64	0.326					
92	9.28	0.054					
93	10.72	0.03	**TS**	GA-II	-2.023	0.043	0.063
94	14.72	0.005	**TS**	GA-I	-2.023	0.043	0.063
95	16.64	0.002	**TS**	GA-I	-2.023	0.043	0.063
96	8.8	0.066					
97	12.16	0.016	**TS**	GA-I	-2.023	0.043	0.063
98	15.36	0.004	**TS**	GA-II	-2.023	0.043	0.063
99	12.32	0.015	**TS**	PSO	-2.023	0.043	0.063
100	16.16	0.003	**TS**	GA-I	-2.023	0.043	0.063

Table 4.21: *The results of the Friedman test for intermediate test cases with post hoc analyses that include the Wilcoxon signed-rank and sign tests.*

	Friedman test		Algorithm		Wilcoxon signed-rank test		Sign test
test case	chi-square statistic	p-value	first	second	z-statistic	p-value	p-value
101	11.68	0.02	**TS**	GA-I	-2.023	0.043	0.063
102	12.16	0.016	**TS**	GA-I	-2.023	0.043	0.063
103	9.76	0.045	GA-I	TS	-0.674	0.5	0.375
104	16.16	0.003	**TS**	GA-I	-2.023	0.043	0.063
105	18.4	0.001	**TS**	GA-I	-2.023	0.043	0.063
106	14.88	0.005	**TS**	GA-II	-2.023	0.043	0.063
107	15.2	0.004	**TS**	GA-I	-2.023	0.043	0.063
108	14.72	0.005	**TS**	GA-I	-2.023	0.043	0.063
109	15.84	0.003	**TS**	GA-II	-2.023	0.043	0.063
110	13.28	0.01	**TS**	GA-II	-2.023	0.043	0.063
111	5.12	0.275					
112	13.44	0.009	**TS**	GA-I	-2.023	0.043	0.063
113	13.28	0.01	**TS**	PSO	-2.023	0.043	0.063
114	15.52	0.004	**TS**	GA-II	-2.023	0.043	0.063
115	12.64	0.013	**TS**	GA-II	-2.023	0.043	0.063
116	19.36	0.001	**TS**	GA-I	-2.032	0.042	0.063
117	14.88	0.005	**TS**	PSO	-2.023	0.043	0.063
118	14.72	0.005	PSO	TS	-0.944	0.345	0.375
119	5.76	0.218					
120	16.69	0.002	**TS**	PSO	-2.023	0.043	0.063
121	12.8	0.012	**TS**	GA-I	-2.023	0.043	0.063
122	15.84	0.003	**TS**	GA-I	-2.023	0.043	0.063
123	15.52	0.004	**TS**	PSO	-2.023	0.043	0.063
124	11.68	0.02	**TS**	GA-II	-2.023	0.043	0.063
125	17.66	0.001	**TS**	GA-I	-2.023	0.043	0.063
126	14.56	0.006	**TS**	GA-I	-2.023	0.043	0.063
127	19.36	0.001	**TS**	GA-I	-2.023	0.043	0.063
128	15.84	0.003	**TS**	GA-I	-2.023	0.043	0.063
129	17.17	0.002	**TS**	GA-I	-2.023	0.043	0.063
130	7.52	0.111					
131	13.28	0.01	**TS**	GA-I	-2.023	0.043	0.063
132	14.88	0.005	**TS**	GA-I	-2.023	0.043	0.063
133	13.28	0.01	**TS**	GA-I	-2.032	0.042	0.063
134	14.72	0.005	**TS**	PSO	-2.023	0.043	0.063
135	14.24	0.007	**TS**	GA-II	-2.023	0.043	0.063
136	14.88	0.005	**TS**	GA-II	-2.023	0.043	0.063
137	17.12	0.002	**TS**	GA-I	-2.023	0.043	0.063
138	6.24	0.182					
139	14.83	0.005	**TS**	GA-II	-2.023	0.043	0.063
140	13.92	0.008	**TS**	GA-II	-2.023	0.043	0.063
141	15.52	0.004	**TS**	GA-II	-2.023	0.043	0.063
142	17.28	0.002	**TS**	GA-II	-2.023	0.043	0.063
143	16.48	0.002	**TS**	GA-II	-2.023	0.043	0.063
144	12.96	0.011	**TS**	GA-II	-2.023	0.043	0.063
145	16.48	0.002	TS	PSO	-1.753	0.08	0.375
146	12.48	0.014	GA-I	TS	-0.944	0.345	1
147	16.64	0.002	**TS**	PSO	-2.023	0.043	0.063
148	14.72	0.005	**TS**	GA-I	-2.023	0.043	0.063
149	16.48	0.002	**TS**	GA-I	-2.023	0.043	0.063
150	12.64	0.013	TS	PSO	-0.944	0.345	1

Table 4.22: *The results of the Friedman test for challenging test cases with post hoc analyses that include the Wilcoxon signed-rank and sign tests.*

5. The dependent variable should be approximately normally distributed for each of the groups in the independent variable. Stated in terms of the problem considered in this thesis, the profits of a test case should be approximately normally distributed for each of the individual metaheuristics. This can be tested using the Kolmogorov-Smirnov (KS) test or the more powerful Shapiro-Wilk (SW) test which will be tested in this section [59]. If the normality assumption is violated, one-way ANOVA cannot be used to compare the means of two or more algorithms. As a result, only non-parametric tests can be used, such as those considered in §4.5.1, as the means of the profits are irrelevant.

6. The final assumption that needs to hold is the homogeneity of variances (or homoscedasticity), *i.e.* the variances of the observations in each of the groups in the independent variable need to be the same. That is, the variances of the profits for each of the metaheuristics for a test case needs to be the same to do a one-way ANOVA on that test case. This can be tested using Levene's test for homogeneity of variances for each of the test cases, which is also tested in this section. If this assumption is violated, the Welch ANOVA can be used instead of the one-way ANOVA with post hoc tests that include Tamhane's T2, Dunnett's T3 and Games-Howell [59]. On the other hand, post hoc tests that can be used if the assumption holds may include the Tukey and Dunnett tests as well as the Bonferroni correction.

The last two assumptions will be addressed in this section. The KS and SW tests will be utilised to test for normality. The KS test is named after Andrey Kolmogorov [54] and Nikolai Smirnov [91]. Kolmogorov contributed towards the KS test statistic and the Kolmogorov distribution while Smirnov produced a table of the distribution. The KS test is used to test whether a set of observations is drawn from a specified continuous distribution in general, *i.e.* the test is not solely dedicated for data which is believed to be from a normal distribution. It has two notable advantages over its usual alternative, the chi-square test. The first of these advantages is that the KS test can be used for a small number of observations whereas the chi-square test will produce unreliable results if the sample size is too small. Secondly, the KS test is more powerful than the chi-square for the most part for a sample of any size [83]. The chi-square test will therefore not be considered in this section.

The procedure for testing whether a sample X is approximately normally distributed using the KS test follows. If there are a total of m'' observations, the value

$$W_{KS} = \max_X |F^*(X) - S_{m''}(X)| \tag{4.20}$$

is calculated where $F^*(X)$ is the cumulative normal distribution function with the sample mean and sample variance (with denominator $n-1$) as parameters, and $S_{m''}(X)$ is the cumulative distribution function of X [63]. If the value of W_{KS} is at least as large as its critical value, obtained by Monte Carlo calculations provided in the table of critical values by Lilliefors [63], the null hypothesis is not rejected in favour of the alternative that the sample does not follow the normal distribution. Alternatively, the p-value should be greater than the significance level for the sample to be approximately normally distributed.

Unlike the KS test which is designed for a specified continuous distribution, the SW test was purely formulated to test whether a sample was drawn from a normally distributed population. The SW test is named after Samuel Shapiro and Martin Wilk [88] who together introduced the method in a paper. It is the most powerful test for normality compared to the KS test, Lilliefors test and Anderson-Darling test [84]. It was noted by Razali *et al.* [84] that the power of both the KS test and SW test is low for small sample sizes. This issue will also be addressed in this section. The SW test was the first statistical test to detect whether skewness or kurtosis causes

	m''									
significance level	5	10	15	20	25	30	35	40	45	50
0.05	0.762	0.842	0.881	0.905	0.918	0.927	0.934	0.940	0.945	0.947
0.1	0.806	0.869	0.901	0.920	0.931	0.939	0.944	0.949	0.953	0.955

Table 4.23: *The critical values for the SW test where m'' ranges from 5 to 50 for significance levels of 0.05 and 0.1.*

observations to depart from normality. It is the preferred test for normality due to its good power properties as other tests have a higher chance of producing a type I error (incorrectly rejecting the true null hypothesis when the data is in fact normally distributed).

The procedure for the SW test starts off by ordering the observations such that $x_{i1} < x_{i2} < \ldots < x_{im''}$ [84] where x_{ij} is the profit for observation j when algorithm i was executed. The test statistic is then given by

$$W_{SW} = \frac{\left(\sum_{j=1}^{m''} a_j x_{ij}\right)^2}{\sum_{j=1}^{m''}(x_{ij} - \bar{x}_i)^2} \tag{4.21}$$

where

$\bar{x}_i$ is the mean of the sample observations when algorithm i was executed and
$\mathbf{a}$ is the vector $(a_1, a_2, \ldots, a_{m''}) = \frac{\mathbf{m}^T V^{-1}}{(\mathbf{m}^T V^{-1} V^{-1} \mathbf{m})^{1/2}}$.

The vector $\mathbf{m} = (\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_{m''})^T$ contains the expected values of the order statistics that are independent and identically distributed random variables sampled from the normal distribution. The matrix V is the covariance matrix of these order statistics. Generating the table of critical values for the test statistic W_{SW} is not an easy task. Shapiro & Wilk [88] sampled many normally distributed variables, plotted empirical percentage points over a range of values for W_{SW} to approximate and used the critical values of the distributions. The critical values of W_{SW} under a significance level of 0.05 and 0.1 are given in Table 4.23 for various values of m''. Alternatively and similar to that of the KS test, if the p-value is greater than 0.05, there is insufficient evidence to reject the null hypothesis that the observations are approximately normally distributed at a confidence interval of 95%. If the p-value is less than 0.05, the sample is drawn from a distribution other than the normal distribution.

A total of 30 observations were generated for each of the test cases to avoid a low power in the KS and SW tests. To keep the computational time of generating results for the testing of normality roughly the same as generating results for other sections in this chapter, a fifth of the 150 test cases are used in the normality tests. The results for the KS and SW tests may be found in Table 4.24. Out of the 30 test cases, 26, 27, 28, 25 and 24 of the test cases' observations are approximately normally distributed for CS, GA-I, GA-II, PSO and TS, respectively, using the KS test. Using the more powerful SW test, 25, 25, 29, 26 and 24 of the 30 test cases' observations are approximately normally distributed for CS, GA-I, GA-II, PSO and TS, respectively, using the SW test. A total of 11 of the 15 test cases' observations are approximately normally distributed for VNS using both the KS and SW tests. Since 85.15% of the 330 normality tests supplied sufficient evidence to not reject the null hypothesis that the observations were drawn from a normal distribution, assumption 5 may hold. As a result, one-way ANOVA may be used to compare the means of the algorithms.

The post hoc tests that can be used, depends on whether the variances of the profit for the different algorithms when running the results of a test case are approximately the same or not. Levene's test for equality of variances can be used to assess whether the variances of two or more groups for a variable are the same. The variable is the algorithms consisting of 5 or 6 groups

(representing each of the algorithms depending on whether VNS is included in the analysis or not) for the considered scheduling problem. The hypotheses

$$\begin{aligned} H_0 &: \text{ the population variances are equal} \\ H_a &: \text{ the population variances are not equal} \end{aligned}$$

are tested. The test statistic for Levene's test [61] is given by

$$W_L = \frac{n''(m''-1)}{n''-1} \frac{\sum_{i=1}^{n''} m''(v_i - v)^2}{\sum_{i=1}^{n''} \sum_{j=1}^{m''} (v_{ij} - v_i)^2} \tag{4.22}$$

where

$$v_{ij} = |x_{ij} - \bar{x}_i| \tag{4.23}$$

$$v_i = \frac{1}{m''} \sum_{j=1}^{m''} v_{ij} \text{ and} \tag{4.24}$$

$$v = \frac{1}{m''n''} \sum_{i=1}^{n''} \sum_{j=1}^{m''} v_{ij}. \tag{4.25}$$

The critical values of W_L are read from an F-table as W_L is approximately F-distributed with $n''-1$ and $n''(m''-1)$ degrees of freedom [9]. The null hypothesis is rejected if W_L is greater than the critical value. Alternatively, if the p-value is less than the significance level, there is insufficient evidence to conclude that the population variances are equal.

There are three other variations of Levene's test. The first of these variations is called the Brown-Forsythe test [9]. In this test $v_{ij} = |x_{ij} - \tilde{x}_i|$ where $\tilde{x}_i$ is the median of the profits for algorithm i. This test performs well when the chi-square distribution with four degrees of freedom best fits the distribution of the profits. The second variation [9] is when the trimmed mean is used in the calculation of $\bar{x}_i$ instead of the mean. The trimmed mean is when a percentage of observations are discarded when calculating the mean, typically between 5% and 25% of the observations. It was shown using Monte Carlo studies that the trimmed mean is appropriate when the profits follow a Cauchy distribution. The final variation is when the median is used and the degrees of freedom in the F-test is adjusted. This variation was designed to address unequal variance issues in independent groups [71].

All four of the variations were tested. The results for the same test cases as those used in the KS and SW tests may be found in Table 4.25. It is evident from the table that the null hypothesis is rejected in favour of the alternative for all of the tested test cases. The variances of the profit for the algorithms are not approximately equal for all of the tested test cases at a significance level of 0.05. This is due to all of the p-values being less than 0.05. As a result, the Welch ANOVA should be employed instead of the one-way ANOVA. Moreover, post hoc tests Tamhane's T2, Dunnett's T3 and Games-Howell need to be used as they were designed for when the observations are approximately normally distributed but the variances are unequal. The Welch ANOVA and its post hoc tests are performed in the following section.

4.5.3 Parametric statistics

The one-way ANOVA is a technique that is used to compare the means of two or more populations. It can be applied if all six of the assumptions mentioned in §4.5.2 hold that includes the assumption that all of the populations that are being compared are approximately normally distributed. Additionally, the variances of the populations need to be equal. If the variances are,

	Kolmogorov-Smirnov test						Shapiro-Wilk test					
test case	CS	GA-I	GA-II	PSO	TS	VNS	CS	GA-I	GA-II	PSO	TS	VNS
5	0.172	0.007	0.022	0.037	0.004	0	0.041	0	0.077	0.189	0	0
10	0.2	0.2	0.129	0.2	0.2	0.2	0.999	0.536	0.071	0.602	0.385	0.345
15	0	0	0	0	0	0	0	0	0	0	0	0
20	0.015	0.001	0.2	0.2	0.007	0	0.139	0.026	0.708	0.118	0	0.001
25	0.04	0.2	0.2	0.186	0.2	0.2	0.019	0.575	0.201	0.327	0.283	0.21
30	0.2	0.2	0.164	0.2	0.2	0.2	0.858	0.93	0.602	0.933	0.788	0.719
35	0.2	0.2	0.2	0.054	0	0.2	0.31	0.418	0.858	0.034	0	0.1
40	0.2	0.193	0.2	0.2	0.022	0.2	0.516	0.507	0.987	0.6	0.127	0.592
45	0.2	0.2	0.2	0.2	0.2	0.2	0.409	0.165	0.669	0.543	0.057	0.493
50	0.2	0.2	0.2	0.113	0.2	0.003	0.196	0.977	0.56	0.041	0.646	0.021
55	0.2	0.2	0.2	0.2	0.108	0.2	0.482	0.604	0.705	0.445	0.052	0.897
60	0.2	0.157	0.2	0.2	0.2	0.2	0.313	0.696	0.132	0.093	0.914	0.862
65	0.2	0.2	0.2	0.2	0.2	0.2	0.025	0.625	0.503	0.981	0.66	0.361
70	0.2	0.2	0.2	0.131	0.173	0.2	0.209	0.328	0.416	0.371	0.022	0.451
75	0.2	0.2	0.2	0.2	0.2	0.2	0.298	0.089	0.889	0.179	0.451	0.258
80	0.2	0.2	0.2	0.2	0.178		0.608	0.714	0.816	0.228	0.612	
85	0.008	0.2	0.2	0.2	0.2		0	0.835	0.738	0.732	0.241	
90	0.2	0.2	0.2	0.005	0.2		0.384	0.065	0.705	0.124	0.447	
95	0.2	0.2	0.2	0.027	0.2		0.538	0.558	0.409	0.073	0.719	
100	0.2	0.191	0.2	0.2	0.167		0.871	0.527	0.054	0.055	0.027	
105	0.2	0.105	0.2	0.2	0.028		0.113	0.425	0.4	0.345	0.1	
110	0.2	0.152	0.2	0.2	0.2		0.866	0.543	0.666	0.121	0.531	
115	0.2	0.2	0.2	0.2	0.145		0.438	0.296	0.749	0.731	0.107	
120	0.179	0.2	0.2	0.2	0.2		0.694	0.726	0.154	0.109	0.49	
125	0.2	0.2	0.2	0.2	0.2		0.624	0.535	0.653	0.167	0.256	
130	0.054	0.2	0.2	0.2	0.2		0.08	0.033	0.297	0.132	0.82	
135	0.2	0.2	0.07	0.2	0.2		0.486	0.713	0.33	0.726	0.16	
140	0.2	0.2	0.2	0.067	0.2		0.494	0.022	0.965	0.118	0.902	
145	0.2	0.125	0.2	0.002	0.2		0.18	0.207	0.107	0.005	0.295	
150	0.2	0.2	0.2	0.2	0.186		0.32	0.16	0.096	0.12	0.544	

Table 4.24: *The results (p-values) of the normality tests that include the Kolmogorov-Smirnov and Shapiro-Wilk tests for 10 selected test cases in each difficulty group.*

test case	p-value for Levene's test			
	Mean	Median	Median with adjusted degrees of freedom	Trimmed mean
5	0	0	0	0
10	0	0	0	0
15	0	0.001	0.001	0
20	0	0	0	0
25	0	0	0	0
30	0	0	0	0
35	0	0	0	0
40	0.001	0.002	0.003	0.001
45	0	0	0	0
50	0	0	0	0
55	0.008	0.017	0.018	0.008
60	0	0	0	0
65	0	0	0	0
70	0	0	0	0
75	0	0	0	0
80	0.001	0.002	0.002	0.002
85	0	0	0	0
90	0	0	0	0
95	0	0.001	0.001	0
100	0.002	0.002	0.003	0.002
105	0	0	0	0
110	0.001	0.002	0.002	0.001
115	0	0	0	0
120	0	0	0	0
125	0	0	0	0
130	0	0	0	0
135	0.005	0.009	0.009	0.006
140	0	0	0	0
145	0	0	0	0
150	0	0	0	0

Table 4.25: *The results (p-values) of the test for equal variances using Levene's test.*

however, heterogeneous but the normality assumption is still not violated, the Welch ANOVA can be used instead [67]. A background of the one-way ANOVA is given which is then extended to the Welch ANOVA.

In order to perform the one-way ANOVA test, the profit is written as

$$x_{ij} = \bar{x}_i + \varepsilon_{ij} \tag{4.26}$$

where ε_{ij} for $i = 1, \ldots, n''$ and $j = 1, \ldots, m''$ are independent and normally distributed with mean 0 and variance σ^2. The variables x_{ij} and $\bar{x}_i$ are the same as those defined in §4.5.2. The total variability of the profits x_{ij}, also called the total sum of squares (SST), needs to be calculated to compare the differences of the means ($\bar{x}_i$) of the algorithms. The SST can then be partitioned into two components called the sum of squared residuals/errors (SSE) and the sum of squares due to regression (SSR). An additional mean is required to calculate SST and SSR, namely the sample grand mean of all the $m''n''$ profits, given by

$$\bar{x} = \sum_{i=1}^{n''} \sum_{j=1}^{m''} x_{ij}. \tag{4.27}$$

The values of SST, SSE and SSR are calculated by the formulas

$$SST = \sum_{i=1}^{n''} \sum_{j=1}^{m''} (x_{ij} - \bar{x})^2, \tag{4.28}$$

$$SSE = \sum_{i=1}^{n''} \sum_{j=1}^{m''} (\bar{x}_i - \bar{x})^2 \text{ and} \tag{4.29}$$

$$SSR = \sum_{i=1}^{n''} m''(x_{ij} - \bar{x}_i)^2, \tag{4.30}$$

$$\tag{4.31}$$

where the relationship $SST = SSE + SSR$ holds. SST has $m'' - 1$ degrees of freedom while SSE and SSR have $m'' - n''$ and $n'' - 1$ degrees of freedom, respectively. An F-test is then performed under the hypotheses

$$\begin{aligned} H_0 &: \quad \bar{x}_1 = \bar{x}_2 = \cdots = \bar{x}_{n''} \\ H_a &: \quad \text{all } \bar{x}_i\text{'s are not equal,} \end{aligned}$$

with the F-statistic given by

$$W_A = \frac{MSR}{MSE} \tag{4.32}$$

$$= \frac{SSR}{\sigma^2(n'' - 1)} \Big/ \frac{SSE}{\sigma^2(m'' - n'')}. \tag{4.33}$$

The critical values of W_A may be found in an F-table, since W_A is approximately F-distributed with $n'' - 1$ and $m'' - n''$ degrees of freedom. The null hypothesis is rejected if W_A is greater than the critical value and accepted otherwise. Alternatively, if the p-value is less than the significance level, there is sufficient evidence to conclude that the means of the profits of the algorithms for a specific test case differ significantly.

The fundamental concept of the Welch ANOVA is to use weights w_i'' for each of the n'' algorithms to reduce heterogeneity[67] . The weight is calculated by the equation

$$w_i = \frac{m''}{s_i^2} \tag{4.34}$$

where s_i^2 is the observed variance of the profits when algorithm i was executed. The adjusted grand mean is then calculated by

$$\bar{x}_W = \frac{\sum_{i=1}^{n''} w_i \bar{x}_i}{\sum_{i=1}^{n''} w_i}. \tag{4.35}$$

In order to calculate the F-statistic for the Welch ANOVA, an adjusted MSR is needed which in turn is dependent of an adjusted SSR. The adjusted SSR and MSR are calculated by

$$SSR_W = \sum_{i=1}^{n''} w_i (\bar{x}_i - \bar{x}_W)^2 \text{ and} \tag{4.36}$$

$$MSR_W = \frac{SSR_W}{n'' - 1}. \tag{4.37}$$

Finally, a second term Λ that is based on the weights is required to calculate the F-statistic, given by

$$\Lambda = \frac{3n''}{(m'' - 1)(n''^2 - 1)} \sum_{i=1}^{n''} \left(1 - \frac{w_i}{\sum_{i=1}^{n''} w_i}\right)^2. \tag{4.38}$$

Subsequently, the adjusted F-statistic is calculated by

$$W_W = \frac{MSR_W}{1 + \frac{2\Lambda(n''-2)}{3}}, \tag{4.39}$$

which is approximately F-distributed with $n'' - 1$ and $1/\Lambda$ degrees of freedom. Again, a p-value of less than the significance level indicates that the null hypothesis is rejected in favour of the alternative that the means of the profits differ significantly. In spite of the Welch test being more reliable when the variances of the populations are not approximately equal, it is less powerful than the one-way ANOVA. This is because it has less degrees of freedom as a result of $1/\Lambda \leq m'' - n''$. Furthermore, the Welch ANOVA may be considerably unstable, since w_i is highly related to m'' and s_i^2 for small sample sizes. Nevertheless, the Welch ANOVA is used instead of the one-way ANOVA in this study due to its reliability when the variances are heterogeneous.

Due to the Welch ANOVA being an omnibus test in that it only tells us whether there is a significant overall difference in the means of the algorithms, post hoc tests can be performed between the pairs of algorithms to find the pairs that caused the overall difference. Post hoc tests that may be carried out after performing the Welch ANOVA test include Tamhane's T2, Dunnett's T3 and Games-Howell [89]. The Games-Howell test is performed by computing the corrected number of degrees of freedom v for a t-test using the formula

$$v = \left\lfloor \frac{(s_1^2/m'' + s_2^2/m'')^2}{\frac{(s_1^2/m'')^2}{m''-1} + \frac{(s_2^2/m'')^2}{m''-1}} \right\rfloor \tag{4.40}$$

$$= \left\lfloor \frac{(m'' - 1)(s_1^2 + s_2^2)^2}{s_1^1 + s_2^2} \right\rfloor, \tag{4.41}$$

where s_1^2 and s_2^2 are the sample variances of the profits for the two algorithms that are being compared. The floor of a value needs to be computed for v, since the degrees of freedom needs to be an integer value. The test statistic is now calculated using

$$W_G = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2 + s_2^2}{m''}}}, \tag{4.42}$$

where $\bar{x}_1$ and $\bar{x}_2$ are the sample means of profits for the algorithms. Lastly, the critical value is calculated by $q_{\alpha,m'',v}/\sqrt{2}$ where $q_{\alpha kv}$ is the upper α point (significance level) of the studentised range distribution with parameter m'' and degree of freedom v. The null hypothesis is rejected if $W_G \geq q_{\alpha,m'',v}/\sqrt{2}$ and not rejected otherwise. That is, the means of the profits for the pair of algorithms differ significantly if the p-value is less than the significance level.

Tamhane's T2 test is based on the Student's t-distribution and is sometimes preferred over Games-Howell's test as it is less liberal. The test statistic for Tamhane's T2 test is calculated in the same way as for Games-Howell's test using equation (4.42). The critical value in this case is denoted $t_{\gamma,v}$ where $\gamma = 1 - (1 - \alpha)^{1/n''}$ (where α is the degrees of freedom) and v is calculated using equation (4.41). The value $t_{\gamma,v}$ is the two sided γ (adjusted degrees of freedom) point of the Student's t-distribution with v degrees of freedom. The null hypothesis is rejected if $W_G \geq t_{\gamma,v}$ and not rejected otherwise. Again, the p-value needs to be less than the significance level to imply a significant difference between the population means.

The last post hoc test that is considered for parametric statistics is Dunnett's T3 which is a modification of Tamhane's T2 test. It is especially powerful if the sample sizes and degrees of freedom are small. Its test statistic is calculated in the same way as for the Games-Howell and Tamhane's T2 tests using equation (4.42). The critical value for Dunnett's T3 test is given by $SMM_{\alpha,n'',v}$ where α is the significance level and v is calculated using equation (4.41). The critical value is the upper α point (significance level) of the Studentised maximum modulus distribution with parameter n'' and degree of freedom v. The null hypothesis is rejected if $W_G \geq SMM_{\alpha,n'',v}$ and not rejected otherwise. Similar to the previous post hoc tests, the p-value needs to be less than the significance level α to imply a significant difference between the means of the profits of the two algorithms being compared.

The Welch ANOVA test with the post hoc tests may be found in Tables 4.26–4.28. Each of the tables display the results for elementary, intermediate and challenging test cases, respectively. For each of the test cases, the first step is to determine whether there is a significant difference between the means of the profits of the algorithms using the Welch ANOVA test. If there is a significant difference in the means, *i.e.* if the Welch ANOVA test's p-value is less than 0.05, the algorithms with the highest and second highest mean are listed. If their means are equal (which is highly unlikely compared to when the medians are considered in §4.5.1), they are not listed. One of the requirements to perform the Welch ANOVA test is that the variance of all the considered populations should be nonzero. Accordingly, the means of the algorithms cannot be compared if the variance of at least one algorithm's profits is zero. This was the case for test cases 3, 13, 14, 15 and 37 that is indicated with a blank line in Table 4.26. A variance of zero typically occurs in elementary test cases when an algorithm obtains the optimal solution every time it is executed.

The second step is to run post hoc analyses for the test cases in which there is a significant difference in the population means, the algorithm with the highest mean profit is not tied with another algorithm and none of the algorithms has a zero variance in its profit. That is to say, post hoc analysis is performed for a test case for which a best and second best algorithm is listed in Tables 4.26–4.28. All tests are performed at a confidence interval of 95%. If all three of the post hoc tests have a p-value less than 0.05 for a specific test case, it is concluded that the best algorithm outperforms the second best algorithm for that test case. Consequently, the best algorithm outperforms all other algorithms for the test case, based on their means. An algorithm that outperforms all other algorithms for a specific test case is in bold (similar to that in Tables 4.20–4.22), evident in Tables 4.26–4.28. VNS outperformed the algorithms in 3 and 1 of the elementary and intermediate test cases, respectively, while TS outperformed the algorithms in 3, 27 and 35 of the three groups of test cases, respectively. For the intermediate test cases that VNS could solve, TS outperformed the algorithms a total of 11 times whereas VNS achieved

this only once. CS, GA-I, GA-II and PSO were unable to outperform the algorithms in any of the test cases, similar to the non-parametric tests.

4.6 Chapter summary

The method that was used to generate 150 test cases to test the metaheuristics developed in Chapter 3 against one another, was given in §4.1. The test cases were entered into AIMMS and solved using CPLEX. Only 23 of the test cases were solved optimally in less than an hour and solutions for 52 of the test cases were obtained after an hour elapsed. The remaining 75 solutions could not be solved, since AIMMS ran out of memory due to too many variables and constraints. Parameter calibration was performed in §4.2 to determine the combination of parameters for each metaheuristic that provides the best solutions. The convergence of the profit was plotted for each metaheuristic in §4.3.1 to assist in choosing the stopping criteria for the metaheuristics. It was concluded that the algorithms should be terminated after approximately 15, 25, 35, 60, 90 and 180 seconds have elapsed for the six difficulty subgroups, respectively. Convergence graphs of the profit were then plotted for the NSTB algorithm to decide on the number of subiterations that should be performed during each step and the stopping criteria that should be used. It was decided that 5 subiterations should be used for the first, third, fourth, fifth and sixth subgroups and 10 subiterations should be used for the second subgroup. The NSTB algorithm should be terminated after approximately 8, 15, 25, 45, 65 and 100 seconds for the six subgroups, respectively.

The metaheuristics were then compared to one another in §4.4 by utilising different measurements. These measurements include the profit of the schedules, the sum of the processing times of the scheduled jobs, the makespan of the schedules, the idle time of the schedules, the number of scheduled jobs, the percentage of scheduled jobs running in parallel, the available space in the schedules and the computational time it took to find the solution with the highest profit. A total of 30 test cases (10 test cases per difficulty group) were randomly chosen to be plotted (all the results may be found in Appendix E). Finally, statistical hypothesis testing was performed in §4.5 using omnibus tests to test whether there is an overall significant difference between the profits of the algorithms for each test case. Non-parametric statistical tests (for when it is assumed that the observations are not necessarily normally distributed) were executed in §4.5.1. The Friedman test was used to test the overall significance while post hoc tests included the Wilcoxon signed-rank test and the sign test to determine whether there is a significant difference between the profits of the solutions provided by the two best performing metaheuristics.

