Particle swarm optimization and differential evolution for multi-objective multiple machine scheduling

by

Jacomine Grobler

Submitted in partial fulfillment of the requirements for the degree
Master of Engineering (Industrial Engineering)
in the Faculty of Engineering, Built Environment and Information Technology
University of Pretoria, Pretoria

September 2008

Publication data:

Jacomine Grobler. Particle swarm optimization and differential evolution for multi-objective multiple machine scheduling. Master's dissertation, University of Pretoria, Department of Industrial and Systems Engineering, Pretoria, South Africa, September 2008.

Electronic, hyperlinked versions of this dissertation are available online, as Adobe PDF files, at:

`http://cirg.cs.up.ac.za/`

`http://upetd.up.ac.za/UPeTD.htm`

Particle swarm optimization and differential evolution for multi-objective multiple machine scheduling

by

Jacomine Grobler
E-mail: jacomine.grobler@gmail.com

Abstract

Production scheduling is one of the most important issues in the planning and operation of manufacturing systems. Customers increasingly expect to receive the right product at the right price at the right time. Various problems experienced in manufacturing, for example low machine utilization and excessive work-in-process, can be attributed directly to inadequate scheduling.

In this dissertation a production scheduling algorithm is developed for *Optimatix*, a South African-based company specializing in supply chain optimization. To address the complex requirements of the customer, the problem was modeled as a flexible job shop scheduling problem with sequence-dependent set-up times, auxiliary resources and production down time.

The algorithm development process focused on investigating the application of both particle swarm optimization (PSO) and differential evolution (DE) to production scheduling environments characterized by multiple machines and multiple objectives. Alternative problem representations, algorithm variations and multi-objective optimization strategies were evaluated to obtain an algorithm which performs well against both existing rule-based algorithms and an existing complex flexible job shop scheduling solution strategy.

Finally, the generality of the priority-based algorithm was evaluated by applying it to the scheduling of production and maintenance activities at *Centurion Ice Cream and Sweets*. The production environment was modeled as a multi-objective uniform parallel machine shop problem with sequence-dependent set-up times and unavailability intervals.

A self-adaptive modified vector evaluated DE algorithm was developed and compared to classical PSO and DE vector evaluated algorithms. Promising results were obtained with respect to the suitability of the algorithms for solving a range of multi-objective multiple machine scheduling problems.

Keywords: Flexible job shop scheduling problem, evolutionary multi-objective optimization, particle swarm optimization, differential evolution.

Supervisors	**:**	Prof. V. S. S. Yadavalli
		Prof. A. P. Engelbrecht
Department	**:**	Department of Industrial and Systems Engineering
Degree	**:**	Master of Engineering (Industrial Engineering)

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Acknowledgements

"*Alone we can do so little; yet together we can do so much.*"

Helen Keller

There are a number of people and institutions which I would like to acknowledge for their help and support during the completion of this dissertation:

- Prof. Sarma Yadavalli for always believing in me and encouraging me to do better.
- Prof. Andries P. Engelbrecht for his excellent mentorship and financial support.
- Prof. Schalk Kok for the numerous insightful conversations and encouraging words.
- Leon Beetge from *Optimatix* and Charles Meeser from *Centurion Ice Cream* for their data, helpful comments and general support.
- The staff of the Department of Industrial and Systems Engineering for their encouragement and interest in my work.
- All my friends and colleagues in and outside of the Computational Intelligence Research Group for their patience and valuable comments.
- The Automated Scheduling Optimization and Planning research group at the University of Nottingham for hosting me at their university.
- The University of Pretoria for their financial support enabling me to attend various national and international conferences.
- The National Research Foundation for their financial assistance towards this research. Opinions expressed in this dissertation and conclusions arrived at, are those of the author and not necessarily to be attributed to the National Research Foundation.
- My parents and grandmother for their unwavering love and support.
- My Creator for His blessing during the completion of this dissertation.

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

Production scheduling is commonly considered to be one of the most important issues in the planning and operation of manufacturing systems [55]. Many production related problems, including low machine utilization and excessive work in process, can be assigned directly to inadequate scheduling. Addressing these problems through improved scheduling can, on the other hand, have a significant impact on cost reduction, customer satisfaction, profitability, and overall competitive advantage.

In addition, recent customer demand for higher variety products have contributed to an increase in product complexity. Subsequently, the number of parts being produced in job shop environments has dramatically increased during the past decade [55]. This increased complexity further emphasizes the need for improved planning and scheduling.

This need has resulted in numerous research papers being published in the field of production scheduling during the last fifty years. It is interesting to note that many researchers ignore the multi-objective nature of the production environment. Loukil *et al.* [85], however, provide a strong motivation for considering production scheduling problems to be multi-objective. More than one decision maker is often involved in decision making in a manufacturing environment resulting in conflicting objectives. For example, the marketing manager is interested in maximizing customer satisfaction by minimizing expected tardiness, while the production manager is concerned with minimizing makespan and work in progress. To accommodate all stake holders, the availability of a set of feasible solutions representing trade-offs between the various objective functions, can be quite valuable. However, due to the extreme complexity of multiple machine multi-objective scheduling problems, very little research has been done in this field.

A large number of solution strategies have also been developed over the years for solving production scheduling problems. The aim of this dissertation is to investigate the application of particle swarm optimization (PSO) [64] and differential evolution (DE) [131] to multiple machine multi-objective scheduling problems. Since the development of PSO and DE in 1995, these algorithms have shown to be effective in solving a large range of problem types. Many production scheduling applications of PSO and DE also exist. However, these tend to consider simple problems while significant research opportunities exist in applying PSO and DE to more complex scheduling scenarios.

The production scheduling problem faced by *Optimatix*, a South African company specializing in supply chain optimization, can be considered a good example of a complex production scheduling problem. The design and development of a production scheduling algorithm for *Optimatix* provides an excellent opportunity for investigating the application of PSO and DE in a multi-objective multiple machine environment. Alternative problem representations, variations on the classical PSO and DE algorithms, and multi-objective optimization strategies can be easily investigated. All of these aspects, as well as a number of other aspects, are considered in this dissertation.

The first objective of this introductory chapter was to provide a rationale for the development of real world production scheduling algorithms. The objectives and contributions of this dissertation are further highlighted in Sections 1.1 and 1.2 before a brief outline of the rest of this dissertation is provided in Section 1.3.

1.1 Objectives

To investigate the application of PSO and DE to multiple machine multi-objective environments, the following sub-objectives have been defined:

- To contextualize the *Optimatix* problem with respect to existing literature to enable the formulation of a realistic yet tractable model of the production environment.

- To develop PSO and DE-based scheduling algorithms capable of addressing the *Optimatix* scheduling requirements.

- To investigate the impact of alternative problem representations on the performance of the developed algorithms.
- To investigate the impact of parameter values and alternative PSO topologies and DE variations on algorithm performance.
- To benchmark the developed algorithms against existing solution strategies.
- To compare alternative multi-objective optimization approaches.
- To investigate the generality of the developed algorithms by applying the algorithms to a different, yet related, multi-objective machine scheduling problem.

1.2 Contributions

The main contributions of this dissertation can be summarized in a similar fashion. These contributions include:

- The first attempt at solving the multi-objective flexible job shop scheduling problem with sequence-dependent set-up times, auxiliary resources and production down times.
- The first attempt at solving the multi-objective uniform parallel machine shop problem with sequence-dependent set-up times and unavailability intervals.
- The first application of a vector evaluated Von Neumann guaranteed convergence PSO (GCPSO) algorithm in the scheduling domain.
- The first application of particle swarm optimization and differential evolution to a flexible job shop scheduling problem with additional constraints.
- The development of priority-based vector evaluated PSO and DE algorithms suitable for solving a range of multiple machine multi-objective scheduling problems.
- The development and application of an adaptive vector evaluated DE algorithm.

1.3 Dissertation Outline

Chapter 2 contextualizes the *Optimatix* problem with respect to existing literature. This is achieved by means of Graham *et al.*'s three-field scheduling notation [51]. This analysis leads to the problem being modeled as a multi-objective flexible job shop scheduling problem with sequence-dependent set-up times, auxiliary resources, and production down time.

An overview and analysis of existing solution strategies which have already been used effectively to solve job shop scheduling problems is provided in Chapter 3. PSO and DE are identified as solution strategies of choice and a detailed introduction of these two paradigms is provided.

Chapter 4 documents the algorithm development process associated with the single objective *Optimatix* algorithm. Alternative problem representations, PSO topologies, and DE based vector selection strategies are investigated. An in depth parameter derivation study is also performed and the priority-based algorithms are benchmarked against existing solution strategies.

The multi-objective nature of the *Optimatix* problem is further investigated in Chapter 5. Alternative multi-objective optimization strategies are investigated before an additional investigation into alternative information exchange strategies in the context of Parsopolous *et al.*'s vector evaluated algorithms [108, 109] is conducted.

The purpose of Chapter 6 is to investigate the performance of the multi-objective priority-based algorithms when applied to a different, yet related, scheduling environment. The *Centurion Ice Cream and Sweets* scheduling problem is modeled as a multi-objective uniform parallel machine shop problem with sequence-dependent set-up times and unavailability intervals.

Chapter 7 concludes the dissertation with a summary of the major findings and future research opportunities identified during the completion of this study. Finally, the definitions of all symbols and acronyms used, as well as publications derived from this dissertation, is described in the three appendices.

Chapter 2

Literature review

Production scheduling has been fascinating researchers since the 1950s [57]. Since then a large number of scheduling models and algorithms have been developed, capable of addressing a wide range of customer requirements. Various mechanisms exist to contextualize complex scheduling problems with respect to existing literature. Problem classification is an important prerequisite to the selection of a suitable solution strategy since information regarding problem complexity and existing algorithms provide useful points of departure for new algorithm development [136]. One of the most well known scheduling classification mechanisms is Graham *et al.*'s three field scheduling notation, $(\alpha/\beta/\gamma)$ [51], which classifies models according to the flow pattern and number of machines (α), the constraints placed on the jobs (β), and the scheduling criteria (γ). The rest of this chapter describes each of these components in more detail with the aim of identifying possible problem characteristics which is useful in modeling and solving the *Optimatix* problem which is discussed in more detail in Section 2.4.

2.1 Classification according to flow pattern and number of machines (α)

The first field of Graham *et. al*'s three field notation [51] is mostly concerned with the configuration of primary resources (usually machines) and the flow of jobs on the production floor. Recently, Zandieh *et al.* [154] have documented a classification for a number of common scheduling problem classes. Since all models differ with respect to the associated resource environments and follow the same convention as Graham *et al.*'s notation, this taxonomy may be considered as a further breakdown of the α component. As can be seen from Figure 2.1, the models range from more generic formulations, for example the job shop scheduling problem with duplicate machines, to more specific formulations, i.e. the single machine shop problem. The various models are primarily classified according to:

- the characteristics of the routings of each of the jobs,
- the number of operations of each of the jobs,
- and the number of resources available to perform the required operations.

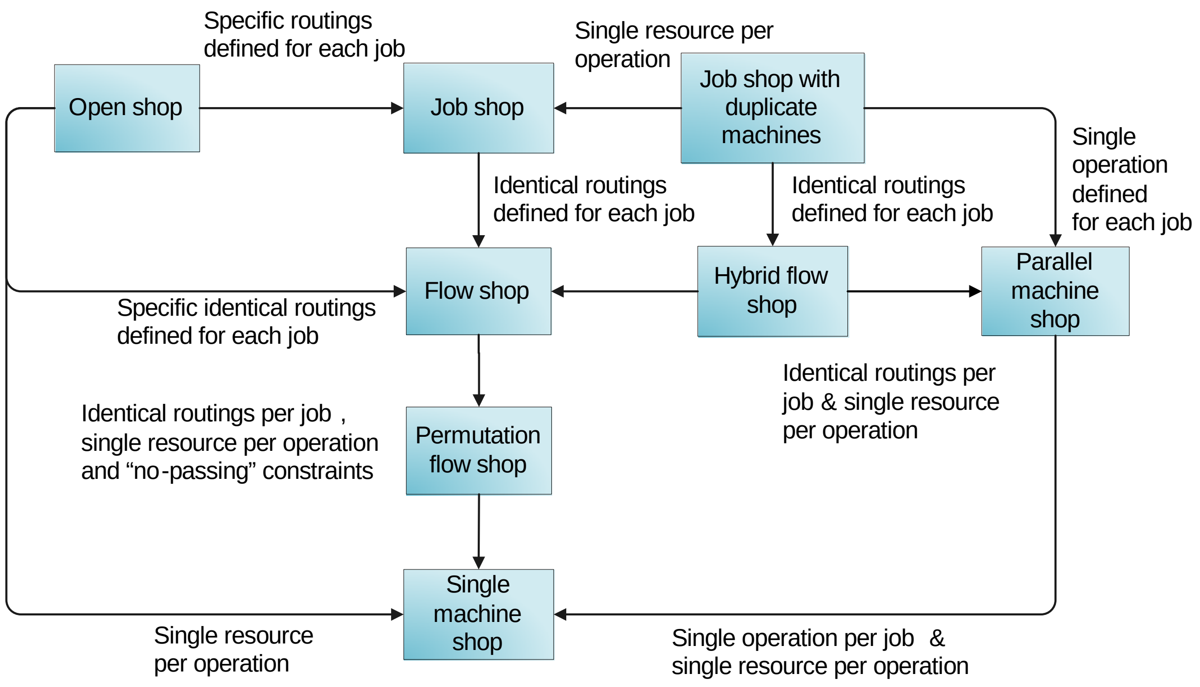

Specific routings defined for each job
Single resource per operation
Open shop
Job shop
Job shop with duplicate machines
Single operation defined for each job
Identical routings defined for each job
Identical routings defined for each job
Flow shop
Hybrid flow shop
Parallel machine shop
Specific identical routings defined for each job
Identical routings per job & single resource per operation
Identical routings per job , single resource per operation and "no-passing" constraints
Permutation flow shop
Single machine shop
Single resource per operation
Single operation per job & single resource per operation

Figure 2.1: Classification of scheduling systems based on resource environments [154]

The most well known scheduling model in the classification is the classical job shop scheduling problem (JSSP). Due to its intractability, the JSSP has been used extensively to test the performance of a wide range of solution strategies, ranging from neural networks to mixed integer linear programming. The purpose of this section is not to provide a detailed review of job shop scheduling. For that the reviews of Blazewicz *et al.* [18] and Jain and Meeran [57] should be consulted. However, due to the importance of the problem, a brief definition is provided nonetheless.

Jain and Meeran [57] describe the problem as consisting of a finite set, $\boldsymbol{J}$, of n_j jobs, i.e. $\boldsymbol{J} = \{J_v\}_{v=1}^{n_j}$ to be processed on a finite set, $\boldsymbol{M}$, of n_m machines, $\{M_l\}_{l=1}^{n_m}$. Each job J_v must be processed on every machine and consists of a chain of n_v operations, $o_{v1}, o_{v2}, \ldots, o_{vn_v}$, that have to be scheduled in a predetermined sequence. There are n operations in total, where $n = \sum_{v=1}^{n_j} n_v$. o_{vl} is the operation of job J_v which has to be processed on machine M_l for an uninterrupted processing time p_{vl}. No operation may be interrupted. Each job has its own independent and individual flow pattern through the machines. Each machine can process only one job at a time and each job can be processed by only one machine at a time.

Upon closer inspection of Zandieh *et al.'s* classification [154], all of the models can, in fact, be described as either generalizations of, or specific instances of the classical job shop scheduling problem. This fact becomes evident when the flow patterns supported by each of the models is compared.

The single machine scheduling problem (SMP), which is indicated in Figure 2.2, is merely a job shop scheduling problem (JSSP) with only one operation per job. The flow shop scheduling problem (FSSP), which is indicated in Figure 2.3, is restricted to specific applications where identical routings are defined for each of the jobs. Thus the problem consists of finding the order in which jobs should be processed on each resource. Closely related to the FSSP, the permutation flow shop scheduling problem (PFSSP) boasts, in addition to identical routings, the addition of a “no passing” constraint. This constraint ensures that the same job sequence is followed for each of the resources.

In contrast, the parallel machine shop problem is limited to jobs consisting of only one operation, which can be performed on any one of a number of resources. Yet another close relation of the FSSP, the hybrid flow shop scheduling problem allows for identical routings along with multiple resources per operation.