Normality and homogeneity in the observations were tested to determine whether the more powerful ANOVA test can be used to test whether there is an overall significant difference between the profits. Using every fifth test case (a total of 30 test cases), it was found that the observations are approximately normally distributed more than 80% of the time from the Kolmogorov-Smirnov and Shapiro-Wilk tests. It was also concluded that the variances of all 30 test cases are significantly different using Levene's test. The Welch ANOVA test was therefore used instead of the standard one-way ANOVA test. Post hoc tests for the Welch ANOVA tests included the Tamhane's T2, Dunnett's T3 and Games-Howell tests. A conclusion on the results is given in §5.2.

	Welch ANOVA		Algorithm		Post hoc test's p-value		
test case	F-statistic	p-value	first	second	Tamhane's T2	Dunnett's T3	Games-Howell
1	13.648	0	VNS	GA-I	0.96	0.8	0.66
2	4.583	0.017	VNS	CS	0.861	0.718	0.553
3							
4	2.195	0.129					
5	5.581	0.009	GA-I	TS	1	1	0.985
6	17.003	0	GA-I	PSO	0.994	0.967	0.857
7	21.789	0	**VNS**	GA-II	0.008	0.006	0.005
8	16.521	0	TS	GA-I	0.129	0.095	0.069
9	24.971	0	**TS**	VNS	0.031	0.021	0.016
10	3.753	0.032	TS	GA-I	1	1	0.993
11	3.543	0.041	VNS	GA-II	0.585	0.388	0.291
12	22.621	0	VNS	GA-II	0.479	0.272	0.207
13							
14							
15							
16	19.073	0	TS	PSO	0.357	0.22	0.163
17	9.799	0.001	TS	GA-I	0.089	0.065	0.048
18	15.753	0	TS	GA-I	1	1	0.999
19	2.255	0.12					
20	15.778	0	VNS	TS	1	1	1
21	6.868	0.004	TS	GA-I	0.952	0.848	0.689
22	11.522	0	VNS	GA-I	1	0.994	0.945
23	23.143	0	VNS	GA-I	0.065	0.038	0.028
24	25.155	0	VNS	GA-I	0.526	0.322	0.243
25	7.756	0.003	TS	GA-I	0.395	0.271	0.198
26	13.23	0	TS	GA-I	0.987	0.933	0.802
27	16.603	0	TS	VNS	1	1	0.994
28	83.318	0	TS	VNS	0.191	0.108	0.08
29	6.218	0.006	TS	VNS	0.999	0.988	0.914
30	47.109	0	VNS	GA-I	0.703	0.538	0.401
31	72.903	0	**VNS**	GA-I	0.002	0.001	0.001
32	34.76	0	VNS	GA-I	0.187	0.131	0.095
33	70.533	0	**VNS**	TS	0.003	0.002	0.002
34	22.255	0	**TS**	GA-I	0.04	0.027	0.02
35	15.404	0	**TS**	GA-I	0.008	0.007	0.005
36	20.204	0	TS	VNS	0.423	0.279	0.206
37							
38	14.251	0	TS	VNS	0.091	0.06	0.044
39	8.07	0.002	TS	VNS	1	1	0.982
40	9.101	0.001	TS	VNS	0.055	0.039	0.029
41	33.897	0	TS	VNS	0.733	0.486	0.375
42	33.526	0	TS	GA-I	0.813	0.614	0.473
43	10.215	0.001	TS	VNS	0.985	0.904	0.771
44	11.041	0.001	TS	VNS	0.791	0.623	0.472
45	21.624	0	TS	GA-I	0.099	0.068	0.05
46	18.838	0	TS	VNS	0.395	0.219	0.166
47	20.214	0	TS	VNS	0.439	0.288	0.213
48	4.851	0.013	VNS	TS	0.997	0.978	0.886
49	22.909	0	VNS	TS	1	0.996	0.95
50	8.842	0.002	VNS	GA-II	0.253	0.169	0.123

Table 4.26: *The results of the Welch ANOVA test for elementary test cases with post hoc analyses that include tests of Tamhane's T2, Dunnett's T3 and Games-Howell.*

	Welch ANOVA		Algorithm		Post hoc test's p-value		
test case	F-statistic	p-value	first	second	Tamhane's T2	Dunnett's T3	Games-Howell
51	10.012	0.001	VNS	TS	1	1	1
52	32.306	0	VNS	GA-I	0.078	0.058	0.042
53	15.275	0	VNS	GA-I	0.985	0.93	0.794
54	32.745	0	**VNS**	GA-I	0.037	0.028	0.021
55	23.655	0	TS	GA-I	0.885	0.724	0.566
56	17.07	0	TS	VNS	0.532	0.388	0.284
57	50.66	0	**TS**	GA-I	0.02	0.015	0.011
58	16.248	0	TS	GA-I	0.285	0.187	0.137
59	18.297	0	TS	GA-I	0.313	0.195	0.144
60	28.932	0	TS	VNS	0.176	0.125	0.091
61	49.943	0	**TS**	GA-I	0.027	0.02	0.015
62	14.499	0	**TS**	GA-I	0.021	0.016	0.012
63	51.167	0	**TS**	GA-I	0.028	0.018	0.013
64	45.31	0	**TS**	GA-I	0.007	0.005	0.004
65	86.447	0	**TS**	GA-II	0.004	0.002	0.002
66	12.946	0	TS	GA-I	0.911	0.733	0.584
67	159.151	0	**TS**	VNS	0.032	0.017	0.013
68	21.481	0	**TS**	GA-I	0.005	0.003	0.002
69	20.67	0	**TS**	PSO	0.01	0.008	0.006
70	41.58	0	TS	GA-I	0.649	0.414	0.316
71	31.474	0	TS	GA-I	0.141	0.084	0.062
72	44.589	0	TS	VNS	0.06	0.038	0.028
73	27.785	0	**TS**	GA-I	0.004	0.003	0.002
74	70.539	0	**TS**	GA-II	0.049	0.03	0.022
75	79.894	0	TS	GA-II	0.126	0.071	0.053
76	3.988	0.039	GA-II	PSO	1	1	1
77	29.231	0	**TS**	GA-I	0.003	0.003	0.002
78	92.675	0	**TS**	GA-I	0.001	0.001	0.001
79	20.517	0	TS	GA-I	0.15	0.112	0.084
80	40.922	0	**TS**	GA-I	0.003	0.003	0.002
81	87.539	0	**TS**	GA-II	0.018	0.012	0.009
82	96.696	0	**TS**	GA-I	0	0	0
83	65.738	0	**TS**	GA-I	0.002	0.002	0.001
84	16.388	0.001	TS	GA-II	1	1	0.999
85	26.807	0	**TS**	GA-I	0.001	0.001	0.001
86	22.69	0	TS	GA-II	0.147	0.113	0.085
87	34.653	0	**TS**	GA-II	0	0	0
88	29.306	0	**TS**	GA-II	0.001	0.001	0.001
89	22.053	0	**TS**	GA-I	0.001	0.001	0.001
90	49.231	0	**TS**	GA-II	0	0	0
91	1.456	0.295					
92	8.967	0.003	GA-I	GA-II	0.96	0.906	0.766
93	26.224	0	**TS**	GA-II	0.002	0.002	0.001
94	96.859	0	**TS**	GA-I	0.012	0.008	0.006
95	15.837	0	**TS**	GA-I	0.029	0.024	0.018
96	15.985	0	TS	GA-I	1	1	0.99
97	21.825	0	TS	GA-I	0.098	0.069	0.052
98	20.463	0	**TS**	GA-II	0.007	0.005	0.004
99	19.82	0	**TS**	GA-I	0.003	0.003	0.002
100	28.615	0	TS	PSO	0.104	0.08	0.06

Table 4.27: *The results of the Welch ANOVA test for intermediate test cases with post hoc analyses that include tests of Tamhane's T2, Dunnett's T3 and Games-Howell.*

	Welch ANOVA		Algorithm		Post hoc test's p-value		
test case	F-statistic	p-value	first	second	Tamhane's T2	Dunnett's T3	Games-Howell
101	64.907	0	**TS**	GA-I	0	0	0
102	106.46	0	**TS**	GA-I	0	0	0
103	13.849	0.001	GA-I	TS	1	1	0.996
104	78.508	0	**TS**	GA-I	0	0	0
105	182.068	0	**TS**	GA-I	0	0	0
106	207.089	0	**TS**	GA-II	0.017	0.011	0.009
107	117.482	0	**TS**	GA-II	0.001	0.001	0.001
108	58.841	0	TS	GA-II	0.16	0.107	0.082
109	93.159	0	TS	GA-II	0.058	0.036	0.027
110	21.171	0	**TS**	GA-II	0.003	0.002	0.002
111	2.726	0.098					
112	10.048	0.002	TS	GA-II	0.124	0.095	0.071
113	40.048	0	**TS**	GA-II	0.001	0	0
114	104.824	0	**TS**	GA-II	0	0	0
115	33.446	0	**TS**	GA-II	0.005	0.004	0.003
116	112.704	0	**TS**	GA-I	0	0	0
117	64.471	0	**TS**	GA-II	0.003	0.003	0.002
118	23.361	0	PSO	GA-I	0.999	0.993	0.94
119	6.138	0.012	TS	GA-I	1	1	1
120	84.964	0	**TS**	PSO	0.001	0.001	0
121	55.341	0	**TS**	GA-I	0.001	0.001	0.001
122	100.938	0	**TS**	GA-I	0	0	0
123	20.883	0	TS	PSO	0.157	0.121	0.091
124	74.309	0	**TS**	GA-II	0.009	0.006	0.005
125	58.289	0	**TS**	GA-I	0.001	0.001	0.001
126	361.185	0	**TS**	GA-I	0	0	0
127	24.213	0	TS	GA-I	0.136	0.104	0.078
128	132.515	0	**TS**	GA-I	0.013	0.009	0.007
129	65.79	0	**TS**	GA-I	0	0	0
130	10.223	0.002	TS	GA-I	1	1	1
131	83.753	0	**TS**	GA-I	0	0	0
132	86.348	0	**TS**	GA-I	0	0	0
133	249.881	0	**TS**	GA-I	0	0	0
134	141.828	0	**TS**	GA-II	0.007	0.005	0.004
135	38.982	0	**TS**	GA-I	0	0	0
136	56.037	0	**TS**	GA-II	0.026	0.019	0.015
137	141.831	0	**TS**	GA-I	0.001	0	0
138	5.525	0.014	TS	GA-II	0.238	0.186	0.139
139	25.683	0	TS	GA-II	0.057	0.042	0.032
140	96.16	0	**TS**	GA-II	0	0	0
141	192.777	0	**TS**	GA-II	0	0	0
142	480.556	0	**TS**	GA-II	0.011	0.007	0.005
143	220.594	0	**TS**	GA-II	0	0	0
144	310.087	0	**TS**	GA-II	0.007	0.005	0.003
145	20.461	0	TS	PSO	0.618	0.495	0.375
146	14.711	0.001	GA-I	TS	1	1	0.997
147	68.333	0	**TS**	GA-I	0	0	0
148	64.744	0	**TS**	GA-I	0	0	0
149	117.868	0	**TS**	GA-I	0	0	0
150	32.512	0	TS	PSO	1	0.999	0.977

Table 4.28: *The results of the Welch ANOVA test for challenging test cases with post hoc analyses that include tests of Tamhane's T2, Dunnett's T3 and Games-Howell.*

CHAPTER 5

Conclusion

Contents

This chapter commences by providing a summary of the work contained in this thesis. Recommendations are then provided in §5.2 as to which parameters and time limit to use for the metaheuristics, the time limit and number of subiterations that should be used for the NSTB algorithm and which metaheuristic is suggested for the considered scheduling problem. The achievement of the objectives for this thesis stated in §1.5 are reviewed in §5.3 followed by the suggestion and thorough discussion of future work in §5.4.

5.1 Thesis summary

The intention of Chapter 2 is to provide an extensive review on the five characteristics of the scheduling problem considered in this thesis. The use of genetic algorithms which is extended to the memetic algorithm is discussed for batch scheduling as well as particle swarm optimisation and tabu search. Heuristics for solving time-dependent problems when minimising the total weighted completion time, when learning effects are present and when machines run in parallel are then looked at. The tabu search method as well as a variety of evolutionary algorithms that include the genetic, self-evolution and hybrid evolutionary algorithms, are reviewed for sequence dependent setup times. Approaches for when the aforementioned characteristic is transformed into a time-dependent travelling salesman problem are also reported. The polynomial time, branch-and-bound and branch-and-cut algorithms are reviewed for when selecting jobs to schedule. Additionally, effective heuristics and metaheuristics are discussed for when selecting jobs. The final characteristic, the use of precedence constraints, is discussed. Single machine scheduling problems with time windows, four different parallel machine scheduling problems that deal with precedence constraints and the use of metaheuristics that include tabu search and a hybrid genetic algorithm, are reviewed.

Chapter 3 contains the exact formulations for the scheduling problem followed by preliminary theory for the metaheuristics that are developed. The exact formulations are divided into four different categories, namely when a job's processing time depends on its start time (and jobs are allowed or not allowed to be implemented in parallel) and when a job's processing time depends on its start time and on the preceding job (and jobs are allowed or not allowed to be implemented in parallel). Preliminary theory for the metaheuristics include the simplification of the models for the metaheuristics, the representation of a solution that is provided as output by the metaheuristic and the conditions that need to hold in a metaheuristic for a solution to be feasible.

A background and the pseudocode are presented for each of the six developed metaheuristics throughout §3.3–3.7. These metaheuristics include the variable neighbourhood search, particle swarm optimisation, cuckoo search, two types of genetic algorithms and tabu search. The definition of VNS's moves and neighbourhoods are defined first which provides groundwork for TS, since TS also uses these moves and neighbourhoods. The first approach of the genetic algorithm (GA-I) is when the chromosome of a solution consists of a string of floating point numbers. The solution's chromosome needs to be decoded into a batch sequence which needs to be made feasible to calculate the fitness value of an individual. The second approach (GA-II) defines a chromosome as a batch sequence containing all the jobs that can be scheduled. An individual's fitness value can then be calculated directly from the feasible solution (no decoding is required). Two additional metaheuristics, particle swarm optimisation and cuckoo search, that make use of decoding mechanisms were developed.

The results for 150 generated test cases (test cases had to be generated due to the unavailability of real-world data) may be found in Chapter 4. The chapter starts off by explaining the method that was used to generate test cases that are close to real-life problems as possible. The test cases were split into three difficulty groups, namely elementary, intermediate and challenging test cases. Parameter calibration was then performed for each of the metaheuristics to determine the combination of parameters that gives the best results. These sets of parameters were then used in the metaheuristics for the remainder of the results. The next step in the chapter involves the plotting of convergence graphs for all the metaheuristics to determine the time limit at which the metaheuristics are terminated. Moreover, convergence graphs are plotted for the NSTB algorithm after solutions have been obtained from the metaheuristics. The convergence graphs are used to determine the computational time at which the last iteration of the NSTB algorithm is performed as well as the number of subiterations in each of the algorithm's steps (excluding the merging of batches).

After all the parameters for the metaheuristics and the NSTB algorithm are concluded, the algorithms can be compared to one another for different difficulty groups. This takes place in §4.4 by utilising different measurements. Amongst these measurements are the profit (the objective to be maximised) of the schedules and the sum of the durations of the scheduled jobs. In terms of time, the makespan of the schedules as well as the number of time periods in which the machine is idle are considered. The number of scheduled jobs, the percentage of scheduled jobs running in parallel and the available space in a schedule are compared across the algorithms. The last measurement is the time at which the solution with the highest profit throughout the search was found. This gives valuable information regarding the general convergence of a metaheuristic for a difficulty group compared to other metaheuristics.

The final stage in the results is to compare the algorithms against one another on a test case level. Omnibus tests are used to test whether there is an overall significant difference between the profits of the algorithms for each of the generated test cases. The first test that is used is the Friedman test. This test is useful when the observations are not necessarily normally distributed. The ANOVA test may, however, be more powerful when the observations are

normally distributed. It is shown in §4.5.2 that the observations are in fact believed to be normally distributed using the Kolmogorov-Smirnov and Shapiro-Wilk tests. It was found that the observations have unequal variances (across the metaheuristics) for each of the tested test cases using Levene's test. Since this violates one of the assumptions that need to hold to use the ANOVA test, the Welch ANOVA test that was specifically designed for heterogeneity in variances is used instead. Post hoc analyses are performed for the Friedman and Welch ANOVA test to determine whether there is a significant difference between the profits of the best and second best algorithms if there is an overall significant difference between the profits of all the algorithms. The Wilcoxon signed-rank test and sign test are utilised for the Friedman test while the Welch ANOVA test uses the Tamhane's T2, Dunnett's T3 and Games-Howell tests. The post hoc tests conclude whether a metaheuristic outperforms all other metaheuristics for a given test case.

5.2 Recommendations

Parameter calibration was performed in §4.2 to determine the combination of parameters to use for each of the metaheuristics. The values of the parameters were chosen for an intermediate test case which is deemed an average instance. These parameters were therefore used for all test cases in all difficulty groups. It is recommended to use the neighbourhoods for VNS and TS that swap two batches in the schedule, remove a batch from the schedule and insert an unscheduled batch, move a batch from one position to another in the schedule, insert a batch into the schedule and lastly, remove a job from the schedule and insert an unscheduled job. A population size of 75, inertia factor of 2/3, acceleration coefficient of 1 in the direction of the best known position of a particle and acceleration coefficient of 0.75 in the direction of the best known position of all the particles should be used for PSO.

It is suggested to set the number of nests to 75, number of eggs in each nest to 5, the probability that a cuckoo will start its Lévy flights from the best egg obtained so far to 0.05, the fraction of the nests that will be abandoned or lose its worse egg to 0.1, the probability that a nest will be abandoned to 0.3 and the number of flights that a cuckoo performs before laying an egg to 10 for CS. Both GA-I and GA-II are recommended to have the population size set to 25 while using FPS as the selection technique and PUX/PUOX as the crossover operator. The mutation probability should be 0.2 and 0.1, the immigration percentage 0.1 and 0.2, and the number of genes to change/moves to make during mutation 6 and 9 for GA-I and GA-II, respectively. It is advised for the final metaheuristic, TS, to use 6 as the population size, 75 as the maximum size of the subset of neighbours, 20% of the total number of jobs (excluding the derived jobs) as the tabu list size and 40% of the total number of jobs (excluding the derived jobs) greater than the tabu list size as the maximum number of iterations that a move stays in the tabu list.

Convergence graphs for the metaheuristics and the NSTB algorithm may be found in §4.3. Each of the three difficulty groups were split into two subgroups containing its first and last 25 test cases. It is recommended to use 15, 25, 35, 60, 90 and 180 seconds as the time limit for the metaheuristics for the six subgroups, respectively. These are the approximate times that all of the convergence graphs for the metaheuristics start approaching near flatness. Furthermore, it is suggested to set the number of subiterations for the NSTB algorithm to 5 for the first, third, fourth, fifth and sixth subgroups and to 10 for the second subgroup. It is also recommended for the NSTB algorithm's time limit to be a maximum of 8, 15, 25, 45, 65 and 100 seconds for the six subgroups, respectively. These are the times at which the convergence graphs for all the metaheuristics are approximately flat. It was found that reducing the time limit for the NSTB algorithm by a few seconds will not affect the results considerably. This is due to jobs not having

much freedom to move around within their batches and the fact that batches cannot be moved around as this will influence the processing times of the jobs in the schedule. A smaller time limit can therefore be used for the NSTB algorithm in some instances.

It is recommended to use either VNS or TS for test cases in the elementary difficulty group and TS for test cases of an intermediate or challenging nature. There is a mere 0.301% increase in the average profit when VNS is used compared to TS for elementary test cases. TS outperforms all other metaheuristics for intermediate and challenging test cases. For intermediate test cases, TS most notably performs (in terms of average profit) 8.658% better than VNS for the 25 test cases that VNS could solve as well as 9.843% and 12.369% better than GA-I and GA-II, respectively. For challenging test cases, TS most notably performs 16.562% and 16.245% better than GA-I and GA-II, respectively, and 18.967% better than PSO. CS is the worst performing metaheuristic in all difficulty groups. In the non-parametric statistics (the Friedman test with the Wolcoxon signed-rank and sign post hoc tests), TS outperformed the metaheuristics in 11, 38 and 41 of the three groups of test cases, respectively, while VNS outperformed the metaheuristics in only 6 and 1 of the elementary and intermediate test cases, respectively. In the parametric statistics (Welch ANOVA test with Tamhane's T2, Dunnett's T3 and Games-Howell post hoc tests), TS outperformed the metaheuristics in 3, 27 and 35 of the three groups of test cases, respectively, while VNS outperformed the metaheuristics in 3 and 1 of the elementary and intermediate test cases, respectively.

5.3 Objectives achieved

Nine objectives were set in §1.5 to develop and compare metaheuristics for this thesis. Objective I was achieved in Chapter 1 where an overview of scheduling in manufacturing and service settings was provided as well as a problem description, applications and the characteristics for the considered scheduling problem. Objective II was addressed in Chapter 2 where an in depth review on the literature of each characteristic for the scheduling problem was given. More attention was put on metaheuristics than on theory and heuristics due to the focus of this thesis being on the development of metaheuristics. Objectives III and IV were fulfilled in §3.1 and §3.2, respectively, where exact formulations were given in the form of integer programming models and preliminary theory necessary for the metaheuristics were discussed.

Objective V was reached in §3.3–3.7 in which a background, the pseudocode and description of each metaheuristic were given. The accomplishment of Objective VI took place in §3.8 where the NSTB algorithm was developed. A population of solutions is obtained from a metaheuristic and used to move single and multiple jobs within a batch, merge batches and insert jobs into batches so that a job has the ability to start in a period other than the first period of the batch it is in. Test instances were generated, the parameters of each metaheuristic were calibrated to provide the best possible solutions and convergence graphs were plotted and analysed to choose stopping criteria for the metaheuristics in §4.1–4.4, respectively, subsequently attaining Objective VII. The metaheuristics were compared to one another using different measurements in §4.4 while statistical hypothesis testing was performed for each test case to determine the best performing metaheuristic in §4.5. Objective VIII was therefore achieved. The final objective, Objective IX, is fulfilled in this chapter in which a conclusion and possible future work are provided.

5.4 Further research

Three possibilities for future work are suggested and discussed in detail. These proposals include a bi-objective scheduling problem which can be extended to a problem containing three or more objectives, multiple identical machine scheduling and several metaheuristics and hybrid metaheuristics.

5.4.1 Bi-objective scheduling problem

In many industries, optimising a single objective is not sufficient to obtain the best possible schedule that satisfies all of a company's constraints [47]. When optimising a single objective, a company will typically realise problems arising as a result of neglecting certain criteria. To overcome this problem, more objectives are added to the problem. Multi-objective machine scheduling algorithms are therefore used when dealing with multiple performance criteria to determine the best possible solution. There is, however, not a single optimal solution but several optimal solutions that lie on the so-called Pareto frontier. When moving from one solution to another in the Pareto set, no objective can be improved without negatively affecting another objective. As a result, trade-offs are present in the solution which only the DM can decide on to find the most-preferred solution.

The scheduling problem considered in this thesis may easily be extended to a multi-objective problem due to two reasons. A fitness function is used in each of the metaheuristics. Additional terms can be included in this function for each of the objectives to be optimised. Secondly, a local/external archive is used in the metaheuristics. Solutions in this archive that are dominated by at least one other solution in the archive, are removed at the end of each iteration. In this way, a best solution is never lost from the archive until a better solution that dominates it is found throughout the search. A literature review of multi-objective machine scheduling is given in §5.4.1.1. Jobs with due dates and due windows are considered in this section. §5.4.1.2 contains the changes that need to be made in the exact formulations, the metaheuristics in this thesis and the NSTB algorithm if a second objective function were to be incorporated in the scheduling problem.

5.4.1.1 Literature review

Eren [31] performed a study on the scheduling of jobs on identical parallel machines with a learning effect in the setup and removal times of jobs. The objectives were to minimise the weighted sum of the total completion time and the total tardiness so that the problem is bi-objective similar to that of Eren & Güner [32]. To deal with the multiple objectives, a weight of α is given to the total completion time and a weight of β is associated with the total tardiness such that $\alpha + \beta = 1$. A mathematical programming model was developed that solves problems of up to only 15 jobs and 5 machines optimally for $\alpha = \beta = 0.5$. As a result, three variations of a heuristic method were developed to solve problems of a larger size. The difference of the three heuristics is the way in which the initial sequence is obtained. These methods are called the Shortest sum of Setup, Processing and Removal Times (SSPRT), Shortest sum of Setup and Removal Times (SSRT) and EDD.

The proposed heuristic starts by applying one of the methods to obtain an initial sequence. The first two jobs in the sequence are then removed and scheduled in a way that minimises the two objectives. The first job from the remaining jobs is then removed from the sequence and inserted into the slot that partially minimises the two objectives. The jobs are inserted in this manner

until there are no jobs remaining in the sequence and all jobs have been scheduled. SSPRT gives the best results for small sized problems with an average error[1] of less than 1%. The average error for SSRT and EDD is 1% and 2%, respectively. For large sized problems, SSPRT again performs the best with an average error of less than 2.5%. An average error of 2.5% and 3.5% is obtained by SSRT and EDD, respectively. Thus, it is recommended to use the heuristic that utilises the SSPRT method for both small and large sized problems.

Arroyo *et al.* [3] used the concepts of the VNS heuristic to schedule jobs with sequence dependent setup times and due windows in a single machine environment. The total weighted earliness or tardiness as well as the total flow time are the objectives to be minimised. A job's due window may be defined as the time interval in which the job must complete its processing. If the job is completed before the interval, an earliness penalty is incurred and for completion after the interval, a tardiness penalty is added to the objective function value. Three multi-objective variable neighbourhood search (MOVNS) algorithms were developed to solve the problem. In the first algorithm (MOVNS1), three initial solutions are generated using greedy heuristics. These solutions are then used to initialise a set of nondominated solutions that is updated at the end of each iteration. At the beginning of each iteration, a random solution is chosen from the set of nondominated solutions and attempted to be improved using one of the two neighbourhood structures, the insertion neighbourhood and exchange neighbourhood. Nondominated solutions may not be selected more than once to ensure diversity and avoid being trapped in a local optimum.

The following algorithms MOVNS2 and MOVNS3 use intensification procedures that are based on scalarising functions and Pareto dominance, respectively. An additional input parameter that is used for the specification of the number of jobs to be removed from the sequence during the intensification procedure, is required to execute MOVNS2 and MOVNS3. The procedure consists of the destruction stage in which the specified number of jobs are removed and the construction stage where the removed jobs are added one for one to obtain a new solution. The authors showed that it is best to remove 6 jobs during the intensification procedure. The three algorithms were tested 5 times each for 72 different test cases consisting of 20, 30, 40, 50, 75 and 100 jobs. It was concluded that MOVNS3 performs the best in all instances except for when there are 20 jobs in which case MOVNS2 presents the best results. MOVNS2 shows better results than MOVNS1 in all test cases.

5.4.1.2 Changes in exact formulations and algorithms

A second objective function that can be included in the exact formulation is the minimisation of the sum of the processing times of the jobs that are scheduled. When a job's processing time depends only on its start time, the objective is to

$$\text{minimise} \quad \sum_{i=1}^{n} \sum_{k=1}^{w} t_{ik} x_{ik}. \tag{5.1}$$

Similarly, when a job's processing time depends on its start time and the preceding job, the objective is to

$$\text{minimise} \quad \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{k=1}^{w} t_{ijk} x_{ijk}. \tag{5.2}$$

When defining a solution Φ_i in §3.2, objective functions (5.1) and (5.2) need to be adjusted for use in the metaheuristics and NSTB algorithm. If each element φ_{ijk} in the solution Φ_i is set to

[1] The error is calculated by the formula (Error) $= \frac{\text{(Heuristic solution)} - \text{(Optimal solution)}}{\text{(Optimal solution)}}$

1 if job j starts implementation during time period k and 0 if job j does not start processing in time period k, the objective functions are given by

$$f_1(\Phi_i) = \sum_{j=1}^{n}\sum_{k=1}^{w} r_j p_{jk} \varphi_{ijk}, \text{ and} \tag{5.3}$$

$$f_2(\Phi_i) = \sum_{j=1}^{n}\sum_{k=1}^{w} t_{jk} \varphi_{ijk}. \tag{5.4}$$

Now that a second objective function has been introduced, the metaheuristics' fitness functions need to be changed. A weight is given to each objective function, depending on the DM's preferences. The more the weight is put on the second objective, the greater the chances will be that the schedules returned by the metaheuristics will have a smaller makespan and/or less jobs running in parallel. Additionally, a target value is given to each objective function. The target value for objective function (5.3) is determined by the DM that depends on the profit that wants to be achieved. On the contrary, the target value for objective function (5.4) can be calculated using the following argument. The maximum value that $f_2(\Phi_i)$ can be for any given scheduling problem is $b_{max}p_{max}$. If $f_2(\Phi_i)$ is being minimised, then $b_{max}p_{max} - f_2(\Phi_i)$ is essentially being maximised. The target value for the maximisation of $b_{max}p_{max} - f_2(\Phi_i)$ is therefore given by $b_{max}p_{max} - p_{min}$, since the smallest value that $f_2(\Phi_i)$ can be is p_{min} if all the scheduled jobs occupy the entire facility and the duration of the schedule is the minimum number of periods that it is allowed to be. The fitness value for particle X_i in PSO is then determined by the fitness function

$$z(\Phi_i) = \frac{w_1 f_1(\Phi_i)}{t_1} + \frac{w_2(b_{max}p_{max} - f_2(\Phi_i))}{b_{max}p_{max} - p_{min}}, \tag{5.5}$$

where

w_j	is the weight assigned to objective function j,
$f_j(\Phi_i)$	is the value of objective function j,
t_1	is the target value of objective function 1 determined by the DM and
b_{max}	is the maximum number of jobs allowed in a batch.

As a result, PSO now additionally requires the minimum number of periods p_{min} that the schedule is allowed to be as well as b_{max}, w_1, w_2 and t_1 as input parameters for the metaheuristic to be executed.

Similar to that of PSO, the fitness value for the egg X_{ij} in CS is determined by the fitness function

$$z(\Phi_{ij}) = \frac{w_1 f_1(\Phi_{ij})}{t_1} - \frac{w_2(b_{max}p_{max} - f_2(\Phi_{ij}))}{b_{max}p_{max} - p_{min}}, \tag{5.6}$$

where $f_k(\Phi_{ij})$ is the value of objective function k for solution X_{ij}. CS now additionally requires b_{max}, w_1, w_2 and t_1. GA-I and GA-II require the same additional parameters as PSO and TS requires the same additional parameters as CS when two objective functions are optimised. GA-I, GA-II and TS all have the same fitness function (5.5).