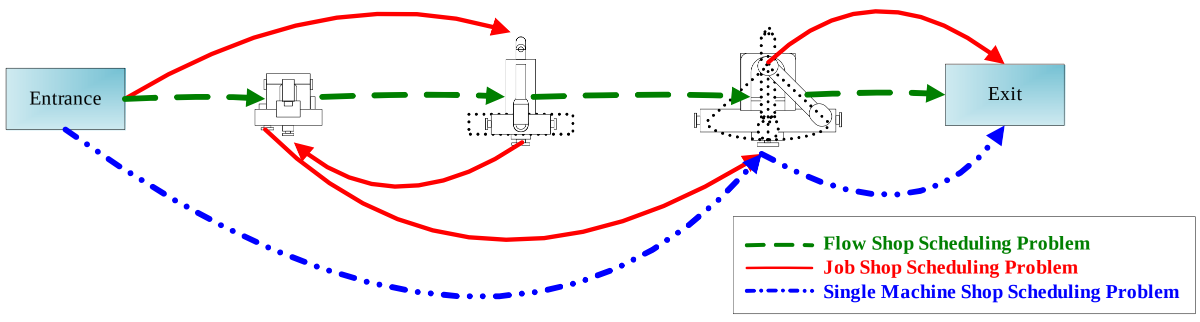

Entrance
Exit
Flow Shop Scheduling Problem
Job Shop Scheduling Problem
Single Machine Shop Scheduling Problem

Figure 2.2: Flow patterns of "single resource per operation" models

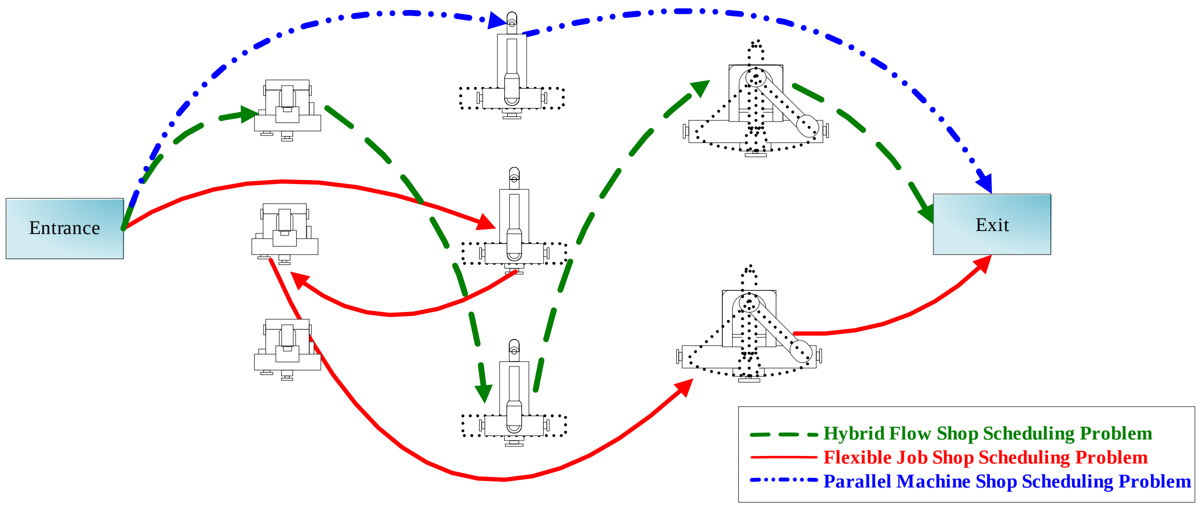

Entrance
Exit
Hybrid Flow Shop Scheduling Problem
Flexible Job Shop Scheduling Problem
Parallel Machine Shop Scheduling Problem

Figure 2.3: Flow patterns of "multiple resources per operation" models

The job shop scheduling problem with duplicate machines can be considered a generalization of the classical JSSP, the only addition being the allowance of more than one resource per operation. This problem is also known as the flexible job shop scheduling problem (FJSP), Kacem *et al.* [62] provide a more formal definition. Consider a finite set, $\mathbf{J}$, of n_j jobs, $\{J_v\}_{v=1}^{n_j}$, to be processed on a finite set, $\mathbf{M}$, of n_m machines, $\{M_l\}_{l=1}^{n_m}$. Each job represents a number, n_v, of nonpreemptable ordered operations $o_{v1}, o_{v2}, \ldots, o_{vn_v}$. The execution of each operation of a job requires one resource or machine selected from a set of available machines, $\boldsymbol{Q}_k \subseteq \mathbf{M}$, where k is a unique operation index. The assignment of the operation k to the machine M_l, where $M_l \in \boldsymbol{Q}_k$, entails the occupation of this machine during a processing time p_{kl}.

The uniqueness of the FJSP results in making provision for the inclusion of additional primary resources such that each operation does not have to be performed on a machine specifically dedicated to it. A selection can be made from any of the available resources belonging to a predefined set of resources. It is important to note, however, that each operation may still only be assigned one resource from the set. The option of alternative resources ensures that the FJSP is useful for scheduling in a wider range of production systems including flexible manufacturing systems and parallel machine shops [35].

Finally, for most of the problem types in Figure 2.1, operations are performed according to a predefined sequence which is derived from the product routing. However, one exception exists in the form of the open shop problem. This formulation results from a relaxation of the operation sequencing constraint. In other words, no specific sequence is specified in which the operations of each of the jobs should be performed.

2.2 Classification according to job constraints (β)

Brucker [21] and T'Kindt and Billaut [136] have done significant work in identifying various types of constraints on the job characteristics, which, when included, may significantly affect the realism of the scheduling model. A number of job constraints listed in Table 2.1, and elsewhere, have already been used in a job shop environment. The purpose of Figure 2.4 is to provide the reader with a glimpse into some of the most common job constraint variations of the classical JSSP. For each variation, just two or three of the solution strategies which have already been effectively employed to solve the specific problem, are mentioned. The variations are further organized into four groups depending on the implications of the extension.

Variations which affect the length of the processing time of operations include the preemptive JSSP, where operations may be interrupted and continued at a later stage [3, 69, 152] and the JSSP with sequence-dependent set-up times [32, 50, 72, 159]. The JSSP with controllable processing times allow the assignment of different processing times to operations, where a pre-determined cost is associated with each process time-reduction alternative [58]. Processing times in the JSSP with batching is determined by the number of similar products which are produced simultaneously as a single operation [111, 112,

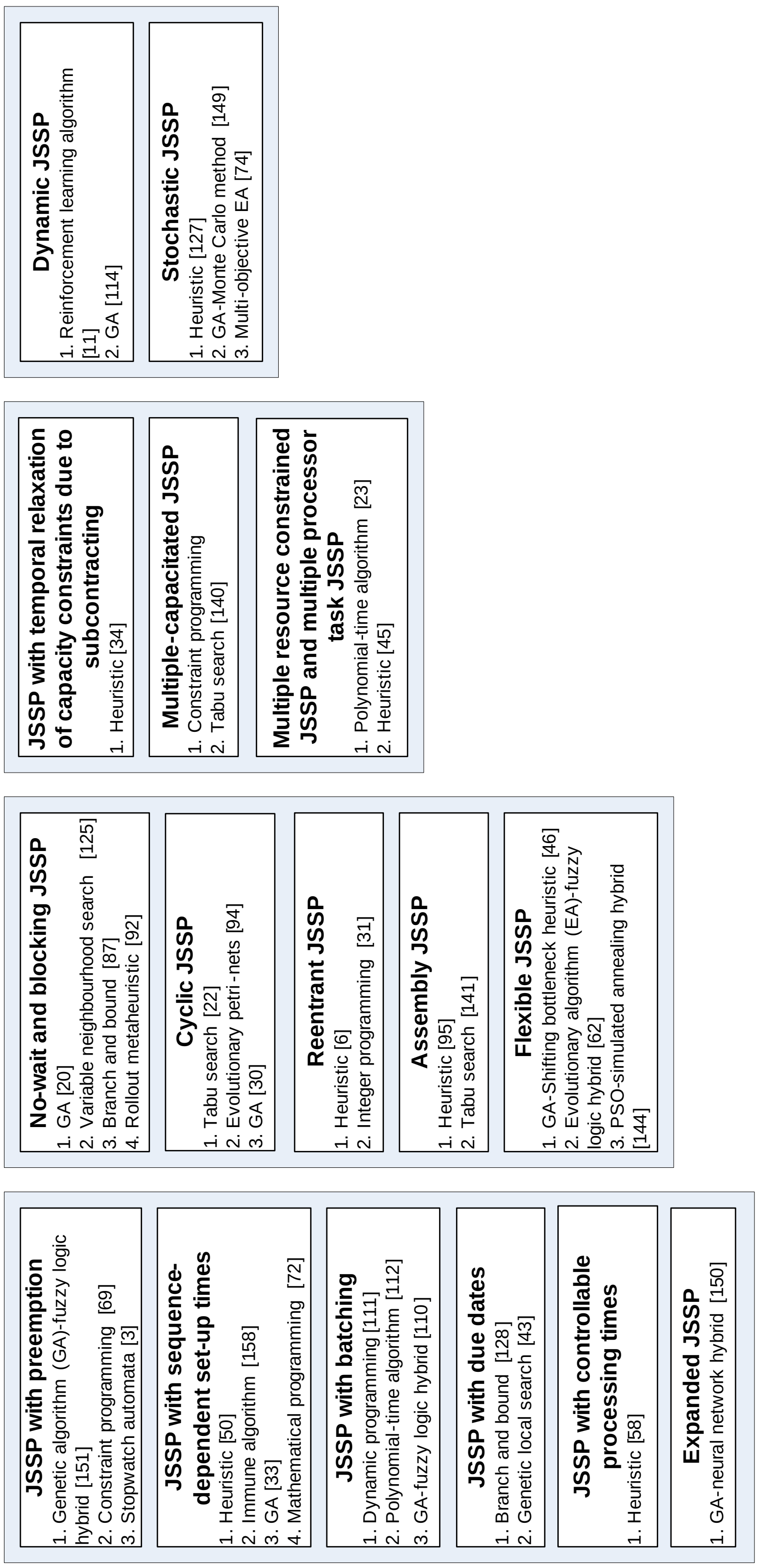

JSSP with preemption
1. Genetic algorithm (GA)-fuzzy logic hybrid [151]
2. Constraint programming [69]
3. Stopwatch automata [3]
JSSP with sequence-dependent set-up times
1. Heuristic [50]
2. Immune algorithm [158]
3. GA [33]
4. Mathematical programming [72]
JSSP with batching
1. Dynamic programming [111]
2. Polynomial-time algorithm [112]
3. GA-fuzzy logic hybrid [110]
JSSP with due dates
1. Branch and bound [128]
2. Genetic local search [43]
JSSP with controllable processing times
1. Heuristic [58]
Expanded JSSP
1. GA-neural network hybrid [150]
No-wait and blocking JSSP
1. GA [20]
2. Variable neighbourhood search [125]
3. Branch and bound [87]
4. Rollout metaheuristic [92]
Cyclic JSSP
1. Tabu search [22]
2. Evolutionary petri-nets [94]
3. GA [30]
Reentrant JSSP
1. Heuristic [6]
2. Integer programming [31]
Assembly JSSP
1. Heuristic [95]
2. Tabu search [141]
Flexible JSSP
1. GA-Shifting bottleneck heuristic [46]
2. Evolutionary algorithm (EA)-fuzzy logic hybrid [62]
3. PSO-simulated annealing hybrid [144]
JSSP with temporal relaxation of capacity constraints due to subcontracting
1. Heuristic [34]
Multiple-capacitated JSSP
1. Constraint programming
2. Tabu search [140]
Multiple resource constrained JSSP and multiple processor task JSSP
1. Polynomial-time algorithm [23]
2. Heuristic [45]
Dynamic JSSP
1. Reinforcement learning algorithm [11]
2. GA [114]
Stochastic JSSP
1. Heuristic [127]
2. GA-Monte Carlo method [149]
3. Multi-objective EA [74]

Figure 2.4: A classification of common variations on the classical JSSP

Table 2.1: Popular job constraints [136]

Constraint	Explanation
pmtn	Preemption is authorized, i.e. jobs may be interrupted
split	Jobs may be split for simultaneous processing
prec	Operations are connected by precedence relations
batch	Operations are grouped into predefined batches
no-wait	No waiting time is allowed between operations
prmu	Operations are processed on all machines in the same order
$d_v = d$	All due dates are identical
$p_v = p$	All processing times are equal
s_{nsd}	Sequence-independent resource set-up times are significant
r_{nsd}	Sequence-independent resource removal times are significant
s_{sd}	Sequence-dependent resource set-up times are significant
r_{sd}	Sequence-dependent resource removal times are significant
a_{k_1,k_2}	A minimum time lag is required between operations k_1 and k_2
blcg	Parallel operations must be completed simultaneously
block	The inventory storage area between machines is limited
recrc	A job may be processed several times on the same machine
unavail$_p$	Significant resource unavailability intervals are present

113]. Minimizing a function of job due dates, as in the JSSP with due dates, indirectly places restrictions on the processing times of jobs [42, 129]. Operation starting and finishing times in an expanded JSSP are restricted by release dates, due dates, and technological enabling constraints [151, 158].

The second group of variations affect the flow of jobs on the shop floor. The no-wait and blocking JSSP consists of a JSSP where at least two operations are constrained such that the second operation has to start immediately upon completion of the first operation [20, 87, 92, 126]. The reentrant JSSP allows the processing of two operations from the same job on the same machine [6, 30], while the cyclic job shop consists of operations which are repeated in a cyclic fashion [22, 29, 94]. When two or more jobs need to be completed before a third may be scheduled, the system can be considered an example of an assembly JSSP [95, 142].

In terms of changes made to the resource requirements of operations, the multiple-capacitated JSSP [100, 141] and JSSP with temporal relaxation of capacity constraints due to subcontracting [33], allow the processing of more than one operation at a time on a single resource. The multiple resource constrained JSSP or multiple processor task JSSP, in turn, requires that an operation be processed simultaneously on two or more resources [23, 44]. The FJSP [45, 63, 145] also belongs to this group.

The dynamic JSSP and stochastic JSSP belong to the final group since both of these variations allow the scheduler to take into account uncertainty in the scheduling process. The dynamic JSSP incorporates uncertainty with respect to the number of jobs and the release dates associated with the jobs which are to be scheduled [11, 115], while the stochastic JSSP focuses on incorporating uncertainty into the process time estimates [74, 128, 150].

Three of the constraints described in this section justify more in depth explanations due to their relevance to the problem considered in this dissertation. These constraints include the addition of sequence-dependent set-up times, availability intervals, and auxiliary resources.

Sequence-dependent set-up times

The inclusion of set-up times is one of the most frequent additional complications in scheduling, and incorporation of set-up times into traditional scheduling models has already been attempted as early as the 1970s [83]. Yang and Liao [149] define set-up times as the times of the tasks which need to be performed before a job can be processed immediately after another job on the same resource. This additional complexity is particularly useful in modeling situations where cleaning operations and tool changes play an important role in production. A typical example includes the manufacturing of different colours of paint.

In the most complex case, sequence-dependent set-up times, where the set-up time depends on the job previously scheduled, as well as the machine on which the current operation is performed, a $n \times n$ (operation $\times$ operation) matrix of set-up time data is required for each resource d_k. The set-up time of operation k is defined by $S_{\kappa_k k d_k}$ where

κ_k is the index of the previous operation scheduled on machine d_k. If the set-up times are machine independent, as is fortunately the case for the *Optimatix* problem, $S_{\kappa_k k d_k}$ can be simplified to $S_{\kappa_k k}$ [21].

A large number of solution strategies, ranging from genetic algorithms to integer programming, have already been applied to production scheduling problems with sequence-dependent set-up times. Detailed reviews can be found in Allahverdi *et al.* [7, 8] and Yang and Liao [149].

Availability intervals

The advent of production calendars, holidays, preventative maintenance and unexpected breakdowns have a significant impact on machine availability and subsequently the scheduling of production resources. If no pre-emption is allowed, the inclusion of machine availability constraints results in the planning horizon being divided into a number of disconnected time windows [10].

Lee [70] differentiates between three types of unavailability intervals, namely resumable, non-resumable, and semiresumable unavailable intervals. When the unavailability intervals are resumable all interrupted operations may continue when the resource is available again without the incurrence of any time penalties. When the unavailability intervals are nonresumable, all interrupted operations need to be reprocessed from the start. Finally, in the semiresumable case, either additional work needs to be performed proportional to the finished part of the operation, or an additional set-up is required. Careful consideration resulted in resumable nonavailability intervals being identified as most appropriate to the *Optimatix* environment [53].

With respect to the available solution strategies, Lee [70] refers to a number of applications in the one machine, parallel machine, and flow shop environments. White and Rogers [143] address a job shop scheduling problem with unavailability intervals by means of a disjunctive graph formulation. Scheduled maintenance is regarded as an operation and is assigned a processing time corresponding to the required maintenance time.

Auxiliary resources

In a complex manufacturing environment, it can happen that the scheduling of operations are constrained by more than one resource type. In addition to machine availability, tooling and labour requirements also play a vital role in the efficient generation of realistic schedules. Studies performed by Mason [88] indicate that 16% of scheduled production cannot be met because tooling is typically not available. Additionally, 40% to 80% of a foreman's time is spent looking for and expediting materials and tools. Therefore incorporating the assignment of workers and tools into the schedule can have a significant effect on production performance.

Gargeya and Deane [46] describe the multiple resource constrained job shop scheduling problem: a job shop in which two or more resource types constrain output. This formulation allows the simultaneous scheduling of machines, labour, and other auxiliary resources, such as tools and jigs. If only two resource types are involved in the scheduling scenario, the problem is referred to as a dual resource constrained JSSP. The two most common dual resource problems are the labour constrained job shop (for scheduling workers and machines) and the auxiliary resource constrained job shop (for scheduling tools and machines).

Brucker [21] defines a class of problems that is very similar to the multiple resource constrained JSSP: the multiprocessor task job shop scheduling problem. This problem requires that a set of resources is linked to each operation. During a processing period $p_{k\boldsymbol{Q}_k}$, each operation k requires all resources belonging to the set $\boldsymbol{Q}_k \subseteq \boldsymbol{M}$. Two tasks which require the same resource cannot be processed simultaneously and are referred to as incompatible tasks.

Relatively little information is available with respect to solution strategies. Exceptions include Brucker [21], who converts a JSSP with multiprocessor tasks, unit processing times, release dates, and precedence constraints between jobs into a shortest path problem and Patel *et al.* [110] who follow a genetic algorithm approach for solving a dual resource constrained scheduling problem.