The NSTB algorithm requires the weights w_1' and w_2' for the profit and duration of a job, respectively, when determining its fitness value for FPS. The average duration of each job j in $\mathcal{T}$ (the set of jobs partaking in PTS when selecting a job to attempt to insert into a batch) if it starts in any of the periods in a batch, is calculated by

$$\bar{t}_j = \frac{\sum_{k=t'_{min}}^{t'_{max}} t_k^{bj}}{t'_{max} - t'_{min} + 1}. \tag{5.7}$$

where

$$t'_{min} = \min_j\{s_b^j\} \text{ and} \tag{5.8}$$

$$t'_{max} = \min\{t'_{min} + b_b - 1, w\}. \tag{5.9}$$

This is similar to equation (3.68) when calculating the average profit of each job j in $\mathcal{T}$. The fitness value of each of the jobs in $\mathcal{T}$ is now calculated by

$$z_j = \frac{w'_1\bar{p}_j}{\max_j\{\bar{p}_j\}} + \frac{w'_2(\max_j\{\bar{t}_j\} - \bar{t}_j)}{\max_j\{\bar{t}_j\} - \min_j\{\bar{t}_j\} + \theta} \tag{5.10}$$

The explanation of equation (5.10) is similar to equation (5.5). If the average profit of a job is maximised, its target value is set to the maximum of the jobs' average profit. If the average duration $\bar{t}_j$ of a job is minimised, it is equivalent to saying that $\max_j\{\bar{t}_j\} - \bar{t}_j$ is maximised. As a result, the target value of maximising $\max_j\{\bar{t}_j\} - \bar{t}_j$ is set to $\max_j\{\bar{t}_j\} - \min_j\{\bar{t}_j\}$ as this is the greatest value that $\max_j\{\bar{t}_j\} - \bar{t}_j$ can be. The variable θ is a very small number to avoid division by zero in the rare event that $\max_j\{\bar{t}_j\} = \min_j\{\bar{t}_j\}$. This will cause the second term in equation (5.10) to be zero so that only the jobs' profits will be considered.

It should be noted that the duration of the schedule (when introducing the additional objective function) is ignored in the calculation of the fitness value z_i (of each solution i in equation (3.66) after moving a single job within a batch) in §3.8. This is due to the duration of the schedule not changing when moving a job in a batch that is not the final batch in the schedule. Secondly, the duration of the schedule does not change considerably when moving a job that's in the final batch. Lastly, it is possible that the job that finishes last in the schedule will be removed when making the solution feasible in line 53 in Algorithm 9, making the usage of the duration of the schedule in this calculation meaningless.

The final change that needs to be made in the NSTB algorithm when introducing the additional objective, is the fitness function of the jobs when moving multiple jobs within a batch in §3.8.3. The fitness value z_j of each job j in the set of jobs $\mathcal{D}$ that can be started in a given period, is calculated by

$$z_j = \frac{w'_1 p_j}{p'_{max}} + \frac{w'_2(t_{max} - t'_j)}{t_{max} - t_{min} + \theta} \tag{5.11}$$

where

p_j	is the profit of job j if it starts in a given period,
p'_{max}	is the maximum profit of the jobs in $\mathcal{D}$,
t'_j	is the duration of job j if it starts in a given period,
t_{max}	is the maximum duration of the jobs in $\mathcal{D}$,
t_{min}	is the minimum duration of the jobs in $\mathcal{D}$ and
θ	is a very small number to avoid division by zero.

Again, the explanation of equation (5.11) is similar to equation (5.10). If the profit of a job is maximised, its target value is set to the maximum of the jobs' profit. If the duration $\bar{t}_j$ of a job is minimised, it is equivalent to saying that $t_{max} - t'_j$ is maximised. As a result, the target value of maximising $t_{max} - t'_j$ is set to $t_{max} - t_{min}$ as this is the greatest value that $t_{max} - t'_j$ can be. The variable θ is used to avoid division by zero in the rare event that $t_{min} = t_{max}$. This occurs when all of the jobs have the same duration. The second term will be ignored so that only the profits of the jobs are considered.

5.4.2 Multiple identical machine scheduling

The scheduling problem considered in this thesis is concerned with the scheduling of jobs on a single machine. The problem can, however, be generalised to more than one machine by adding

a dimension to the processing time, profit and the two output variables of each job. This extra dimension represents the machine that a job is scheduled on. The additional objective mentioned in §5.4.1 is also included in the models in this section. The parameters that are used as input for the model when the processing times of the jobs depend on their start time only are defined by letting

- t_{ikl} be the processing time of job i when its implementation starts during period k on machine l,
- p_{ikl} be the profit of job i when its implementation starts during period k on machine l,
- h_i be the fraction of the machine that job i requires for the full period of implementation, and
- r_i be the relative weight representing the relative importance of job i.

The sets of variables whose values need to be determined by the model as output are denoted by

$$x_{ikl} = \begin{cases} 1, & \text{if the implementation of job } i \text{ starts during period } k \text{ on machine } l \\ 0, & \text{otherwise} \end{cases} \tag{5.12}$$

and

$$b_{ikl} = \begin{cases} 1, & \text{if job } i \text{ is implemented during period } k \text{ on machine } l \\ 0, & \text{otherwise.} \end{cases} \tag{5.13}$$

The assumptions mentioned in §3.1.1 also hold for this model. The objectives are to

$$\text{maximise} \quad \sum_{i=1}^{n}\sum_{k=1}^{w}\sum_{l=1}^{m} r_i p_{ikl} x_{ikl}, \text{ and} \tag{5.14}$$

$$\text{minimise} \quad \sum_{i=1}^{n}\sum_{k=1}^{w}\sum_{l=1}^{m} t_{ikl} x_{ikl} \tag{5.15}$$

subject to

$$\sum_{k=1}^{w}\sum_{l=1}^{m} x_{ikl} \leq 1 \qquad i = 1, \ldots, n, \tag{5.16}$$

$$\sum_{i=1}^{n} x_{i1l} \geq 1 \qquad l = 1, \ldots, m, \tag{5.17}$$

$$\sum_{i=1}^{n} h_i b_{ikl} \leq 1 \qquad k = 1, \ldots, w,\ l = 1, \ldots, m, \tag{5.18}$$

$$\sum_{q=k}^{k+t_{ik}-1} b_{iql} \geq t_{ikl} x_{ikl} \qquad i = 1, \ldots, n,\ k = 1, \ldots, w,\ l = 1, \ldots, m, \tag{5.19}$$

$$(1-w) y_{ikl} + x_{ikl} \leq 0 \qquad i = 1, \ldots, n,\ k = 1, \ldots, w,\ l = 1, \ldots, m, \tag{5.20}$$

$$(w-1) y_{ikl} + t_{ikl} \leq 2w - k \qquad i = 1, \ldots, n,\ k = 1, \ldots, w,\ l = 1, \ldots, m, \tag{5.21}$$

$$x_{skl} = 0 \qquad \forall\ (s, k, l) \in \mathcal{S}_k \tag{5.22}$$

$$\sum_{v \in \mathcal{H}_i}\sum_{k=1}^{w}\sum_{l=1}^{m} x_{vkl} \leq 1 \qquad \forall\ \mathcal{H}_i, \tag{5.23}$$

$$b_{ikl}, x_{ikl}, y_{ikl} \in \{0, 1\} \qquad i = 1, \ldots, n,\ k = 1, \ldots, w,\ l = 1, \ldots, m, \text{ and} \tag{5.24}$$

$$t_{ikl} \in \mathbb{N}^+ \qquad i = 1, \ldots, n,\ k = 1, \ldots, w,\ l = 1, \ldots, m. \tag{5.25}$$

The objectives (5.14) and (5.15) as well as the constraint and variable sets (5.16)–(5.25) are similar to those found in §3.1.1 and §5.4.1.2.

In the event that the processing times of the jobs depend on their start time and the preceding job, the parameters required as input for the modelling of categories 3 and 4 are given below. Let

t_{ijkl} be the processing time of job i when it follows on job j and its implementation starts during period k on machine l,
p_{ijkl} be the profit of job i when it follows on job j and its implementation starts during period k on machine l,
h_{il} be the fraction of the machine that job i requires for the full period of implementation on machine l, and
r_i be the relative weight representing the relative importance of job i.

The sets of variables that are presented as output are given by

$$x_{ijkl} = \begin{cases} 1, & \text{if the running of job } i \text{ follows on job } j \text{ and starts during period } k \\ & \text{on machine } l \\ 0, & \text{otherwise} \end{cases} \tag{5.26}$$

and

$$b_{ikl} = \begin{cases} 1, & \text{if job } i \text{ is implemented during period } k \text{ on machine } l \\ 0, & \text{otherwise.} \end{cases} \tag{5.27}$$

Let q'_{ijkl} be the dummy variable that is set to 1 if the running of job i does not follow on job j and start during period k on machine l or set to 0 if job i indeed follows on job j and starts during period k on machine l. In other words, $q'_{ijkl} = 0$ if $x_{ijkl} = 1$ and $q'_{ijkl} = 1$ if $x_{ijkl} = 0$. Furthermore, suppose job 0, a dummy job with a profit of 0, a duration of 1 as well as no setup time and setup cost, starts during period 0. Then

1. $x_{0,0,0} = 1$, $q'_{0,0,0} = 0$ and $b_{0,0,0} = 1$ for $l = 1, \ldots, m$,
2. $x_{0,0,k} = 0$, $q'_{0,0,k} = 1$ and $b_{0,0,k} = 0$ for $k = 1, \ldots, w$, $l = 1, \ldots, m$,
3. $x_{0,j,k} = 0$, $q'_{0,j,k} = 1$ and $b_{0,j,k} = 0$ for $j = 1, \ldots, n, k = 0, \ldots, w$, $l = 1, \ldots, m$,
4. $x_{i,0,0} = 0$, $q'_{i,0,0} = 1$ and $b_{i,0,0} = 0$ for $i = 1, \ldots, n$, $l = 1, \ldots, m$,
5. $x_{i,j,0} = 0$, $q'_{i,j,0} = 1$ and $b_{i,j,0} = 0$ for $i = 1, \ldots, n, j = 1, \ldots, n$, $l = 1, \ldots, m$,
6. $x_{i,i,k} = 0$, $q'_{i,i,k} = 1$ and $b_{i,i,k} = 0$ for $i = 1, \ldots, n, k = 0, \ldots, w$, $l = 1, \ldots, m$, and
7. $x_{i,j,1} = 0$, $q'_{i,j,1} = 1$ and $b_{i,j,1} = 0$ for $i = 0, \ldots, n, j = 1, \ldots, n$, $l = 1, \ldots, m$.

Explanations for the assignments mentioned above may be found in §3.1.2.

Again, all the assumptions mentioned in §3.1.1 in addition to those mentioned in §3.1.2 need to hold for this model. The objectives are to

$$\text{maximise} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{k=1}^{w} \sum_{l=1}^{m} r_i p_{ijkl} x_{ijkl}, \text{ and} \tag{5.28}$$

$$\text{minimise} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{k=1}^{w} \sum_{l=1}^{m} t_{ijkl} x_{ijkl} \tag{5.29}$$

subject to

$$\sum_{\substack{j=1\\j\neq i}}^{n}\sum_{k=1}^{w}\sum_{l=1}^{m} x_{ijkl} \leq 1 \qquad i = 1,\ldots,n, \tag{5.30}$$

$$\sum_{i=1}^{n}\sum_{\substack{j=1\\j\neq i}}^{n} x_{ij1l} \geq 1 \qquad l = 1,\ldots,m, \tag{5.31}$$

$$\sum_{i=1}^{n} h_{il} b_{ikl} \leq 1 \qquad k = 1,\ldots,w,\ l = 1,\ldots,m, \tag{5.32}$$

$$x_{ijkl} + q'_{ijkl} = 1 \qquad \begin{array}{l} i = 1,\ldots,n,\ j = 1,\ldots,n,\ k = 1,\ldots,w,\\ l = 1,\ldots,m, \end{array} \tag{5.33}$$

$$\sum_{q=k}^{k+t_{ijkl}-1} b_{ijql} + q'_{ijkl} t_{ijkl} \geq t_{ijkl} \qquad \begin{array}{l} i = 1,\ldots,n,\ j = 1,\ldots,n,\ k = 1,\ldots,w,\\ l = 1,\ldots,m, \end{array} \tag{5.34}$$

$$(1-w) y_{ijkl} + x_{ijkl} \leq 0 \qquad \begin{array}{l} i = 1,\ldots,n,\ j = 1,\ldots,n,\ k = 1,\ldots,w,\\ l = 1,\ldots,m, \end{array} \tag{5.35}$$

$$(w-1) y_{ijkl} + t_{ijkl} \leq 2w - k \qquad \begin{array}{l} i = 1,\ldots,n,\ j = 1,\ldots,n,\ k = 1,\ldots,w,\\ l = 1,\ldots,m, \end{array} \tag{5.36}$$

$$\sum_{\substack{i=1\\i\neq j}}^{n} x_{ijkl} \leq n q''_{jkl} \qquad j = 1,\ldots,n,\ k = 2,\ldots,w,\ l = 1,\ldots,m, \tag{5.37}$$

$$\sum_{\substack{i=1\\i\neq j}}^{n}\sum_{a=1}^{k-1} x_{jial} \geq q''_{jkl} \qquad j = 1,\ldots,n,\ k = 2,\ldots,w,\ l = 1,\ldots,m, \tag{5.38}$$

$$x_{ijkl} \leq n w q'''_{ja'_{ijkl}} \qquad \begin{array}{l} i = 1,\ldots,n,\ j = 1,\ldots,n,\ k = 1,\ldots,w,\\ l = 1,\ldots,m,\ v_{ijk} = 0, \end{array} \tag{5.39}$$

$$\sum_{\substack{i=1\\i\neq j}}^{n}\sum_{a=k+1}^{w} x_{ijal} \leq nw(1 - q'''_{jkl}) \qquad j = 1,\ldots,n,\ k = 1,\ldots,w-1,\ l = 1,\ldots,m, \tag{5.40}$$

$$\sum_{i=1}^{n}\sum_{\substack{i=1\\i\neq j}}^{n}\sum_{k=1}^{w}(1 - v_{ijk}) x_{ijk} = 0 \tag{5.41}$$

$$x_{sjkl} = 0 \qquad \forall\ (s,j,k,l) \in \mathcal{S}_k \tag{5.42}$$

$$\sum_{v\in\mathcal{H}_i}\sum_{j=1}^{n}\sum_{k=1}^{w}\sum_{l=1}^{m} x_{vjkl} \leq 1 \qquad \forall\ \mathcal{H}_i, \tag{5.43}$$

$$b_{ijkl}, x_{ijkl}, q'_{ijkl} \in \{0,1\} \qquad \begin{array}{l} i = 0,\ldots,n,\ j = 0,\ldots,n,\ k = 0,\ldots,w,\\ l = 1,\ldots,m, \end{array} \tag{5.44}$$

$$y_{ijkl} \in \{0,1\} \qquad \begin{array}{l} i = 1,\ldots,n,\ j = 1,\ldots,n,\ k = 1,\ldots,w,\\ l = 1,\ldots,m, \end{array} \tag{5.45}$$

$$q''_{jkl} \in \{0,1\} \qquad j = 1,\ldots,n,\ k = 2,\ldots,w,\ l = 1,\ldots,m, \tag{5.46}$$

$$q'''_{jkl} \in \{0,1\} \qquad j = 1,\ldots,n,\ k = 1,\ldots,w,\ l = 1,\ldots,m,\ \text{and} \tag{5.47}$$

$$t_{ijkl} \in \mathbb{N}^{+} \qquad \begin{array}{l} i = 1,\ldots,n,\ j = 1,\ldots,n,\ k = 1,\ldots,w,\\ l = 1,\ldots,m. \end{array} \tag{5.48}$$

The objectives (5.28) and (5.29) as well as the constraint and variable sets (5.30)–(5.48) are similar to those found in §3.1.2 and §5.4.1.2.

5.4.3 Metaheuristics and hybrid metaheuristics

Additional metaheuristics in the literature may be applied to the scheduling problem. Their results can then be compared to that of the metaheuristics considered in this thesis. Two well-known metaheuristics are recommended for future work, namely simulated annealing and ant colony systems. A background and a review of the literature for simulated annealing may be found in §5.4.3.1. Simulated annealing is mostly applied to scheduling problems with sequence dependent setup times. A review is therefore provided for scheduling problems of this nature. An introduction and past research of ant colony systems (including systems for multi-objective problems) are given in §5.4.3.2. Ant colony systems are discussed for scheduling problems when batch scheduling and precedence constraints are involved. Finally, §5.4.3.3 contains different hybrid metaheuristics that may be implemented.

5.4.3.1 Simulated annealing

Simulated annealing is a trajectory metaheuristic for finding good solutions to large combinatorial optimisation problems [30]. The development of the algorithm was inspired by the metallurgic process of heating, called annealing. Annealing is the procedure whereby a metal is heated which softens the internal structure of the metal. This allows the shape and size of the metal to be easily changed. As the metal cools down, after all alterations have been made, the new physical appearance of the metal becomes fixed. Simulated annealing uses these concepts by defining a variable, initially high, for the temperature to simulate the heating and cooling operation. As more iterations are executed, the temperature cools down. A high temperature implies that the probability of selecting a worse solution is high while the lowest virtual temperature restricts the algorithm to choosing a solution only if it is an improvement to the current one. The initial high temperature allows the algorithm to remove itself from any possible local optima. The cooling process allows the algorithm to focus on promising regions of the search space as the probability of accepting worse solutions decreases.

Simulated annealing (SA) was applied by Zarandi *et al.* [110] to solve a two-machine robotic cell scheduling problem with sequence dependent setup times with the presence of different loading/unloading times for each part. Robotic cells consist of an input device, several machines in series, an output device and several robots that process the parts between the machines. The objective is to determine the sequence of robot moves and the order of the parts to minimise the total cycle time or equivalently, to maximise the throughput. The authors used the Gilmore and Gomory algorithm [39] to determine a lower bound for the problem with a complexity of $\mathcal{O}(n^2)$. This lower bound is used to verify the performance of the SA algorithm.

Parts requiring similar operations are grouped into the same family that are then manufactured in specialised cells using cyclic production. Production is separated into several short cycles to fulfil the demand. The minimal part set (MPS), consisting of a new set of parts, is then repeatedly manufactured to satisfy the total demand. The SA algorithm uses the MPS to represent a solution as a sequence of MPS elements. A swapping scheme is implemented to provide a new solution by swapping two random elements. Two types of cooling schemes are used, that is when the temperature is reduced with increments of one and when the temperature is reduced by 0.5%. The dominance condition that was used to evaluate a solution in each iteration showed an improvement in the running time and overall quality of the algorithm. A

total of up to 200 parts was solved to optimality or near-optimality in less than 10 seconds using the SA algorithm.

Behnamian *et al.* [6] proposed a more complex algorithm comprising of three components to form a hybrid metaheuristic. These components include the use of ant colony optimisation (ACO) to generate the initial population, SA to evaluate each solution and a variable neighbourhood search (VNS) that attempts to improve the population in each iteration. Identical machines that operate in parallel with independent jobs that are available at the beginning of the schedule are considered. The results obtained after executing the algorithm indicate a statistically significant difference between the hybrid metaheuristic, SA-VNS hybrid, ACO-VNS hybrid and VNS with the hybrid metaheuristic performing the best in all test instances with the number of jobs ranging from 6 to 100 jobs.

The hybrid metaheuristic begins by iteratively generating an initial solution according to the concept of ACO. This is performed by using the max-min ant system (MMAS) [93] to choose an unscheduled job probabilistically for a specific position in the sequence until there are no unscheduled jobs left. The trail intensities are then updated after which moves are performed amongst the ants. In each iteration a random solution is selected from the neighbourhood and accepted according to the probability defined in the SA technique. This probability depends on the temperature at the specific iteration which decreases after each iteration according to a logarithmic cooling scheme. However, If the solution is rejected, the neighbourhood structure is changed according to the manner of the VNS method. The VNS used by the authors is based on the changing of three different neighbourhoods[2], namely the swapping of jobs on one machine, swapping of jobs between two machines and transferring of a job from one machine to another.

5.4.3.2 Ant colony system

The ant colony system is a probabilistic metaheuristic that is based on the pheromone trails formed by ants between their colony and food sources [28]. Ants move in a system by means of pheromone trails that are straight lines containing pheromone deposited by the ants. Ants prefer to follow a path that is comprised of more pheromone. When ants are met by an obstacle, a decision needs to be made as to which path to follow around the obstacle to continue to and from their food sources. Ants are assumed to move at approximately the same speed and deposit pheromone at roughly the same rate. The ants that choose to follow, by chance, the shorter path around the obstacle will reach the other side of the obstacle earlier than the ants that chose the longer route. As a consequence, the shorter path will collect a greater amount of pheromone at an increased rate over a long period of time compared to the other path. A larger number of ants will thus choose the shorter path that is richer in pheromone. This natural metaphor can be applied to the scheduling problem to generate a sequence of batches (path) that provides the greatest profit (pheromone).

Xu *et al.* [104] implemented an ant colony system (ACS) to minimise the makespan on a single batch processing machine. The ACS is a constructive based metaheuristic, since it generates a solution from scratch by iteratively adding elements to construct a feasible solution. They made use of two unique components to guarantee efficiency and effectiveness in the algorithm. The first component is described by a candidate list strategy to select only a number of different choices at each construction step so as to avoid a large neighbourhood. The second component involves constructing heuristic information by means of a new method. The candidate list considers the relationship between unscheduled jobs and the jobs in a given batch. It comprises a set of encouraging solutions instead of analysing the entire neighbourhood to ensure that

[2]It is often best to use a maximum of three neighbourhoods so as to avoid computational inefficiency.

improved solutions are achieved in a shorter computational time. The method to construct heuristic information is based on a theorem which states that the minimisation of the makespan is equivalent to determining the lower bound for the waste and idle space for the schedule.

Pheromone trails and heuristic information are then used by the ants to construct solutions. Each ant starts with an empty batch adding jobs to the batch until their current batch is full. An ant will then close the batch and start adding jobs to a new batch. The pheromone trails are updated after each ant has a feasible solution. The ACS shows similar results to the GA approach by Chou *et al.* [18] for small instances. However, there is a significant improvement in the ACS compared to the GA for large test cases demonstrating the ACS's ability to solve large scale problems.

The Pareto-based ACS (PACS) was developed by Xu *et al.* [105] to deal with parallel batch scheduling problems when two objectives are present in contrast to the single objective ACS by Xu *et al.* [104] to solve for a single machine. The objectives to be minimised are the makespan and maximum tardiness. They extended their ACS framework to a multi-objective optimisation problem (MOOP) by using multiple matrices for both the pheromone trails and the heuristic information with each objective having its own matrix. A set of nondominated solutions is returned as output for the decision maker (DM) to decide which trade-offs need to be made.

In addition to the two features mentioned by Xu *et al.* [104], Xu *et al.* employed a new solution construction mechanism to encode the batch formation, machine assignment and batch sequencing directly as a complete solution. In this mechanism, each ant selects the first available machine as its current machine. It will then form a batch depending on the machine's available time and assign it to the chosen machine. Pheromone trails are updated which is defined as the desirability of having two specific jobs in the same batch. The heuristic information with respect to the makespan is based on the waste and idle space while the maximum tardiness concerns the use of the corresponding space-related concept of tardiness space. The performance of the PACS was compared to the SMOGA introduced by Kashan *et al.* [51] after which it was concluded that the PACS performs better 92% of the time.

Finally, Afzalirad & Rezaeian [1] developed a hybrid ant colony optimisation algorithm (HACOA) that combines an ACO algorithm with the acceptance strategy of the SA algorithm to solve an unrelated parallel machine scheduling problem with precedence constraints. Release dates of jobs, sequence dependent setup times between jobs and machine eligibility[3] are considered while minimising the total late work[4]. The graph used for the HACOA is constructed by associating each job with a supernode that contains nodes that represent the eligible machines that the job may be processed on. This is opposed to the graph used by Pedersen *et al.* [77] due to machine eligibility. A dummy node that is connected to each supernode in the graph is defined which may be viewed as the ants' nest from where the ants move and return to in the algorithm to construct a solution.

A solution is constructed by an ant travelling from its nest (dummy node) to each supernode exactly once, visiting an eligible machine on that supernode, before returning to its nest. Once an ant returns after visiting each supernode, a complete sequence of the job assignments is achieved by assigning the jobs to the machines in the order that the ant visited a node in each supernode. A corrective algorithm is then applied to the sequence if the solution is infeasible due to the precedence constraints not being satisfied. The pheromone is deposited using an acceptance strategy based on the SA algorithm by applying stochastic elitism strategy instead of the more common deterministic variant. It updates the pheromone in such a way so as to

[3]Machine eligibility constraints occur when a job is able to be processed on a subset of all machines opposed to the convention that all machines are able to process any job.

[4]The late work is defined as the number of units of a job that is processed after its due date.

avoid local convergence by allowing other ants to also undertake the elitism scheme. Seven small-, seven medium- and seven large-sized randomly generated test cases are used to compare the performance of the HACOA with the HGA. The processing times, setup times and release dates were also randomly generated using integer uniform distributions. Each algorithm was executed ten times after which the solution with the smallest total late work was used as the final solution. It was concluded that both algorithms performed exceptional in the small- and medium-sized problems, but that the HACOA displayed better results for large-sized problems.

5.4.3.3 Hybrid metaheuristics

Not much research has been performed on the application of hybrid metaheuristics (compared to pure metaheuristics) to solve scheduling problems. A hybrid metaheuristic is an algorithm that incorporates several concepts from heuristics in the branches of operations research, artificial intelligence and computer science [8]. Its broad definition allows for the discovery of unique algorithms that are not confined to the ideas of a single metaheuristic. Thus, there are virtually endless possibilities as to the number of hybrid metaheuristics that can be developed. The principal reasoning for combining ideas from different algorithms is to obtain results that are better than that obtained from the individual algorithms. Knowing which combination of concepts to integrate and in which sequence these concepts should be performed in the algorithm are key to achieving success in the hybrid metaheuristic. The algorithmic ideas that are used are mostly problem specific and it is for this reason a difficult task to establish an effective hybridisation for the scheduling problem at hand.

Concepts from the metaheuristics considered in this thesis can be combined to develop hybrid metaheuristics. It is suggested to integrate the best performing metaheuristic, the tabu search method, into any of the other metaheuristics in this thesis, notably the genetic algorithms and VNS. Additional metaheuristics that can be exploited may include those mentioned in §5.4.3.1 and §5.4.3.2, namely simulated annealing and an ant colony system, respectively. Apart from the merging of the genetic algorithms with TS, the concepts from genetic algorithms or evolutionary algorithms in general, may be used in conjunction with other metaheuristics to obtain results that are possibly better than those obtained by using TS exclusively.

Another type of metaheuristic that may be implemented is the hyper-heuristic. These heuristics do not work directly on the search space of the problem like metaheuristics or hybrid metaheuristics [8]. Instead, they manage the metaheuristics that have been hybridised. The heuristic determines at which time instances or iterations during the running of an algorithm a certain metaheuristical concept should be executed or which combination of metaheuristical concepts should be utilised for a given type of test case. Its search space therefore consists of the lower-level metaheuristics. The basic idea of the metaheuristics in this thesis is to evolve a population of solutions to a strong set of schedules of which a subset is sent to the NSTB algorithm for further processing. Another type of hyper-heuristic is one that instead manages a population of metaheuristics (each containing a population of solutions) and evolves these metaheuristics [12].

List of References

[1] Afzalirad M & Rezaeian J, 2015, *Design of high-performing hybrid meta-heuristics for unrelated parallel machine scheduling with machine eligibility and precedence constraints*, Engineering Optimization, pp. 1–21.

[2] AIMMS Software & Solutions, *AIMMS*, [Online], Available from `https://aimms.com/english/software-solutions/software`, [Accessed July 24th, 2017].

[3] Arroyo JEC, dos Santos Ottoni R & de Paiva Oliveira A, 2011, *Multi-objective variable neighborhood search algorithms for a single machine scheduling problem with distinct due windows*, Electronic Notes in Theoretical Computer Science, **281(1)**, pp. 5–19.

[4] Bachman A, Cheng T, Janiak A, Ng C *et al.*, 2002, *Scheduling start time dependent jobs to minimize the total weighted completion time*, Journal of the Operational Research Society, **53(6)**, pp. 688–693.

[5] Bean JC, 1994, *Genetic algorithms and random keys for sequencing and optimization*, ORSA journal on computing, **6(2)**, pp. 154–160.

[6] Behnamian J, Zandieh M & Ghomi SF, 2009, *Parallel-machine scheduling problems with sequence-dependent setup times using an aco, sa and vns hybrid algorithm*, Expert Systems with Applications, **36(6)**, pp. 9637–9644.

[7] Bigras LP, Gamache M & Savard G, 2008, *The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times*, Discrete Optimization, **5(4)**, pp. 685–699.

[8] Blum C, Puchinger J, Raidl G & Roli A, 2010, *A brief survey on hybrid metaheuristics*, Proceedings of BIOMA, pp. 3–18.

[9] Brown MB & Forsythe AB, 1974, *Robust tests for the equality of variances*, Journal of the American Statistical Association, **69(346)**, pp. 364–367.

[10] Brucker P, 2007, *Scheduling algorithms*, 5th Edition, Springer, New York (NY).

[11] Brucker P, Kovalyov MY, Shafransky YM & Werner F, 1998, *Batch scheduling with deadlines on parallel machines*, Annals of Operations Research, **83**, pp. 23–40.

[12] Burke E, Kendall G, Newall J, Hart E, Ross P & Schulenburg S, 2003, *Hyper-heuristics: An emerging direction in modern search technology*, Handbook of metaheuristics, pp. 457–474.

[13] Cai X, Chen J, Xiao Y & Xu X, 2008, *Product selection, machine time allocation, and scheduling decisions for manufacturing perishable products subject to a deadline*, Computers & Operations Research, **35(5)**, pp. 1671–1683.

[14] Chang Jl, Chen Y & Ma Xp, 2013, *A hybrid particle swarm optimization algorithm for parallel batch processing machines scheduling*, Proceedings of the Computational Intelligence (UKCI), 2013 13th UK Workshop on, pp. 252–257.

[15] Cheng BY, Leung JT, Li K & Yang SL, 2015, *Single batch machine scheduling with deliveries*, Naval Research Logistics (NRL), **62(6)**, pp. 470–482.

[16] Chiang TC, Cheng HC & Fu LC, 2010, *A memetic algorithm for minimizing total weighted tardiness on parallel batch machines with incompatible job families and dynamic job arrival*, Computers & Operations Research, **37(12)**, pp. 2257–2269.

[17] Choobineh FF, Mohebbi E & Khoo H, 2006, *A multi-objective tabu search for a single-machine scheduling problem with sequence-dependent setup times*, European Journal of Operational Research, **175(1)**, pp. 318–337.

[18] Chou FD, Chang PC & Wang HM, 2006, *A hybrid genetic algorithm to minimize makespan for the single batch machine dynamic scheduling problem*, The International Journal of Advanced Manufacturing Technology, **31(3-4)**, pp. 350–359.