2.3 Classification according to objective function (γ)

A large number of standard objective functions have already been used to evaluate schedule quality. The final field of Graham *et al.*'s notation [51] focuses on this aspect of schedule optimization. There are obviously an infinite number of variations of cost-related objective functions which may be used. However, accurate cost information is not always available in the *Optimatix* environment and thus the more standardized objective functions were considered to be more suitable.

Brucker [21] provides a list of the most common measurements which can be used for objective function formulation. Any of the five measures in Table 2.2 can be used to formulate at least four different objective functions of the form: $\max\{q_v \mid v = 1\ldots, n_j\}$, $\sum_{v=1}^{n_j} q_v$, $\sum_{v=1}^{n_j} w_v q_v$ and $\max\{w_v q_v \mid v = 1\ldots, n_j\}$, where q_v denotes the measurement associated with job v, w_v and d_v denote the weight and the due date of job v, and n_j is the total number of jobs to be scheduled. For example, the following four objective functions can be formulated for job completion time, where C_v denotes the completion time of job v: makespan ($\max\{C_v \mid v = 1\ldots, n_j\}$), total flow time ($\sum_{v=1}^{n_j} C_v$) and weighted total flow time ($\sum_{v=1}^{n_j} w_v C_v$).

Table 2.2: Commonly used job shop scheduling measurements

JSSP measurement	Formulation
Lateness	$L_v = C_v - d_v$
Earliness	$E_v = \max\{0, d_v - C_v\}$
Tardiness	$T_v = \max\{0, C_v - d_v\}$
Absolute deviation	$D_{1v} = \mid C_v - d_v \mid$
Squared deviation	$D_{2v} = (C_v - d_v)^2$

The description of the this final field of Graham's three field notation concludes the analysis of existing job shop based scheduling literature. The next section will introduce the *Optimatix* problem in more detail before attempting to contextualize the problem within existing literature.

2.4 The *Optimatix* problem

Optimatix is a South African company which can be best described as a best-of-breed software vendor specializing in supply chain optimization. The focus is on providing clients with customized demand and supply planning solutions for addressing various strategic, tactical and operational issues which may arise during the day-to-day running of a business.

The software consists of a number of modules. This dissertation focuses on the scheduling module: *Tactix Scheduling*. Classified as a forwards finite capacity production scheduling tool, *Optimatix* boasts a number of successful implementations in low-volume-high-variety manufacturing environments.

Currently, the underlying algorithms of this module consists of rule-based heuristic methods. Priority rules are the most frequent heuristic method applied to job shop scheduling problems, due to their ease of implementation and low time complexity [18]. However, recent customer requirements have resulted in the investigation of more sophisticated solution techniques for the *Optimatix* problem. Opportunities for improvement exist in terms of solution quality and model formulation. Furthermore, management is specifically interested in the potential use of metaheuristics to obtain improved schedules. Any improvements made should retain the functionality of the existing production scheduling algorithms, but a compromise may be made with respect to the time required to obtain a solution.

The *Optimatix* scheduling requirements can be described most effectively by means of the generic example indicated in Figure 2.5. Here two parts, namely subassembly A and subassembly B, have to be manufactured and assembled to produce a specific product. If each subassembly is defined as a different job, each job consists of a number of operations which denote the manufacturing processes through which each job has to be routed. Each operation can be performed on any machine from a set of primary resources. Tools and labour may be required and can be selected from a set of auxiliary resources. The processing time of an operation includes sequence-dependent set-up times and is dependent on the resource on which it is produced. Scheduled maintenance, machine breakdowns and production calendars also need to be incorporated into the schedule.

Additional requirements in terms of problem size and algorithm generality also exist. The average number of operations and the maximum number of auxiliary and primary resources which need to be considered have, respectively, been estimated at 100 and 216. Furthermore, definite variations exists between the various production environments serviced by *Optimatix*. Since it would be beneficial to reduce the development time associated with adapting the existing algorithms to each new client environment, algorithm generality is an additional important requirement.

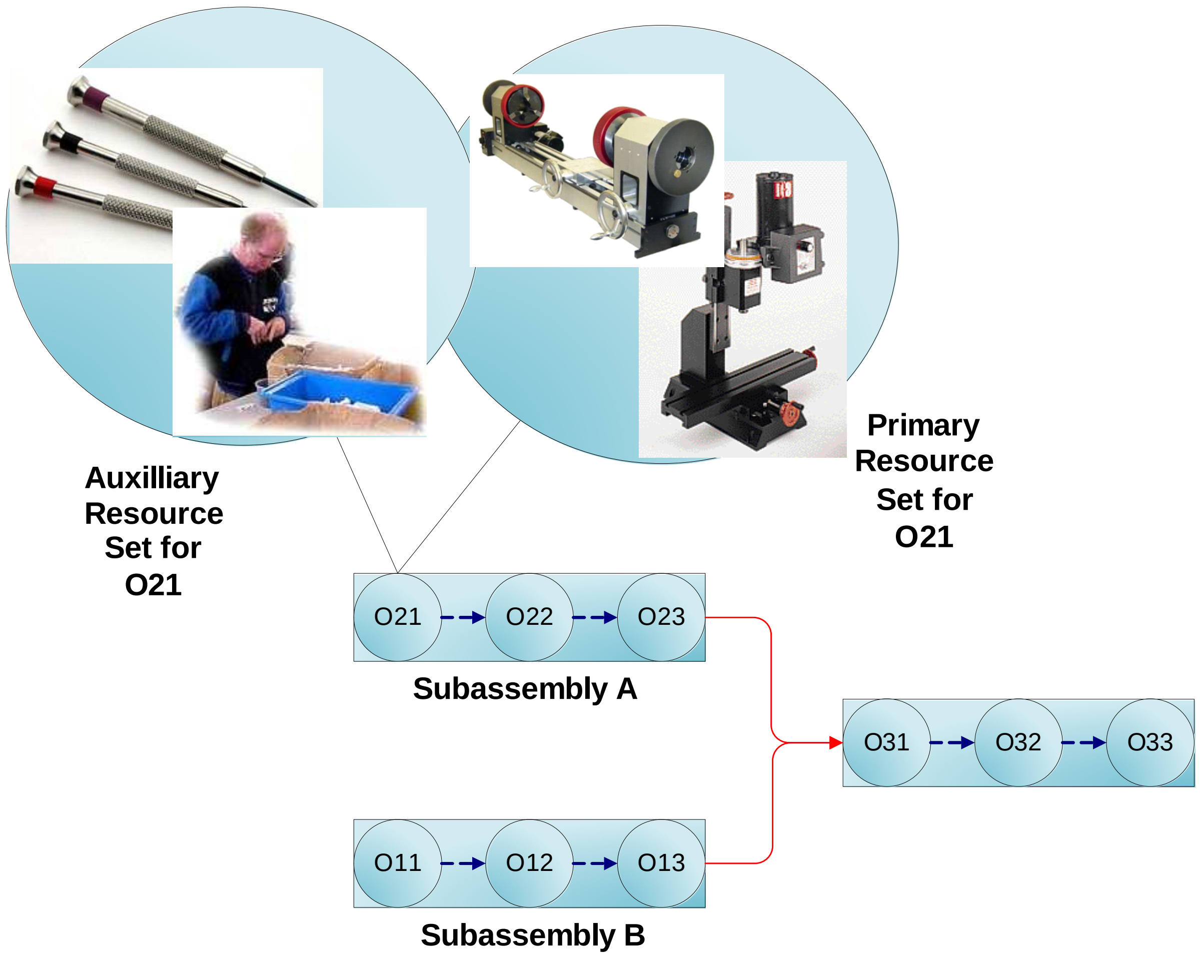


Figure 2.5: A schematic representation of the *Optimatix* problem

An analysis of the business requirements of *Optimatix* and a consideration of existing literature, resulted in the job shop with duplicate machines problem, the parallel machine scheduling problem and the single machine scheduling problem identified as suitable points of departure. However, by solving a problem belonging to the class of job shop scheduling problems, all three of the identified problem instances can be addressed by means of judicial selection of the input parameters. The *Optimatix* problem can thus be classified as a job shop with duplicate machines. More specifically, the problem may be modeled as a **flexible job shop scheduling problem with sequence-dependent set-up times, auxiliary resources, and unavailability in-**

tervals. Minimum makespan, earliness/tardiness, and queue time were defined to be the most important objective functions. To address the multi-objective nature of the client schedule environments, all three of these functions need to be minimized simultaneously.

Combining a number of JSSP variations to meet specific scheduling requirements, as is required for the *Optimatix* problem, is fortunately not new. In fact, as scheduling models have become more and more complex, this practice is quite common. Hoitomt *et al.* [55] solve a JSSP with a number of additional constraints by means of an augmented Lagrangian formulation. Bertel and Billaut [15] develops both a greedy algorithm and a genetic algorithm for a hybrid flow shop scheduling problem with re-entrance and release dates. Hwang and Sun [56] uses a dynamic programming formulation for a re-entrant JSSP with sequence-dependent set-up times. Numerous examples of complex job shop scheduling problems also exist in the semi-conductor manufacturing industry [89, 90].

However, incorporation of auxiliary resources along with a relatively large number of additional constraints and problem features, as is the case with the proposed problem, is not commonly found in literature. One notable exception is the work done by Norman and Bean [99] in the application of a random keys genetic algorithm to a complex production problem, which is in many respects similar to the problem faced by *Optimatix*. Multiple machines, ready times, sequence-dependent set-up times, machine down time, and scarce tools are addressed.

It is important to note that most of the existing algorithms mentioned in Figure 2.4 specializes in solving only the specific variation for which it was developed. Based on this observation, this dissertation will follow a similar strategy and the algorithms developed should subsequently be considered as specialized algorithms for the FJSP with sequence-dependent set-up times, auxiliary resources, and machine down time.

2.5 Summary

The most important contribution of this chapter lies in the identification of a job shop-based scheduling model which adequately addresses the unique business requirements of *Optimatix*. Now that the problem has been suitably described, the next step is to select an appropriate solution strategy. Numerous research papers propose solution strategies ranging from complex metaheuristic implementations to simple rule-based approaches. Every solution strategy has its merits — the challenge is to find the best one for the purpose at hand.

Chapter 3

Selecting an appropriate solution strategy

The design of a suitable algorithm requires the identification of the most appropriate solution strategy for the given scheduling problem. This chapter provides an overview of existing solution strategies which have already been employed effectively in a job shop environment. Based on this information a suitable strategy is selected for the *Optimatix* environment.

3.1 An overview of existing solution strategies

Figure 3.1 shows that a large number of solution strategies have already been applied to the classical job shop scheduling problem over the last 50 years. A more generic classification of the various search methodologies is, however, more useful for the purposes of this dissertation. Feoktistov [43] differentiates between optimal solution strategies and approximate methods (refer to Figure 3.2). The approximate methods are again categorized into heuristics and metaheuristics. This section is aimed at providing a brief introduction into each of these strategies and describing a number of popular examples of each strategy. Comments are also made regarding the suitability of the different search methodologies for solving job shop-based scheduling problems.

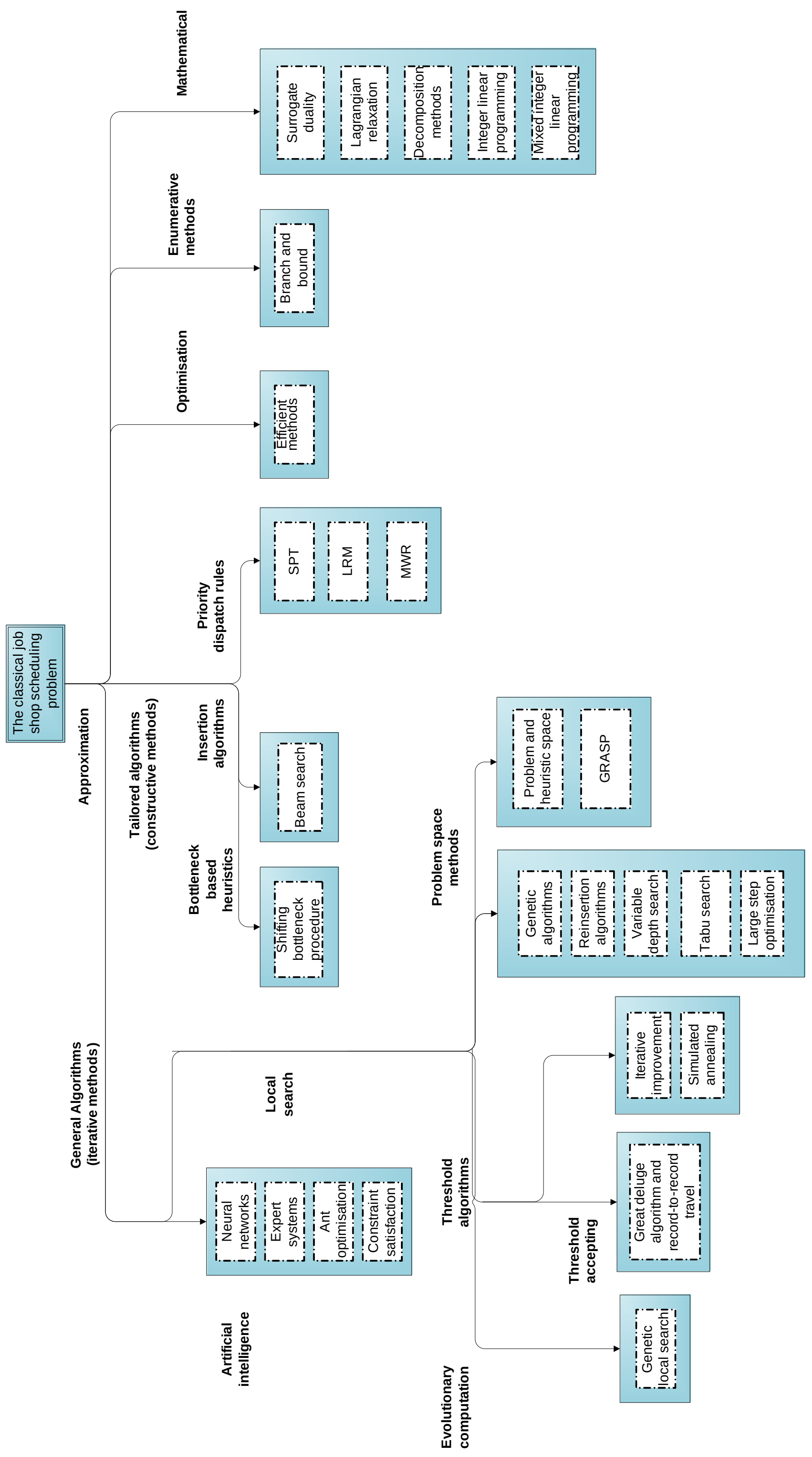


Figure 3.1: Solution strategies of the classical job shop scheduling problem [57]

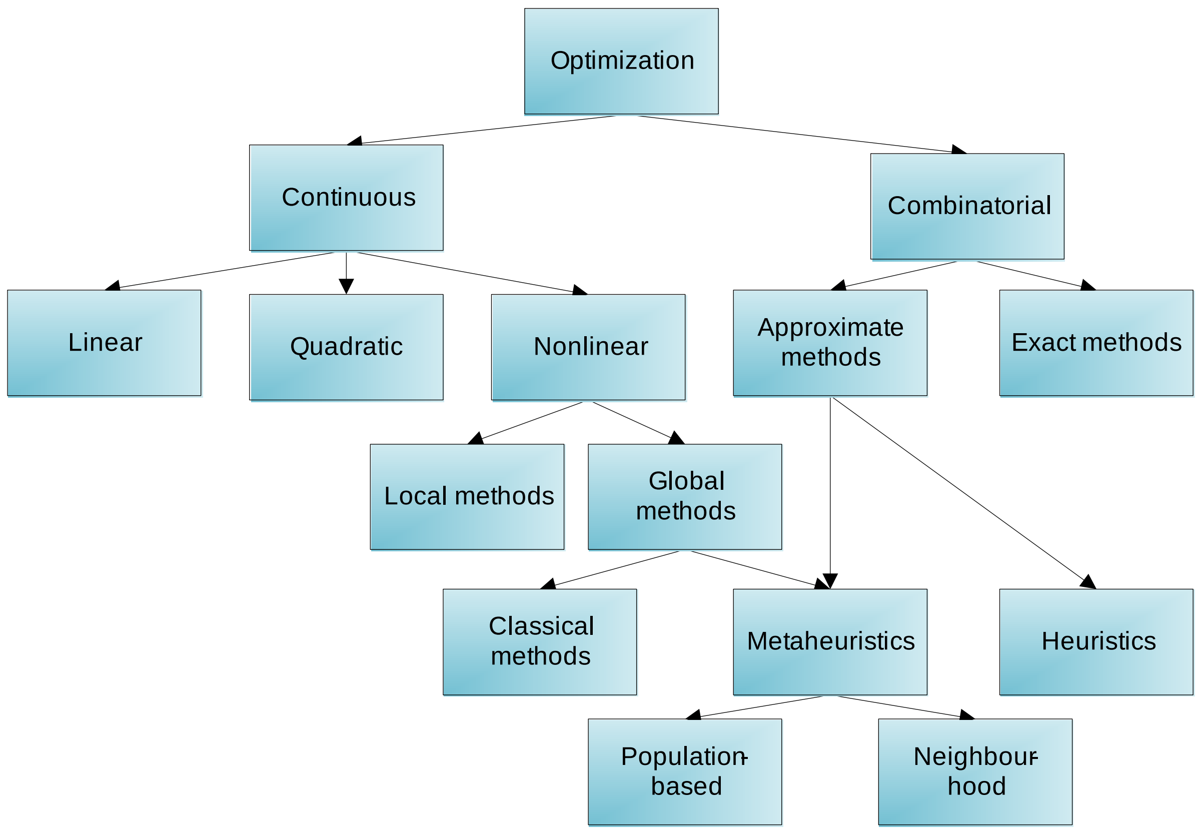


Figure 3.2: A classification of popular search methodologies [43]

3.1.1 Optimal solution strategies

During the 1960s significant emphasis was placed on finding exact solutions by means of elaborate and sophisticated mathematical constructs. The most widely used enumerative strategy is the branch-and-bound algorithm [68] which entails an implicit search of a tree structure representing the solution space. A number of procedures have been developed to exclude large portions of the tree to speed up the search process. Unfortunately, apart from the excessive computational burden, this strategy's performance is also relatively problem dependent and is sensitive to the initial upper or lower bound values [57].

The suitability of optimal solution strategies is also highly dependent on the complexity of the problem. Research from the 1970s clearly highlights the extreme intractability of the job shop scheduling problem [21]. The problem can be classified as strongly *NP*-hard. Therefore, only a small number of special instances of the JSSP are solvable within polynomial time and optimal solution strategies are probably of limited use. Approximation methods, on the other hand, becomes an attractive alternative. Even though the optimality of the solutions cannot be guaranteed, larger problems can be solved more efficiently.

3.1.2 Heuristic methods

In general, heuristic methods simply aim to obtain a "good enough" solution by selecting decision variables to obtain solutions which continuously progress towards a superior solution. The general local search procedure, the shifting bottleneck heuristic, and various priority-based rules are often applied to job shop scheduling problems.

General local search procedure

The simplest heuristic method for solving the JSSP is the general local search procedure. This method consists of iteratively evaluating the current solution and determining the direction in which movement should take place to improve the objective function. Search directions and step lengths can be determined using conjugate gradients, newton methods, or steepest gradient descent [39].

Priority dispatch rules

One of the earliest heuristic methods developed for scheduling applications, priority dispatch rules (PDRs) [107], is based on the assignment of priorities to all operations available for sequencing. These priorities can be assigned according to a large number of heuristic rules, for example, shortest processing time (SPT) and earliest due date (EDD). Although very easy to implement with a low computational burden, PDRs are highly problem dependent and solution quality degrades significantly as dimensionality increases [57].

The shifting bottleneck heuristic

The shifting bottleneck heuristic (SBH) [4] is commonly considered to have had the greatest influence on approximation methods for production scheduling applications. This is largely due to its exploitation of the well-developed algorithms for the single machine shop scheduling problem. The strategy involves relaxing the problem into a number of single machine problems, which are solved one at a time and ranked according to objective function value. The schedule for the more complex job shop scheduling problem can then be generated by sequentially scheduling each machine based on its rank [57].

The SBH heuristic is often used in conjunction with the disjunctive graph formulation, which consists of a set of nodes representing all the operations to be processed on the set of machines. Two fictitious nodes are also added, namely the source node (at the beginning of the network) and the sink node (at the end of the network). The set of nodes are indicated by **V**. A weight, proportional to the processing time of the operation, is assigned to each of the nodes. Precedence relationships between operations are indicated by means of a set of directed arcs denoted by **C**. Capacity constraints ensure that two jobs which require the same machine cannot be processed simultaneously. These constraints are enforced by means of a set of undirected arcs, **D**. Potential feasible solutions are obtained by defining directions for each of the disjunctive arcs [21]. When solving a makespan minimization problem, shortest path algorithms are traditionally used to find the optimal solution. An example of a disjunctive graph formulation for a 3 machine 4 job problem is illustrated in Figure 3.3.

Although the shifting bottleneck heuristic with disjunctive graph representation is robust and useful for solving real life job shop problems, concurrent or parallel processing and indefinite cyclical process flows cannot be modelled directly [143]. Feasibility of solutions can also be a problem since an acyclic graph is required for schedule feasibility. However, even though these difficulties exist, Mason *et al.* [89, 90] have documented a number of successful applications of a modified SBH in the semi-conductor manufacturing industry.

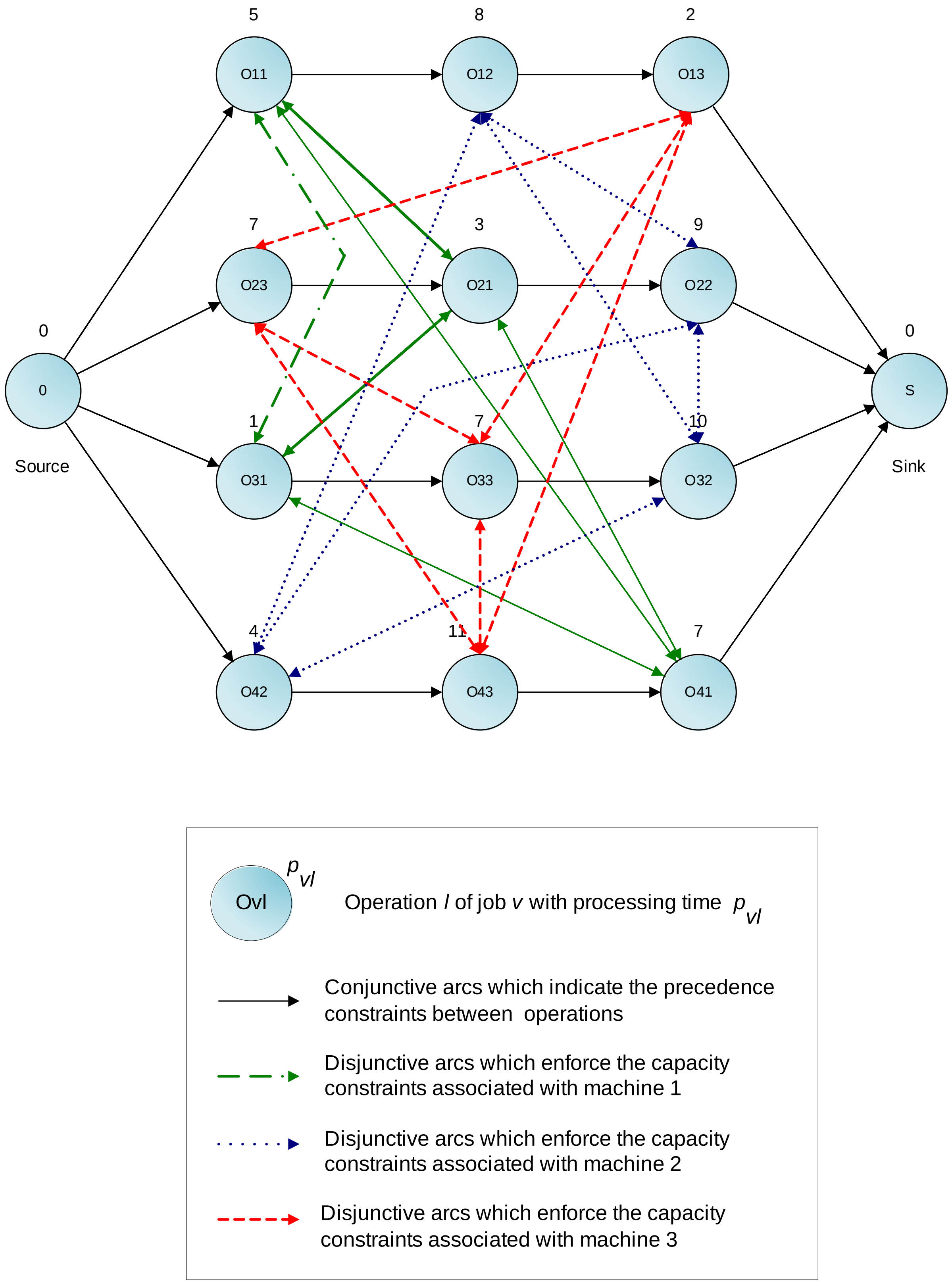

5
8
2
O11
O12
O13
7
3
9
O23
O21
O22
0
0
0
S
Source
Sink
1
7
10
O31
O33
O32
4
11
7
O42
O43
O41
Ovl
p_{vl}
Operation l of job v with processing time p_{vl}
Conjunctive arcs which indicate the precedence constraints between operations
Disjunctive arcs which enforce the capacity constraints associated with machine 1
Disjunctive arcs which enforce the capacity constraints associated with machine 2
Disjunctive arcs which enforce the capacity constraints associated with machine 3

Figure 3.3: An example of the disjunctive graph formulation [57]

3.1.3 Metaheuristics

The inability of heuristic methods to escape local optima have resulted in the development of metaheuristics. These “intelligent heuristics” temporarily allow non-improving feasible moves which have a positive impact on the algorithm’s ability to explore the search space [119]. For reference purposes a number of the more common metaheuristics are indicated in Figure 3.4.

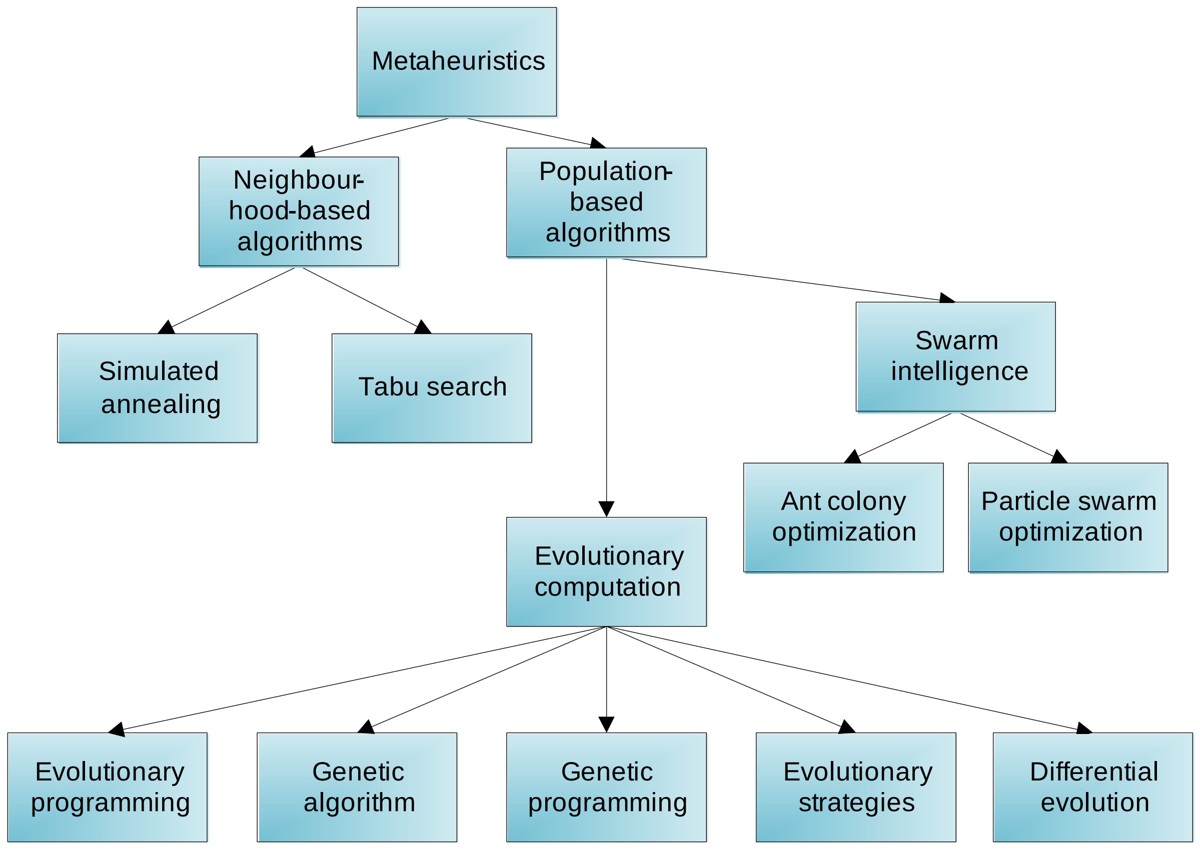


Figure 3.4: A number of common metaheuristics

Neighbourhood metaheuristics refer to those search methodologies where a single solution is transformed over time by making use of predefined neighbourhoods. Population-based metaheuristics, on the other hand, are characterized by a population of candidate solutions which are adapted over time. The candidate solutions in an evolutionary algorithm compete for survival [27], whereas the agents in a swarm communicate and cooperate with each other by acting on the environment [39].

Of the listed examples, simulated annealing (SA), tabu search (TS), and genetic algorithms (GAs) have been most frequently applied to job shop scheduling problems [49]. However, it will become evident throughout the rest of this section that each of these search methodologies have their own advantages and disadvantages.

Tabu search

TS can be defined as an algorithm which deals with cycling by temporarily forbidding moves that would return to a solution recently visited [48]. This is accomplished by means of a tabu list which records the most recent solutions and prevents the search from continuing with these now non-feasible moves. This list can act as both a recency-based memory (where the list classifies solutions according to the length of time they have spent in the list) and frequency-based memory (where the number of times a solution occurs has an influence). Additionally, an incumbent solution [155] is used to keep track of the best solution found thus far and certain aspiration criteria can also be defined to override the tabu list if this should become necessary. This solution strategy has led to a number of successful solutions of job shop scheduling problems [57].

Simulated annealing

SA is an optimization process based on the cooling process of liquids and solids. As a substance cools, the molecules tend to align themselves in a crystalline structure associated with the minimum energy state of the system. This is analogous to the algorithm converging to the optimal solution of an optimization problem. As the temperature of the metals decrease, the alignment of the atoms in the structure continually changes. This alignment is analogous to the fitness of the solution: an alignment which results in a lower energy state also results in an improved solution. Alignments of atoms are probabilistically accepted based on the Boltzmann–Gibbs distribution:

$$P_{i_1 i_2}(t) \triangleq \begin{cases} 1 & \text{if } f(\boldsymbol{x}_{i_2}) < f(\boldsymbol{x}_{i_1}) \\ e^{\frac{f(\boldsymbol{x}_{i_2}) - f(\boldsymbol{x}_{i_1})}{a\Upsilon}} & \text{otherwise,} \end{cases} \tag{3.1}$$

where $P_{i_1 i_2}(t)$ is the probability of moving from point $\boldsymbol{x}_{i_1}$ to $\boldsymbol{x}_{i_2}$, a is a positive constant and Υ is the temperature of the system [39].

Jain and Meeran [57] describe SA as a generic technique requiring excessive computational effort due to its inability to achieve good solutions quickly. However, the hybridization of SA with other solution strategies, including genetic algorithms, has greatly improved its competitiveness.

Evolutionary algorithms

Evolutionary algorithms attempt to mimic the process of biological evolution to find better and better solutions [119]. A number of operators (for example selection, crossover, mutation, and cloning) act upon a population of randomly initialized individuals to transform these individuals into better solutions.

Opinions seem to be conflicting with respect to the success of evolutionary algorithms (EA)s in solving complex scheduling problems. Jain and Meeran [57] criticize GAs for being inefficient, stating that GAs are unable to successfully represent the classical job shop scheduling problem since traditional crossover operators cannot generate feasible schedules. Yet many successful JSSP applications of modified EAs have been recorded [62, 91, 99].

Particle swarm optimization

PSO can be classified as a stochastic population-based optimization technique [64], which was developed as a model of the flocking behaviour of birds. Since its development, the algorithm has established itself as a competitive solution strategy for a wide range of real-world problems. However, due to its relatively recent development, very few complex scheduling applications have been documented.

Ant colony optimization

The first ant colony optimization (ACO) algorithm was developed to model the foraging behaviour of ants [39]. This search methodology is traditionally associated with discrete combinatorial optimization problems which can be modeled as decision graphs [27]. Each ant is tasked with constructing a candidate solution or path through the decision graph. Good solutions are marked with a high pheromone concentration, which ensures that they are revisited with a higher probability. As an ant is progressing through the graph, the transition probability, $P_{i_1i_2}(t)$, associated with visiting node i_2 immediately after node i_1 at time t is given by

$$P_{i_1i_2}(t) = \frac{\pi^a_{i_1i_2}(t)}{\sum_{i_2 \in N_{i_1}} \pi^a_{i_1i_2}(t)} \tag{3.2}$$

where $\pi_{i_1i_2}$ denotes the pheromone concentration of the link between node i_1 and node i_2, a is a positive constant used to amplify the pheromone influence, and N_{i_1} is the set of feasible nodes connected to node i_1. Although this technique has been used successfully in solving simpler scheduling problems [13, 19, 36], not many complex scheduling applications have been reported.

3.1.4 Selecting a suitable solution strategy

Based on the above analysis, a population-based metaheuristic was selected for further investigation. Apart from the metaheuristic's ability to escape from local minima, there are distinct advantages associated with using a population-based algorithm for optimizing multiple conflicting objectives. Since a population of candidate solutions is adapted over time, different individuals can simultaneously converge to different regions of the objective space. This results in significantly less effort required to generate a set of trade-off solutions.

Of all the population-based algorithms, genetic algorithms seem to be the most successful at addressing complex scheduling problems. However, as a direct result of the no free lunch theorem [144], there is no way of predicting that a GA would indeed be the best solution strategy for the *Optimatix* problem. Technically, any of the search methodologies in Figure 3.4 could be used.