[19] Clerc M, 2011, *From theory to practice in particle swarm optimization*, Handbook of Swarm Intelligence, **8(1)**, pp. 3–36.

[20] Conover WJ, 1999, *Practical Nonparametric Statistics*, 3rd Edition, Wiley, New York (NY).

[21] Conway RW, Maxwell WL & Miller LW, 2003, *Theory of scheduling*, 1st Edition, Dover Publications, New York (NY).

[22] Correa JR & Schulz AS, 2005, *Single-machine scheduling with precedence constraints*, Mathematics of Operations Research, **30(4)**, pp. 1005–1021.

[23] Davari M, Demeulemeester E, Leus R & Nobibon FT, 2016, *Exact algorithms for single-machine scheduling with time windows and precedence constraints*, Journal of Scheduling, **19(3)**, pp. 309–334.

[24] De P, Ghosh JB & Wells CE, 1993, *Job selection and sequencing on a single machine in a random environment*, European Journal of Operational Research, **70(3)**, pp. 425–431.

[25] Deb K, Pratap A, Agarwal S & Meyarivan T, 2002, *A fast and elitist multiobjective genetic algorithm: NSGA-II*, IEEE Transactions on Evolutionary Computation, **6(2)**, pp. 182–197.

[26] Demšar J, 2006, *Statistical comparisons of classifiers over multiple data sets*, Journal of Machine Learning Research, **7(1)**, pp. 1–30.

[27] Dolgui A, Gordon V & Strusevich V, 2012, *Single machine scheduling with precedence constraints and positionally dependent processing times*, Computers & Operations Research, **39(6)**, pp. 1218–1224.

[28] Dorigo M & Gambardella LM, 1997, *Ant colony system: a cooperative learning approach to the traveling salesman problem*, IEEE Transactions on evolutionary computation, **1(1)**, pp. 53–66.

[29] Eberhart RC & Kennedy J, 1995, *A new optimizer using particle swarm theory*, Proceedings of the sixth international symposium on micro machine and human science, pp. 39–43.

[30] Eglese R, 1990, *Simulated annealing: a tool for operational research*, European journal of operational research, **46(3)**, pp. 271–281.

[31] Eren T, 2009, *A bicriteria parallel machine scheduling with a learning effect of setup and removal times*, Applied Mathematical Modelling, **33(2)**, pp. 1141–1150.

[32] Eren T & Güner E, 2006, *A bicriteria scheduling with sequence-dependent setup times*, Applied Mathematics and Computation, **179(1)**, pp. 378–385.

[33] Fan B, Yuan J & Li S, 2012, *Bi-criteria scheduling on a single parallel-batch machine*, Applied Mathematical Modelling, **36(3)**, pp. 1338–1346.

[34] Fanjul-Peyro L & Ruiz R, 2012, *Scheduling unrelated parallel machines with optional machines and jobs selection*, Computers & Operations Research, **39(7)**, pp. 1745–1753.

[35] Friedman M, 1937, *The use of ranks to avoid the assumption of normality implicit in the analysis of variance*, Journal of the American Statistical Association, **32(200)**, pp. 675–701.

[36] Gacias B, Artigues C & Lopez P, 2010, *Parallel machine scheduling with precedence constraints and setup times*, Computers & Operations Research, **37(12)**, pp. 2141–2151.

[37] Gandibleux X, 2006, *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys*, International Series in Operations Research & Management Science, Springer Science & Business Media, Berlin.

[38] Gao WJ, Huang X & Wang JB, 2010, *Single-machine scheduling with precedence constraints and decreasing start-time dependent processing times*, The International Journal of Advanced Manufacturing Technology, **46(1-4)**, pp. 291–299.

[39] Gilmore PC & Gomory RE, 1964, *Sequencing a one state-variable machine: A solvable case of the traveling salesman problem*, Operations Research, **12(5)**, pp. 655–679.

[40] Glover F, 1989, *Tabu search — part I*, ORSA Journal on Computing, **1(3)**, pp. 190–206.

[41] Glover F, 1990, *Tabu search — part II*, ORSA Journal on Computing, **2(1)**, pp. 4–32.

[42] Glover F, 1990, *Tabu search: A tutorial*, Interfaces, **20(4)**, pp. 74–94.

[43] Glover F & Taillard E, 1993, *A user's guide to tabu search*, Annals of operations research, **41(1)**, pp. 1–28.

[44] Graham RL, Lawler EL, Lenstra JK & Kan AR, 1979, *Optimization and approximation in deterministic sequencing and scheduling: a survey*, Annals of discrete mathematics, **5(1)**, pp. 287–326.

[45] Hald A, 1998, *A History of Mathematical Statistics*, 1st Edition, Wiley, New York (NY).

[46] Holland JH, 1975, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor (MI).

[47] Hoogeveen H, 2005, *Multicriteria scheduling*, European Journal of operational research, **167(3)**, pp. 592–623.

[48] IBM ILOG CPLEX Otimisation Studio, *CPLEX*, [Online], Available from `https://www.ibm.com/bs-en/marketplace/ibm-ilog-cplex`, [Accessed July 24th, 2017].

[49] Iranpoor M, Ghomi SF & Zandieh M, 2012, *Machine scheduling in the presence of sequence-dependent setup times and a rate-modifying activity*, International Journal of Production Research, **50(24)**, pp. 7401–7414.

[50] Joo CM & Kim BS, 2012, *Parallel machine scheduling problem with ready times, due times and sequence-dependent setup times using meta-heuristic algorithms*, Engineering Optimization, **44(9)**, pp. 1021–1034.

[51] Kashan AH, Karimi B & Jolai F, 2010, *An effective hybrid multi-objective genetic algorithm for bi-criteria scheduling on a single batch processing machine with non-identical job sizes*, Engineering Applications of Artificial Intelligence, **23(6)**, pp. 911–922.

[52] Kim ES & Posner ME, 2010, *Parallel machine scheduling with s-precedence constraints*, IIE Transactions, **42(7)**, pp. 525–537.

[53] Koehler F & Khuller S, 2013, *Optimal batch schedules for parallel machines*, pp. 475–486 in *Algorithms and Data Structures*, pp. 475–486. Springer, Berlin, Heidelberg.

[54] Kolmogorov AN, 1933, *Sulla determinazione empirica di una legge di distribuzione*, Giornale dell'Istituto Italiano degli Attuari, **4(1)**, pp. 83–91.

[55] Kumar VA, Marathe MV, Parthasarathy S & Srinivasan A, 2009, *Scheduling on unrelated machines under tree-like precedence constraints*, Algorithmica, **55(1)**, pp. 205–226.

[56] Kuo WH, Hsu CJ & Yang DL, 2009, *A note on unrelated parallel machine scheduling with time-dependent processing times*, Journal of the Operational Research Society, **60(3)**, pp. 431–434.

[57] Kuo WH & Yang DL, 2008, *Parallel-machine scheduling with time dependent processing times*, Theoretical Computer Science, **393(1)**, pp. 204–210.

[58] Kyparisis J & Douligeris C, 1993, *Single machine scheduling and selection to minimize total flow time with minimum number tardy*, Journal of the Operational Research Society, pp. 835–838.

[59] Laerd Statistics, *One-way ANOVA in SPSS Statistics*, [Online], Available from `https://statistics.laerd.com/spss-tutorials/one-way-anova-using-spss-statistics.php`, [Accessed July 12th, 2017].

[60] Lenstra JK & Rinnooy Kan A, 1978, *Complexity of scheduling under precedence constraints*, Operations Research, **26(1)**, pp. 22–35.

[61] Levene H, 1960, *Robust tests for equality of variances*, Contributions to probability and statistics, **1(1)**, pp. 278–292.

[62] Li S & Yuan J, 2010, *Parallel-machine parallel-batching scheduling with family jobs and release dates to minimize makespan*, Journal of Combinatorial Optimization, **19(1)**, pp. 84–93.

[63] Lilliefors HW, 1967, *On the Kolmogorov-Smirnov test for normality with mean and variance unknown*, Journal of the American Statistical Association, **62(318)**, pp. 399–402.

[64] Little JD, Murty KG, Sweeney DW & Karel C, 1963, *An algorithm for the traveling salesman problem*, Operations research, **11(6)**, pp. 972–989.

[65] Liu C, 2013, *A hybrid genetic algorithm to minimize total tardiness for unrelated parallel machine scheduling with precedence constraints*, Mathematical Problems in Engineering, **1(1)**.

[66] Liu C & Yang S, 2011, *A heuristic serial schedule algorithm for unrelated parallel machine scheduling with precedence constraints*, Journal of Software, **6(6)**, pp. 1146–1153.

[67] Liu H, 2015, *Comparing Welch ANOVA, a Kruskal-Wallis test, and traditional ANOVA in case of heterogeneity of variance*, Masters Thesis, Virginia Commonwealth University, Richmond (VA).

[68] Liu L, Ng C & Cheng TE, 2010, *On the complexity of bi-criteria scheduling on a single batch processing machine*, Journal of Scheduling, **13(6)**, pp. 629–638.

[69] Liu Z, 2010, *Single machine scheduling to minimize maximum lateness subject to release dates and precedence constraints*, Computers & Operations Research, **37(9)**, pp. 1537–1543.

[70] Lu YY, Wang JJ & Wang JB, 2014, *Single machine group scheduling with decreasing time-dependent processing times subject to release dates*, Applied Mathematics and Computation, **234(1)**, pp. 286–292.

[71] Mara CA, 2013, *Testing for equivalence of group variances*, Doctoral Dissertation, York University, Toronto (ON).

[72] Meng Y & Tang L, 2010, *A tabu search heuristic to solve the scheduling problem for a batch-processing machine with non-identical job sizes*, Proceedings of the 2010 International Conference on Logistics Systems and Intelligent Management, pp. 1703–1707.

[73] Mladenović N & Hansen P, 1997, *Variable neighborhood search*, Computers & Operations Research, **24(11)**, pp. 1097–1100.

[74] Mönch L, Balasubramanian H, Fowler JW & Pfund ME, 2005, *Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times*, Computers & Operations Research, **32(11)**, pp. 2731–2750.

[75] Montoya-Torres JR, Soto-Ferrari M, Gonzalez-Solano F & Alfonso-Lizarazo EH, 2009, *Machine scheduling with sequence-dependent setup times using a randomized search heuristic*, Proceedings of the International Conference on Computers & Industrial Engineering, pp. 28–33.

[76] Oğuz C, Salman FS, Yalçin ZB *et al.*, 2010, *Order acceptance and scheduling decisions in make-to-order systems*, International Journal of Production Economics, **125(1)**, pp. 200–211.

[77] Pedersen CR, Rasmussen RV & Andersen KA, 2007, *Solving a large-scale precedence constrained scheduling problem with elastic jobs using tabu search*, Computers & operations research, **34(7)**, pp. 2025–2042.

[78] Picard JC & Queyranne M, 1978, *The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling*, Operations Research, **26(1)**, pp. 86–110.

[79] Pinedo M, 2002, *Scheduling: Theory, Algorithms, and Systems*, 2nd Edition, Prentice-Hall, Inc., New Jersey (NJ).

[80] Pinedo M, 2005, *Planning and scheduling in manufacturing and services*, 1st Edition, Springer, New York (NY).

[81] Poon KC & Zhang P, 2004, *Minimizing makespan in batch machine scheduling*, Algorithmica, **39(2)**, pp. 155–174.

[82] Rabadi G, Moraga RJ & Al-Salem A, 2006, *Heuristics for the unrelated parallel machine scheduling problem with setup times*, Journal of Intelligent Manufacturing, **17(1)**, pp. 85–97.

[83] Rajarshi Guha, *Testing Fit to Distributions*, [Online], Available from `http://www.rguha.net/writing/notes/stats/node11.html`, [Accessed October 9th, 2017].

[84] Razali NM, Wah YB *et al.*, 2011, *Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests*, Journal of statistical modeling and analytics, **2(1)**, pp. 21–33.

[85] Rowe G & Wright G, 1999, *The Delphi technique as a forecasting tool: issues and analysis*, International journal of forecasting, **15(4)**, pp. 353–375.

[86] Salhi S, 2002, *Defining tabu list size and aspiration criterion within tabu search methods*, Computers & Operations Research, **29(1)**, pp. 67–86.

[87] Seegmuller D, Visagie S, De Kock H & Pienaar W, 2008, *Selection and scheduling of jobs with time-dependent duration*, ORiON: The Journal of ORSSA, **23(1)**, pp. 17–28.

[88] Shapiro SS & Wilk MB, 1965, *An analysis of variance test for normality (complete samples)*, Biometrika, **52(3)**, pp. 591–611.

[89] Shingala MC & Rajyaguru A, 2015, *Comparison of post hoc tests for unequal variance*, International Journal of New Technologies in Science and Engineering, **2(5)**, pp. 22–33.

[90] Sioud A, Gravel M & Gagné C, 2012, *A hybrid genetic algorithm for the single machine scheduling problem with sequence-dependent setup times*, Computers & Operations Research, **39(10)**, pp. 2415–2424.

[91] Smirnov N, 1948, *Table for estimating the goodness of fit of empirical distributions*, The annals of mathematical statistics, **19(2)**, pp. 279–281.

[92] Stadje W, 1995, *Selecting jobs for scheduling on a machine subject to failure*, Discrete applied mathematics, **63(3)**, pp. 257–265.

[93] Stützle T & Hoos HH, 2000, *Max–min ant system*, Future generation computer systems, **16(8)**, pp. 889–914.

[94] Sule DR, 2007, *Production planning and industrial scheduling: examples, case studies and applications*, 2nd Edition, CRC press, Boca Raton (FL).

[95] Sundararaghavan P & Kunnathur A, 1994, *Single machine scheduling with start time dependent processing times: some solvable cases*, European Journal of Operational Research, **78(3)**, pp. 394–403.

[96] Taha Hamdy A, 2010, *Operations research: an introduction*, 9th Edition, Pearson, London.

[97] Thevenin S, Zufferey N & Widmer M, 2015, *Metaheuristics for a scheduling problem with rejection and tardiness penalties*, Journal of Scheduling, **18(1)**, pp. 89–105.

[98] Tian J, Cheng T, Ng C & Yuan J, 2012, *An improved on-line algorithm for single parallel-batch machine scheduling with delivery times*, Discrete Applied Mathematics, **160(7)**, pp. 1191–1210.

[99] Toomey J, 2000, *Inventory management: principles, concepts and techniques*, 12th Edition, Springer Science & Business Media, Berlin.

[100] Vallada E & Ruiz R, 2011, *A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times*, European Journal of Operational Research, **211(3)**, pp. 612–622.

[101] Wilcoxon F, 1945, *Individual comparisons by ranking methods*, Biometrics bulletin, **1(6)**, pp. 80–83.

[102] Wilson JH & Keating B, 2009, *Business forecasting with accompanying Excel-based ForeastX software*, 6th Edition, McGraw-Hill, New York (NY).

[103] Xu H, Lü Z, Yin A, Shen L & Buscher U, 2014, *A study of hybrid evolutionary algorithms for single machine scheduling problem with sequence-dependent setup times*, Computers & Operations Research, **50(1)**, pp. 47–60.

[104] Xu R, Chen H & Li X, 2012, *Makespan minimization on single batch-processing machine via ant colony optimization*, Computers & Operations Research, **39(3)**, pp. 582–593.

[105] Xu R, Chen H & Li X, 2013, *A bi-objective scheduling problem on batch machines via a pareto-based ant colony system*, International Journal of Production Economics, **145(1)**, pp. 371–386.

[106] Yalaoui F & Chu C, 2003, *An efficient heuristic approach for parallel machine scheduling with job splitting and sequence-dependent setup times*, IIE Transactions, **35(2)**, pp. 183–190.

[107] Yang B & Geunes J, 2007, *A single resource scheduling problem with job-selection flexibility, tardiness costs and controllable processing times*, Computers & Industrial Engineering, **53(3)**, pp. 420–432.

[108] Yang XS & Deb S, 2009, *Cuckoo search via Lévy flights*, Proceedings of the World Congress on Nature & Biologically Inspired Computing, pp. 210–214.

[109] Yin Y, Xu D, Sun K & Li H, 2009, *Some scheduling problems with general position-dependent and time-dependent learning effects*, Information Sciences, **179(14)**, pp. 2416–2425.

[110] Zarandi MF, Mosadegh H & Fattahi M, 2013, *Two-machine robotic cell scheduling problem with sequence-dependent setup times*, Computers & Operations Research, **40(5)**, pp. 1420–1434.

[111] Zou J, Zhang Y & Miao C, 2013, *Uniform parallel-machine scheduling with time dependent processing times*, Journal of the Operations Research Society of China, **1(2)**, pp. 239–252.

APPENDIX A

Solving test cases using CPLEX

The information and CPLEX results of the first 25 test cases in the elementary difficulty group may be found in Table A.1. Table A.2 contains the information and CPLEX results of the last 25 test cases in the elementary difficulty group and the first 25 test cases in the intermediate difficulty group. Information include the number of constraints, variables and nonzeros (the number of coefficients in the constraints) for the test cases. The results are comprised of the number of iterations that CPLEX performed, the best upper bound that it could find for the profit, the profit of the best solution that was found and the solving time. A solving time of 3 600 indicates that the optimal solution could not be found.

Test case	Constraints	Variables	Nonzeros	Iterations	Best upper bound	Best solution	Solving time
1	434	540	1 832	745	123	123	0.14
2	468	540	1 951	597	133	133	0.11
3	444	540	2 003	1 153	81	81	0.97
4	692	834	3 149	68 670	244	244	0.98
5	774	932	3 421	6 072	246	246	0.58
6	2 018	2 605	10 047	221 074	623	623	6.03
7	2 158	2 605	13 165	234 571	378	378	10.67
8	3 552	4 589	23 400	1 946 220	561	561	153.17
9	3 354	4 093	23 708	781 295	446	446	43.45
10	5 578	7 565	45 665	2 751 905	784	784	1 785.63
11	6 955	6 325	52 199	3 314	262	262	3.05
12	6 768	6 320	52 861	5 848	379	379	2.08
13	7 207	6 314	54 110	7 481	196	196	2.14
14	6 983	6 314	54 210	6 488	317	317	1.88
15	6 984	6 314	55 060	7 932	150	150	3.03
16	9 316	11 957	77 840	3 904 892	2 001.91	1 092	3 600
17	10 678	13 421	89 393	2 319 119	1 935.08	1 096	3 600
18	16 241	14 654	138 430	630 240	434	434	193.77
19	16 967	14 654	139 247	294 603	325	325	36.58
20	16 273	14 654	139 368	2 222 218	395	395	400.16
21	16 946	14 654	140 051	787 328	377	377	256.24
22	16 818	14 654	141 678	438 597	311	311	47.44
23	27 166	25 264	401 936	192 886	787	787	111.63
24	27 237	25 275	404 640	64 300	503	503	305.7
25	67 188	60 425	1 030 393	1 541 717	923.73	610	3 600

Table A.1: *The information and results of the first 25 test cases after solving in CPLEX.*

Test case	Constraints	Variables	Nonzeros	Iterations	Best upper bound	Best solution	Solving time
26	68 178	60 400	1 048 535	890 113	965.85	486	3 600
27	71 271	60 400	1 058 439	229 247	900.88	388	3 600
28	66 930	60 400	1 066 575	238 864	1 011.6	525	3 600
29	70 388	65 234	1 118 166	3 825 403	1 371	816	3 600
30	62 741	56 791	1 192 832	1 250 700	585.6	556	3 600
31	61 874	56 784	1 213 940	314 519	570	570	2 544.06
32	60 502	56 784	1 218 741	2 942 521	568.68	561	3 600
33	60 578	56 784	1 239 437	933 209	524.82	500	3 600
34	81 009	75 482	1 311 857	250 928	1 520.99	849	3 600
35	82 216	75 460	1 317 338	1 697 417	1 153.87	794	3 600
36	81 533	75 460	1 318 630	176 079	1 186.48	791	3 600
37	85 899	75 460	1 356 589	2 186 712	617.67	547	3 600
38	89 843	80 881	1 413 860	985 974	973.49	775	3 600
39	104 162	92 194	1 602 775	1 319 019	1 081.69	716	3 600
40	113 137	98 144	1 740 244	184 004	1 208.57	687	3 600
41	116 416	104 301	1 824 789	2 080 717	1 515.71	1 008	3 600
42	152 185	137 973	3 054 563	4 397 801	1 419.64	824	3 600
43	150 092	137 973	3 080 219	4 061 462	1 397.65	749	3 600
44	150 905	137 969	3 087 636	5 263 816	1 722.27	918	3 600
45	149 408	137 973	3 110 381	4 196 514	1 322.38	726	3 600
46	152 583	137 940	3 121 803	2 929 021	1 972.26	890	3 600
47	151 442	137 940	3 150 581	258 186	1 573.67	666	3 600
48	134 322	128 414	3 998 004	331 070	2 214.82	903	3 600
49	134 796	128 414	4 036 516	3 409 398	1 760.48	846	3 600
50	258 142	241 024	10 000 581	154 488	2 095.24	544	3 600
51	251 920	240 994	10 005 436	210 723	2 822.52	269	3 600
52	252 848	240 994	10 074 880	832 578	2 614.67	197	3 600
53	253 523	240 994	10 135 258	4 036 833	2 526.53	503	3 600
54	60 941	56 784	11 922 006	3 112 668	784.41	759	3 600
55	446 329	395 595	13 160 537	137 487	2 352.54	700	3 600
56	479 317	430 358	14 417 816	1 340 612	2 408.54	823	3 600
57	470 654	430 368	14 636 748	2 725 345	2 628.07	789	3 600
58	497 018	448 252	14 956 451	620 601	2 666.27	564	3 600
59	479 848	448 252	15 124 950	2 813 494	2 665.35	792	3 600
60	514 390	466 550	15 826 579	154 114	2 621.64	857	3 600
61	583 168	485 248	16 236 288	8 256 232	3 441.1	1 623	3 600
62	526 066	485 214	16 567 697	4 507 452	2 787.16	914	3 600
63	639 959	523 640	17 499 511	6 953 489	3 176.1	1 556	3 600
64	711 223	584 024	19 530 227	5 904 847	3 700.78	1 462	3 600
65	687 850	584 024	20 029 662	2 469 200	3 280.59	1 141	3 600
66	693 194	604 941	20 161 796	2 048 248	2 670.18	1 285	3 600
67	767 710	647 702	21 940 697	179 420	3 766.33	1 211	3 600
68	650 101	595 252	25 296 630	6 610 641	3 300.93	1 132	3 600
69	629 608	595 252	25 326 581	5 145 812	2 886.52	1 150	3 600
70	634 732	595 301	25 373 349	2 037 325	2 558.68	1 034	3 600
71	638 862	595 252	25 730 217	3 761 872	3 361.2	1 226	3 600
72	647 957	595 300	25 759 048	2 354 247	2 701.89	256	3 600
73	790 411	748 330	32 216 228	249 086	3 203.68	545	3 600
74	915 036	860 102	36 922 248	2 557 493	3 505.72	877	3 600
75	932 756	889 307	38 493 786	2 040 206	3 278.08	542	3 600

Table A.2: *The information and results of test cases 26–75 after solving in CPLEX.*

APPENDIX B

Solving relaxed test cases using CPLEX

The information and results of the elementary, intermediate and challenging difficulty groups may be found in Tables B.1–B.3, respectively, for the relaxation of the test cases. The problems were relaxed by converting all Category 3 and 4 problems (formulated in §3.1.2) to Category 1 and 2 problems (formulated in §3.1.1) so that the processing time of a job depends only on its start time and not the jobs that preceded it. Also, the relaxation contains no restrictions as to certain jobs that are not allowed to run in parallel with other specified jobs and no restrictions as to certain jobs that are not allowed to follow other specified jobs and start in a given period. Information in the tables include the number of constraints, variables and nonzeros for the relaxed test cases. The results are comprised of the number of iterations that CPLEX performed, the best upper bound that it could find for the profit, the profit of the best solution that was found and the solving time. A solving time of 3 600 indicates that the optimal solution could not be found for the relaxation.

Test case	Constraints	Variables	Nonzeros	Iterations	Best upper bound	Best solution	Solving time
1	434	540	1 832	745	123	123	0.19
2	468	540	1 951	597	133	133	0.09
3	444	540	2 003	1153	81	81	1.5
4	686	834	2 969	55567	285	285	1.23
5	768	932	3 211	15307	277	277	1.16
6	2 018	2 605	10 047	221 074	623	623	6.7
7	2 158	2 605	13 165	234 571	378	378	11.44
8	3 540	4 589	22 280	1 555 556	614	614	108.09
9	3 302	4 093	22 668	1 461 042	479	479	111.95
10	5 578	7 565	45 665	2 751 905	784	784	1 128.47
11	420	540	1 588	645	275	275	0.13
12	384	540	1 545	6456	379	379	0.84
13	464	540	1 931	752	196	196	0.13
14	426	540	1 813	686	324	324	0.2
15	426	540	1 945	1084	150	150	0.34
16	9 298	11 957	74 960	3 776 517	2 703.13	1 183	3 600
17	10 660	13 421	86 033	2 241 620	2 370.03	1 124	3 600
18	642	834	2 711	27 438	484	484	2.14
19	664	834	2 767	43 503	366	366	2.69
20	618	834	2 658	22 904	453	453	1.73
21	690	834	2 954	12 384	452	452	1.42
22	676	834	3 039	7 661	365	365	1.44
23	1 136	1 505	5 924	2 808	825	825	0.88
24	1 137	1 505	6 246	184 411	504	504	5.83
25	1 862	2 351	9 355	1 530 254	791	791	63.53
26	1 932	2 351	10 471	306 995	711	711	32.16
27	2 005	2 351	11 242	578 996	597	597	24.17
28	1 851	2 351	10 958	345 609	706	706	25.47
29	1 897	2 445	9 657	357 031	1 076	1 076	29.31
30	2 162	2 605	11 921	25 943	591	591	11.2
31	2 086	2 605	12 432	198 483	584	584	18.5
32	1 960	2 605	12 008	229 380	580	580	14.73
33	1 945	2 605	12 769	195 326	543	543	9.45
34	1 965	2 633	10 398	1 929 828	1 311	1 311	102.31
35	1 935	2 633	10 504	416 233	1 197	1 197	34.47
36	2 003	2 633	10 906	295 843	1 173	1 173	79.63
37	2 183	2 633	13 270	273 307	734	734	30.38
38	2 072	2 727	11 067	660 803	1 122	1 122	71.55
39	2 513	2 915	12 801	836 283	1 032	1 032	31.91
40	2 379	3 009	12 556	590 049	1 078	1 078	127.61
41	2 428	3 103	12 104	736 055	1 655	1 655	95.55
42	3 108	4 093	16 061	7 868 414	1 302	1 302	731.59
43	3 134	4 093	17 165	2 977 180	1 015	1 015	536.78
44	3 194	4 093	17 904	2 290 069	1 268	1 268	600.63
45	2 996	4 093	17 416	2 316 350	1 077	1 077	259.17
46	3 056	4 093	17 966	2 249 450	1 559	1 559	802.97
47	3 128	4 093	19 235	1 545 507	1 010	1 010	158.84
48	3 447	4 785	22 500	438 827	1 425	1 425	51.28
49	3 469	4 785	23 891	841 038	1 132	1 132	89.19
50	5 896	7 565	45 424	3 043 548	985	985	1 261.45

Table B.1: *Information and results of the relaxed problem for elementary test cases after solving in CPLEX.*

Test case	Constraints	Variables	Nonzeros	Iterations	Best upper bound	Best solution	Solving time
51	5 478	7 565	41 656	1 720 091	1 226	1 226	490.28
52	5 578	7 565	45 788	3 333 670	1 090	1 090	709
53	5 568	7 565	46 985	6 440 221	996	996	1 803.84
54	2 000	2 605	10 889	7 656	801	801	4.08
55	7 107	8 465	50 907	6 736 999	1 738.4	1 482	3 600
56	7 209	8 833	52 407	6 551 473	1 748.41	1 461	3 600
57	6 735	8 833	54 165	4 030 544	2 504.92	1 380	3 600
58	7 385	9 017	51 998	4 893 147	1 613.89	1 344	3 600
59	6 822	9 017	51 676	2 571 348	2 506.43	1 404	3 600
60	7 437	9 201	56 994	5 055 444	2 300.47	1 592	3 600
61	7 308	9 385	48 805	1 425 363	3 758.27	2 280	3 600
62	7 228	9 385	57 691	8 660 205	2 675.45	1 725	3 600
63	7 028	9 753	45 277	11 405 854	3 941.85	3 129	3 600
64	7 917	10 305	50 818	2 581 532	4 451.45	2 776	3 600
65	7 889	10 305	60 353	1 306 082	5 611.1	1 927	3 600
66	8 012	10 489	50 924	1 573 345	3 720.85	2 309	3 600
67	7 934	10 857	55 268	4 856 174	5 327.25	2 880	3 600
68	8 958	11 957	59 709	3 156 807	4 181.4	2 510	3 600
69	8 448	11 957	57 387	2 846 891	3 516.77	2 005	3 600
70	9 136	11 957	63 360	6 880 625	3 145.37	1 758	3 600
71	8 669	11 957	67 039	3 256 285	4 240.3	2 097	3 600
72	9 368	11 957	73 668	2 784 604	3 696.73	1 297	3 600
73	10 025	13 421	71 490	1 763 241	3 640.73	1 963	3 600
74	10 668	14 397	71 913	4 184 701	4 255.76	2 373	3 600
75	10 593	14 641	76 538	877 757	4 347.44	2 347	3 600
76	7 769	10 945	60 394	5 704 591	2 338.07	1 938	3 600
77	15 451	20 673	118 364	2 348 712	6 603.1	2 698	3 600
78	15 201	20 673	137 645	1 631 232	5 721.09	1 861	3 600
79	18 722	24 625	152 959	2 753 433	8 768.18	3 392	3 600
80	16 636	21 585	146 349	2 424 571	7 696.13	2 686	3 600
81	18 265	24 929	170 971	3 605 569	10 035.77	2 644	3 600
82	17 464	24 017	146 731	640 496	9 721.79	3 326	3 600
83	18 011	23 713	140 174	218 814	11 094.83	4 264	3 600
84	7 711	10 945	55 527	3 973 176	3 415.62	2 936	3 600
85	18 070	25 233	134 298	3 848 901	12 398.94	4 797	3 600
86	17 896	24 017	138 735	273 691	9 937.38	3 812	3 600
87	19 252	25 841	150 571	337 627	12 687.74	4 224	3 600
88	19 038	27 057	139 926	255 649	12 394.27	5 433	3 600
89	19 229	26 145	143 059	239 410	12 728.64	5 095	3 600
90	18 677	26 145	149 362	257 756	11 953.41	4 493	3 600
91	11 412	16 155	92 934	3 171 107	2 642.82	2 016	3 600
92	11 872	16 155	94 118	2 636 651	2 993.36	2 203	3 600
93	18 052	25 611	157 662	2 403 894	7 142.26	2 938	3 600
94	18 228	25 611	151 365	17 310 394	6 890.54	2 688	3 600
95	18 726	25 611	148 690	884 632	6 725.49	2 896	3 600
96	11 549	16 155	85 531	2 172 960	2 535	2 535	3 447.88
97	17 766	25 611	127 953	4 911 340	8 158.04	4 273	3 600
98	22 689	31 521	164 824	331 414	7 604.86	3 315	3 600
99	21 019	29 945	164 138	307 684	8 299.86	3 989	3 600
100	21 164	29 551	145 368	1 983 451	8 670.77	4 588	3 600