For the purposes of this dissertation, the two latest additions to the metaheuristics depicted in Figure 3.4, namely particle swarm optimization and differential evolution, were selected for further investigation. The inherent simplicity [64, 114] and proven success on simpler scheduling problems have further added to the desirability of the two algorithms. Further research opportunities are also present in investigating the use of continuous optimization algorithms for solving discrete combinatorial optimization problems.

3.2 Particle swarm optimization

Kennedy and Eberhart [64] trace the origins of the particle swarm optimization (PSO) algorithm back to Reynold's "boid" simulations [121]. The initial objectives of this study and the other collective behaviour studies of the late 80s was to simulate the graceful, unpredictable choreography of collision-proof birds in a flock [37]. However, the optimization potential, of what was at that stage only a conceptual model, soon became apparent. Simplification and parameter derivation resulted in the first simplistic implementation by Kennedy and Eberhart [64] in 1995.

Since its humble beginnings, PSO has established itself as a simple and computationally efficient optimization method in both the fields of artificial intelligence and mathematical optimization. Applications range from more traditional implementations such as training artificial neural networks [40, 137] and task allocation [124], to more specific applications, such as the design of aircraft wings [140] and the generation of interactive, improvised music [17]. The rest of this section introduces the basic concepts of PSO before the actual algorithm, associated algorithm parameters, and variations are discussed in more detail.

3.2.1 The basic algorithm

The PSO algorithm represents each potential problem solution by the position of a particle in multi-dimensional hyperspace. Throughout the optimization process velocity and displacement updates are applied to each particle to move it to a different position and thus a different solution in the search space.

The velocity update is often thought to be the most critical component of the PSO algorithm since it incorporates the concepts of emergence and social intelligence. Figure 3.5 illustrates that the magnitude and direction of a particle's velocity at time t is considered to be the resultant of three vectors: the particle velocity vector at time $t-1$, the cognitive component ($pbest$), which is a vector representation of the best solution found to date by the specific particle, and the social component ($gbest$), which is a vector representation of the best solution found to date by all the particles in the swarm. The $gbest$ model [64] calculates the velocity of particle i in dimension j at time $t+1$ using

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_{1j}(t)[\hat{x}_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[x^*_j(t) - x_{ij}(t)] \tag{3.3}$$

where $v_{ij}(t)$ represents the velocity of particle i in dimension j at time t, c_1 and c_2 are the cognitive and social acceleration constants, $\hat{x}_{ij}(t)$ and $x_{ij}(t)$ respectively denotes the personal best position ($pbest$) and the position of particle i in dimension j at time t. $x^*_j(t)$ denote the global best position ($gbest$) in dimension j, w refers to the inertia weight, and $r_{1j}(t)$ and $r_{2j}(t)$ are sampled from a uniform random distribution, $U(0,1)$.

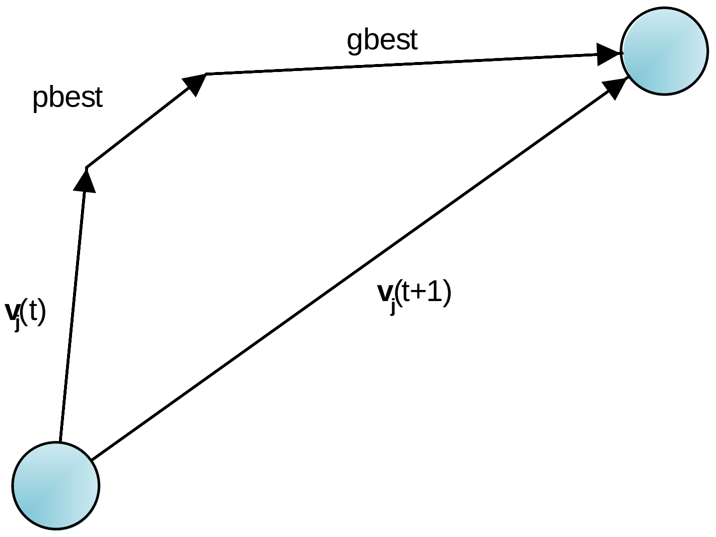


Figure 3.5: Particle velocity as resultant of three components

The displacement of particle i at time t is simply derived from the calculation of $v_{ij}(t+1)$ in equation (3.3) and is given as

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \tag{3.4}$$

This simultaneous movement of particles towards their own previous best solutions ($pbest$) and the best solution found by the entire swarm ($gbest$) results in the particles converging to one or more good solutions in the search space. For the sake of completeness, pseudocode of the basic PSO algorithm is provided in Algorithm 3.1.

```
Initialize an $n_x$-dimensional swarm of $n_s$ particles
$t = 1$
**while** $t < I_{max}$ **do**
    **for** *All particles i* **do**
        **if** $f(\boldsymbol{x}_i(t)) < f(\hat{\boldsymbol{x}}_i)$ **then**
            $\hat{\boldsymbol{x}}_i = \boldsymbol{x}_i(t)$
        **end**
        **if** $f(\hat{\boldsymbol{x}}_i) < f(\boldsymbol{x}^*)$ **then**
            $\boldsymbol{x}^* = \hat{\boldsymbol{x}}_i$
        **end**
    **end**
    **for** *All particles i* **do**
        Update the particle velocity using equation (3.3)
        Update the particle position using equation (3.4)
    **end**
    $t = t + 1$
**end**
```

Algorithm 3.1: The basic *gbest* PSO algorithm [39]

3.2.2 The guaranteed convergence PSO algorithm

Unfortunately, the basic PSO algorithm has a potentially dangerous property. The algorithm is "driven" by the fact that as a particle moves through the decision space, it is always attracted towards its *pbest* value and the flock's *gbest* value. If any of the particles reach a position in the search space where

$$\hat{\boldsymbol{x}} = \boldsymbol{x}(t) = \boldsymbol{x}^* \tag{3.5}$$

only the momentum term ($wv_{ij}(t)$ in equation (3.3)) remains to act as a driving force for the specific particle to continue exploring the rest of the search space. However, if the condition described in equation (3.5) persists, it can result in the swarm stagnating on a solution which is not necessarily a local optimum [138]. The guaranteed convergence particle swarm optimization (GCPSO) algorithm [138] has been shown to address this problem effectively and have thus been used for all simulations in this dissertation. This algorithm (Algorithm 3.2) requires that different velocity and displacement updates, defined as

$$v_{\tau j}(t+1) = -\, x_{\tau j}(t) + x_j^*(t) + wv_{\tau j}(t) + \rho(t)(1 - 2r_j(t)) \tag{3.6}$$

and

$$x_{\tau j}(t+1) = x_j^*(t) + wv_{\tau j}(t) + \rho(t)(1 - 2r_j(t)), \tag{3.7}$$

are applied to the global best particle, where $\rho(t)$ is a time-dependent scaling factor, $r_j(t)$ is sampled from a uniform random distribution, $U(0,1)$, and all other particles are updated by means of equations (3.3) and (3.4). This algorithm forces the *gbest* particle into a random search around the global best position. The size of the search space is then adjusted on the basis of the number of consecutive successes or failures of the particle, where success is defined as an improvement in the objective function value. In Algorithm 3.2, the number of consecutive successes is denoted by ζ and the number of consecutive failures are denoted by η.

3.2.3 Algorithm parameters

A section on PSO would not be complete without discussing the various parameters which have an effect on algorithm performance. Eberhart *et al.* [65] and Engelbrecht [39] provide a detailed description of each of the PSO parameters. Thus this dissertation focuses only on those parameters which directly impacts on scheduling performance.

- The swarm size, n_s, and maximum number of iterations, I_{max}, largely determined the amount of time available to produce a scheduling solution.

```
Initialize an $n_x$-dimensional swarm of $n_s$ particles
$t = 1$
$\rho(t) = 1$
$\zeta = 0$
$\eta = 0$
while $t < I_{max}$ do
    for All particles $i$ do
        if $f(\boldsymbol{x}_i(t)) < f(\hat{\boldsymbol{x}}_i)$ then
            $\hat{\boldsymbol{x}}_i = \boldsymbol{x}_i(t)$
        end
        if $f(\hat{\boldsymbol{x}}_i) < f(\boldsymbol{x}^*)$ then
            $\zeta = \zeta + 1$
            $\eta = 0$
            $\boldsymbol{x}^* = \hat{\boldsymbol{x}}_i$
        else
            $\eta = \eta + 1$
            $\zeta = 0$
        end
    end
    for All particles $i|i \neq \tau$ do
        Update the particle velocity using equation (3.3)
        Update the particle position using equation (3.4)
    end
    Update the gbest particle velocity using equation (3.6)
    Update the gbest particle position using equation (3.7)
    $t = t + 1$
end
```

Algorithm 3.2: The guaranteed convergence PSO (GCPSO) algorithm [138]

- V_{max} places an upper bound on the maximum velocity that a particle may obtain. This parameter was incorporated into the PSO algorithm to prevent the particle velocities from becoming excessively large as particles move further away from the *gbest* and *pbest* positions.

- The inertia weight, w, is important in ensuring that a suitable trade-off is obtained between exploration of different areas of the search space and further exploitation of good areas. As can be seen from equation (3.3), w weighs the contribution of the previous velocity. Large values of w thus encourage exploration while smaller values encourage exploitation.

- The two acceleration coefficients, c_1 and c_2, also have a significant impact on the exploration-exploitation trade-off of the algorithm, since they control the stochastic influence of the cognitive and social components on particle velocity [39].

3.2.4 Variations on the basic PSO algorithm

In order to address the inherent limitations and requirements of the PSO algorithm, a number of variations on the *gbest* PSO algorithm have been developed over the years. As described in Engelbrecht [39] these variations, indicated in Figure 3.6, can be organized into six main categories:

- Social-based algorithms use different social topologies or network structures. Algorithms using alternative means of calculating *pbest* and *gbest* are also classified as social-based algorithms.

- Hybrid algorithms refer to all PSO variations that incorporate concepts from other metaheuristics. Examples include the use of EA-based concepts such as selection, reproduction, and mutation.

- Sub-swarm-based algorithms are based on some explicit or implicit grouping of particles in sub-swarms and can be divided into cooperative and competitive sub-swarm-based algorithms.

- Memetic algorithms incorporate local search procedures within the standard PSO to enhance the exploitation ability of the algorithm.
- Multi-start algorithms inject chaos into the swarm to increase diversity through random initialization of particles.
- Algorithms which utilize various repelling methods also have as their main objective the diversification of the swarm. This class of algorithms include all variations where specific mechanisms are employed to avoid particle collisions or where adjacent particles are repelled.

Of all the PSO variations developed over the past decade, the degree of social interaction between particles has probably received the most attention. A number of alternative social network structures have been developed to explore different information exchange mechanisms between the particles within a swarm. Kennedy and Mendes [66] have already empirically evaluated a number of these social network structures, including the *gbest*, *lbest*, *pyramid*, *star* and Von Neumann structures. The *gbest* and Von Neumann topologies (refer to Figure 3.7) are the most important variations for the purposes of this dissertation.

It is well known in PSO literature that the *gbest* PSO algorithm converges fairly quickly [65], since all particles are partially attracted to the best position found by the swarm. Depending on the problem, this relatively fast loss of diversity can result in the algorithm finding a suboptimal solution within relatively few iterations.

The Von Neumann PSO organizes the particles into a lattice according to the particle indices. Each particle belongs to a neighbourhood consisting of its nearest neighbours in the cubic structure. Instead of being partially attracted to *gbest*, the velocity of a particle is influenced by the best solution found by the other particles in the same neighbourhood. Since these neighbourhoods overlap, information about good solutions is eventually propagated throughout the swarm, but at a much slower rate. In so doing more diversity and subsequent slower convergence is obtained, leading to significantly improved chances of finding a good solution.

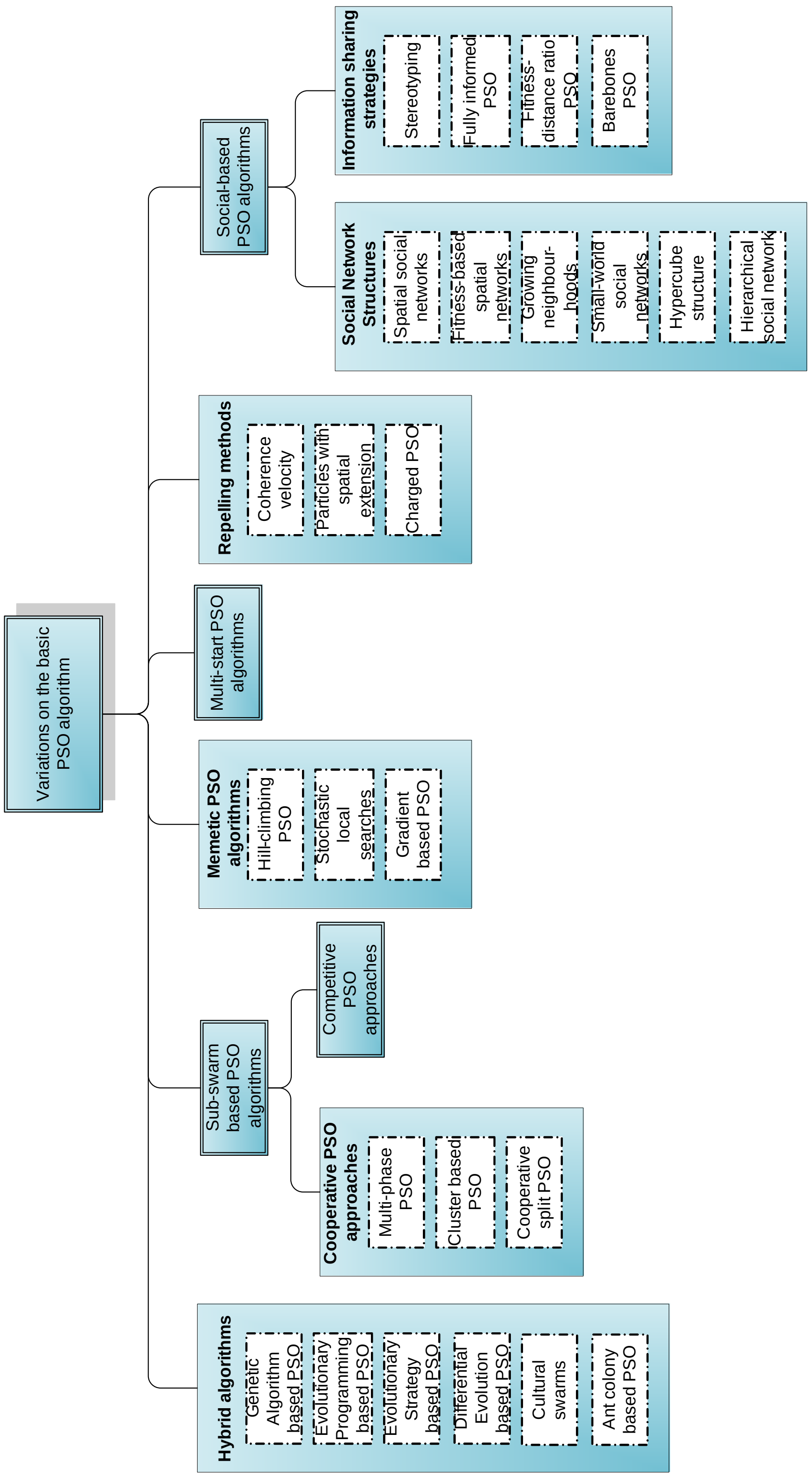

Variations on the basic PSO algorithm
Social-based PSO algorithms
Information sharing strategies
Stereotyping
Fully informed PSO
Fitness-distance ratio PSO
Barebones PSO
Social Network Structures
Spatial social networks
Fitness-based spatial networks
Growing neighbour-hoods
Small-world social networks
Hypercube structure
Hierarchical social network
Repelling methods
Coherence velocity
Particles with spatial extension
Charged PSO
Multi-start PSO algorithms
Memetic PSO algorithms
Hill-climbing PSO
Stochastic local searches
Gradient based PSO
Sub-swarm based PSO algorithms
Competitive PSO approaches
Cooperative PSO approaches
Multi-phase PSO
Cluster based PSO
Cooperative split PSO
Hybrid algorithms
Genetic Algorithm based PSO
Evolutionary Programming based PSO
Evolutionary Strategy based PSO
Differential Evolution based PSO
Cultural swarms
Ant colony based PSO

Figure 3.6: Variations on the basic PSO algorithm [39]

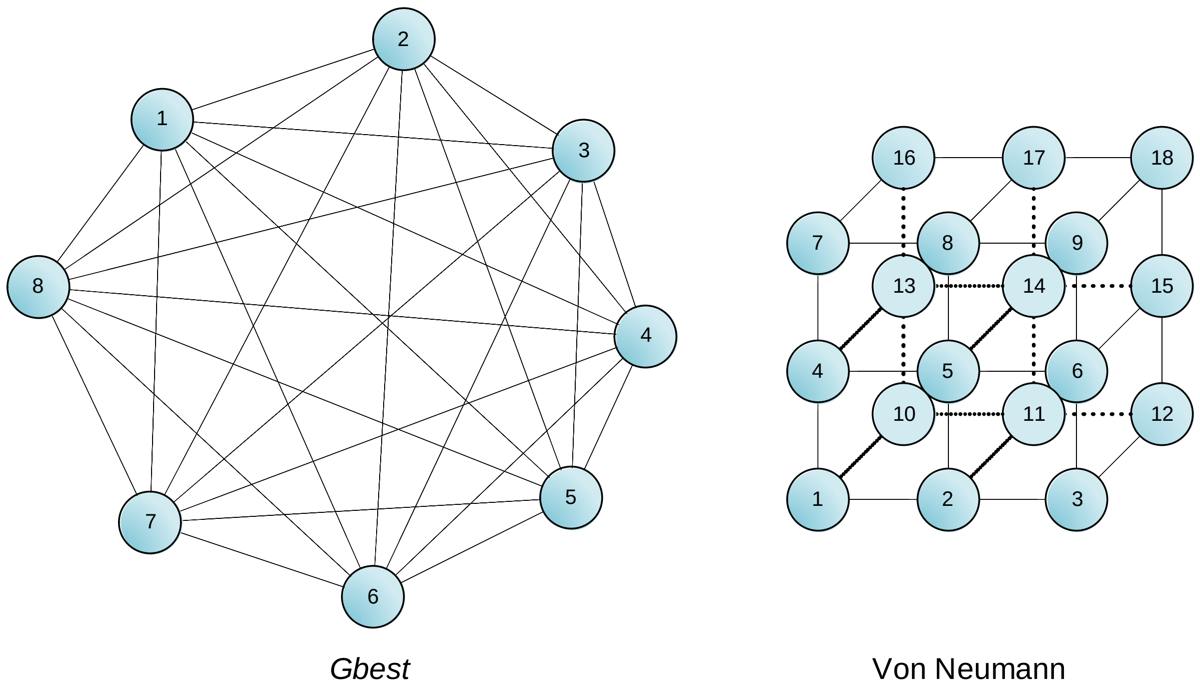


Figure 3.7: The *gbest* and Von Neumann topologies [39]. The lines between particles indicate the propagation of information through the swarm.