Table B.2: *Information and results of the relaxed problem for intermediate test cases after solving in CPLEX.*

Test case	Constraints	Variables	Nonzeros	Iterations	Best upper bound	Best solution	Solving time
101	22 532	31 127	176 537	371 712	9 092.13	4 063	3 600
102	21 385	30 339	173 922	450 638	7 976.64	3 733	3 600
103	14 659	20 885	127 238	3 818 861	3 789.19	2 678	3 600
104	30 258	40 407	284 125	2 226 045	12 917.65	3 890	3 600
105	34 653	48 125	367 922	2 055 515	16 255.11	4 782	3 600
106	33 075	45 855	326 563	316 616	14 580.5	3 627	3 600
107	36 459	49 033	359 377	412 773	16 773.33	4 264	3 600
108	36 524	48 579	379 224	400 130	15 709.96	3 752	3 600
109	34 839	49 033	341 730	437 317	15 630.77	5 907	3 600
110	35 156	47 671	316 175	481 935	14 748.05	5 866	3 600
111	14 509	20 885	109 362	2 525 399	4 927.97	3 906	3 600
112	34 814	48 579	258 567	340 246	15 728.73	5 990	3 600
113	38 593	53 573	318 734	374 110	19 854.46	6 427	3 600
114	42 709	59 929	398 049	217 763	21 384.18	5 335	3 600
115	37 995	54 481	290 831	285 533	19 804.64	6 812	3 600
116	39 431	54 481	327 997	225 045	20 042.1	5 666	3 600
117	42 236	60 383	338 788	229 441	26 030.53	9 160	3 600
118	18 170	26 215	141 204	4 022 233	3 385.08	2 715	3 600
119	18 770	26 215	148 856	1 373 650	3 633.21	2 983	3 600
120	29 253	41 635	275 920	1 368 139	8 368.67	3 068	3 600
121	29 144	41 635	236 742	3 506 053	8 637.46	3 596	3 600
122	29 942	41 635	247 480	1 174 159	8 731.18	3 322	3 600
123	18 444	26 215	140 179	1 799 405	4 379.58	3 474	3 600
124	28 614	41 635	214 738	2 040 223	9 807.19	4 825	3 600
125	36 274	50 887	287 356	330 223	11 400.24	4 248	3 600
126	35 757	51 401	291 446	248 009	11 497.12	4 794	3 600
127	34 011	48 317	234 553	279 636	10 222.58	4 981	3 600
128	33 882	47 803	269 585	301 966	12 184.25	4 448	3 600
129	33 922	48 831	279 934	257 778	13 122.31	4 462	3 600
130	22 423	32 145	205 396	2 340 471	4 668.53	3 203	3 600
131	56 721	76 917	583 943	310 675	21 909.86	5 272	3 600
132	47 561	67 159	517 723	206 756	17 920.95	4 699	3 600
133	49 444	69 455	511 641	266 671	19 042.48	5 241	3 600
134	49 949	68 881	473 768	328 751	16 990.05	5 557	3 600
135	57 442	79 213	574 439	275 866	21 545.84	6 247	3 600
136	56 902	80 935	586 113	319 808	21 736.65	6 628	3 600
137	58 385	80 361	564 025	356 772	20 500.89	6 167	3 600
138	22 100	32 145	168 494	1 782 521	5 183.72	5 010	3 600
139	57 376	80 935	465 739	437 467	20 171.25	6 885	3 600
140	67 139	94 137	606 810	371 498	29 484.99	7 706	3 600
141	65 658	93 563	618 117	335 944	28 646.11	6 749	3 600
142	62 262	90 693	456 494	302 256	23 603.45	8 215	3 600
143	57 954	82 083	450 357	213 424	25 485.75	8 826	3 600
144	61 804	88 971	521 190	235 632	29 217.57	9 963	3 600
145	26 584	38 675	212 141	2 526 382	4 265.11	3 296	3 600
146	27 436	38 675	229 146	317 336	4 796.14	3 521	3 600
147	42 583	61 499	381 267	210 359	10 589.9	4 254	3 600
148	42 689	61 499	351 489	283 561	10 298.07	4 356	3 600
149	43 577	61 499	372 616	346 869	10 726.7	4 383	3 600
150	26 784	38 675	196 148	3 126 784	5 448.36	4 580	3 600

Table B.3: *Information and results of the relaxed problem for challenging test cases after solving in CPLEX.*

APPENDIX C

Measurements of CPLEX results

The measurements of the results from CPLEX for the first 25 original and relaxed test cases, original and relaxed test cases 26–75, relaxed test cases 76–125 and relaxed test cases 126–150 may be found in Tables C.1–C.4, respectively. The measurements include the profit of the schedule (A), the sum of job durations (B), the makespan of the schedule (C), the idle time of the schedule (D), the number of scheduled jobs (E), the percentage of jobs running in parallel (F) and the available space in the schedule (G).

	Original problem							Relaxed problem						
test case	A	B	C	D	E	F	G	A	B	C	D	E	F	G
1	123	15	15	0	0	6	0	123	15	15	0	0	6	0
2	133	13	15	0	2	5	2	133	13	15	0	2	5	2
3	81	14	15	0	1	4	1	81	14	15	0	1	4	1
4	244	23	15	0.4	0.5	7	0	285	23	15	0.4	0.5	8	0
5	246	20	15	0.2	0.05	7	0	277	23	15	0.4	0.41	8	0
6	623	40	40	0	0	15	0	623	40	40	0	0	15	0
7	378	39	40	0	1	8	1	378	39	40	0	1	8	1
8	561	76	40	0.3	0.54	11	0	614	68	40	0.4	1.6	13	0
9	446	70	39	0.35	1	8	0	479	63	39	0.3	1.5	9	0
10	784	80	80	0	0	13	0	784	80	80	0	0	13	0
11	262	13	14	0	1	7	1	275	15	15	0	0	9	0
12	379	14	14	0	0	7	0	379	15	15	0	0	8	0
13	196	13	14	0	1	4	1	196	14	15	0	1	4	1
14	317	14	14	0	0	6	0	324	14	14	0	0	6	0
15	150	14	14	0	0	4	0	150	15	15	0	0	4	0
16	1 092	128	80	0.3	3	15	1	1 183	116	80	0.267	8	16	1
17	1 096	128	80	0.2	5.68	15	2	1 124	111	80	0.2	6.97	15	2
18	434	21	14	0.4	0.5	7	0	484	22	15	0.5	1	9	0
19	325	14	14	0	0	6	0	366	22	15	0.4	0.5	7	0
20	395	16	14	0.2	0	7	0	453	25	15	0.6	0.5	9	0
21	377	19	14	0.2	0.5	5	0	452	23	15	0.4	0	8	0
22	311	18	14	0.2	0.5	6	0	365	23	15	0.4	2	6	1
23	787	28	29	0	1	8	1	825	29	30	0	1	9	1
24	503	29	29	0	0	7	0	504	30	30	0	0	8	0
25	610	42	29	0.267	0.5	9	0	791	49	30	0.467	0	12	0

Table C.1: *The measurements of the results from CPLEX for the first 25 original and relaxed test cases.*

	Original problem							Relaxed problem						
test case	A	B	C	D	E	F	G	A	B	C	D	E	F	G
26	486	39	29	0.267	2.5	7	0	711	47	30	0.333	0	10	0
27	388	32	29	0.133	3	5	0	597	46	30	0.267	0.5	7	0
28	525	35	29	0.133	4	6	4	706	52	30	0.4	1	8	0
29	816	44	29	0.2	1.75	9	1	1 076	52	30	0.4	1.85	11	0
30	556	39	39	0	0	11	0	591	38	39	0	1	11	1
31	570	39	39	0	0	9	0	584	40	40	0	0	9	0
32	561	39	39	0	0	10	0	580	40	40	0	0	10	0
33	500	35	39	0	4	8	4	543	38	40	0	2	9	2
34	849	45	29	0.2	1.42	8	0	1 311	60	30	0.533	1.53	14	0
35	794	46	29	0.4	4.16	9	2	1 197	56	30	0.667	0.53	15	0
36	791	26	28	0	2	7	2	1 173	41	30	0.333	1.16	10	0
37	547	29	29	0.067	0.11	6	0	734	46	30	0.2	2.76	8	0
38	775	36	29	0.133	0.74	8	0	1 122	46	30	0.4	0.81	12	0
39	716	50	29	0.333	4.46	9	1	1 032	51	30	0.533	0.49	14	0
40	687	41	29	0.2	4.08	8	2	1 078	63	29	0.533	0.86	12	0
41	1 008	40	29	0.333	1.81	9	0	1 655	56	30	0.533	0.2	13	0
42	824	41	39	0.1	2.5	11	1	1 302	62	40	0.45	0.5	16	0
43	749	58	39	0.3	2	11	0	1 015	60	40	0.3	0.5	14	0
44	918	51	39	0.25	3	10	1	1 268	67	40	0.45	1	14	0
45	726	56	39	0.25	3.5	11	1	1 077	64	40	0.4	0.5	15	0
46	890	43	39	0.1	6.5	9	5	1 559	57	40	0.35	0.5	15	0
47	666	48	39	0.15	4	7	0	1 010	58	40	0.3	2	11	1
48	903	45	59	0	14	10	14	1 425	60	60	0	0	14	0
49	846	52	58	0	6	10	6	1 132	60	60	0	0	13	0
50	544	66	77	0	11	8	11	985	80	80	0	0	13	0
51	269	19	63	0	44	3	44	1 226	78	80	0	2	13	2
52	197	14	75	0	61	3	61	1 090	79	80	0	1	13	1
53	503	57	79	0	22	6	22	996	80	80	0	0	13	0
54	759	37	39	0	2	11	2	801	40	40	0	0	12	0
55	700	78	57	0.12	4.92	9	1	1 482	116	60	0.36	3.03	18	2
56	823	69	59	0.16	7.59	9	4	1 461	93	60	0.32	5.3	13	1
57	789	69	59	0.2	11.68	9	6	1 380	129	60	0.36	4.75	14	0
58	564	41	58	0.08	30.13	6	17	1 344	110	60	0.28	3.02	13	1
59	792	53	58	0.04	5.05	9	5	1 404	106	60	0.32	1.4	15	0
60	857	80	56	0.2	7.68	8	2	1 592	108	60	0.28	2.3	14	0
61	1 623	90	59	0.24	8.5	13	7	2 280	153	60	0.48	3.5	18	0
62	914	77	59	0.2	8.93	10	3	1 725	93	60	0.32	3.35	17	1
63	1 556	43	59	0	16	13	16	3 129	138	60	0.64	4	25	0
64	1 462	70	59	0.16	15	12	4	2 776	133	60	0.56	3	20	0
65	1 141	97	59	0.2	9.25	9	0	1 927	141	60	0.36	2.75	14	0
66	1 285	82	59	0.16	9	12	0	2 309	132	60	0.44	1.5	20	0
67	1 211	72	57	0.24	13.75	9	6	2 880	133	59	0.68	6.75	19	1
68	1 132	81	79	0.1	18	11	11	2 510	118	80	0.4	4.5	26	1
69	1 150	85	74	0.167	14.5	13	11	2 055	118	80	0.367	6.5	22	1
70	1 034	104	79	0.233	13.5	14	9	1 758	121	80	0.367	2	21	0
71	1 226	87	79	0.133	13	12	8	2 097	114	80	0.333	5.5	19	2
72	256	16	78	0	62	3	62	1 297	138	80	0.367	3.5	15	1
73	545	37	62	0.067	37.06	6	34	1 963	133	80	0.3	3.39	17	1
74	877	45	71	0.067	26.1	11	26	2 373	152	80	0.5	3.23	24	0
75	542	73	61	0.1	5.52	6	4	2 347	134	80	0.433	3.14	23	0

Table C.2: *The measurements of the results from CPLEX for the original and relaxed test cases 26–75.*

	Relaxed problem						
test case	A	B	C	D	E	F	G
76	1 938	100	100	0	0	19	0
77	2 698	158	99	0.314	13.46	21	8
78	1 861	183	100	0.4	13.1	17	2
79	3 392	213	100	0.486	10	26	3
80	2 686	218	100	0.4	12.5	21	2
81	2 644	253	100	0.486	18.5	19	6
82	3 326	197	100	0.457	7.75	23	1
83	4 264	210	100	0.543	5.75	32	0
84	2 936	97	100	0	3	24	3
85	4 797	242	100	0.571	6.6	30	0
86	3 812	212	99	0.4	10.4	26	4
87	4 224	208	100	0.514	12.4	29	3
88	5 433	244	100	0.743	4.4	37	0
89	5 087	260	100	0.686	5.2	31	2
90	4 493	243	100	0.6	6.6	29	1
91	2 016	129	130	0	1	21	1
92	2 203	125	130	0	5	23	5
93	2 938	185	130	0.35	12	30	5
94	2 688	193	130	0.375	5	28	3
95	2 896	176	130	0.325	13	30	8
96	2 535	129	129	0	0	25	0
97	4 274	175	130	0.325	6.5	36	4
98	3 315	196	130	0.4	13.08	29	3
99	3 989	184	130	0.375	15.88	32	2
100	4 588	212	130	0.55	9.05	39	1
101	4 063	205	130	0.4	13.76	28	3
102	3 733	179	130	0.375	16.64	25	9
103	2 678	145	150	0	5	22	5
104	3 890	308	150	0.378	23.75	27	8
105	4 782	309	150	0.489	26.5	30	6
106	3 627	270	150	0.356	18.25	27	8
107	4 264	312	150	0.4	26.5	26	8
108	3 752	344	149	0.378	20.2	24	6
109	5 907	353	150	0.622	6.8	39	0
110	5 866	340	150	0.667	20	38	0
111	3 906	146	150	0	4	26	4
112	5 990	289	150	0.578	18	40	1
113	6 427	278	150	0.511	21.48	37	1
114	5 335	337	150	0.489	31.24	32	10
115	6 812	282	149	0.511	13.16	41	2
116	5 540	275	150	0.444	25.04	30	5
117	9 160	304	150	0.689	16.24	47	3
118	2 715	167	170	0	3	31	3
119	2 983	164	170	0	6	31	6
120	3 068	234	169	0.36	29.5	30	14
121	3 538	224	170	0.34	16.5	33	6
122	3 322	228	170	0.34	15	30	9
123	3 474	161	170	0	9	31	9
124	4 834	225	170	0.34	8.5	42	3
125	4 248	242	170	0.3	18.09	33	9

Table C.3: *The measurements of the results from CPLEX for the relaxed test cases 76–125.*

	Relaxed problem						
test case	A	B	C	D	E	F	G
126	4 794	251	170	0.4	25.76	39	14
127	4 981	242	170	0.42	9.53	43	3
128	4 448	257	170	0.34	15.21	32	5
129	4 462	259	170	0.4	29.79	31	12
130	3 203	189	190	0	1	27	1
131	5 272	353	190	0.382	23.5	35	4
132	4 699	364	190	0.418	37.5	35	12
133	5 241	339	190	0.364	14.75	35	6
134	5 557	368	190	0.473	26.25	40	8
135	6 247	339	190	0.455	24	41	5
136	6 628	390	190	0.6	14.4	46	1
137	6 167	404	190	0.527	38	38	10
138	5 010	184	190	0	6	36	6
139	6 885	300	190	0.364	8	45	1
140	7 706	384	190	0.527	30.12	42	7
141	6 749	347	190	0.345	34.24	36	12
142	8 215	329	190	0.564	15	57	1
143	8 826	311	190	0.527	25.12	50	1
144	9 963	320	190	0.618	33.96	52	8
145	3 296	204	210	0	6	38	6
146	3 521	203	210	0	7	35	7
147	4 254	256	210	0.25	24	41	11
148	4 356	270	210	0.3	16.5	45	10
149	4 383	285	210	0.317	10	40	1
150	4 580	207	210	0	3	42	3

Table C.4: *The measurements of the results from CPLEX for the relaxed test cases 126–150.*

APPENDIX D

Parameter calibration

This chapter contains six sets of tables (Tables D.1–D.12) with each set representing a metaheuristic. Each set contains the average of the time it took for the solution with the highest profit to be found and the profit of the solution with highest profit of the five times the algorithm was tested for different combinations of parameters. Cells that are highlighted in a table are the same cells highlighted for the metaheuristic's average profit table in §4.2.

			$\boldsymbol{N_1 = T}$				$N_1 = F$			
			$\boldsymbol{N_3 = T}$		$N_3 = F$		$N_3 = T$		$N_3 = F$	
N_5	N_7	N_8	$\boldsymbol{N_4 = T}$	$N_4 = F$	$N_4 = T$	$N_4 = F$	$N_4 = T$	$N_4 = F$	$N_4 = T$	$N_4 = F$
T	T	T	16.596	13.919	9.215	7.638	9.632	9.882	19.593	22.767
T	T	F	16.48	13.981	18.428	11.915	4.387	12.654	16.487	17.446
T	F	T	18.413	9.058	8.97	11.501	16.684	3.799	9.812	11.122
T	F	F	15.109	6.543	7.09	17.441	10.342	13.69	3.078	13.687
F	T	T	15.148	15.697	12.448	9.723	18.701	10.258	14.605	10.643
F	T	F	12.519	12.579	12.654	10.525	5.543	16.37	14.048	1.183
F	F	T	9.928	8.264	10.523	17.271	5.988	11.659	18.375	10.153
F	**F**	**F**	**16.936**	9.621	14.049	16.537	16.101	19.906	14.902	13.146

Table D.1: *Average time found for VNS over different parameter settings.*

			$\boldsymbol{N_1 = T}$				$N_1 = F$			
			$\boldsymbol{N_3 = T}$		$N_3 = F$		$N_3 = T$		$N_3 = F$	
N_5	N_7	N_8	$\boldsymbol{N_4 = T}$	$N_4 = F$	$N_4 = T$	$N_4 = F$	$N_4 = T$	$N_4 = F$	$N_4 = T$	$N_4 = F$
T	T	T	1317	1466	1442	1433	1493	1483	1529	1407
T	T	F	1332	1384	1540	1442	1439	1451	1429	1332
T	F	T	1467	1423	1429	1444	1337	1334	1416	1455
T	F	F	1385	1512	1482	1341	1443	1454	1418	1325
F	T	T	1481	1369	1411	1324	1340	1412	1382	1428
F	T	F	1420	1482	1430	1577	1463	1405	1384	1438
F	F	T	1468	1359	1415	1575	1452	1397	1360	1421
F	**F**	**F**	**1601**	1329	1437	1558	1398	1487	1327	1291

Table D.2: *Maximum profit (in R) for VNS over different parameter settings.*

		$\lvert\mathcal{P}\rvert = 25$			$\lvert\mathcal{P}\rvert = 50$			$\lvert\boldsymbol{\mathcal{P}}\rvert = \mathbf{75}$		
φ_p	φ_g	$\omega = \frac{1}{3}$	$\omega = \frac{2}{3}$	$\omega = 1$	$\omega = \frac{1}{3}$	$\omega = \frac{2}{3}$	$\omega = 1$	$\omega = \frac{1}{3}$	$\boldsymbol{\omega = \frac{2}{3}}$	$\omega = 1$
0.25	0.25	4.084	3.315	2.813	6.562	4.705	4.463	8.395	10.431	6.025
0.25	0.5	0.941	2.205	5.236	3.561	4.454	4.785	9.092	2.963	2.606
0.25	0.75	1.047	1.838	6.721	2.958	2.654	7.653	4.55	3.676	4.678
0.25	1.0	0.68	1.51	7.329	1.261	1.704	6.568	1.795	5.855	4.723
0.5	0.25	2.696	3.524	1.05	7.39	4.693	5.918	5.015	6.087	7.768
0.5	0.5	2.111	3.441	8.969	3.354	3.062	2.191	4.506	2.288	4.865
0.5	0.75	1.231	0.693	10.085	1.62	4.057	9.679	3.777	4.908	14.226
0.5	1.0	1.35	1.262	4.35	2.706	2.358	7.02	2.454	1.588	1.499
0.75	0.25	4.908	2.96	2.287	6.2	4.866	4.855	12.044	8.706	2.658
0.75	0.5	1.562	1.572	7.047	2.651	6.466	7.437	5.933	3.142	6.012
0.75	0.75	1.18	0.772	11.862	2.299	1.839	5.697	3.489	4.364	3.064
0.75	1.0	0.895	1.233	16.713	1.602	1.272	6.282	4.336	2.638	8.215
1.0	0.25	3.074	1.475	2.611	5.515	6.679	8.734	8.815	7.005	10.589
1.0	0.5	1.889	1.957	2.885	2.295	4.536	3.754	5.289	2.577	8.559
1.0	**0.75**	1.337	1.042	5.258	2.141	7.1	4.777	4.535	**4.478**	1.728
1.0	1.0	2.704	1.696	6.603	1.559	1.38	9.295	3.281	2.904	9.742

Table D.3: *Average time found for PSO over different parameter settings.*

		$\lvert\mathcal{P}\rvert = 25$			$\lvert\mathcal{P}\rvert = 50$			$\lvert\boldsymbol{\mathcal{P}}\rvert = \mathbf{75}$		
φ_p	φ_g	$\omega = \frac{1}{3}$	$\omega = \frac{2}{3}$	$\omega = 1$	$\omega = \frac{1}{3}$	$\omega = \frac{2}{3}$	$\omega = 1$	$\omega = \frac{1}{3}$	$\boldsymbol{\omega = \frac{2}{3}}$	$\omega = 1$
0.25	0.25	1645	1472	1470	1626	1619	1577	1611	1537	1522
0.25	0.5	1636	1687	1569	1588	1575	1556	1628	1577	1610
0.25	0.75	1509	1568	1659	1630	1635	1519	1544	1586	1578
0.25	1.0	1464	1408	1525	1532	1749	1647	1598	1576	1601
0.5	0.25	1557	1546	1505	1686	1679	1525	1630	1628	1652
0.5	0.5	1584	1538	1468	1579	1525	1583	1620	1573	1540
0.5	0.75	1484	1559	1619	1637	1684	1557	1576	1548	1592
0.5	1.0	1604	1599	1531	1566	1560	1621	1622	1580	1607
0.75	0.25	1620	1524	1597	1592	1605	1652	1606	1595	1594
0.75	0.5	1522	1653	1465	1644	1650	1599	1669	1649	1588
0.75	0.75	1570	1550	1507	1631	1600	1585	1595	1525	1590
0.75	1.0	1662	1611	1600	1528	1561	1592	1648	1633	1551
1.0	0.25	1569	1545	1621	1617	1565	1668	1590	1625	1659
1.0	0.5	1536	1586	1475	1590	1542	1625	1635	1690	1689
1.0	**0.75**	1545	1585	1563	1580	1638	1649	1618	**1726**	1547
1.0	1.0	1637	1534	1563	1517	1540	1595	1577	1644	1619

Table D.4: *Maximum profit (in R) for PSO over different parameter settings.*

				$\delta = 25$			$\delta = 50$			$\boldsymbol{\delta = 75}$		
ψ	ζ_w	ζ_n	γ	$\varepsilon = 1$	$\varepsilon = 3$	$\varepsilon = 5$	$\varepsilon = 1$	$\varepsilon = 3$	$\varepsilon = 5$	$\varepsilon = 1$	$\varepsilon = 3$	$\boldsymbol{\varepsilon = 5}$
0.05	0.1	0.15	5	7.124	20.958	9.307	13.486	12.324	15.663	17.196	15.645	9.669
0.05	0.1	0.15	10	15.605	14.149	13.708	12.609	9.657	17.49	10.886	7.766	10.316
0.05	0.1	0.15	15	7.562	13.494	20.068	15.363	9.281	12.073	14.408	16.393	13.845
0.05	0.1	0.3	5	14.827	16.761	15.408	16.896	12.021	7.33	5.189	16.592	12.162
0.05	**0.1**	**0.3**	**10**	11.818	13.695	17.336	14.247	11.367	19.226	12.68	15.145	**7.4**
0.05	0.1	0.3	15	13.932	17.156	19.235	13.509	12.229	19.332	12.644	18.729	12.863
0.05	0.2	0.15	5	12.162	12.255	8.913	15.796	11.472	18.904	17.082	13.701	11.481
0.05	0.2	0.15	10	19.202	12.992	15.482	13.111	14.28	11.417	7.729	15.895	9.342
0.05	0.2	0.15	15	20.597	15.153	7.83	6.573	15.376	15.52	11.586	15.955	7.798
0.05	0.2	0.3	5	20.257	16.316	16.46	11.382	10.402	8.124	13.069	18.679	12.681
0.05	0.2	0.3	10	16.167	8.722	11.856	8.329	16.111	14.498	8.894	16.489	15.653
0.05	0.2	0.3	15	16.744	11.454	12.846	19.111	16.407	19.491	13.043	7.913	11.556
0.1	0.1	0.15	5	15.501	16.008	15.519	10.293	9.798	8.134	14.726	19.384	14.001
0.1	0.1	0.15	10	12.653	14.495	9.099	13.02	15.552	17.322	4.827	15.816	20.284
0.1	0.1	0.15	15	18.177	10.125	14.201	18.026	2.109	10.597	15.731	13.584	11.636
0.1	0.1	0.3	5	17.374	19.435	16.606	24.685	13.2	11.592	13.31	18.763	15.48
0.1	0.1	0.3	10	12.68	16.889	9.546	20.017	16.336	14.33	8.908	10.642	20.715
0.1	0.1	0.3	15	12.038	15.707	16.346	15.348	12.064	10.886	12.703	19.055	12.467
0.1	0.2	0.15	5	15.005	12.384	12.772	10.997	14.059	23.842	18.996	12.317	16.699
0.1	0.2	0.15	10	15.482	14.597	17.304	9.766	7.884	15.019	15.945	15.011	11.248
0.1	0.2	0.15	15	12.237	8.755	14.239	15.514	22.12	12.498	23.199	14.529	15.415
0.1	0.2	0.3	5	14.339	12.315	22.603	16.373	18.77	7.923	14.593	6.741	14.328
0.1	0.2	0.3	10	13.282	16.409	10.438	18.567	22.807	18.858	11.106	11.848	13.831
0.1	0.2	0.3	15	13.248	20.976	11.891	10.211	16.02	14.245	18.94	6.393	13.148

Table D.5: *Average time found for CS over different parameter settings.*

				$\delta = 25$			$\delta = 50$			$\boldsymbol{\delta = 75}$		
ψ	ζ_w	ζ_n	γ	$\varepsilon = 1$	$\varepsilon = 3$	$\varepsilon = 5$	$\varepsilon = 1$	$\varepsilon = 3$	$\varepsilon = 5$	$\varepsilon = 1$	$\varepsilon = 3$	$\boldsymbol{\varepsilon = 5}$
0.05	0.1	0.15	5	1525	1490	1486	1506	1555	1546	1582	1610	1571
0.05	0.1	0.15	10	1504	1484	1579	1519	1520	1489	1510	1520	1521
0.05	0.1	0.15	15	1589	1526	1484	1544	1504	1511	1489	1499	1521
0.05	0.1	0.3	5	1495	1535	1494	1490	1543	1545	1530	1584	1529
0.05	**0.1**	**0.3**	**10**	1508	1506	1512	1500	1492	1489	1546	1563	**1607**
0.05	0.1	0.3	15	1531	1510	1488	1505	1496	1520	1575	1557	1498
0.05	0.2	0.15	5	1497	1490	1546	1498	1476	1477	1542	1527	1518
0.05	0.2	0.15	10	1538	1541	1533	1542	1553	1473	1511	1529	1494
0.05	0.2	0.15	15	1512	1505	1513	1567	1570	1505	1602	1546	1521
0.05	0.2	0.3	5	1514	1524	1483	1497	1555	1514	1499	1568	1526
0.05	0.2	0.3	10	1496	1478	1519	1565	1526	1493	1511	1542	1494
0.05	0.2	0.3	15	1509	1544	1488	1495	1541	1546	1486	1532	1513
0.1	0.1	0.15	5	1476	1532	1513	1559	1525	1529	1551	1493	1491
0.1	0.1	0.15	10	1490	1554	1510	1539	1522	1493	1523	1481	1491
0.1	0.1	0.15	15	1548	1548	1557	1492	1534	1561	1488	1583	1462
0.1	0.1	0.3	5	1514	1529	1559	1542	1465	1544	1522	1511	1507
0.1	0.1	0.3	10	1508	1532	1534	1506	1581	1557	1517	1500	1516
0.1	0.1	0.3	15	1573	1467	1481	1504	1489	1495	1500	1575	1508
0.1	0.2	0.15	5	1558	1590	1563	1510	1511	1501	1468	1551	1504
0.1	0.2	0.15	10	1462	1549	1493	1536	1569	1504	1556	1552	1548
0.1	0.2	0.15	15	1492	1546	1514	1555	1495	1540	1576	1508	1531
0.1	0.2	0.3	5	1508	1515	1577	1570	1489	1494	1527	1520	1605
0.1	0.2	0.3	10	1578	1603	1541	1546	1507	1514	1504	1556	1491
0.1	0.2	0.3	15	1556	1646	1502	1494	1534	1501	1492	1562	1544

Table D.6: *Maximum profit (in R) for CS over different parameter settings.*

				$\|\mathcal{P}\| = 25$			$\|\mathcal{P}\| = 50$			$\|\mathcal{P}\| = 75$		
α	β	selection	crossover	$\alpha_{max} = 3$	$\alpha_{max} = 6$	$\alpha_{max} = 9$	$\alpha_{max} = 3$	$\alpha_{max} = 6$	$\alpha_{max} = 9$	$\alpha_{max} = 3$	$\alpha_{max} = 6$	$\alpha_{max} = 9$
0.1	0.1	t	o	13.115	20.798	15.904	12.284	14.386	13.127	17.381	12.9	15.557
0.1	0.1	t	u	12.797	11.371	10.211	14.068	14.302	11.651	13.235	14.683	9.781
0.1	0.1	r	o	24.741	25.609	25.702	21.51	22.053	24.656	19.124	22.293	18.861
0.1	0.1	r	u	19.595	26.587	26.797	24.184	26.815	25.112	15.987	17.502	21.132
0.1	0.1	s	o	21.374	21.575	22.408	24.648	20.581	23.153	20.277	17.523	20.476
0.1	0.1	s	u	22.158	26.377	25.187	19.634	20.491	20.51	17.738	19.57	16.635
0.1	0.2	t	o	5.999	17.511	17.391	12.69	13.64	11.465	16.223	12.871	21.073
0.1	0.2	t	u	15.369	11.701	15.422	18.554	19.701	14.089	23.092	12.304	13.311
0.1	0.2	r	o	25.119	26.585	25.076	19.504	22.112	20.85	19.876	21.075	18.184
0.1	0.2	r	u	22.862	26.118	20.612	23.978	18.94	23.933	7.529	19.285	23.442
0.1	0.2	s	o	20.184	27.618	21.153	22.992	28.209	27.313	23.698	18.385	18.67
0.1	0.2	s	u	21.942	20.758	22.404	15.937	15.946	19.962	19.355	20.875	21.883
0.2	0.1	t	o	14.594	12.245	10.869	19.419	13.751	11.216	11.925	12.246	15.452
0.2	0.1	t	u	21.973	19.144	11.729	14.198	10.383	12.269	11.078	19.015	19.217
0.2	0.1	r	o	23.369	20.841	25.313	26.199	16.25	20.048	18.06	20.332	16.657
0.2	**0.1**	**r**	**u**	24.82	**23.871**	26.598	24.691	25.198	26.171	26.334	22.366	21.755
0.2	0.1	s	o	18.751	24.036	23.585	25.647	20.397	25.467	21.423	19.891	22.195
0.2	0.1	s	u	21.561	24.754	20.196	16.752	20.756	22.131	23.4	21.036	13.144
0.2	0.2	t	o	12.563	5.315	7.292	21.005	13.761	17.764	15.815	16.389	14.36
0.2	0.2	t	u	13.786	10.981	14.827	19.121	9.398	17.898	16.869	17.317	14.692
0.2	0.2	r	o	22.27	21.159	26.046	24.267	23.844	25.271	18.899	16.885	23.491
0.2	0.2	r	u	25.81	24.893	24.428	16.938	22.421	16.956	20.949	14.258	15.326
0.2	0.2	s	o	21.605	24.494	25.534	23.237	25.595	24.853	23.212	20.302	23.435
0.2	0.2	s	u	24.903	16.42	25.875	15.107	15.981	18.923	19.991	21.971	19.236

Table D.7: *Average time found for GA-I over different parameter settings.*

				$\|\mathcal{P}\| = 25$			$\|\mathcal{P}\| = 50$			$\|\mathcal{P}\| = 75$		
α	β	selection	crossover	$\alpha_{max} = 3$	$\alpha_{max} = 6$	$\alpha_{max} = 9$	$\alpha_{max} = 3$	$\alpha_{max} = 6$	$\alpha_{max} = 9$	$\alpha_{max} = 3$	$\alpha_{max} = 6$	$\alpha_{max} = 9$
0.1	0.1	t	o	1530	1445	1475	1546	1538	1469	1468	1523	1444
0.1	0.1	t	u	1482	1554	1430	1627	1427	1482	1474	1470	1448
0.1	0.1	r	o	1656	1711	1653	1624	1707	1594	1706	1589	1656
0.1	0.1	r	u	1702	1726	1679	1621	1545	1556	1553	1541	1572
0.1	0.1	s	o	1642	1603	1663	1613	1609	1630	1547	1591	1578
0.1	0.1	s	u	1602	1639	1597	1637	1532	1512	1506	1608	1459
0.1	0.2	t	o	1525	1481	1458	1461	1469	1458	1414	1421	1439
0.1	0.2	t	u	1479	1520	1508	1526	1420	1505	1477	1424	1472
0.1	0.2	r	o	1676	1622	1681	1615	1638	1595	1580	1565	1645
0.1	0.2	r	u	1711	1655	1671	1593	1665	1591	1524	1525	1527
0.1	0.2	s	o	1667	1650	1613	1556	1628	1652	1635	1563	1670
0.1	0.2	s	u	1642	1643	1646	1629	1552	1589	1643	1502	1507
0.2	0.1	t	o	1462	1444	1460	1485	1463	1412	1412	1471	1427
0.2	0.1	t	u	1414	1456	1422	1444	1398	1403	1410	1446	1480
0.2	0.1	r	o	1657	1692	1653	1718	1549	1604	1599	1519	1582
0.2	**0.1**	**r**	**u**	1656	**1721**	1712	1637	1763	1562	1617	1521	1576
0.2	0.1	s	o	1616	1670	1685	1611	1632	1606	1549	1590	1595
0.2	0.1	s	u	1717	1655	1664	1531	1572	1560	1553	1547	1527
0.2	0.2	t	o	1488	1437	1443	1539	1443	1475	1499	1544	1475
0.2	0.2	t	u	1423	1475	1439	1446	1516	1445	1506	1551	1454
0.2	0.2	r	o	1714	1640	1660	1649	1582	1601	1612	1548	1657
0.2	0.2	r	u	1664	1604	1658	1506	1568	1648	1561	1532	1553
0.2	0.2	s	o	1643	1627	1639	1635	1591	1648	1568	1559	1590
0.2	0.2	s	u	1640	1610	1676	1557	1560	1585	1525	1531	1541

Table D.8: *Maximum profit (in R) for GA-I over different parameter settings.*

				$\|\mathcal{P}\| = 25$			$\|\mathcal{P}\| = 50$			$\|\mathcal{P}\| = 75$		
α	β	selection	crossover	$\alpha'_{max} = 3$	$\alpha'_{max} = 6$	$\alpha'_{max} = 9$	$\alpha'_{max} = 3$	$\alpha'_{max} = 6$	$\alpha'_{max} = 9$	$\alpha'_{max} = 3$	$\alpha'_{max} = 6$	$\alpha'_{max} = 9$
0.1	0.1	t	o	10.698	6.076	16.178	11.395	6.157	11.129	7.482	10.114	8.998
0.1	0.1	t	u	8.793	16.362	8.897	8.445	14.526	17.696	14.355	12.269	17.209
0.1	0.1	r	o	10.642	10.082	14.731	11.66	7.37	14.771	17.319	13.334	8.856
0.1	0.1	r	u	17.567	11.503	17.96	20.039	23.405	23.36	18.032	20.14	18.231
0.1	0.1	s	o	9.228	12.372	11.531	8.465	13.654	15.267	14.237	22.431	7.425
0.1	0.1	s	u	16.891	18.525	24.297	19.676	19.447	21.072	22.876	17.948	25.328
0.1	0.2	t	o	9.288	12.269	17.182	10.024	10.742	9.596	10.351	6.844	13.243
0.1	0.2	t	u	7.183	11.12	6.939	13.264	22.393	14.955	12.415	24.007	13.898
0.1	0.2	r	o	17.833	14.255	12.263	19.53	15.876	17.608	21.706	22.958	6.857
0.1	**0.2**	**r**	**u**	19.561	26.622	**12.86**	20.558	18.048	18.119	22.529	21.517	20.959
0.1	0.2	s	o	10.126	17.235	11.23	10.803	17.31	8.416	15.776	16.01	11.64
0.1	0.2	s	u	19.232	22.403	17.598	21.813	19.608	23.762	21.112	17.699	20.557
0.2	0.1	t	o	9.147	15.039	5.18	13.341	13.025	17.819	19.024	15.548	6.256
0.2	0.1	t	u	18.756	9.782	14.599	8.858	10.891	15.22	13.703	9.744	8.321
0.2	0.1	r	o	15.985	5.809	14.855	11.353	8.496	8.602	16.172	5.733	13.753
0.2	0.1	r	u	15.647	16.561	17.633	24.31	12.65	17.775	23.444	14.098	19.946
0.2	0.1	s	o	17.121	10.154	20.159	10.476	16.284	16.14	15.584	4.211	13.545
0.2	0.1	s	u	15.688	18.083	24.063	15.47	18.092	18.123	17.399	19.229	17.047
0.2	0.2	t	o	8.169	13.168	8.326	17.883	1.86	12.688	7.901	13.33	16.632
0.2	0.2	t	u	11.073	16.596	7.027	8.639	9.869	10.253	9.361	6.746	10.519
0.2	0.2	r	o	19.732	9.981	13.757	20.712	9.928	17.875	12.064	17.027	17.795
0.2	0.2	r	u	18.125	20.127	11.554	21.989	15.722	22.593	17.179	17.885	18.933
0.2	0.2	s	o	5.788	10.09	13.356	18.445	7.388	13.272	6.445	5.376	12.286
0.2	0.2	s	u	11.024	16.166	19.574	21.47	22.993	21.098	22.919	13.86	13.988

Table D.9: *Average time found for GA-II over different parameter settings.*

				$\|\mathcal{P}\| = 25$			$\|\mathcal{P}\| = 50$			$\|\mathcal{P}\| = 75$		
α	β	selection	crossover	$\alpha'_{max} = 3$	$\alpha'_{max} = 6$	$\alpha'_{max} = 9$	$\alpha'_{max} = 3$	$\alpha'_{max} = 6$	$\alpha'_{max} = 9$	$\alpha'_{max} = 3$	$\alpha'_{max} = 6$	$\alpha'_{max} = 9$
0.1	0.1	t	o	1469	1441	1455	1518	1588	1505	1465	1463	1505
0.1	0.1	t	u	1453	1443	1504	1486	1531	1462	1450	1463	1441
0.1	0.1	r	o	1483	1575	1544	1476	1525	1570	1536	1638	1531
0.1	0.1	r	u	1675	1607	1635	1631	1623	1682	1657	1601	1647
0.1	0.1	s	o	1478	1530	1487	1555	1499	1480	1524	1530	1694
0.1	0.1	s	u	1658	1719	1642	1593	1580	1543	1559	1548	1639
0.1	0.2	t	o	1455	1471	1448	1449	1533	1505	1517	1415	1450
0.1	0.2	t	u	1477	1449	1515	1455	1460	1485	1485	1469	1429
0.1	0.2	r	o	1569	1576	1590	1507	1554	1478	1575	1539	1542
0.1	**0.2**	**r**	**u**	1584	1757	**1769**	1653	1682	1662	1626	1593	1579
0.1	0.2	s	o	1504	1506	1498	1480	1462	1505	1478	1474	1637
0.1	0.2	s	u	1718	1611	1675	1596	1627	1691	1615	1681	1612
0.2	0.1	t	o	1424	1447	1406	1582	1469	1423	1482	1481	1467
0.2	0.1	t	u	1433	1429	1428	1446	1449	1483	1398	1468	1397
0.2	0.1	r	o	1545	1467	1560	1474	1561	1597	1603	1531	1674
0.2	0.1	r	u	1601	1600	1714	1658	1600	1734	1635	1598	1596
0.2	0.1	s	o	1517	1542	1520	1485	1621	1504	1609	1492	1495
0.2	0.1	s	u	1616	1636	1635	1666	1574	1636	1659	1571	1617
0.2	0.2	t	o	1482	1541	1457	1486	1528	1487	1455	1442	1443
0.2	0.2	t	u	1494	1492	1494	1484	1442	1426	1510	1451	1451
0.2	0.2	r	o	1570	1600	1585	1511	1569	1551	1497	1475	1523
0.2	0.2	r	u	1699	1668	1624	1649	1666	1656	1644	1574	1603
0.2	0.2	s	o	1512	1603	1539	1568	1472	1559	1479	1510	1543
0.2	0.2	s	u	1582	1611	1621	1565	1649	1634	1645	1579	1628

Table D.10: *Maximum profit (in R) for GA-II over different parameter settings.*

		$\|\mathcal{P}\| = 3$			$\|\mathcal{P}\| = 6$			$\|\mathcal{P}\| = 9$		
υ	τ	$\|\bar{\mathcal{N}}\| = 25$	$\|\bar{\mathcal{N}}\| = 50$	$\|\bar{\mathcal{N}}\| = 75$	$\|\bar{\mathcal{N}}\| = 25$	$\|\bar{\mathcal{N}}\| = 50$	$\|\bar{\mathcal{N}}\| = 75$	$\|\bar{\mathcal{N}}\| = 25$	$\|\bar{\mathcal{N}}\| = 50$	$\|\bar{\mathcal{N}}\| = 75$
τ	$0.2n'$	15.974	25.407	24.739	20.455	16.444	20.144	17.063	20.306	25.294
τ	$0.4n'$	17.713	26.143	29.55	14.4	22.154	24.856	25.616	25.718	23.22
τ	$0.6n'$	24.45	22.752	21.607	16.842	25.465	27.569	17.796	21.812	25.848
$\tau + 0.2n'$	$0.2n'$	22.641	21.794	20.559	20.542	20.514	21.247	16.996	19.208	19.316
$\tau + 0.2n'$	$0.4n'$	28.072	19.873	28.241	22.262	23.519	24.703	13.484	20.34	21.324
$\tau + 0.2n'$	$0.6n'$	18.078	22.115	22.503	25.317	17.563	24.435	22.678	21.075	17.475
$\boldsymbol{\tau + 0.4n'}$	$\boldsymbol{0.2n'}$	22.026	19.831	18.148	18.892	20.175	**21.36**	19.576	17.383	13.28
$\tau + 0.4n'$	$0.4n'$	26.063	25.086	22.449	22.027	19.764	26.202	21.906	21.413	24.712
$\tau + 0.4n'$	$0.6n'$	16.838	25.626	25.622	23.618	30.44	21.091	20.729	24.144	27.332

Table D.11: *Average time found for TS over different parameter settings.*

		$\lvert\mathcal{P}\rvert = 3$			$\lvert\boldsymbol{\mathcal{P}}\rvert = \mathbf{6}$			$\lvert\mathcal{P}\rvert = 9$		
υ	τ	$\lvert\bar{\mathcal{N}}\rvert = 25$	$\lvert\bar{\mathcal{N}}\rvert = 50$	$\lvert\bar{\mathcal{N}}\rvert = 75$	$\lvert\bar{\mathcal{N}}\rvert = 25$	$\lvert\bar{\mathcal{N}}\rvert = 50$	$\boldsymbol{\lvert\bar{\mathcal{N}}\rvert = 75}$	$\lvert\bar{\mathcal{N}}\rvert = 25$	$\lvert\bar{\mathcal{N}}\rvert = 50$	$\lvert\bar{\mathcal{N}}\rvert = 75$
τ	$0.2n'$	1804	1783	1788	1779	1794	1826	1765	1764	1759
τ	$0.4n'$	1784	1843	1838	1800	1764	1789	1751	1796	1767
τ	$0.6n'$	1786	1794	1748	1781	1728	1769	1739	1748	1813
$\tau + 0.2n'$	$0.2n'$	1783	1817	1800	1764	1795	1763	1848	1792	1814
$\tau + 0.2n'$	$0.4n'$	1770	1814	1795	1846	1746	1809	1734	1744	1781
$\tau + 0.2n'$	$0.6n'$	1726	1758	1829	1764	1745	1754	1775	1749	1784
$\boldsymbol{\tau + 0.4n'}$	$\boldsymbol{0.2n'}$	1777	1836	1825	1786	1795	**1836**	1823	1781	1821
$\tau + 0.4n'$	$0.4n'$	1812	1798	1830	1745	1745	1797	1782	1808	1809
$\tau + 0.4n'$	$0.6n'$	1750	1763	1746	1754	1813	1746	1815	1749	1827

Table D.12: *Maximum profit (in R) for TS over different parameter settings.*

APPENDIX E

Algorithm comparisons

A total of 30 test cases (10 test cases per difficulty group) were randomly chosen to be plotted in §4.4.1–4.4.3 to compare the algorithms against one another. This appendix contains the results of the seven measurements (which excludes the measurement representing the computational times at which the solution with the highest profit was found) mentioned in §4.4 for all 150 test cases.

E.1 Profit of schedule and sum of job durations

Tables E.1–E.3 contain the profit of the schedule and the sum of job durations for elementary, intermediate and challenging test cases, respectively.

	Profit of the schedule						Sum of job durations					
test case	CS	GA-I	GA-II	PSO	TS	VNS	CS	GA-I	GA-II	PSO	TS	VNS
1	116	122	121	121	121	123	15	14	14	14	13	15
2	133	133	128	125	133	133	13	14	14	13	13	15
3	81	81	81	80	81	81	15	15	15	14	14	15
4	239	242	237	240	242	242	19	20	18	19	21	20
5	239	246	241	241	246	246	17	19	15	17	20	19
6	521	618	556	584	482	584	35	39	38	39	38	39
7	307	354	339	338	340	373	32	38	38	37	38	40
8	471	516	495	449	540	500	53	54	54	52	70	57
9	362	378	377	377	402	384	49	53	58	56	66	57
10	608	680	660	658	655	657	75	78	78	77	77	77
11	258	259	259	257	256	262	15	14	15	13	13	15
12	365	377	377	364	371	379	14	15	14	14	15	15
13	196	196	196	196	196	196	15	15	15	13	14	15
14	281	285	278	280	285	285	14	14	14	14	14	13
15	150	150	150	150	150	150	15	15	15	15	15	15
16	827	923	845	959	1 005	818	93	98	86	98	107	89
17	853	927	929	860	1 010	886	117	90	69	94	120	108
18	425	425	408	397	434	434	21	18	18	22	23	19
19	324	324	325	319	324	324	18	18	18	19	17	18
20	387	390	365	390	392	395	20	21	18	19	21	18
21	350	364	355	349	377	364	18	19	20	18	20	18
22	306	311	300	309	309	311	17	17	17	17	18	18
23	683	722	721	734	755	780	27	29	27	29	29	29
24	455	496	500	496	503	501	28	29	30	29	30	29
25	660	682	636	652	691	675	44	34	35	43	47	35
26	537	605	550	562	616	598	42	40	35	39	47	42
27	519	536	518	460	540	538	42	37	40	38	44	37
28	552	622	609	593	632	610	44	45	44	43	51	44
29	862	899	926	841	934	933	42	44	36	35	43	39
30	433	510	511	522	447	530	33	37	39	37	38	39
31	460	512	502	488	511	547	35	37	38	37	39	37
32	483	537	524	525	497	547	36	39	38	38	40	40
33	381	475	435	451	441	495	38	38	39	38	40	39
34	1035	1100	1070	1037	1160	1092	42	42	42	42	58	39
35	830	845	823	834	883	846	42	44	39	45	55	43
36	891	954	953	954	994	989	35	33	33	35	39	36
37	606	665	665	665	665	665	46	40	43	45	49	41
38	835	865	857	870	916	876	33	35	31	38	37	35
39	651	723	705	680	779	785	42	41	36	40	45	45
40	798	813	822	820	863	824	45	42	44	43	55	46
41	1 074	1 177	1 110	1 121	1 231	1 209	39	42	33	39	51	40
42	875	966	895	918	976	934	50	53	38	48	62	47
43	767	815	769	768	833	852	50	52	45	49	55	49
44	963	1 011	974	1 005	1 097	1 069	52	50	47	54	58	56
45	762	830	773	753	872	810	55	52	48	51	64	53
46	1 096	1 136	1 133	1 135	1 186	1 165	55	50	42	51	59	42
47	770	844	753	819	898	855	51	51	45	54	62	58
48	1 069	1 187	1 160	1 141	1 199	1241	58	58	58	57	59	58
49	839	1 035	932	979	1 036	1 094	56	58	59	58	58	59
50	724	832	786	802	792	875	68	72	76	77	77	76

Table E.1: *The profit of the schedule (in R) and sum of job durations for elementary test cases.*

	Profit of the schedule						Sum of job durations					
test case	CS	GA-I	GA-II	PSO	TS	VNS	CS	GA-I	GA-II	PSO	TS	VNS
51	891	980	979	962	983	985	79	76	76	76	80	79
52	760	890	846	935	917	962	79	79	79	77	79	80
53	779	908	891	846	864	932	75	77	78	78	79	78
54	589	672	601	668	673	721	37	39	39	39	39	38
55	855	1 025	978	919	1077	979	81	91	75	82	109	93
56	965	1 103	1 010	1 009	1 201	1 153	66	84	74	84	95	83
57	973	1 137	1 124	1 070	1 253	1 117	96	90	78	94	121	99
58	997	1 126	1 061	1 091	1 179	1 061	74	74	75	75	94	92
59	997	1 110	1 087	1 082	1 151	1 096	79	77	60	89	91	76
60	981	1 159	1 077	1 077	1 365	1 246	93	84	68	89	110	95
61	1 575	1 832	1 908	1 785	2 029	1 604	98	94	85	89	127	89
62	1 178	1 236	1 119	1 250	1 379	1 211	78	72	60	93	94	81
63	1 992	2 073	2 040	1 931	2 184	1 885	111	96	67	88	124	105
64	1 695	1 910	1 832	1 894	2 132	1 716	87	86	69	88	118	104
65	1 550	1 637	1 558	1 475	1 757	1 587	106	98	86	111	130	84
66	1 594	1 768	1 669	1 433	1 744	1 544	95	85	68	78	131	103
67	1 621	1 865	1 783	1 764	2 110	1 934	98	90	78	94	124	96
68	1 633	1 699	1 719	1 666	1 917	1 592	91	91	80	106	120	87
69	1 448	1 574	1 512	1 564	1 716	1 505	98	98	80	99	104	99
70	1 301	1 400	1 277	1 390	1 446	1 324	99	95	87	99	109	102
71	1 442	1 675	1 517	1 597	1 762	1 408	104	95	83	101	116	97
72	989	1 145	1 050	1 086	1 268	1 187	109	102	98	115	131	115
73	1 519	1 596	1 582	1 488	1 743	1 525	96	105	87	97	128	119
74	1 564	1 475	1 785	1 553	1 948	1 793	94	96	93	109	137	126
75	1 477	1 638	1 711	1 702	1 847	1 441	114	104	99	116	138	103
76	1 442	1 663	1 640	1 632	1 665		95	97	99	99	99	
77	1 837	1 989	1 969	1 772	2 301		142	124	100	136	153	
78	1 411	1 603	1 560	1 520	1 830		155	152	131	156	205	
79	1 953	2 135	2 194	2 038	2 265		130	138	107	144	203	
80	1 668	1 820	1 626	1 691	2 160		154	144	109	152	225	
81	1 954	2 007	2 269	1 949	2 590		169	158	149	186	266	
82	2 110	2 372	2 354	2 345	2 915		165	150	128	157	219	
83	2 293	2 581	2 682	2 384	3 023		134	137	96	141	198	
84	1 862	2 114	2 296	2 396	2 250		93	99	96	97	97	
85	3 078	3 196	3 194	3 033	3 706		151	151	118	170	201	
86	2 340	2 519	2 642	2 332	2 813		154	151	134	146	193	
87	2 499	2 614	2 491	2 568	3 151		117	118	109	143	189	
88	2 867	3 087	3 016	2 832	3 466		157	142	110	159	182	
89	2 773	2 885	2 756	2 692	3 398		140	133	105	143	181	
90	2 904	2 956	2 949	2 806	3 330		154	134	136	154	202	
91	1 473	1 680	1 586	1 687	1 630		123	128	127	127	129	
92	1 528	1 868	1 752	1 840	1 731		111	122	128	123	122	
93	1 920	1 982	2 019	2 167	2 345		161	146	128	165	200	
94	1 921	2 145	1 963	1 960	2 386		171	162	146	163	213	
95	1 998	2 186	2 106	2 122	2 463		159	152	140	170	189	
96	1 675	1 871	1 787	1 916	1 945		125	127	123	127	127	
97	3 095	3 294	3 224	3 173	3 425		161	157	134	160	190	
98	2 446	2 610	2 595	2 466	2 879		152	154	129	172	205	
99	2 744	2 806	2 763	2 734	3 045		162	148	131	165	191	
100	2 840	3 129	3 071	3 185	3 350		170	164	138	169	190	

Table E.2: *The profit of the schedule (in R) and sum of job durations for intermediate test cases.*

	Profit of the schedule					Sum of job durations				
test case	CS	GA-I	GA-II	PSO	TS	CS	GA-I	GA-II	PSO	TS
101	2 799	2 886	2 844	2 765	3 480	153	166	129	168	210
102	2 396	2 684	2 674	2 781	3 131	157	158	146	163	190
103	1 943	2 400	2 406	2 095	2 214	141	149	148	139	149
104	2 592	2 913	2 818	2 736	3 453	223	222	159	229	308
105	2 727	3 007	2 943	2 737	3 857	236	222	197	254	362
106	2 603	2 949	3 019	2 925	3 455	238	215	173	230	329
107	2 941	3 157	3 271	3 258	3 947	252	200	173	240	360
108	2 422	2 762	3 068	2 676	3 217	222	208	188	199	312
109	2 998	3 358	3 542	3 296	3 875	224	195	178	239	321
110	2 757	2 974	3 023	2 809	3 762	187	196	159	204	271
111	2 684	2 979	2 886	2 945	3 006	145	147	146	143	147
112	3 643	3 810	3 931	3 745	4 272	213	202	148	204	245
113	3 818	3 957	3 909	3 926	4 939	230	214	171	231	278
114	3 387	3 642	3 861	3 566	4 529	258	229	228	231	341
115	4 484	4 667	4 923	4 697	5 352	216	206	187	220	288
116	3 615	4 102	4 033	3 653	4 872	222	204	171	225	285
117	4 622	5 029	5 318	5 115	6 037	212	208	162	227	291
118	1 863	2 198	2 031	2 142	2 144	162	166	165	165	169
119	2 076	2 573	2 371	2 491	2 428	164	164	163	165	165
120	2 055	2 369	2 268	2 407	2 849	212	196	186	214	257
121	2 557	2 909	2 808	2 837	3 288	217	204	176	223	270
122	2 325	2 620	2 612	2 434	3 085	212	212	193	220	257
123	2 332	2 629	2 475	2 566	2 719	160	167	165	165	167
124	3 371	3 392	3 496	3 585	3 988	197	198	174	215	233
125	3 062	3 319	3 390	3 281	3 767	209	211	177	223	261
126	3 037	3 366	3 383	3 373	3 818	234	210	173	225	274
127	3 301	3 667	3 556	3 511	3 929	208	211	169	225	241
128	3 393	3 977	3 607	3 793	4 399	211	209	168	205	279
129	3 242	3 492	3 436	3 380	4 339	222	207	202	217	277
130	2 300	2 896	2 561	2 776	2 787	186	187	184	184	189
131	3 264	3 534	3 470	3 556	4 553	284	261	214	259	371
132	3 166	3 411	3 373	3 293	4 511	332	266	232	278	446
133	3 250	3 602	3 555	3 544	4 756	286	279	216	300	466
134	3 359	4 006	4 206	3 974	4 766	270	262	249	320	447
135	3 524	3 675	3 699	3 793	4 569	244	231	190	255	357
136	3 356	3 688	4 355	3 604	4 942	291	215	223	302	344
137	3 602	4 072	3 970	4 102	4 756	223	245	218	295	359
138	3 473	3 745	3 694	3 703	3 908	186	189	187	185	189
139	4 103	4 507	4 532	4 261	5 159	244	230	190	272	293
140	4 882	5 281	5 497	5 356	6 461	289	302	200	285	434
141	4 635	5 045	5 061	4 880	6 401	316	282	221	251	468
142	5 001	5 217	5 699	5 451	6 332	278	258	226	287	352
143	5 312	5 587	5 763	5 486	6 880	251	220	188	274	351
144	5 175	5 697	5 913	5 266	7 072	242	247	197	257	354
145	2 282	2 466	2 512	2 622	2 789	204	208	199	204	206
146	2 377	2 809	2 807	2 654	2 823	199	203	202	207	199
147	3 083	3 202	3 206	3 219	3 974	258	247	224	261	322
148	3 138	3 566	3 450	3 436	4 273	257	261	220	243	315
149	2 930	3 312	3 292	3 177	3 938	268	242	214	265	323
150	3 000	3 301	3 242	3 505	3 502	206	205	205	206	208

Table E.3: *The profit of the schedule (in R) and sum of job durations for challenging test cases.*

E.2 Makespan and idle time of schedules

The makespan and idle time of the schedule for elementary, intermediate and challenging test cases may be found in Tables E.4–E.6, respectively.