3.3 Differential evolution

Originally developed from work done on Chebyshev's polynomial fitting problems, differential evolution (DE) finds its roots in the genetic annealing algorithm of Storn and Price [131]. Classified as a parallel direct search method [130], DE achieved third place on benchmark problems at the first international contest on evolutionary optimization in 1996. Since then, the number of DE research papers increased significantly every year and DE is now well-known in the evolutionary computation community as an alternative to traditional EAs. The algorithm is considered to be easy to understand, simple to implement, reliable, and fast [114]. Application areas are just as diverse as is the case for the PSO algorithm and range from function optimization [131] to the determination of earthquake hypocenters [123].

Similar to the previous section, the rest of this section first introduces the basic concepts of DE before the actual algorithm, associated algorithm parameters, and variations are discussed in more detail.

3.3.1 The basic algorithm

The success of DE can be mainly attributed to the use of a difference vector which determines the step size of the algorithm as the population, consisting of vectors of floating point numbers, moves through the search space. Information regarding the difference between two existing solutions is, in other words, used to guide the algorithm towards better solutions [131].

More specifically, for each individual, i, in the population, a base vector, $\boldsymbol{x}_{i_1}(t)$, as well as two other vectors, $\boldsymbol{x}_{i_2}(t)$ and $\boldsymbol{x}_{i_3}(t)$, are randomly selected from the current population, where $x_{ij}(t)$ denotes the j^{th} dimension of individual i of generation t and $i \neq i_1 \neq i_2 \neq i_3$. The target vector, $\boldsymbol{T}_i$, can then be obtained through the application of a differential mutation operator defined as

$$T_{ij}(t) = x_{i_1j}(t) + F(x_{i_2j}(t) - x_{i_3j}(t)) \tag{3.8}$$

Then, for all dimensions, j, if $r \sim U(0,1) \leq p_r$ or $j = \nu \sim U(1, ..., n_x)$

$$c_{ij}(t) = T_{ij}(t) \tag{3.9}$$

otherwise $c_{ij}(t) = x_{ij}(t)$, where p_r is the probability of reproduction, n_x is the number of dimensions, F is the scaling factor, and $\boldsymbol{c}_i$ is known as the trial vector.

An individual may only be replaced by an individual with a better fitness function value. In other words, if the fitness of $\boldsymbol{c}_i(t)$ is better than the fitness of the i^{th} individual of the original population, this individual is replaced by $\boldsymbol{c}_i(t)$ [38]. For the sake of completeness, pseudocode of the basic DE algorithm is provided in Algorithm 3.3.

The differential mutation operator in equation (6.4) has the desirable property that it allows the step sizes of the algorithm to automatically adapt to the objective function landscape [114]. For example, before the population has converged around a specific optimum, the randomly sampled individuals could still be far apart in different areas of the search space. This allows for larger step sizes during the initial iterations of the optimization algorithm when greater exploration of the search space and the ability to escape from local optima is desirable. Later on, when all of the individuals are converging

```
Initialize an $n_x$-dimensional population of $n_s$ individuals
$t = 1$
$i_1, i_2, i_3 = 0$
while $t < I_{max}$ do
    for All individuals $i$ do
        Randomly select an individual, $i_1$, from the population
        Randomly select an individual, $i_2$, from the population
        while $i_1 = i_2$ do
            Randomly select an individual, $i_2$, from the population
        end
        Randomly select an individual, $i_3$, from the population
        while $i_2 = i_3$ or $i_1 = i_3$ do
            Randomly select an individual, $i_3$, from the population
        end
        Randomly select a dimension, $\nu$
        for All dimensions $j$ do
            if $r \sim (0, 1) \leq p_r$ or $j = \nu$ then
                Calculate $c_{ij}(t)$ using equation (6.4)
            else
                $c_{ij}(t) = x_{ij}(t)$
            end
        end
    end
    for All individuals $i$ do
        if $f(\boldsymbol{c}_i(t)) \leq f(\boldsymbol{x}_i(t))$ then
            $\boldsymbol{x}_i(t + 1) = \boldsymbol{c}_i(t)$
        end
    end
    $t = t + 1$
end
```

Algorithm 3.3: The classic DE algorithm [114]

around a single optimum, smaller step sizes are automatically taken since all individuals are close to each other in the search space. This strategy allows the algorithm to more effectively exploit the area around the optimum in search of a better solution.

3.3.2 Algorithm parameters

DE has surprisingly few parameters when compared to PSO. Since the population size and maximum number of iterations have already been considered in Section 3.2.3, only the scaling factor, F, and reproduction probability, p_r, are discussed below:

- Price *et al.* [114] define F as a positive real-valued number that controls the rate at which the population evolves. Large F values tend to allow larger step sizes which are traditionally considered to be better for exploration purposes, whereas smaller F values are better for exploitation.
- The reproduction probability controls the fraction of parameter values that are inherited from the target vector. Larger values of p_r are associated with improved exploration capability, whereas smaller p_r values focus on intensifying the search around good solutions.

3.3.3 Variations on the basic DE algorithm

The number of DE research papers increases exponentially each year and as the algorithm is refined, results continue to improve and additional research opportunities become visible. As a result, several variants of DE have been defined over the years. This section describes a number of these variants according to Storn and Price's DE/x/y/z notation [131], where x refers to the method used to select the target vector, y refers to the number of difference vectors used and z denotes the crossover mechanism used.

Alternative base vector selection mechanisms

Storn [130] identifies three different base vector selection mechanisms: DE/rand/y/z, DE/best/y/z and DE/rand-to-best/y/z (DE/R2B/y/z):

- A randomly selected population member serves as the base vector in DE/rand/y/z.
- The base vector is selected as the population member with the best fitness function, i.e. the best individual, in DE/best/y/z. Incorporating the best individual into the equation is meant to enable faster convergence, with a subsequent decrease in population diversity.
- DE/R2B/y/z aims to address the limitations of DE/best/y/z in terms of potential premature convergence. A linearly or exponentially decreasing value ($\gamma \in (0,1)$) is incorporated into the equation used to calculate the base vector. This ensures that more emphasis is placed on random base vector selection at the start of the optimization run when population diversity is important. Towards the end, the DE/best/y/z strategy is emphasized when convergence to the best solution is desirable. The adjusted target vector equation becomes

  $$T_{ij}(t) = \gamma x_{\tau j}(t) + (1-\gamma)x_{i_1 j}(t) + F(x_{i_2 j}(t) - x_{i_3 j}(t)), \tag{3.10}$$

  where $x_{\tau j}(t)$ denotes the j^{th} component of the best individual in the population at time t.

Alternative difference vector selection strategies

Storn [130] defines an additional variant which uses y difference vectors to "shift the random variation slightly into a gaussian direction". Preliminary results have indicated that this seems to be beneficial for a large number of functions. As an example, the trial vector for DE/x/2/z is given as

$$T_{ij}(t) = x_{i_1 j}(t) + F(x_{i_2 j}(t) + x_{i_3 j}(t) - x_{i_4 j}(t) - x_{i_5 j}(t)), \tag{3.11}$$

where $i_1 \neq i_2 \neq i_3 \neq i_4 \neq i_5$.

Alternative cross-over mechanisms

Two different crossover mechanisms, namely exponential and binomial crossover, have also been employed effectively in DE-based algorithms. Exponential crossover operators are applied to each of the n_x dimensions until $\nu \sim U(0,1) \geq p_r$. Binomial crossover operators are applied per dimension whenever $\nu \sim U(0,1) \leq p_r$. For high values of p_r, the exponential and binomial crossovers thus yield similar results [12].

The identification of a large number of DE variants is important because different strategies perform differently on different problems. Since no single strategy can be proved to outperform all other strategies for all problems, finding the best performing algorithm variation for each data set may result in significant performance improvements. However, it is not always practical to test all strategies, thus careful consideration should be given to determine which strategies result in the greatest improvement for a specific problem. For the purposes of this dissertation, an investigation into alternative base vector selection mechanisms was considered to be the most useful. This investigation is described in more detail in Section 4.2.

3.4 A brief analysis of existing PSO- and DE-based scheduling algorithms

The number of papers where PSO and DE are applied to scheduling has dramatically increased over the past few years [73]. This section provides an overview of PSO and DE machine scheduling literature before making a number of interesting observations.

3.4.1 General observations

Effective production scheduling requires solving a complex combinatorial and discrete optimization problem. Subsequently, two main approaches for solving scheduling problems by means of continuous optimization algorithms, such as PSO and DE, can be identified from literature [73]. Figures 3.8 and 3.9 firstly organize the relevant machine scheduling literature into the two main approaches followed, and then also distinguishes between different types of scheduling models which have been solved by these two approaches.

The first approach is based on redefining the standard operators of the classical PSO and DE algorithms, allowing the algorithms to function in a discrete domain. Secondly,

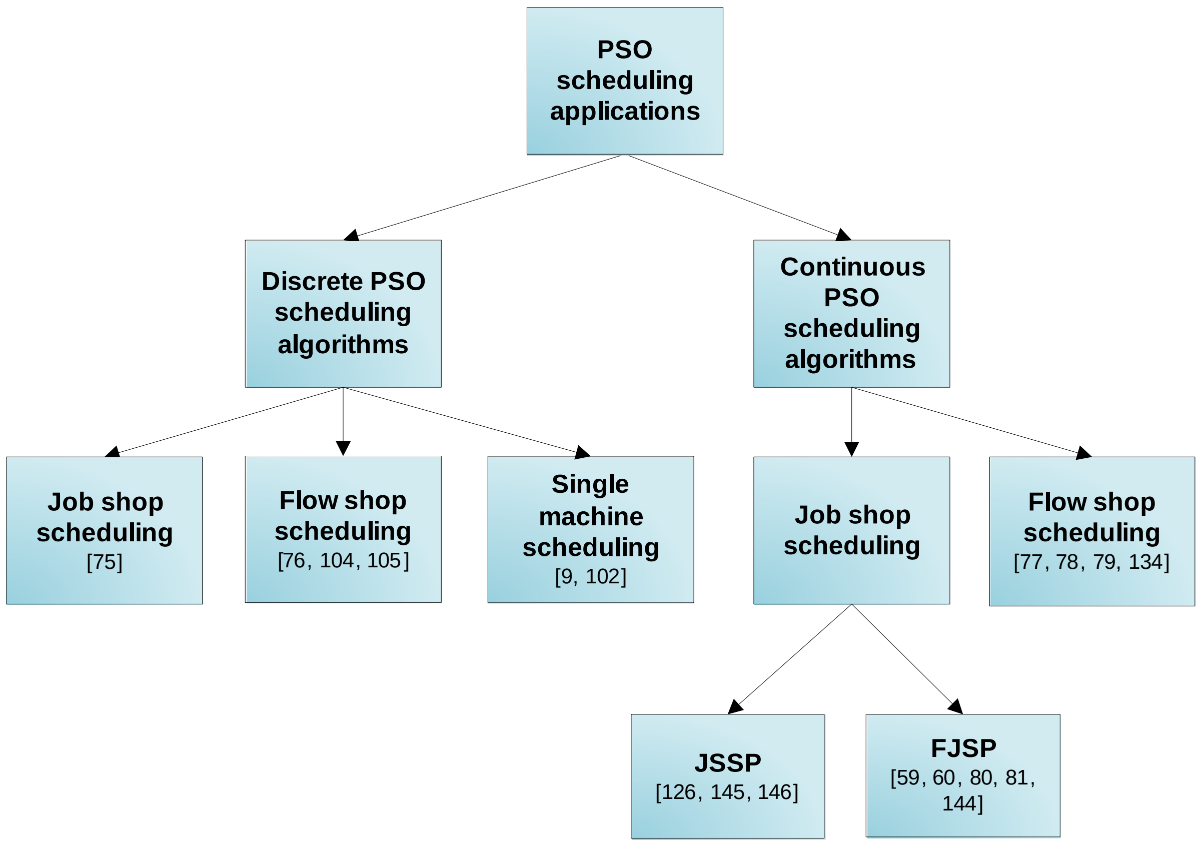

PSO scheduling applications
Discrete PSO scheduling algorithms
Continuous PSO scheduling algorithms
Job shop scheduling [75]
Flow shop scheduling [76, 104, 105]
Single machine scheduling [9, 102]
Job shop scheduling
Flow shop scheduling [77, 78, 79, 134]
JSSP [126, 145, 146]
FJSP [59, 60, 80, 81, 144]

Figure 3.8: Applications of PSO to machine scheduling literature

the scheduling problem may be converted into a continuous problem which can be solved easily by means of a continuous optimization algorithm.

According to Lei [73], the redefinition of the standard PSO operators often lead to poor performance in scheduling. Although the structure of the problem can be more easily exploited, it is quite difficult to retain the trade-off between social and individual learning which is largely responsible for the success of the continuous PSO. The inherent structure of the information exchange between particles is, after all, changed. A similar argument may be used for DE. The definition of the difference vector may be problematic when the continuous DE operators are discretized. Nonetheless, as can be seen from Figure 3.8, discrete PSO algorithms have already been applied successfully to single machine scheduling [9, 103], flow shop scheduling [76, 105, 106], and job shop scheduling [75]. Discrete DE algorithms have been used to solve flow shop scheduling [5, 104, 133] and

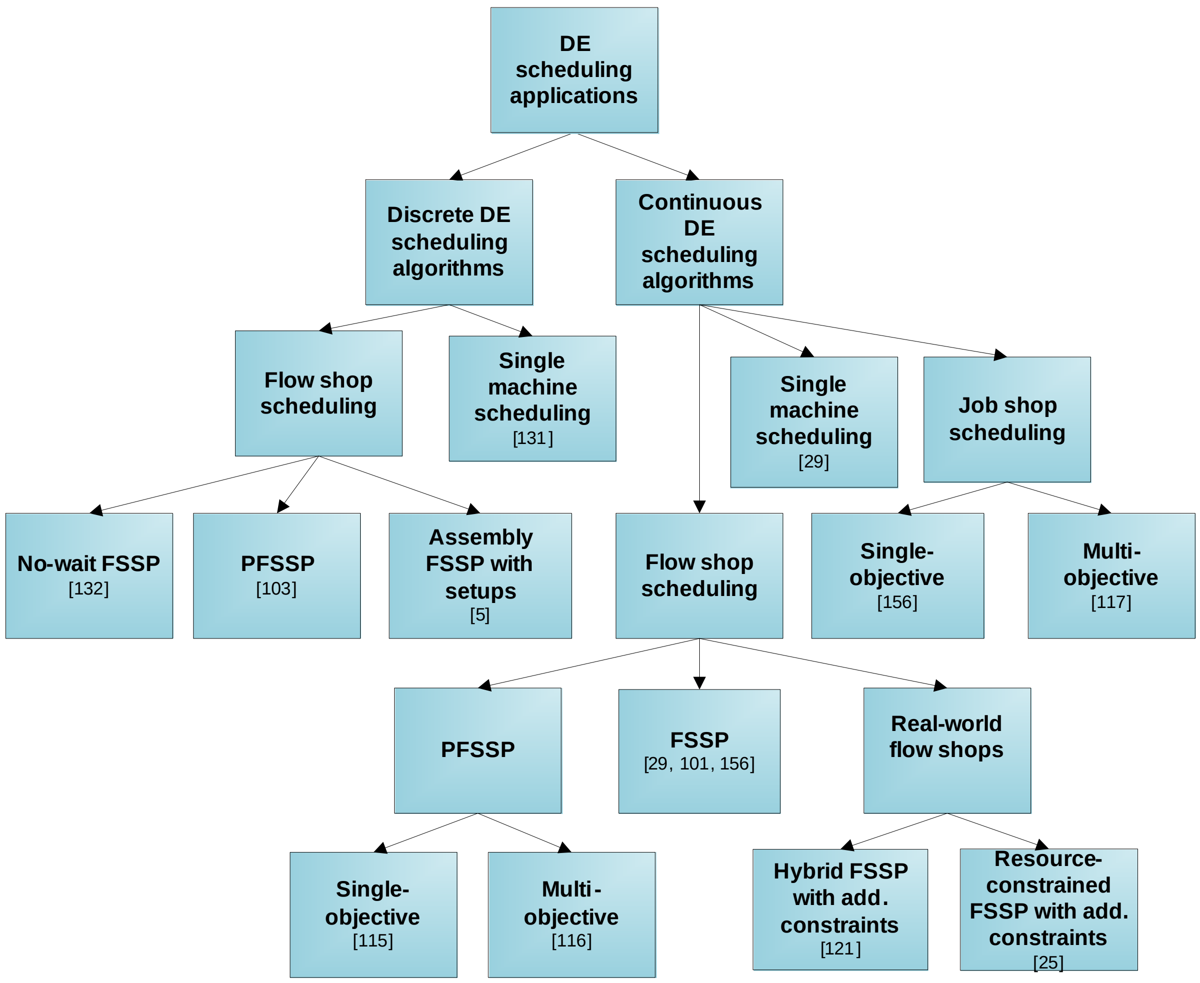

DE scheduling applications
Discrete DE scheduling algorithms
Continuous DE scheduling algorithms
Flow shop scheduling
Single machine scheduling [131]
Single machine scheduling [29]
Job shop scheduling
No-wait FSSP [132]
PFSSP [103]
Assembly FSSP with setups [5]
Flow shop scheduling
Single-objective [156]
Multi-objective [117]
PFSSP
FSSP [29, 101, 156]
Real-world flow shops
Single-objective [115]
Multi-objective [116]
Hybrid FSSP with add. constraints [121]
Resource-constrained FSSP with add. constraints [25]

Figure 3.9: Applications of DE to machine scheduling literature

single machine scheduling problems [132].

In this dissertation, the second approach will be followed where the scheduling problem is converted to a continuous problem which can be solved directly by the PSO and DE algorithms. Following this approach requires the algorithms to be able to operate effectively in two separate search spaces: (1) The schedule allocation space, $\boldsymbol{S}_s$, which consists of all the feasible schedules associated with the problem to be addressed, and (2) the particle space, $\boldsymbol{P}_s$, which consists of all the possible positions of the particles within the search space. To evaluate the fitness of a particle, the solution $\boldsymbol{x} \in \boldsymbol{P}_s$ must first be mapped to $\boldsymbol{y} \in \boldsymbol{S}_s$ before the fitness function value $\boldsymbol{f} \in \boldsymbol{F}$ (where $\boldsymbol{F}$ denotes the objective space) can be calculated.

In the PSO domain this approach has already been applied successfully to various flow shop [77, 78, 79, 135], job shop [127, 146, 147], and multiprocessor task scheduling problems [41]. Continuous DE implementations, on the other hand, have focused on the single machine scheduling problem (SMP) [97, 96, 134], flow shop scheduling problem (FSSP) [25, 97, 102, 116, 117, 122, 157], and job shop scheduling problem (JSSP) [118, 157].

3.4.2 Addressing more complex scheduling problems

It is obvious from the analysis of Section 3.4.1 that PSO and DE are establishing themselves as solution strategies of note for simpler scheduling problems. The applications to more complex production scheduling problems are still, however, considered to be relatively sparse. This is especially true for scheduling problems where both the sequencing of operations and the allocation of these operations to resources need to be addressed.

All the PSO and DE scheduling applications discussed in Section 3.4.1 can be reduced effectively to the problem of finding an "optimal" sequence of operations subject to a number of problem-specific constraints. However, when the processing of operations on alternative resources can lead to a reduction in the overall processing time, for example as in the case of a flexible job shop scheduling problem, the allocation of operations to resources becomes an important part of the optimization problem. This additional complexity understandably creates a number of additional challenges for the scheduling algorithm.

To the best of the author's knowledge no applications of differential evolution to a flexible job shop scheduling problem exist. However, five papers addressing flexible job shop scheduling problems by means of PSO could be identified:

- Xia and Wu [145] developed a simulated annealing-PSO-based hybrid solution strategy. It is notable that only the allocation of operations to resources is done by means of PSO. The actual sequencing of the assigned operations is performed by a simulated annealing (SA) algorithm and the multiple objectives are addressed by combining all relevant objectives into a single weighted sum objective.

- Liu *et al.* [80, 81] solved the multi-objective flexible job shop scheduling problem with minimum makespan and flowtime by means of a variable neighbourhood PSO algorithm, which employed a variable neighbourhood-based local search mechanism to enhance the exploitation ability of the swarm. Dynamic weighted aggregation is used to simultaneously minimize the two objective functions.

- Jia *et al.* [59] have minimized makespan, total workload, and maximum workload by means of a PSO algorithm employing a chaotic local search around the gbest particle. The multiple objectives are addressed by further minimizing the objective with the smallest fitness value at each function evaluation.

- Jia *et al.* [60] used a fully informed pareto-based PSO algorithm to minimize the makespan and maximum lateness in a flexible job shop (FJSP) environment. A problem-specific mutation operator was also defined to improve the diversity of the swarm.

A number of interesting observations can be made from this brief analysis. Firstly, it can indeed be confirmed that PSO and DE do not seem to be considered as established solution strategies for more complex scheduling environments when compared to, for example, genetic algorithms. Real world scheduling problems, where a large number of additional scheduling-specific constraints need to be addressed, have not been solved frequently by means of either PSO or DE. In fact, no additional constraints are even considered in the listed examples.

Secondly, the largest problem attempted by any of these PSO-based algorithms only consider the scheduling of 56 operations on at most 15 resources, which is currently considered to be a problem of "great size" in FJSP scheduling literature [63]. This is in sharp contrast with the *Optimatix* problem where a multi-objective FJSP consisting of up to 256 operations on 216 resources, need to be considered.

Thirdly, apart from the PSO-SA hybrid of Xia and Wu [145], all the algorithms make use of a two-part particle representation and resource allocation is addressed by means of rounding off the continuous PSO particle dimensions to the nearest integer value representing a resource index.

Finally, a number of the algorithms [59, 80, 81] employ complex local search mechanisms. Apart from the increased computational complexity, extending these algorithms to include sequence-dependent set-up times, auxiliary resources, and production down time, as is required in the proposed problem, would require that major structural changes be made to the algorithms currently in use at *Optimatix*.

3.5 Summary

This chapter documented the process of selecting a suitable solution strategy. Particle swarm optimization and differential evolution were selected as the solution strategies of choice. A brief introduction to each one of these search methodologies was provided before the existing PSO and DE scheduling work was described in more detail. The next chapter focuses on describing the actual implementation of these two strategies in a scheduling environment.

Chapter 4

Solving the single objective *Optimatix* problem

A large number of options are available when particle swarm optimization (PSO) and differential evolution (DE) are applied to a complex production scheduling problem such as the problem faced by *Optimatix*. The focus of this chapter is to motivate the initial "design choices" which were made during algorithm development. Investigations into appropriate problem representations are conducted within the context of PSO, suitable parameter values are derived and alternative PSO topologies and DE base vector selection mechanisms are implemented. Finally, the resulting PSO and DE algorithms are benchmarked against existing solution strategies.

4.1 Alternative particle representations

It is well known that the existence of separate search spaces have a significant impact on the performance of optimization algorithms [99], thus justifying research into alternative problem representations for the *Optimatix* problem. In the context of PSO and DE, the performance of problem representations may be considered to be relatively independent of search methodology. The purpose of this section is, thus, to investigate alternative problem representations and mapping mechanisms in the context of PSO. To achieve this objective, three PSO-based heuristics, namely the priority-based PSO (P-PSO), the random keys PSO (RKPSO), and the rule-based PSO (RBPSO), were developed.

Although all empirical performance evaluations throughout this and the next chapter were conducted on data sets ranging from 56 to 256 operations, a smaller problem, similar in complexity to the *Optimatix* problem, can be used to compare the various algorithms on a conceptual level. Four operations, indicated by means of the network diagram in Figure 4.1, need to be scheduled. Precedence constraints, denoted by arrows between the operations, are also present. Each operation needs to be scheduled on a single resource from a set of primary resources and a single auxiliary resource may also be required. The primary and auxiliary resource sets associated with each operation are respectively denoted by oval and rectangular blocks linked to each operation. Consider the fourth operation in Figure 4.1 as an example. This operation may be scheduled on either the first or second resource (primary resource allocation), but also requires the third resource as an auxiliary resource. The network diagram can, subsequently, be converted to a particle representation. The final solution (before the inclusion of production down time) can then be converted into the feasible schedule indicated by means of a gantt chart.

Throughout the rest of this section each of the three PSO-based heuristics are described and illustrated by means of the example problem. The experimental conditions of the empirical analysis are also described and the results of the investigation are presented.

4.1.1 The priority-based PSO algorithm

The priority-based algorithm is based on the idea that evolving operation priorities over time may lead to superior operation sequences. The particle representation of the priority-based PSO (P-PSO) algorithm consists of a $(2n - \varphi)$-dimensional vector, where φ is the number of operations which may be processed on only one primary resource and n is the total number of operations which need to be scheduled. The allocation of auxiliary resources are enforced as additional problem constraints.

Dimensions $n + 1$ to $2n - \varphi$ are used to represent the allocation of operations to primary resources. This is done by discretizing the search space as follows: For each

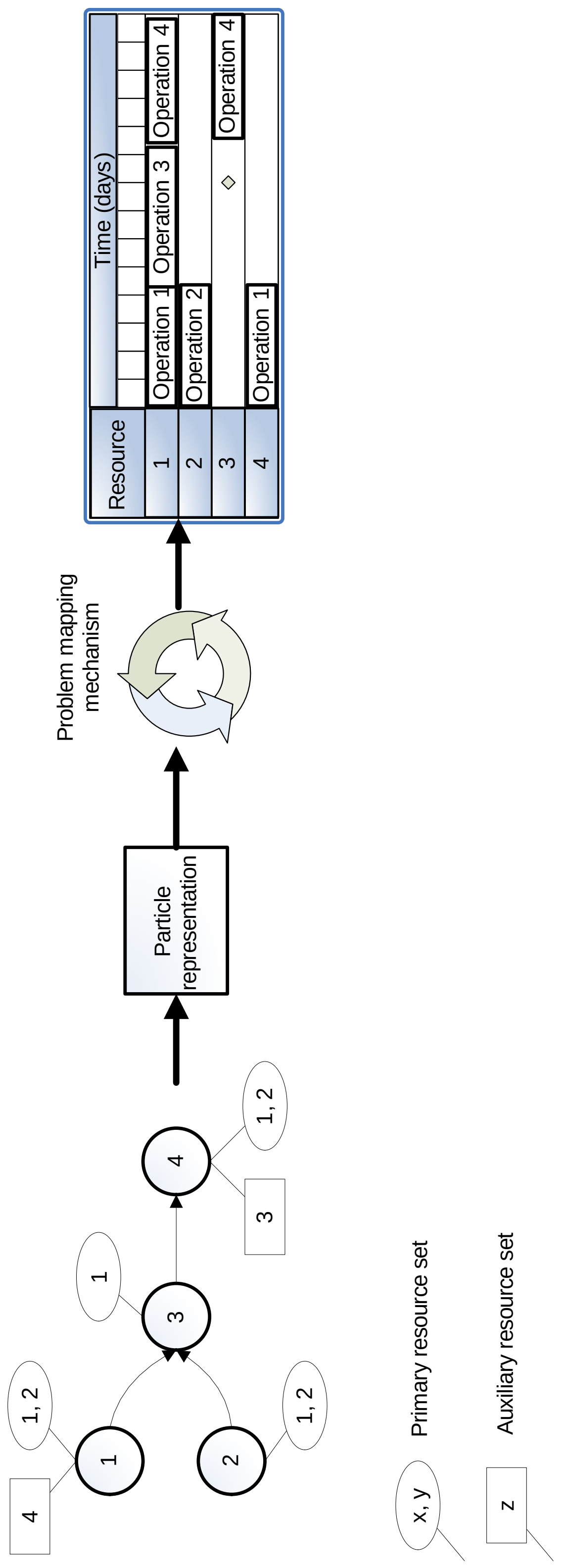


Figure 4.1: A small example problem and possible solution

operation k, the k^{th} dimension of the $\boldsymbol{P}_s$ space is divided into $|\boldsymbol{Q}_k|$ intervals, where $|\boldsymbol{Q}_k|$ denotes the number of primary resources on which operation k can be processed. Since each interval is associated with a unique integer number or resource index, dimensions $n+1$ to $2n-\varphi$ of the position vector can easily be interpreted as resource allocation variables.

Figure 4.1 shows that operation 1 may be scheduled on either resource 1 or resource 2. The first dimension of the $\boldsymbol{P}_s$ space is subsequently divided into two intervals, namely $\{-1500, 0\}$ and $\{0, 1500\}$, given that $\boldsymbol{x} \in \{-1500, 1500\}$. In the example particle representation i in Figure 4.3, $x_{i5} = 10.2$, where x_{ij} denotes the j^{th} dimension of particle i. Since $10.2 \in \{0, 1500\}$ operation 1 is scheduled on resource 2. Similarly, operation 2 and 4 are also scheduled on resource 2 and operation 3 is scheduled on resource 1.

The sequencing variables of dimensions 1 to n denote the priority values of each of the operations. These priorities are used as input to a schedule-building heuristic which attempts to schedule each operation at the earliest available time on its associated resource. In this dissertation, Giffler and Thompson's heuristic [47] (initially developed in 1960 and since then successfully used by Sha and Hsu [127] and Gao *et al.* [45]) was extended to include the unique problem characteristics of the *Optimatix* environment.

The modified Giffler and Thompson heuristic is given in Algorithm 4.1. The basic idea of the algorithm is to maintain two sets of operations. As can be seen in the pseudocode, the first set, $\boldsymbol{\Theta}$, stores all operations which are available for scheduling at a specific time. The operations in $\boldsymbol{\Theta}$ are inserted one by one into the partial schedule, $\boldsymbol{\Phi}$, according to their associated priority values. Inserting the operation with the highest priority, k, into $\boldsymbol{\Phi}$ involves considering each element in a set of possible starting times, $\boldsymbol{\Psi}_k$ (refer to Figure 4.2). The earliest starting time which allows for a feasible auxiliary resource allocation determines the position of operation k in the partial schedule. As soon as an operation is scheduled all its successors, which have no other unscheduled predecessors, become available for scheduling. These operations can be inserted into $\boldsymbol{\Theta}$ before the next operation is scheduled. As an example, consider the priorities of the example problem in Figure 4.3. These priorities can be converted into the job permutation: $\{1, 4, 3, 2\}$. The application of the modified Giffler and Thompson's heuristic [47] results in the schedule in Figure 4.3 being obtained.

```
Initialize $\mathbf{\Phi} = \emptyset$
Initialize $\mathbf{\Theta}$ to contain all operations without any predecessors
while $\mathbf{\Theta} \neq \emptyset$ do
    Select $k$ from $\mathbf{\Theta}$ as the operation with the highest priority
    Determine $\mathbf{\Psi}_k$ (the set of possible starting times for operation $k$ on resource $d_k$)
    while $k \in \mathbf{\Theta}$ do
        Set $t_k = min\{\mathbf{\Psi}_k\}$
        Calculate the finishing time associated with $t_k$: $z_k$ = Finish($t_k$)
        (Refer to Algorithm 4.2)
        Determine feasibility of schedule with respect to auxiliary resources
        (Refer to Algorithm 4.3)
        if $t_k$ results in a feasible schedule then
            Delete $k$ from $\mathbf{\Theta}$
        else
            Delete $t_k$ from $\mathbf{\Psi}_k$
        end
    end
    Insert $k$ into $\mathbf{\Phi}$
    for All successors $m$ of $k$ do
        if All other predecessors of operation $m \in \mathbf{\Phi}$ then
            Insert $m$ into $\mathbf{\Theta}$
        end
    end
end
```

Algorithm 4.1: The priority-based PSO mapping mechanism

The most involved step of the modified Giffler and Thompson algorithm is determining whether a feasible auxiliary resource allocation exists for a specific operation starting time. A two step process is required. Firstly, the effect of the resource-specific down time intervals on the operation duration needs to be determined in order to obtain an

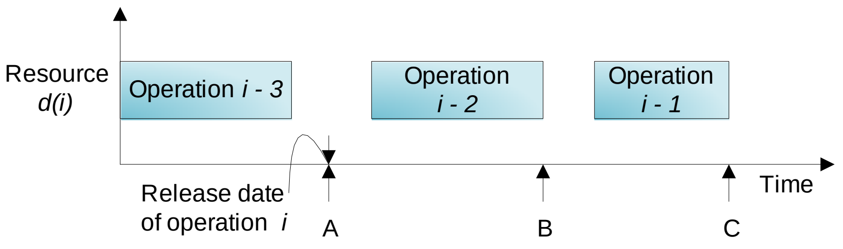


Figure 4.2: When operations $k-3$,$k-2$ and $k-1$ are already scheduled on resource d_k, the possible starting times of operation k, $\mathbf{\Psi}_k$, are A, B and C

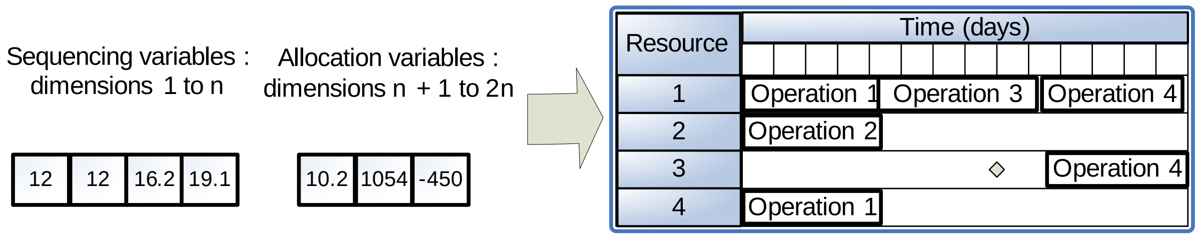


Figure 4.3: An example PPSO particle representation and corresponding solution to the example problem

accurate finishing time. Secondly, the availability of the required auxiliary resources during the operation processing time needs to be evaluated.