	Makespan of the schedule						Idle time of the schedule					
test case	CS	GA-I	GA-II	PSO	TS	VNS	CS	GA-I	GA-II	PSO	TS	VNS
1	15	14.6	14.8	14.8	14.4	15	0	0	0	0	0	0
2	14.6	14.8	14.8	14.2	14.6	15	0	0	0	0	0	0
3	15	15	15	14.8	14.8	15	0	0	0	0	0	0
4	14.8	15	14.4	14.8	15	15	0	0.2	0	0	0	0
5	14.8	15	14.8	14.6	15	15	0	0	0	0	0	0
6	37.2	39.8	39	39.4	39.4	39.8	0	0	0	0	0	0
7	37.6	39.2	39.6	39	39.2	40	0	0	0	0	0	0
8	38.4	39.6	38.8	38.8	38.4	39.2	0	0	0	0.6	1	0.6
9	37.8	39.4	38.4	39.2	39	39.4	0	0.4	0.2	0.2	0	0.2
10	78.6	79.2	79	78.4	78.8	79	0	0	0	0	0	0
11	15	14.8	15	14.4	14.6	15	0	0	0	0	0	0
12	14.8	15	14.8	14.6	15	15	0	0	0	0	0	0
13	15	15	15	14.4	14.6	15	0	0	0	0	0	0
14	14.4	14.4	14.2	14	14.2	13	0	0	0	0	0	0
15	15	15	15	15	15	15	0	0	0	0	0	0
16	74.2	78.8	78.8	77.8	79.4	77	0.6	0	0	0	0	0.8
17	78.6	77.8	76.8	77	79.6	78.4	0.2	0	0.2	0.4	0.2	0.2
18	14.8	15	14.8	12.4	15	15	0	0	0	0	0.2	0
19	15	15	15	15	14.8	15	0	0	0	0	0	0
20	14.8	14.6	14	14	14.6	14.8	0	0	0	0	0	0
21	15	15	15	15	15	15	0.2	0	0	0.2	0	0
22	14.2	14	14	14.4	14.6	14	0.6	0.2	0.4	0.4	0	0
23	29	29.6	29.2	29.8	29.8	29.6	0	0	0	0	0	0
24	29.6	29.8	30	29.4	30	29.8	0	0	0	0	0	0
25	29.8	30	29.8	29.2	30	29.6	0.2	0	0	0	0.2	0
26	29	29.6	29.4	27.8	29.2	29	0	0	0	0	0	0
27	29.2	29.2	30	29.2	29.6	29.8	0	0	0	0	0	0
28	29.4	29.4	29.8	29.6	29.2	29.8	0.2	0	0	0	0	0
29	29	29.6	29.2	29.4	29.6	29.8	0	0	0	0	0	0.2
30	37.8	39	39.4	39	39.2	39.4	0	0	0	0	0	0
31	37.6	39	39	38.8	39.6	38.8	0	0	0	0	0	0
32	38.8	39.6	39.2	39	40	40	0	0	0	0	0	0
33	38.6	39.4	39.4	39.4	40	39.8	0	0	0	0	0	0
34	28.8	30	29.6	29.4	30	29.6	0.2	0	0	0	0.2	0.4
35	29.4	29.8	28.8	28.8	29.6	29.8	0	0.4	0.6	0	0.8	0
36	29	29.6	29	29.4	28	29	0	0	0	0.4	0.2	0
37	29.2	30	29.8	29.6	30	30	0.4	0.2	0	0.2	0	0
38	29.2	29.6	29.6	28.2	30	30	0	0	0	0.4	0	0
39	28.8	29.2	29.6	29.2	29.2	29.2	0	0	0.2	0	0	0
40	29	29.8	29.2	28.8	29.8	29.6	0	0	0	0	0.2	0
41	28.8	28.8	28.6	29	29.6	29.8	0.2	0	0	0	0	0
42	38.6	40	39.6	39.2	39.8	39.6	0	0	0	0.8	0	0
43	39	39.8	39.4	39.4	39.6	39.8	0	0	0.2	0	0.2	0.2
44	38	40	39.4	37.6	39.4	39.4	0	0.2	0	0.4	0.2	0
45	39.4	39.6	37.4	39.4	33.2	38.8	0.2	0.2	0	0	0	0
46	38.6	33.8	39	38	38.4	39.2	0	0.2	0	0.4	0.2	0
47	39.6	39.8	38.2	39.2	39.6	39.4	0.4	0	0	0.6	0	0
48	59.4	58.4	59.4	58.4	59.4	59.6	0	0	0	0	0	0
49	57.8	59.2	59.2	59.2	59.2	59.4	0	0	0	0	0	0
50	76.2	77.8	78.4	78	78.4	78.2	0	0	0	0	0	0

Table E.4: *The finish and idle time of the schedule for elementary test cases.*

	Makespan of the schedule						Idle time of the schedule					
test case	CS	GA-I	GA-II	PSO	TS	VNS	CS	GA-I	GA-II	PSO	TS	VNS
51	79.6	78.6	78.8	79	80	79.8	0	0	0	0	0	0
52	79.4	79.4	79.4	79.2	79.6	80	0	0	0	0	0	0
53	78.2	78.8	79.2	79	79.8	79	0	0	0	0	0	0
54	39.2	39.8	39.6	39.6	39.8	39.6	0	0	0	0	0	0
55	49.6	57.8	58.4	58.6	57.4	59.6	0	0	0	0	0	0
56	56.2	58.8	59.4	58	59.2	59.2	0	0	0	0.2	0	0
57	58.8	59.2	58.8	59.2	59.8	59.6	0	0	0	0	0	0.2
58	57.6	58	59	59.2	59.6	57.8	1.2	0.2	0.4	0	0	1.2
59	52.8	58.2	58.2	58.6	53.4	58.2	0	0	0	0.2	0.2	0.2
60	57.6	56.6	56.4	57.2	59	59	0	0	0	0.2	0	0.2
61	54	59.2	58.4	58.4	59.4	58.6	0	0.2	0	0	0	0
62	57.4	50.8	58.4	53.2	59.2	54.4	0	1	0	0	0	0
63	58.6	55	59	59.4	59.4	58.6	0	0	0	0	1	0
64	59.4	58.6	57.8	58.4	59.4	59.4	0	0	0	0.2	0.2	0
65	59.4	59.2	59.6	52.4	59.4	59.2	0.2	0	0.8	0	1.6	1
66	54.6	58.6	58.4	57.6	59.8	59	0	0	0	0	0.6	1
67	57.8	59.2	58.6	56.8	59.8	58.4	0	0.2	0	1.6	0	0.6
68	75.8	68.4	77.8	77	80	77	0	0.6	0.2	0.2	0	0.4
69	78	78.6	79.4	78.8	80	79	0	0	0	0.2	1.8	0.2
70	75.8	78.8	78.6	78	79.8	78.2	0	0	0	0	1.8	1.2
71	79.8	79.4	78.6	79.2	79.8	77.8	0.4	0.6	0.8	0.8	1.2	0.8
72	77.6	78.2	79.2	77.4	79.4	79.6	0	0	0	0	0	0.4
73	78.8	78.6	78.4	78.4	79.2	78.4	0.2	0.2	0	0.4	0.2	0.4
74	78.6	78.4	78.6	79.4	79.6	77.8	0	0	0	0.2	0	0
75	75.8	78.8	80	77.2	79.4	76.6	0.8	0	0.2	0.2	2.2	0
76	98	98.6	99.2	99.8	99.6		0	0	0	0	0	
77	97.8	98.8	98.6	97.2	97.2		0	0	0	0	0.2	
78	99.2	98.6	99.4	99.2	98.6		0.6	0.2	0	0	0.6	
79	97.4	98.8	90	97.2	99.6		0	0	0	0	0.2	
80	97.6	99	94.2	95.2	97.8		0.2	0	0	1	0	
81	98.8	99	99.2	98.6	98		0.2	0	0	0.2	0.2	
82	97.8	98	99.4	98	99.4		0	0.2	0.4	0	0.8	
83	96.8	95.8	98.8	96.2	99.2		0	0.4	0	0	0.4	
84	96.8	99.2	99.2	99.4	99		0	0	0	0	0	
85	97.6	97.8	97.8	82.2	99.2		1.8	0	0	1.2	0.2	
86	98.4	97.6	98.4	97.4	98.6		0	0	0	0	0.2	
87	96.2	97.2	99.2	96	99.6		0	0.8	0.2	0	0	
88	98.2	97.8	98.2	96.2	99		0	0	0.2	0	2.6	
89	98.4	96.6	98.8	80.6	98.2		0	0	0	1	2.2	
90	98.6	95	99.6	95	99		0.2	0	1	0.2	5.4	
91	125.6	129	128	128.4	129.8		0	0	0	0	0	
92	121.4	126.6	129.4	127	127		0	0	0	0	0	
93	127.6	126.6	128.2	127	130		1	2.2	0.2	0.2	2.4	
94	125.2	129.2	128.6	126	130		0.6	1.4	0	0	1.2	
95	128.4	129.2	127.8	127.6	128.6		0.4	0	0.4	0.2	2.2	
96	129	128.8	127.6	128.8	128.8		0	0	0	0	0	
97	129.6	128.2	129.2	129.6	129.6		0.6	0.4	0	0.2	2.6	
98	127.6	128	128.8	127.2	129.8		0	0.2	0	0	0	
99	127.4	128.2	128.2	128.8	129.6		1.2	0.8	0.4	0	0.2	
100	129.2	127.4	128.8	130	130		0	0	0	0	0	

Table E.5: *The finish and idle time of the schedule for intermediate test cases.*

	Makespan of the schedule					Idle time of the schedule				
test case	CS	GA-I	GA-II	PSO	TS	CS	GA-I	GA-II	PSO	TS
101	129.2	129.8	128.6	127	129.2	0.2	0	0	0.4	0
102	126.8	128.6	129	128.6	129.8	1.4	0.2	0.2	0	0
103	146.8	149.4	149.2	145.8	149.8	0	0	0	0	0
104	145	148.2	148.6	144.2	149.2	0.2	0	0	1.2	0
105	146	147.6	148.8	146.2	148.8	0	1	0.6	1.4	3.4
106	118	148.2	149	146.4	148.4	1.8	1.6	0	0.6	0.4
107	142.2	139.8	147	148.8	125.6	0	1	0	0	0
108	144.6	146.6	148	128.6	147.4	1	2	0	0.4	0
109	144.2	147.2	148.8	144.8	149.2	1.2	0.2	0	1.2	1.6
110	142.8	147	149.4	145.6	149.2	1.2	0.4	0.6	2.4	0.6
111	147.4	148.8	148.6	148.2	148.8	0	0	0	0	0
112	146.2	148.6	149	126.4	149.8	0.4	0	0	0.6	0.4
113	146.8	148	149	148.6	149.4	0.2	0.2	0	0.4	0
114	146.4	146.6	149.2	146.2	148	0	1	1	2	7
115	146.6	146	148.8	146.2	147.4	0	0.6	0	1.2	2.6
116	146.6	149.2	148.6	143.4	149.4	0.4	0.6	0	0	0.6
117	145.6	145.4	148.6	148	147.8	0.2	1.4	0	0.2	1.2
118	166.8	168.6	168.4	168.6	169.6	0	0	0	0	0
119	167.4	167.4	166.2	169	167.4	0	0	0	0	0
120	167	152.8	167.8	167.8	169.6	0.6	0.2	0.8	1.4	0.2
121	166.8	168.4	169.2	167.4	169.6	0	0	0	2	1.8
122	164.2	166	168.8	151.8	169	0	2	0.8	0.4	0
123	165.6	168.4	167.2	167.2	168.4	0	0	0	0	0
124	169	167	168.4	166.8	165.8	0.4	1	0	2.2	1.2
125	168.2	165.8	169.8	166.4	169.4	0	0	0	0.2	0.4
126	166.6	166.2	168.8	168.8	169	0.6	0.2	0.2	1	1.8
127	166	169.4	169.6	167.4	169.4	0	0	0	0	0
128	167.2	168.4	168.8	168.2	169.4	0	0.4	0.2	0.2	0.6
129	157.8	167	168.8	166.8	169.2	0.6	0	0.2	0	1.2
130	188.6	188.6	188.6	188.6	189.6	0	0	0	0	0
131	184.8	184.6	188	185.6	189	0.6	0	0.2	2.2	0
132	187.6	187.8	189.2	186.2	188.4	0.2	0.8	0	1.4	0
133	186.6	185.2	189.2	185.6	187.4	2.4	0.4	0	1.2	1.8
134	182.4	186.2	188.6	183	188	0.2	1.4	1.6	0	1.8
135	184.8	186.4	185.8	156	188.8	1.4	1	0	1.6	0
136	176.6	183	188.4	181.2	189.2	5.4	0.2	0	2.4	2.8
137	182.6	184.6	186.8	165	188.2	0	0.8	0.2	2	0.4
138	187.6	189.4	188.6	188	189.4	0	0	0	0	0
139	185.8	185.6	188.2	188.4	189.2	3	0	0.2	0	0.2
140	186.6	184.4	188.6	186.6	189.6	0	0.4	0	0.8	0
141	187	187.6	189.2	185	189.4	0	0.4	0.6	2	1.6
142	187.4	186.4	187.2	186.4	189.6	0.2	0	0	0	2.6
143	187.8	183.8	188.4	189	189.6	0	0.2	0	0	2.2
144	186	187	188	186.6	189.4	0	0	0	2.4	0.6
145	207.4	209	206.8	208.4	209	0	0	0	0	0
146	206.4	207.8	207.8	208.2	206.8	0	0	0	0	0
147	205.6	207.6	208.2	207.4	209.4	1	0.4	0	0.2	1
148	207	207	209.2	196.2	209.6	0	0.6	0	0	0.8
149	204.4	208.4	208.2	179	208.8	0.4	0.4	0	0	0.4
150	208.2	208.2	208.2	208.8	208.8	0	0	0	0	0

Table E.6: *The finish and idle time of the schedule for challenging test cases.*

E.3 Jobs and available space in schedules

The number of scheduled jobs and the percentage of jobs running in parallel for elementary, intermediate and challenging test cases are displayed in Tables E.7–E.9, respectively. Tables E.10–E.12 contain the results for the available space in the schedule for elementary, intermediate and challenging test cases, respectively.

	Number of scheduled jobs						Percentage of jobs in parallel					
test case	CS	GA-I	GA-II	PSO	TS	VNS	CS	GA-I	GA-II	PSO	TS	VNS
1	5.2	5.6	5.2	5	5.2	6	0	0	0	0	0	0
2	5	4.6	4.2	4.6	4.4	5	0	0	0	0	0	0
3	4	4	4	3.4	4	4	0	0	0	0	0	0
4	7	7	6.4	6.8	7	7	0.24	0.32	0.22	0.24	0.32	0.24
5	6.6	7	6.4	6.4	7	7	0.24	0.2	0.16	0.28	0.28	0.2
6	12.6	14	13.2	13.8	11.6	12.8	0	0	0	0	0	0
7	6.4	6.8	7	7	6.2	7.8	0	0	0	0	0	0
8	9.2	9.8	10	9	10.4	9.4	0.25	0.23	0.18	0.25	0.32	0.29
9	6.2	6.4	6.4	6.6	7	7	0.19	0.18	0.22	0.24	0.29	0.2
10	10	10.4	10.4	10.2	10	9.8	0	0	0	0	0	0
11	8	8.2	8	8	7.6	8.8	0	0	0	0	0	0
12	8	7.8	8	7.6	8	8	0	0	0	0	0	0
13	4	4	4	3.8	3.8	4	0	0	0	0	0	0
14	5	5	5	5	5	5	0	0	0	0	0	0
15	4	4	4	4	4	4	0	0	0	0	0	0
16	11.6	12	12	12.2	13.2	10.2	0.22	0.22	0.093	0.207	0.293	0.24
17	11.4	12.2	11.6	11.2	12.2	10.6	0.173	0.113	0.08	0.14	0.22	0.187
18	7	7.8	7.2	6.6	7.2	7.4	0.44	0.24	0.32	0.42	0.56	0.28
19	6.2	6.4	6.4	6.2	6	6.4	0.24	0.32	0.26	0.26	0.28	0.28
20	7.8	8	7	7.6	8	8	0.44	0.48	0.36	0.48	0.52	0.44
21	5.8	6	5.2	5.4	6.2	5.8	0.32	0.2	0.3	0.34	0.36	0.24
22	5.2	5.8	5	5.4	5.8	6	0.24	0.2	0.28	0.28	0.32	0.2
23	7	7.4	7	7.6	8	8	0	0	0	0	0	0
24	6.8	7	7	6.4	7	7	0	0	0	0	0	0
25	9.4	9.6	9.4	9.4	9.6	9.4	0.347	0.24	0.213	0.347	0.453	0.293
26	7.4	8	7.4	7.2	8	8	0.333	0.267	0.213	0.333	0.387	0.334
27	6.2	6.6	6.6	5.8	6.6	6.2	0.28	0.293	0.267	0.24	0.347	0.267
28	6.2	6.6	6.6	6.4	7	6.8	0.294	0.294	0.267	0.333	0.4	0.294
29	9	9.6	9	8.6	10	9.2	0.306	0.28	0.16	0.187	0.306	0.293
30	8.4	9.2	8.6	8	7.6	9.4	0	0	0	0	0	0
31	7	8	7.2	7.2	7.4	8.2	0	0	0	0	0	0
32	8.2	9	8.8	8.4	8	9	0	0	0	0	0	0
33	6	7.4	6.8	6.8	7.2	8	0	0	0	0	0	0
34	10.6	11.4	11	10.6	11.8	11	0.253	0.227	0.2	0.32	0.4	0.24
35	9.2	10	9.8	9.4	10.8	9.8	0.293	0.306	0.267	0.36	0.413	0.347
36	7	7.6	7.6	7.6	8	8	0.24	0.24	0.16	0.213	0.267	0.187
37	6.2	6.8	6.6	6	7	7	0.213	0.173	0.187	0.2	0.2	0.146
38	8.2	8.4	8	8.2	9.6	8.8	0.173	0.213	0.146	0.28	0.293	0.213
39	8	8.4	8.4	8	9.6	8.8	0.28	0.2	0.227	0.333	0.32	0.28
40	8.8	8.8	8.4	7.8	9.8	8.8	0.333	0.28	0.267	0.333	0.48	0.28
41	9.6	10.2	9.6	9.4	11	10.4	0.333	0.36	0.267	0.28	0.427	0.32
42	11.4	12.4	11.2	11.8	12.4	11.8	0.35	0.35	0.16	0.33	0.44	0.28
43	10.4	10.8	10.4	10.6	10.8	10.8	0.33	0.22	0.14	0.24	0.32	0.24
44	10.4	11.4	10.4	10.6	11.6	11.4	0.31	0.3	0.27	0.35	0.4	0.37
45	10.8	11.8	11.2	11.2	12.8	11.2	0.29	0.26	0.23	0.27	0.41	0.32
46	10	10.6	10.4	10	11.2	10.6	0.34	0.32	0.24	0.34	0.42	0.3
47	8.6	8.6	7.8	8.6	9.8	9.2	0.29	0.22	0.15	0.34	0.36	0.32
48	9.2	10	9.6	10	10.4	10.2	0	0	0	0	0	0
49	9.2	11.2	10.6	10.2	11	11	0	0	0	0	0	0
50	10.4	10.2	10.4	9.8	9.8	10.4	0	0	0	0	0	0

Table E.7: *The number of scheduled and percentage of parallel jobs for elementary test cases.*

	Number of scheduled jobs						Percentage of jobs in parallel					
test case	CS	GA-I	GA-II	PSO	TS	VNS	CS	GA-I	GA-II	PSO	TS	VNS
51	9	9.8	9.4	9.6	9.6	9.6	0	0	0	0	0	0
52	8.8	9.8	9.2	9.8	10.2	10.6	0	0	0	0	0	0
53	9.8	10.8	10.4	10.2	10.6	11	0	0	0	0	0	0
54	8.2	9.6	8.2	8	9.4	10	0	0	0	0	0	0
55	10.6	12.8	12.2	11.4	12.8	12.2	0.232	0.256	0.176	0.248	0.256	0.296
56	8.6	10.2	9.2	9.2	10.4	10	0.168	0.216	0.192	0.224	0.272	0.248
57	10.2	10.4	10.6	10.2	12.2	11	0.192	0.2	0.24	0.184	0.272	0.264
58	9.2	10	9.4	9.6	10.8	9.4	0.224	0.216	0.176	0.16	0.24	0.232
59	10.8	11.6	10.8	10.6	12.2	10.8	0.152	0.136	0.064	0.2	0.184	0.184
60	8.6	9.6	9.6	9	11.4	10.2	0.128	0.16	0.168	0.208	0.256	0.288
61	13.4	14.8	14.2	14	16	12.6	0.344	0.328	0.288	0.312	0.472	0.36
62	11	11.4	10.2	11.2	12	10.8	0.176	0.144	0.08	0.224	0.248	0.24
63	16.2	16.8	16.6	16	18	14.8	0.368	0.304	0.184	0.336	0.472	0.424
64	13.2	13	12.6	13.6	16	13.4	0.28	0.224	0.16	0.304	0.448	0.384
65	10.8	10.4	10.8	10.6	12.6	9.8	0.304	0.184	0.224	0.304	0.4	0.264
66	15.4	16	16.2	13.4	16.6	14	0.296	0.272	0.192	0.256	0.504	0.368
67	11.2	12.4	10.8	11.4	14.8	12.6	0.344	0.328	0.232	0.36	0.512	0.4
68	17	18.6	17.6	17	19.8	16	0.26	0.233	0.16	0.3	0.44	0.28
69	16	17.2	16.4	17.2	17.8	15.8	0.227	0.22	0.074	0.24	0.333	0.307
70	15.2	16.2	14.6	15.6	17	14.8	0.28	0.233	0.147	0.3	0.353	0.32
71	13.4	13.4	13.2	13.6	15.8	12.4	0.227	0.207	0.173	0.227	0.387	0.207
72	11.8	13.2	12.6	13.2	15.4	13.4	0.267	0.233	0.193	0.3	0.433	0.353
73	13	13.8	13.2	12.6	15.4	13.6	0.193	0.18	0.127	0.187	0.32	0.267
74	15.6	15	16.8	15.6	18.4	17.2	0.24	0.16	0.18	0.267	0.36	0.42
75	15.2	15.4	16	15.6	18	13.4	0.213	0.2	0.127	0.26	0.34	0.273
76	14.4	15	14.8	15.4	14.2		0	0	0	0	0	
77	15.2	16.6	16	15	18		0.251	0.2	0.137	0.229	0.32	
78	12.8	13.6	12.6	12.8	15.8		0.206	0.188	0.154	0.206	0.337	
79	16.6	17.8	17.2	17	19.2		0.251	0.189	0.143	0.251	0.412	
80	12.6	13	12.2	12.6	16.2		0.28	0.183	0.109	0.229	0.366	
81	14.2	14.4	15.6	14.8	19.6		0.286	0.24	0.229	0.303	0.451	
82	16.8	18	17	17.6	21.8		0.309	0.229	0.16	0.229	0.423	
83	16.8	19.2	18.2	17.6	21.2		0.274	0.32	0.12	0.32	0.44	
84	14.4	16.6	17.2	16.2	17.8		0	0	0	0	0	
85	21.2	21.4	21.8	21	24		0.251	0.331	0.137	0.309	0.48	
86	16.4	18.4	18.6	16.4	19.2		0.223	0.257	0.154	0.251	0.4	
87	16.6	15.8	16.4	16.2	19.8		0.257	0.206	0.12	0.234	0.389	
88	20.2	21	21.6	19.6	23.8		0.337	0.297	0.206	0.32	0.44	
89	18.2	18.4	17.6	17.6	20.8		0.309	0.229	0.114	0.291	0.423	
90	17.2	17.4	17.8	17.2	21.4		0.263	0.234	0.217	0.269	0.469	
91	16.2	15.8	16	15	16.4		0	0	0	0	0	
92	15.4	17.6	16.8	16.8	17		0	0	0	0	0	
93	18.8	18.6	19.6	19.4	22.6		0.245	0.21	0.135	0.25	0.385	
94	19	20	19.6	18.6	23		0.305	0.275	0.18	0.265	0.495	
95	21.2	21.8	21.2	20.4	23.8		0.24	0.195	0.165	0.24	0.37	
96	16	17.4	16.2	16.6	17.4		0	0	0	0	0	
97	26.4	27.8	26.6	26.2	27.8		0.265	0.225	0.105	0.255	0.4	
98	21.6	22.4	22.2	22	25.2		0.17	0.2	0.085	0.23	0.345	
99	20.4	22.2	20.8	21.2	23.8		0.23	0.21	0.1	0.215	0.325	
100	24.6	26	25.8	26.2	26.4		0.295	0.26	0.115	0.285	0.375	

Table E.8: *The number of scheduled and percentage of parallel jobs for intermediate test cases.*

	Number of scheduled jobs					Percentage of jobs in parallel				
test case	CS	GA-I	GA-II	PSO	TS	CS	GA-I	GA-II	PSO	TS
101	20.4	20.8	19.8	20	24.6	0.2	0.21	0.085	0.225	0.385
102	17	17.4	16.6	17.6	21.2	0.205	0.175	0.125	0.205	0.335
103	16.6	19	18.6	16.4	18.4	0	0	0	0	0
104	20.2	23.2	21.6	21	26.2	0.258	0.262	0.155	0.271	0.409
105	18.4	19.8	19.2	18	24	0.236	0.231	0.142	0.244	0.418
106	18	19	20.6	19.4	24	0.249	0.205	0.156	0.276	0.404
107	19.4	20.2	20	20	25.2	0.271	0.227	0.16	0.253	0.467
108	15.4	16.8	16.6	16.6	19.8	0.2	0.222	0.178	0.227	0.382
109	21	23	23	21.8	27.2	0.307	0.24	0.173	0.307	0.44
110	19	19.2	19.2	18.8	24.4	0.262	0.204	0.151	0.253	0.378
111	18.8	19.6	19.2	19.6	20.2	0	0	0	0	0
112	26.4	27	26.8	26.2	28.2	0.293	0.284	0.062	0.311	0.449
113	23.4	24.2	23.6	24	28.6	0.262	0.218	0.116	0.262	0.391
114	20.8	21.6	22.6	21.4	26.6	0.266	0.24	0.213	0.244	0.44
115	28	28.4	29.4	29.2	31.4	0.293	0.244	0.129	0.289	0.449
116	23.4	25.2	22.8	24.2	28.4	0.267	0.258	0.089	0.293	0.413
117	26.2	27	26.6	26.8	30.8	0.267	0.293	0.116	0.298	0.476
118	21.6	24.2	22.2	24.6	23.2	0	0	0	0	0
119	22.4	24.4	23.4	24	24.8	0	0	0	0	0
120	20.8	23.6	23.2	23	27.2	0.192	0.196	0.128	0.252	0.372
121	26	28	26.8	27.2	32.4	0.256	0.224	0.116	0.3	0.476
122	22.8	24.2	22.8	23.2	28.6	0.252	0.256	0.156	0.24	0.44
123	21.8	21.6	22.2	23	24	0	0	0	0	0
124	29.6	29.4	29.8	30	34	0.22	0.216	0.148	0.276	0.44
125	25.2	26.4	26.2	26	29.2	0.184	0.192	0.096	0.216	0.308
126	24.8	27	27.4	26.6	30.8	0.264	0.224	0.112	0.232	0.372
127	32.2	35.4	33.4	34.4	35.2	0.212	0.24	0.052	0.264	0.364
128	25.2	28.8	26.8	25.8	31	0.212	0.228	0.084	0.208	0.352
129	23.8	24.8	24	23.6	30.8	0.216	0.208	0.144	0.208	0.332
130	19.8	22	20.6	20.2	22.6	0	0	0	0	0
131	22.4	24	23.6	23.2	29.2	0.244	0.2	0.131	0.236	0.371
132	21.4	23.6	23.2	22.8	30.6	0.218	0.229	0.127	0.273	0.404
133	22.6	24.2	23.6	23.2	30.6	0.294	0.225	0.12	0.28	0.447
134	24.2	27.2	26.4	26.8	33.2	0.244	0.247	0.193	0.273	0.498
135	23.6	24.8	23.4	23.6	29.4	0.225	0.189	0.084	0.225	0.378
136	23.2	23.8	26.6	24.6	31.6	0.244	0.215	0.153	0.284	0.414
137	23.2	26.8	24.6	26.4	32.2	0.193	0.247	0.134	0.32	0.444
138	24.6	25	24.4	24.4	28.2	0	0	0	0	0
139	29.4	32.2	31	30.6	33	0.225	0.2	0.087	0.247	0.371
140	28.4	30.2	29.2	28.6	35	0.287	0.262	0.113	0.294	0.498
141	26.8	26.6	27.4	25.8	35.2	0.244	0.204	0.106	0.24	0.455
142	37.4	38.4	40.8	40	44.6	0.356	0.309	0.138	0.389	0.487
143	34.6	33.2	34	34.4	41.6	0.262	0.229	0.029	0.295	0.469
144	29.8	30.4	31.8	30.6	38.4	0.236	0.204	0.076	0.269	0.44
145	27.2	29.6	28.2	29.6	30.6	0	0	0	0	0
146	25.4	28.2	27	27	28	0	0	0	0	0
147	30.8	31.8	31	32.2	37	0.21	0.173	0.067	0.243	0.383
148	31.6	35.4	32.8	32.4	41.4	0.223	0.27	0.07	0.233	0.433
149	27.4	29.4	27.8	28	35	0.243	0.2	0.09	0.243	0.407
150	29.4	31.2	31	31.8	32.2	0	0	0	0	0

Table E.9: *The number of scheduled and percentage of parallel jobs for challenging test cases.*

	Available space in the schedule					
test case	CS	GA-I	GA-II	PSO	TS	VNS
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	1.1	1.5	0.5	0.8	1.3	1
5	0.714	0.048	0.43	0.854	0.466	0.048
6	0	0	0	0	0	0
7	0	0	0	0	0	0
8	2.41	3.146	2.994	3.812	2.768	4.672
9	0.7	2.3	1.9	2.6	1.7	1.8
10	0	0	0	0	0	0
11	0	0	0	0	0	0
12	0	0	0	0	0	0
13	0	0	0	0	0	0
14	0	0	0	0	0	0
15	0	0	0	0	0	0
16	4	3.6	1.7	3.4	5.7	6.8
17	3.958	3.272	2.88	4.476	5.254	4.694
18	0.7	0.4	0.7	0.5	0.9	0.3
19	0.6	0.6	0.4	0.8	0.8	0.7
20	0.6	0.4	0.4	0.4	0.3	0.3
21	1.3	0.7	1.3	1.3	0.4	0.8
22	1.5	0.5	1.4	1.3	0.2	0.1
23	0	0	0	0	0	0
24	0	0	0	0	0	0
25	1.1	0.7	1	1.5	1.4	1.6
26	1.5	1.7	1.5	1.6	1.8	1
27	2.1	1.5	2.4	2.4	2	2.5
28	2.2	1.7	1.7	1.6	1.2	0.9
29	1.524	1.314	0.882	1.006	1.664	2.186
30	0	0	0	0	0	0
31	0	0	0	0	0	0
32	0	0	0	0	0	0
33	0	0	0	0	0	0
34	2.568	1.144	1.478	2.854	1.958	1.614
35	1.844	2.058	2.632	2.2	1.814	2.732
36	1.392	1.32	0.61	1.106	1.272	0.57
37	3.914	3.034	2.642	3.908	3.11	3.29
38	1.66	1.89	2.118	2.186	1.926	1.566
39	3.052	2.022	2.236	4.81	2.526	1.596
40	2.136	2.248	1.516	2.276	2.982	2.112
41	2.646	2.062	1.822	1.894	1.944	2.642
42	1.8	2.1	0.8	3.4	1.5	1.4
43	1.7	0.8	0.7	2.1	2.1	1.9
44	2.3	3	2.1	3.2	1.4	2.7
45	2.1	1	0.9	2.1	1.7	1.5
46	1.7	2	1.6	2.4	2.2	3.1
47	2	1.9	1.2	3	1.4	1.6
48	0	0	0	0	0	0
49	0	0	0	0	0	0
50	0	0	0	0	0	0

Table E.10: *The available space in the schedule for elementary test cases.*

	Available space in the schedule					
test case	CS	GA-I	GA-II	PSO	TS	VNS
51	0	0	0	0	0	0
52	0	0	0	0	0	0
53	0	0	0	0	0	0
54	0	0	0	0	0	0
55	5.362	4.088	2.922	3.198	4.784	5.594
56	3.454	4.488	4.29	4.102	3.548	5.232
57	4.274	4.638	3.872	2.548	3.094	4.116
58	4.426	4.622	3.184	3.264	3.752	4.288
59	3.258	1.712	1.968	3.466	2.31	1.964
60	1.538	2.698	3.07	3.544	3.852	5.31
61	6.45	5.2	4.1	4.65	7.5	6.75
62	3.474	4.292	2.752	3.612	6.786	4.382
63	3.95	4.8	4.4	5.65	7.5	7.4
64	3.25	2.75	3.05	5.6	7.25	6.6
65	6.3	3.75	4.4	4.35	6.9	7.2
66	4.3	4.75	2.8	5.3	11.35	8.6
67	4.75	3.65	3.1	6.05	4.95	5.7
68	3	4.1	2.7	4.6	7.2	4.5
69	2.2	2.2	1.3	3.8	3.8	3.5
70	3.6	3.2	2.2	5.8	3.9	5.5
71	5	4.1	2.7	2.8	6.8	3.7
72	4.5	3.3	2.6	5.2	4.9	6.4
73	4.006	3.98	2.37	3.608	5.454	4.118
74	5.196	2.364	3.624	4.352	5.452	4.404
75	5.196	5.396	3.348	2.888	7.348	6.81
76	0	0	0	0	0	
77	8.452	5.524	3.492	9.264	9.98	
78	6.19	4.118	6.524	6.966	6.678	
79	8.15	6.65	5.6	8.7	16.85	
80	13.2	4.05	2.95	8.75	10.3	
81	7.25	7.25	7.25	10.1	9.85	
82	8.25	6.5	5.25	5.95	10.45	
83	9.3	11.8	3.9	10.1	12.25	
84	0	0	0	0	0	
85	9.24	9.76	3.44	11.08	22.52	
86	5.72	6.04	3.8	7.8	10.64	
87	9.24	8.08	3.6	5.4	16.76	
88	7.16	6.44	5.2	7.2	14.72	
89	9.88	7.84	3.36	9.6	20.32	
90	5.52	6.56	8.96	8.72	18	
91	0	0	0	0	0	
92	0	0	0	0	0	
93	6.7	7.5	3.4	7.3	11.7	
94	8.8	6.5	3	5	10.1	
95	6.5	5.9	3.8	4.9	7.8	
96	0	0	0	0	0	
97	5.6	5.6	2.8	5.1	11.9	
98	5.308	6.788	2.27	5.294	8.232	
99	7.886	5.584	3.826	5.708	8.182	
100	6.324	6.626	2.726	6.704	11.514	

Table E.11: *The available space in the schedule for intermediate test cases.*

	Available space in the schedule				
test case	CS	GA-I	GA-II	PSO	TS
101	6.778	6.066	3.252	6.658	9.998
102	6.102	4.392	3.412	5.476	6.27
103	0	0	0	0	0
104	12.2	16.2	5.7	11.9	17.45
105	12.35	10.9	9.65	14.35	17.8
106	14.95	12	6.4	11.6	19.65
107	11.2	13.3	8.7	13.6	20.75
108	12.8	12.32	9.28	13.48	23.36
109	18.32	12.6	9.12	12.92	27.64
110	15.24	8.84	8.32	16.48	30
111	0	0	0	0	0
112	10.76	15.36	1.8	13.64	22.6
113	11.864	10.8	5.4	12.4	19.88
114	16.424	15.632	11.536	12.592	27.496
115	11.64	10.632	5.008	13.856	21.384
116	18.152	11.864	3.04	13.608	18.608
117	12.56	13.72	6.352	15.816	20.52
118	0	0	0	0	0
119	0	0	0	0	0
120	8.8	8.3	4.7	10.6	12.4
121	7.7	7.9	3.1	10.5	11.8
122	8.8	11.2	5.4	6.1	11.9
123	0	0	0	0	0
124	9.1	6.5	3.8	10.5	15.2
125	4.872	7.95	2.996	6.988	11.256
126	10.8	10.528	4.444	9.158	14.518
127	6.126	7.656	1.08	7.32	10.96
128	8.934	11.342	1.744	8.71	11.6
129	9.758	9.626	5.268	7.834	9.76
130	0	0	0	0	0
131	20.2	16.25	7.55	19.5	27.1
132	14.35	15.25	5.2	14.8	16.75
133	17.8	13.45	5.95	16.65	18.65
134	13.5	12.7	13	15.2	22.2
135	16.76	12.48	5.56	12.88	41.64
136	18.28	17.44	12.96	17	35.16
137	14.16	16	4.88	17.32	36.12
138	0	0	0	0	0
139	16.84	12.8	4.76	17.04	27.8
140	15.928	15.384	7.496	19.968	38.808
141	14.408	14.8	8.032	21.368	22.336
142	19.424	13.104	4.784	14.712	21.464
143	15.592	14.96	1.552	16.16	31.904
144	14.912	14.824	4.456	19.088	19.352
145	0	0	0	0	0
146	0	0	0	0	0
147	10	9.2	2.4	11.8	13.8
148	8	6.9	1.3	9.1	15
149	12.8	8.6	4.6	9.7	13.1
150	0	0	0	0	0

Table E.12: *The available space in the schedule for challenging test cases.*

APPENDIX F

Measures of central tendency

Two measures of central tendency were used during the statistical hypothesis testing in §4.5. If there was an overall statistical difference between the profits of the algorithms for a specific test case, the median of the profit from the two best performing algorithms were compared in §4.5.1 while a statistical difference between the means of the profit from the two best performing algorithms were tested in §4.5.3. If there was an overall statistical difference between the profits of all the metaheuristics and a statistical difference between the profits of the two best performing metaheuristics for a specific test case, the best performing metaheuristic was said to outperform all other metaheuristics for that test case. This appendix contains the median, mean and standard deviation for the 150 test cases and all the developed metaheuristics.

F.1 Median of the profit of the schedules

Tables F.1–F.3 contain the median of the profit of the schedule for elementary, intermediate and challenging test cases, respectively.

	Median					
test case	CS	GA-I	GA-II	PSO	TS	VNS
1	114	119	110	108	106	121
2	127	122	119	118	113	133
3	81	81	81	66	81	81
4	235	239	215	236	240	236
5	232	246	214	214	246	242
6	506	581	527	570	474	548
7	290	331	334	331	304	369
8	461	484	478	435	529	469
9	354	354	361	362	397	382
10	585	609	626	645	647	631
11	255	258	258	256	233	261
12	358	371	369	333	360	379
13	193	196	196	182	196	196
14	280	281	278	272	278	285
15	149	149	149	148	149	150
16	823	826	833	861	952	769
17	824	860	895	824	976	759
18	410	425	402	383	425	417
19	319	321	319	304	317	324
20	373	389	355	386	385	391
21	345	364	340	333	361	361
22	296	311	296	297	306	311
23	661	713	714	722	716	767
24	439	468	449	434	470	496
25	619	634	630	609	680	643
26	529	572	535	514	598	573
27	457	511	495	438	517	519
28	543	590	578	526	626	578
29	833	878	885	781	888	864
30	423	491	491	430	417	506
31	444	504	469	477	493	543
32	460	532	509	473	474	542
33	379	403	420	418	434	486
34	967	1 054	1 060	1 002	1 130	1 052
35	780	841	814	765	882	796
36	878	934	898	891	987	924
37	591	644	574	536	665	644
38	816	849	817	802	916	858
39	639	689	614	611	736	712
40	787	780	772	668	843	787
41	1 042	1 146	1 104	1 044	1 224	1 193
42	862	944	868	908	955	919
43	732	787	739	734	812	778
44	916	978	905	940	1 062	1 025
45	733	789	750	734	858	787
46	1 073	1 110	1 037	1 012	1 180	1 118
47	717	816	740	738	886	831
48	1 044	1 121	1 072	1 092	1 141	1 195
49	805	957	899	924	1 021	999
50	699	737	759	779	722	826

Table F.1: *The median of the profit of the schedule (in R) for elementary test cases.*

	Median					
test case	CS	GA-I	GA-II	PSO	TS	VNS
51	848	937	963	931	944	979
52	746	851	801	816	801	920
53	724	866	833	822	848	924
54	566	659	583	572	642	697
55	822	1 009	954	881	1 022	944
56	913	1 085	960	920	1 150	1 082
57	964	1 065	1 061	969	1 194	1 085
58	918	1 061	957	938	1 131	1 044
59	977	1 075	990	959	1 135	1 066
60	954	1 115	1 038	923	1 241	1 124
61	1 554	1 784	1 693	1 599	1 942	1 525
62	1 129	1 214	1 069	1 110	1 321	1 176
63	1 861	1 992	1 967	1 903	2 141	1 689
64	1 633	1 782	1 690	1 720	2 090	1 621
65	1 436	1 466	1 550	1 441	1 743	1 413
66	1 407	1 536	1 597	1 365	1 706	1 456
67	1 589	1 818	1 689	1 612	2 107	1 766
68	1 561	1 687	1 613	1 573	1 851	1 523
69	1 436	1 512	1 437	1 522	1 672	1 418
70	1 222	1 371	1 252	1 304	1 427	1 217
71	1 380	1 527	1 420	1 393	1 736	1 313
72	934	1 064	992	1 043	1 220	1 068
73	1 369	1 532	1 451	1 340	1 706	1 489
74	1 469	1 458	1 562	1 434	1 863	1 581
75	1 461	1 558	1 635	1 560	1 832	1 297
76	1 378	1 476	1 497	1 591	1 453	
77	1 730	1 899	1 903	1 716	2 181	
78	1 376	1 544	1 443	1 459	1 796	
79	1 823	2 058	1 917	1 842	2 183	
80	1 615	1 754	1 559	1 562	2 048	
81	1 761	1 872	1 963	1 853	2 534	
82	2 062	2 173	2 222	2 192	2 846	
83	2 184	2 425	2 301	2 276	2 936	
84	1 835	2 052	2 054	1 993	2 158	
85	2 953	3 105	3 100	2 897	3 628	
86	2 144	2 364	2 511	2 229	2 646	
87	2 326	2 446	2 410	2 270	2 967	
88	2 711	2 865	2 979	2 688	3 270	
89	2 602	2 697	2 559	2 574	3 291	
90	2 726	2 898	2 828	2 654	3 285	
91	1 446	1 509	1 524	1 353	1 568	
92	1 425	1 737	1 663	1 613	1 642	
93	1 792	1 840	1 928	1 880	2 274	
94	1 877	2 012	1 902	1 906	2 347	
95	1 934	2 042	2 028	1 831	2 277	
96	1 602	1 809	1 721	1 663	1 901	
97	2 952	3 155	3 009	2 996	3 379	
98	2 392	2 491	2 559	2 453	2 738	
99	2 398	2 652	2 636	2 721	3 016	
100	2 765	3 006	2 989	3 004	3 215	

Table F.2: *The median of the profit of the schedule (in R) for intermediate test cases.*

	Median				
test case	CS	GA-I	GA-II	PSO	TS
101	2 614	2 751	2 731	2 526	3 435
102	2 387	2 574	2 534	2 349	3 042
103	1 893	2 300	2 094	1 918	2 158
104	2 540	2 894	2 660	2 590	3 383
105	2 611	2 870	2 794	2 657	3 738
106	2 507	2 641	2 775	2 745	3 445
107	2 842	3 060	2 991	2 737	3 938
108	2 363	2 638	2 523	2 425	3 114
109	2 852	3 180	3 285	3 056	3 847
110	2 690	2 811	2 820	2 566	3 450
111	2 633	2 961	2 725	2 862	2 775
112	3 613	3 701	3 626	3 487	4 025
113	3 760	3 856	3 857	3 868	4 721
114	3 364	3 519	3 654	3 329	4 356
115	4 378	4 586	4 656	4 635	5 188
116	3 569	3 979	3 712	3 610	4 717
117	4 442	4 840	4 893	4 902	5 831
118	1 831	2 013	1 888	2 112	2 067
119	2 003	2 200	2 174	2 125	2 270
120	2 024	2 217	2 211	2 248	2 715
121	2 499	2 709	2 514	2 601	3 200
122	2 242	2 491	2 428	2 317	3 025
123	2 318	2 275	2 431	2 501	2 633
124	3 289	3 326	3 438	3 253	3 961
125	3 034	3 235	3 193	3 152	3 659
126	3 020	3 248	3 189	3 214	3 800
127	3 237	3 555	3 423	3 374	3 728
128	3 309	3 721	3 500	3 372	4 297
129	3 180	3 329	3 281	3 147	4 189
130	2 223	2 772	2 371	2 297	2 646
131	3 149	3 354	3 300	3 192	4 475
132	2 991	3 294	3 200	3 158	4 374
133	3 147	3 537	3 389	3 437	4 691
134	3 325	3 660	3 806	3 860	4 703
135	3 454	3 561	3 588	3 380	4 534
136	3 260	3 469	3 765	3 384	4 788
137	3 455	4 061	3 807	3 738	4 724
138	3 366	3 475	3 473	3 463	3 647
139	4 034	4 352	4 515	4 136	5 105
140	4 728	5 111	5 114	4 783	6 286
141	4 557	4 781	4 950	4 337	6 338
142	4 897	5 133	5 370	5 211	6 258
143	5 152	5 230	5 482	5 107	6 831
144	5 137	5 386	5 398	5 216	6 883
145	2 190	2 355	2 425	2 484	2 626
146	2 340	2 744	2 570	2 578	2 671
147	2 960	3 117	3 091	3 119	3 874
148	2 962	3 341	3 187	3 045	4 063
149	2 802	3 100	2 883	2 938	3 816
150	2 968	3 282	3 227	3 325	3 394

Table F.3: *The median of the profit of the schedule (in R) for challenging test cases.*

F.2 Mean and standard deviation of the profit of the schedules

The mean and standard deviation of the profit of the schedule for elementary, intermediate and challenging test cases may be found in Tables F.4–F.6, respectively.

	Mean						Standard deviation					
test case	CS	GA-I	GA-II	PSO	TS	VNS	CS	GA-I	GA-II	PSO	TS	VNS
1	114	117.6	111.2	109.2	109	121.4	2	5.413	5.63	10.281	7.45	0.894
2	127.8	124.6	117.8	118	118.8	131	3.115	5.727	7.95	7.314	10.895	2.739
3	80.8	81	81	71	80.6	81	0.447	0	0	7.349	0.548	0
4	234.4	238.4	221.8	231.6	239.2	237	3.578	2.881	13.554	12.198	3.033	2.828
5	229	245	221.8	221.8	244	243.4	9.487	2.236	14.237	16.115	2.739	2.408
6	504.8	581.2	530	562.2	470.4	535.2	15.912	25.223	20.211	27.914	13.867	38.324
7	294.4	328.2	330	327	310	365	10.407	22.399	10.344	13.874	17.678	9.354
8	460.2	488.6	478.8	432	524	479.6	7.596	16.607	15.482	15.621	16.171	18.743
9	352.8	357.6	358.8	357	396.8	377.6	9.011	15.502	12.853	17.706	4.147	7.436
10	590.4	628	623.2	609.4	638	625.4	15.566	39.198	30.376	59.087	17.088	24.825
11	253.8	257	257.8	252.4	240.6	260.2	4.439	2.828	0.837	6.656	13.795	2.049
12	358.8	363.4	367.6	338.6	362.2	378.6	4.95	15.176	8.385	19.781	6.181	0.548
13	194.2	195.4	194.2	179.2	182.8	196	1.643	1.342	4.025	14.755	19.677	0
14	279.4	280.6	275.4	261.2	277.6	285	2.074	4.506	3.975	20.377	5.857	0
15	149	149.4	149	144.2	149.2	150	0.707	0.548	1.225	6.648	0.045	0
16	807.8	853.4	826.4	856.6	962.8	763.4	27.05	64.057	19.23	73.901	27.95	45.992
17	826	871.6	865.6	810.6	970.2	777.4	22.327	44.072	64.497	47.836	40.202	67.597
18	411.4	424.8	399.8	381.2	425.8	416.6	8.474	0.447	8.786	14.22	6.611	14.467
19	317	320	318	307.4	318.4	320	5.431	4.301	6.325	6.804	4.506	5.523
20	374.4	385.4	356	373.8	386	386.2	8.591	5.413	5.568	24.641	5.788	12.598
21	342.6	356.8	341	330	364.8	354.4	8.143	9.96	8.485	16.062	6.943	12.7
22	297.6	306.6	295.8	299.2	303.4	310	5.505	8.764	3.633	5.541	5.273	2.236
23	657.4	697.2	692.2	674.4	695.8	768.2	21.709	29.895	39.392	72.002	50.987	8.983
24	442.8	468.4	460.4	437.6	461	495.4	7.497	21.836	25.225	37.554	37.047	5.983
25	619	642	630	603.2	675.4	641.2	27.175	23.707	4.743	41.433	14.588	23.963
26	522	581.6	527.8	514.2	596.6	573.8	16.248	22.59	26.499	37.057	14.485	15.418
27	465	512.4	491	443.2	524.6	520.8	33.742	15.077	29.715	16.423	13.221	10.159
28	543.8	578	570	541.8	626	580.8	4.97	41.875	31.804	34.989	6.364	25.173
29	838.6	876.8	869	800.8	900.2	879.2	19.139	14.856	53.418	34.252	27.932	39.373
30	422.8	493.8	480	428.4	421.2	510.6	6.723	14.601	29.309	61.57	16.843	11.171
31	447.2	503	468.4	473	489.8	541.8	9.445	8.888	22.832	20.273	20.018	5.07
32	459.6	532	510	479.2	476.4	541.8	14.792	5.523	12.39	31.356	20.586	3.899
33	374.8	429.8	416.8	412	430.4	483	7.19	36.867	16.634	32.78	13.831	9.11
34	977.8	1 065.6	1 040.8	992.8	1 140	1 049.4	32.676	30.27	38.376	54.366	16.538	31.077
35	786	834.8	803.4	776.4	876.2	809.2	28.697	11.671	22.546	43.466	12.029	24.273
36	874.8	919.6	904.2	892.2	979.4	936.4	14.078	31.182	28.683	47.484	15.915	31.77
37	593.4	636.2	588.6	554.2	665	645.2	7.503	31.108	43.918	64.743	0	12.834
38	818.6	835.2	811	808.6	902.6	861.2	11.149	38.219	33.399	50.915	20.07	10.569
39	632.6	685.6	630	619.2	737	720.6	19.424	30.493	51	44.634	39.019	40.041
40	775.4	783.8	763.4	706.6	844	789.8	26.969	23.931	48.138	68.948	15.716	23.69
41	1 041	1 138.6	1 083.4	1 048.6	1 218	1 162.2	31.241	31.777	37.813	59.618	12.186	54.797
42	860.8	938.2	860	894.6	961.8	908.4	11.3	24.844	37.809	28.893	11.735	23.416
43	729.8	780.6	741.6	736.2	811	787.2	28.42	28.554	21.536	28.986	14.916	37.699
44	907.2	984.2	918.6	929.4	1 063.2	1 015.4	47.442	17.824	32.639	62.123	32.461	46.199
45	731	787.2	749	729.8	851.4	782.6	22.76	31.776	26.805	18.82	19.021	25.442
46	1 048.4	1 090	1 052	1 023.2	1 181.4	1 108.6	51.578	41.719	50.334	75.51	2.608	50.974
47	731.4	799	727.2	752.4	872.6	833.2	24.317	44.939	33.522	44.015	29.577	14.114
48	1 034.2	1 109.6	1 078.4	1 083.2	1 139.6	1 174.6	35.138	62.176	52.767	44.724	40.71	62.308
49	809.6	966	888.6	893.4	979	1 010.4	17.242	40.181	42.235	75.577	67.354	48.962
50	702.2	740.8	758.8	738.2	737.4	826.4	12.578	76.621	24.046	98.746	51.174	42.176

Table F.4: *The mean and standard deviation of the profit of the schedule (in R) for elementary test cases.*

	Mean						Standard deviation					
test case	CS	GA-I	GA-II	PSO	TS	VNS	CS	GA-I	GA-II	PSO	TS	VNS
51	859.4	928.8	947.8	929.4	951	956.2	20.428	46.965	32.299	29.288	22.383	38.842
52	745.8	848.6	805	822.8	816.2	925.2	10.06	33.702	24.156	76.166	58.968	30.012
53	740.2	862.8	834.8	819.4	851.6	894.4	25.975	33.641	40.202	20.84	9.45	44.92
54	563.8	655.8	581	572.4	633	698.8	20.487	16.115	18.762	58.892	30.765	15.255
55	820.6	983.8	935.6	871.8	1 034.6	938.4	25.403	57.295	37.859	49.221	33.381	47.826
56	903	1 072.4	949.6	919.8	1 146.2	1 081	62.837	26.978	52.257	62.022	37.539	49.573
57	962.2	1 074	1 052.8	969.8	1 200.4	1 073.8	8.928	45.596	51.247	72.413	30.956	45.196
58	926.2	1 052.4	969.6	968.8	1 133.2	1 041	42.435	52.305	61.374	76.627	28.057	18.722
59	961.2	1 066.8	993.2	998.6	1 135	1 063.6	36.732	45.494	58.87	66.546	18.987	35.275
60	950.4	1 110.6	1 040	948.4	1 267.4	1 128.8	24.069	33.156	38.177	124.134	57.265	76.077
61	1 556.2	1 774.8	1 701.8	1 615.6	1 959.6	1 522	15.271	69.898	166.035	115.199	52.286	66.813
62	1 106.6	1 199.4	1 058.8	1 124	1 323.2	1 148.6	61.435	41.914	64.368	90.131	39.752	68.948
63	1 895	1 995	1 958.2	1 904.4	2 157.2	1 745.2	90.653	58.664	68.613	26.083	24.035	92.947
64	1 643.8	1 794	1 679.6	1 723	2 075.2	1 610.2	35.857	82.901	112.066	128.221	47.971	94.93
65	1 446.8	1 504.6	1 519	1 444.6	1 743.6	1 405.6	67.968	87.457	47.576	32.192	13.069	124.987
66	1 465.2	1 593	1 588.2	1 350	1 690.2	1 438	104.222	119.327	73.421	91.137	47.473	79.699
67	1 579.8	1 807.2	1 687.2	1 628	2 098	1 809.6	36.793	69.478	97.561	127.413	15.443	95.06
68	1 565.6	1 682.8	1 628.2	1 551.8	1 859.8	1 493.6	58.226	17.584	59.893	88.689	44.24	94.991
69	1 424	1 515.8	1 448.2	1 522	1 660.6	1 435.6	27.065	45.279	52.557	39.768	42.045	42.571
70	1 237.4	1 338.6	1 257.8	1 311.4	1 422.6	1 224.6	40.827	75.956	16.604	49.576	18.298	63.374
71	1 378.2	1 463	1 426.2	1 447.6	1 717.2	1 312	45.091	132.69	64.867	101.256	45.406	68.604
72	941.4	1 073.6	992.4	1 044.6	1 225.4	1 082.6	31.262	67.055	43.004	25.852	26.567	61.703
73	1 387.6	1 541.4	1 467	1 357	1 709	1 465.4	76.549	46.28	78.648	90.868	32.241	61.505
74	1 484.2	1 452	1 616.4	1 448.4	1 886	1 574	47.31	20.579	109.322	85.342	42.065	159.499
75	1 452.2	1 551.8	1 595.4	1 580.6	1 825.6	1 328	28.735	66.515	113.725	71.213	29.117	76.246
76	1 388.4	1 477.2	1 528.4	1 524.4	1 492.4		35.592	168.943	72.023	152.014	133.562	
77	1 746.8	1 889.8	1 836.6	1 710	2 188.2		61.941	66.27	122.592	43.96	83.563	
78	1 377	1 537.6	1 460.6	1 438.2	1 794		28.914	58.389	78.408	88.737	34.11	
79	1 845.6	2 047.4	1 951.2	1 880.6	2 189.8		63.92	84.931	147.681	13.85	50.766	
80	1 611.8	1 731.6	1 573.6	1 570.6	2 060		44.807	94.039	35.218	71.759	72.336	
81	1 791	1 881.4	2 021.6	1 839	2 536.2		93.87	78.29	175.645	97.804	56.113	
82	2 048	2 237.2	2 232.4	2 115.6	2 835.8		58.125	103.253	109.519	195.495	61.325	
83	2 203	2 420.2	2 339.2	2 283.6	2 921.2		61.871	127.174	200.429	62.963	73.734	
84	1 837.8	2 038.6	2 118.4	1 983.6	2 137		19.728	73.361	144.716	276.459	116.248	
85	2 941.8	3 085.6	3 056.6	2 863.6	3 594.8		128.245	120.712	126.045	148.264	102.979	
86	2 180.6	2 364	2 503.8	2 188	2 702		93.225	112.158	114.611	132.558	83.633	
87	2 307.6	2 391	2 406.6	2 361.8	3 004		158.216	190.991	81.322	176.859	87.416	
88	2 725.6	2 868.6	2 941.4	2 696.2	3 328.6		117.294	181.544	74.184	89.469	93.417	
89	2 605.4	2 743.8	2 599.8	2 509	3 266.4		108.643	101.423	147.346	163.825	128.138	
90	2 684.8	2 791.4	2 846.6	2 628.6	3 279.6		204.424	203.457	61.411	169.041	43.907	
91	1 447.8	1 498.8	1 518	1 422.2	1 523.2		24.924	180.392	65.364	176.3	112.842	
92	1 447	1 728.6	1 649.4	1 636.4	1 637.2		54.722	117.953	94.18	156.161	69.93	
93	1 815.6	1 878.8	1 918.6	1 904.6	2 265		73.731	95.602	91.985	168.09	65.689	
94	1 859	2 035.6	1 922.2	1 866.2	2 346.2		74.34	98.989	32.714	116.781	34.91	
95	1 934.4	2 065.8	2 025.2	1 878.2	2 311.6		51.096	96.627	60.093	140.704	86.976	
96	1 619	1 820.6	1 725.4	1 705.4	1 846.6		37.303	37.614	60.719	174.613	127.275	
97	2 996.8	3 126.6	3 039.2	2 957.8	3 362.8		68.831	126.156	135.646	165.912	55.188	
98	2 386	2 502.8	2 529.8	2 442.8	2 781.6		48.513	86.676	61.552	36.307	79.98	
99	2 470.4	2 685.4	2 643.6	2 637.6	2 994		156.721	88.404	91.762	131.135	65.548	
100	2 773	2 988.2	2 970.6	3 016.6	3 223.4		44.738	104.222	127.426	111.476	76.82	

Table F.5: *The mean and standard deviation of the profit of the schedule (in R) for intermediate test cases.*

	Mean					Standard deviation				
test case	CS	GA-I	GA-II	PSO	TS	CS	GA-I	GA-II	PSO	TS
101	2 629.6	2 759.8	2 739.2	2 594.8	3 412.2	100.206	105.604	82.05	152.6	75.649
102	2 373.6	2 596.6	2 522.8	2 453	3 043.8	30.624	85.512	125.35	237.233	58.367
103	1 902	2 198.2	2 133	1 953	2 151.8	33.196	288.573	178.821	106.37	64.438
104	2 530	2 866.4	2 659	2 553.4	3 362	71.179	49.115	100.305	172.056	73.192
105	2 616.8	2 905.4	2 814.8	2 598	3 768.2	94.904	73.691	98.584	137.557	60.313
106	2 531	2 670.8	2 787.4	2 716.8	3 428.6	55.05	183.652	202.249	168.295	36.115
107	2 829.2	3 044.4	3 067.6	2 864.8	3 905.6	88.661	84.63	174.43	284.191	71.322
108	2 360.8	2 630.6	2 666	2 428.8	3 110.6	41.704	105.605	267.197	191.358	91.781
109	2 838.6	3 175.6	3 263.8	3 008	3 847.8	123.89	166.839	246.595	272.809	20.729
110	2 684.4	2 740	2 854	2 564.2	3 500	47.247	207.379	121.649	244.971	179.169
111	2 639.2	2 822.4	2 731.4	2 789	2 786.8	38.648	204.178	94.081	199.533	139.763
112	3 560.6	3 704	3 714	3 537.2	4 058.8	99.432	89.138	191.997	172.446	137.447
113	3 765.4	3 848	3 849.4	3 820.4	4 753.6	41.44	80.7	52.662	122.12	153.495
114	3 369.6	3 511	3 706.8	3 367.6	4 412.2	14.977	136.406	120.342	132.058	104.975
115	4 404.8	4 511.6	4 673.4	4 508.8	5 208	58.862	149.853	159.892	231.407	129.684
116	3 545.4	3 967.8	3 751.6	3 615.2	4 714.4	74.982	99.407	163.949	31.854	106.105
117	4 499	4 811.6	4 983	4 817.8	5 849.4	104.477	236.172	240.607	243.349	137.169
118	1 824.8	2 055.4	1 892.6	2 091.8	2 052.2	33.305	95.762	87.643	54.61	84.206
119	2 019.4	2 253.8	2 234.2	2 144.4	2 257.2	42.794	249.676	114.92	258.845	133.876
120	2 024.8	2 216.8	2 194.4	2 284	2 729	29.66	94.975	77.713	99.431	73.352
121	2 494	2 713.8	2 587	2 602.8	3 207.2	61.131	121.242	160.798	177.314	78.158
122	2 252.2	2 499	2 426.6	2 329	3 010	45.604	79.803	122.938	66.701	64.661
123	2 303.6	2 342	2 435.2	2 477.4	2 635.8	36.964	168.375	28.831	92.235	69.665
124	3 264.4	3 304.2	3 342.8	3 323	3 940.2	91.481	97.158	171.423	155.459	49.676
125	3 021.4	3 262.2	3 227.6	3 165.8	3 654.2	33.968	43.58	96.909	96.147	84.987
126	3 008.6	3 274.2	3 225.4	3 209.2	3 788.6	25.066	83.87	100.451	127.874	33.306
127	3 238.2	3 561.6	3 445.2	3 401.2	3 777.6	50.475	88.35	75.873	78.462	122.786
128	3 320.2	3 735.8	3 523.6	3 395	4 304.6	46.569	188.26	54.939	265.531	74.396
129	3 177.4	3 361.4	3 319.6	3 141.8	4 193.8	51.335	100.997	99.52	196.598	117.698
130	2 215.2	2 622.8	2 396.4	2 358.6	2 648.8	67.659	358.964	120.436	168.735	129.421
131	3 131.8	3 371	3 311.4	3 187	4 435.8	122.616	137.737	128.278	321.839	111.147
132	2 982.2	3 264.6	3 223	3 056.2	4 369.6	121.664	116.68	94.385	232.993	120.976
133	3 140.8	3 525.6	3 391.4	3 321	4 692.6	101.443	82.175	157.719	240.249	64.748
134	3 308.6	3 723.4	3 814.4	3 687.6	4 676.2	56.145	217.981	250.653	342.413	102.111
135	3 410.8	3 571	3 568.6	3 286	4 452.8	142.873	98.089	105.308	367.494	139.917
136	3 260	3 340.2	3 793.2	3 453.8	4 732.6	82.417	390.479	375.948	135.507	183.624
137	3 428.6	3 981.2	3 815.8	3 821	4 712.2	158.151	119.267	112.119	221.874	40.289
138	3 346.2	3 462.8	3 487	3 416.8	3 714	99.833	251.853	138.73	251.589	125.784
139	4 001.2	4 370.4	4 372	4 085.2	5 015.8	125.927	91.081	307.315	212.513	168.864
140	4 722.2	5 119	5 137	4 879.2	6 290.6	107.048	122.199	221.52	400.396	127.911
141	4 542.6	4 781.6	4 918	4 491.4	6 303	74.279	178.78	152.94	286.493	108.593
142	4 923.2	5 073.2	5 411.4	5 283	6 271.2	54.815	161.266	239.517	146.214	35.351
143	5 197.4	5 320.4	5 470.4	5 082.2	6 794	76.787	178.326	198.971	266.084	85.694
144	5 143.2	5 361.4	5 397.2	5 165	6 903.8	20.789	247.909	393.668	115.646	98.309
145	2 200	2 354.2	2 423.4	2 498.6	2 620	48.949	87.953	64.026	80.857	113.813
146	2 335	2 668.2	2 591.6	2 489.6	2 663	40.268	124.791	140.251	195.214	124.634
147	2 940.4	3 145.8	3 071.6	3 103.4	3 868.4	118.69	48.618	118.018	123.046	86.515
148	2 997.4	3 402.8	3 199.4	3 044.8	4 090.6	92.964	133.172	166.148	247.509	109.061
149	2 835.2	3 150.2	2 971.8	2 942.4	3 853.4	67.674	123.265	180.601	233.342	71.273
150	2 972.6	3 176.4	3 191.4	3 326.4	3 367.6	15.582	160.759	63.177	122.816	107.537

Table F.6: *The mean and standard deviation of the profit of the schedule (in R) for challenging test cases.*