More detail with respect to the inclusion of production down time into the schedule is provided in Algorithm 4.2. Depending on the available data, either a sequence-independent resource-dependent set-up time or a sequence-dependent resource-independent set-up time may be used to calculate the processing time of an operation. Here $S_{\kappa_k k}$ denotes the sequence-dependent set-up time if operation k is processed immediately after operation κ_k on the same resource.

The second half of the pseudocode of Algorithm 4.2 describes the procedure followed to incorporate the down time intervals into the operation processing time. The starting times, $\boldsymbol{\psi}$, and ending times, $\boldsymbol{\chi}$, of all the resource down time intervals of resource d_k needs to be provided as input data. If the proposed starting time of operation k falls within any of the predefined down time intervals, the operation will only start once the resource is available again. If a resource is expected to become unavailable in the midst of an operation's processing time, the operation will be scheduled to resume on the same resource once it becomes available again. The processing time of the operation is then adjusted to allow for this period of inactivity.

```
for All operations $k$ do
    $z_k = t_k + p_{kd_k} + S_{\kappa_k k}$
    for All downtime intervals $p$ do
        if $z_k \leq \psi_p$ or $t_k \geq \chi_p$ then
            Interval $p$ is an intersected downtime interval of operation $k$
        end
    end
    for Intersected downtime intervals $p$ of operation $k$ do
        if $\psi_p \leq z_k$ and $t_k \leq \chi_p$ then
            $z_k = z_k + \chi_p - \psi_p$
        end
        if $\psi_p < t_k$ and $t_k \leq \chi_p$ then
            $z_k = z_k + \chi_p - t_k$
        end
    end
    for All downtime intervals $q$ from $p$ to $P$ do
        if $z_k > \psi_q$ then
            $z_k = z_k + \chi_q - \psi_q$
        else
            Break to operation $k + 1$
        end
    end
end
```

Algorithm 4.2: Calculating the finishing time associated with starting time t_k of operation k (Finish(t_k))

Once an accurate idea of the operation processing time on the required primary resources is obtained, the pseudocode of Algorithm 4.3 can be applied. For each auxiliary resource in the set of available auxiliary resources, $\boldsymbol{V}_k$, the algorithm checks whether an auxiliary resource is available during the proposed processing time of operation k. If no such resource exists, operation k cannot be scheduled at the proposed starting time and needs to be rescheduled on its associated primary resource.

```
for All operations $k$ do
    if $\boldsymbol{V}_k \neq \emptyset$ then
        for All resources $l \in \boldsymbol{V}_k$ do
            for All scheduled intervals $p$ do
                if $z_k \leq \chi_{pl}$ or $t_k \geq \psi_{pl}$ then
                    Operation $k$ will overlap interval $p$
                end
            end
            if Operation $k$ overlaps any intervals then
                Operation $k$ cannot be scheduled on resource $l \in \boldsymbol{V}_k$
                if $l = |\boldsymbol{V}_k|$ then
                    Operation $k$ is infeasible
                    Break to operation $k+1$
                else
                    Break to resource $l+1$
                end
            else
                Schedule operation $k$ on resource $l \in \boldsymbol{V}_k$
            end
        end
    end
end
```

Algorithm 4.3: Allocation of operations to auxiliary resources

The P-PSO algorithm is considered useful since the continuous nature of the PSO algorithm can be exploited to solve a very complex, discrete combinatorial optimization problem [127]. However, Sha and Hsu [127] also mention a characteristic of concern for scheduling algorithms which utilize a priority-based fitness function evaluation mechanism. A very small change in the position of the particle within the $\boldsymbol{P}_s$ space may result in a very large change in the $\boldsymbol{S}_s$ space. The algorithm, additionally, has the property that many different solutions within $\boldsymbol{P}_s$ map to the same solution in $\boldsymbol{S}_s$.

The last problematic aspect of a priority-based mapping mechanism is the effect which the schedule-building heuristic has on the simultaneous optimization of the specific set of multiple objectives required by *Optimatix*. Since the schedule-building mechanism attempts to position each operation at the earliest possible time, the algorithm is, in fact biased towards the minimization of makespan. However, this statement is also heavily dependent on the characteristics of the problem being solved. As an example, consider the scheduling of a single operation on a single resource while simultaneously minimizing earliness/tardiness and makespan. If the earliest starting time is larger than the due date of the job to which the operation belongs, minimizing makespan also minimizes the earliness/tardiness objective. However, if the earliest starting time is smaller than the due date of the associated job, the two objectives become conflicting, and the solution and subsequent fitness calculation is distorted.

4.1.2 The random keys PSO algorithm

The random keys PSO (RKPSO) algorithm is a direct application of Norman and Bean's random keys genetic algorithm (RKGA) [99] to the PSO paradigm. The gene representation of the RKGA consists of an n-dimensional vector in contrast with the $2n - \varphi$ dimensions required for the P-PSO algorithm. A sorting mechanism (which is given in Algorithm 4.4) is used to decode the real-valued n-dimensional vector into its corresponding resource indices and priorities. The mechanism simply interprets the integer component of the particle dimension as a primary resource index. The decimal component is taken to be the priority of the associated operation for sequencing purposes. Giffler and Thompson's heuristic [47] can then be applied directly. The example in Figure 4.4 shows that the resource allocation decision is again addressed through discretization of the search space. However, now $x_{ij} \in \{0, 1\}$, where x_{ij} is the j^{th} dimension of the i^{th} particle representation.

The RKPSO has the important advantage that the dimensionality of the $\boldsymbol{P}_s$ space is

```
for All operations k do
    $l = (|\lfloor x_{ik} \rfloor| \bmod |\boldsymbol{Q}_k|) + 1$
    $d_k = Q_{kl}$
    $t_k = x_{ik} - |\lfloor x_{ik} \rfloor|$
end
```

Algorithm 4.4: The random keys PSO sorting mechanism as applied to particle i

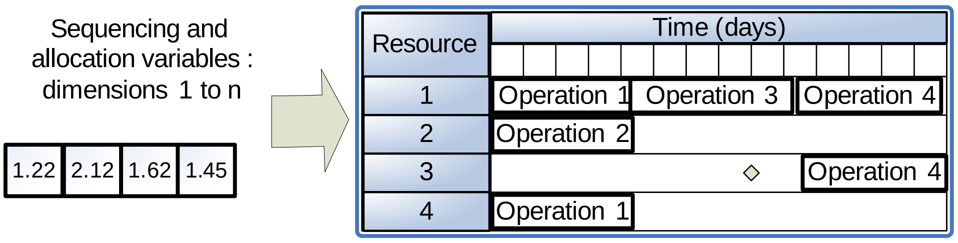


Figure 4.4: An example RKPSO particle representation and corresponding solution to the example problem

halved. However, the limitations of Giffler and Thompson's heuristic [47], as identified in the previous section, are still applicable. Furthermore, in most traditional optimization applications one dimension is used to denote one unique and separate concept. Here both operation sequence and resource allocation is represented by a single dimension, which could have significant implications for algorithm performance.

4.1.3 The rule-based PSO algorithm

The rule-based PSO (RBPSO) algorithm [53] is another attempt at reducing the $\boldsymbol{P}_s$ space. This strategy was inspired by both the rule-based algorithms currently used by *Optimatix*, as well as elements of Kacem *et. al*'s genetic algorithm-based approach to flexible job shop scheduling [62]. Similar to the RKPSO, the particle representation consists of one n-dimensional vector which represents the sequencing variables. However, the resource allocation is now performed within the schedule-building mechanism as described in Algorithm 4.5.

The only difference between Algorithm 4.5 and Algorithm 4.1 is that the resource allocation now involves considering each primary resource in the set, $\boldsymbol{Q}_k$. The resource allocation, l_{min}, which results in the smallest finishing time, $z_{kl_{min}}$, is subsequently selected. An illustration of the algorithm is, again, provided in Figure 4.5.

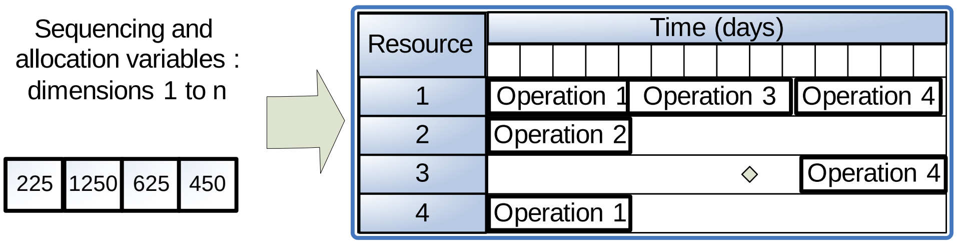


Figure 4.5: An example RBPSO particle representation and corresponding solution to the example problem

It should, however, be noted that even though the dimensionality of the $\boldsymbol{P}_s$ space is reduced, this is done at the cost of a more computationally complex algorithm since an explicit search of all possible resource allocations are performed for each particle during the schedule construction phase.

4.1.4 Comparative analysis of alternative particle representations

The purpose of this section is to evaluate alternative particle representations by comparing the performance of the P-PSO, RKPSO, and RBPSO algorithms. However, for an investigation into algorithm performance to be most effective, it is important to conduct such an experimental analysis under the same conditions under which the algorithms will eventually be used. It should be noted that this study is not only focused on improving the existing scheduling algorithms of *Optimatix*, but rather in identifying those requirements that if addressed will best meet the needs of *Optimatix*' clients and then to develop an effective solution which addresses all of these requirements. To achieve these objectives, three test problems corresponding to actual problem size and complexity were derived from actual customer data and were adapted, as described in the rest of this section, to incorporate the changing customer requirements of *Optimatix*. Un-

```
Initialize $\boldsymbol{\Phi} = \emptyset$
Initialize $\boldsymbol{\Theta}$ to contain all operations without any predecessors
while $\boldsymbol{\Theta} \neq \emptyset$ do
    Select $i$ from $\boldsymbol{\Theta}$ as the operation with the highest priority
    for All resources $l \in \boldsymbol{Q}_k$ on which operation $k$ may be scheduled do
        Determine $\boldsymbol{\Psi}_k$ (the set of possible starting times for operation $k$ on resource $d_k$)
        while $k \in \boldsymbol{\Theta}$ do
            Set $t_{kl} = min\{\boldsymbol{\Psi}_k\}$
            Calculate the finishing time associated with $t_{kl}$: $z_{kl} = \text{Finish}(t_{kl})$
            (Refer to Algorithm 4.2)
            Determine feasibility of schedule with respect to auxiliary resources
            (Refer to Algorithm 4.3)
            if $t_{kl}$ results in a feasible schedule then
                Delete $k$ from $\boldsymbol{\Theta}$
            else
                Delete $t_{kl}$ from $\boldsymbol{\Psi}_k$
            end
        end
        Insert $k$ into $\boldsymbol{\Theta}$
    end
    $z_{kl_{min}} \leftarrow min_{l \in \boldsymbol{Q}_k}\{z_{kl}\}$
    Set $t_k = t_{kl_{min}}$
    Delete $k$ from $\boldsymbol{\Theta}$
    Insert $k$ into $\boldsymbol{\Phi}$
    for All successors $m$ of $k$ do
        if All other predecessors of operation $m \in \boldsymbol{\Phi}$ then
            Insert $m$ into $\boldsymbol{\Theta}$
        end
    end
end
```

Algorithm 4.5: The rule-based PSO mapping mechanism

less otherwise indicated, all performance analyses in this dissertation were conducted on these data sets.

The effectiveness of a scheduling solution is, furthermore, highly dependent on the realism of the solution. In other words, it is important that the actual solution obtained corresponds to the requirements of the production environment which is to be scheduled. By effectively modeling the actual production environment, the number of times rescheduling is required as well as the disruptions associated with frequent rescheduling, can be drastically reduced. The time required to customize scheduling algorithms for each client's unique production environment can also be reduced when the level of generality of a consulting firm's scheduling algorithms is increased. To achieve these objectives, the existing customer data sets were extended to include resource-dependent processing times and sequence-dependent set-up times.

The variation, σ_B^2, of all operation process times processed on different resources was calculated from Kacem *et. al*'s benchmark data set for flexible job shop scheduling problems [63], which address both the allocation of operations to resources and the sequencing of these operations on their associated resources. Subsequently, the processing times and set-up times of the data sets used in this dissertation were randomly generated within the interval $[\mu_B - \sigma_B, \mu_B + \sigma_B]$, where μ_B denotes the operation-dependent data point as obtained from the original customer data set. In alignment with customer requirements, half of the sequence-dependent set-up times were initialized to zero. The data sets range in size from 56 to 256 operations which are to be scheduled on 216 resources and are available for comparison purposes from the author.

The results of the performance evaluation for the three PSO-based heuristics were recorded over 30 independent simulation runs over each of the three data sets. Both accuracy and computational complexity were considered to be important performance measurements. Throughout the rest of this dissertation, f_1 denotes makespan, f_2 the earliness/tardiness criteria, and f_3 the queue time. In this chapter, goal programming, which minimizes the weighted deviation between each fitness function value and a target value set for it, was used to address the multiple objectives. The aggregated fitness function, f_4, is given as

$$f_4 = \sum_{s=1}^{3} |(f_s - g_s)| + \upsilon f_s \tag{4.1}$$

where g_s denotes the target value of the s^{th} fitness function, f_s, and υ is selected as sufficiently small. Throughout the rest of this dissertation, μ and σ respectively denote the mean and standard deviation associated with the corresponding performance measurement and $\boldsymbol{CI}_{0.05}$ is a 95% confidence interval on μ. Finally, g_1 and g_2 were set to 250, and g_3 was set to 0.

Analysis of the behaviour of the priority-based PSO on the 56-operation problem resulted in the parameter values listed in Table 4.1 being defined as suitable for initial comparison purposes. The number of particles in the swarm is denoted by n_s and the size of the discretization intervals, δ_k, of operation k is dependent on λ, where

$$\delta_k = \frac{2\lambda}{|\boldsymbol{Q}_k|}. \tag{4.2}$$

$|\boldsymbol{Q}_k|$ is the number of resources on which operation k may be scheduled, λ is the number of discretization intervals which are allocated to a single operation-resource pair, and $a \longrightarrow b$ indicates that the associated parameter is decreased linearly from a to b over 95% of the total number of iterations, I_{max}. Finally, the sequencing and resource allocation variables were initialized from a rectangular distribution within the intervals $\{-500, 500\}$ and $\{-1500, 1500\}$.

The actual results of the investigation into alternative problem mapping mechanisms and particle representations are recorded in Tables 4.2 and 4.3 and Figure 4.6. The RBPSO outperformed both the RKPSO and P-PSO with respect to all four objective functions for the two larger problem instances. No statistically significant difference could be identified between the RBPSO and the P-PSO on the 56 operation problem when the aggregated objective function was considered. The RBPSO did, however, outperform the P-PSO with respect to the makespan objective.

Even though the RBPSO was clearly the best algorithm with respect to solution quality, this was not the only criterion which had to be considered for the *Optimatix* environment. During the empirical evaluation, it was evident that the RBPSO algorithm

Table 4.1: There are a number of parameters which have a significant effect on the performance of PSO

Parameter	**Value used**
n_s	27
λ	3
I_{max}	200
c_1	$2.0 \longrightarrow 1.0$
c_2	$2.8 - c_1$
w	$0.8 \longrightarrow 0.4$

took significantly longer to return a solution than the other two algorithms. An analysis of the complexity of the three algorithms was subsequently performed.

Table 4.2: Experimental results of alternative mapping strategies and particle representations with respect to makespan and earliness/tardiness

Pro-	**Algo-**	f_1			f_2		
blem	**rithm**	μ	σ	$CI_{0.05}$	μ	σ	$CI_{0.05}$
56-op	P-PSO	1582.37	7.76	± 2.90	3563.24	316.19	± 118.05
	RKPSO	2086.04	109.26	± 3.83	4060.96	353.53	± 132.00
	RBPSO	1567.71	0.00	± 0	3546.47	331.85	± 123.90
100-op	P-PSO	1862.46	13.90	± 5.19	7045.34	378.46	± 141.31
	RKPSO	2271.42	128.58	± 48.01	8024.63	809.16	± 302.11
	RBPSO	1799.55	50.99	± 19.04	6431.98	369.90	± 138.11
256-op	P-PSO	5059.85	153.17	± 57.19	39007.07	2789.33	± 1041.44
	RKPSO	6191.17	731.57	± 273.14	38382.25	4102.06	± 1531.56
	RBPSO	4922.67	62.13	± 23.20	30892.79	3508.45	± 1309.93

Assume that the complexity of the initialization of $\mathbf{\Phi}$ and $\mathbf{\Theta}$ (in Algorithm 4.1) is

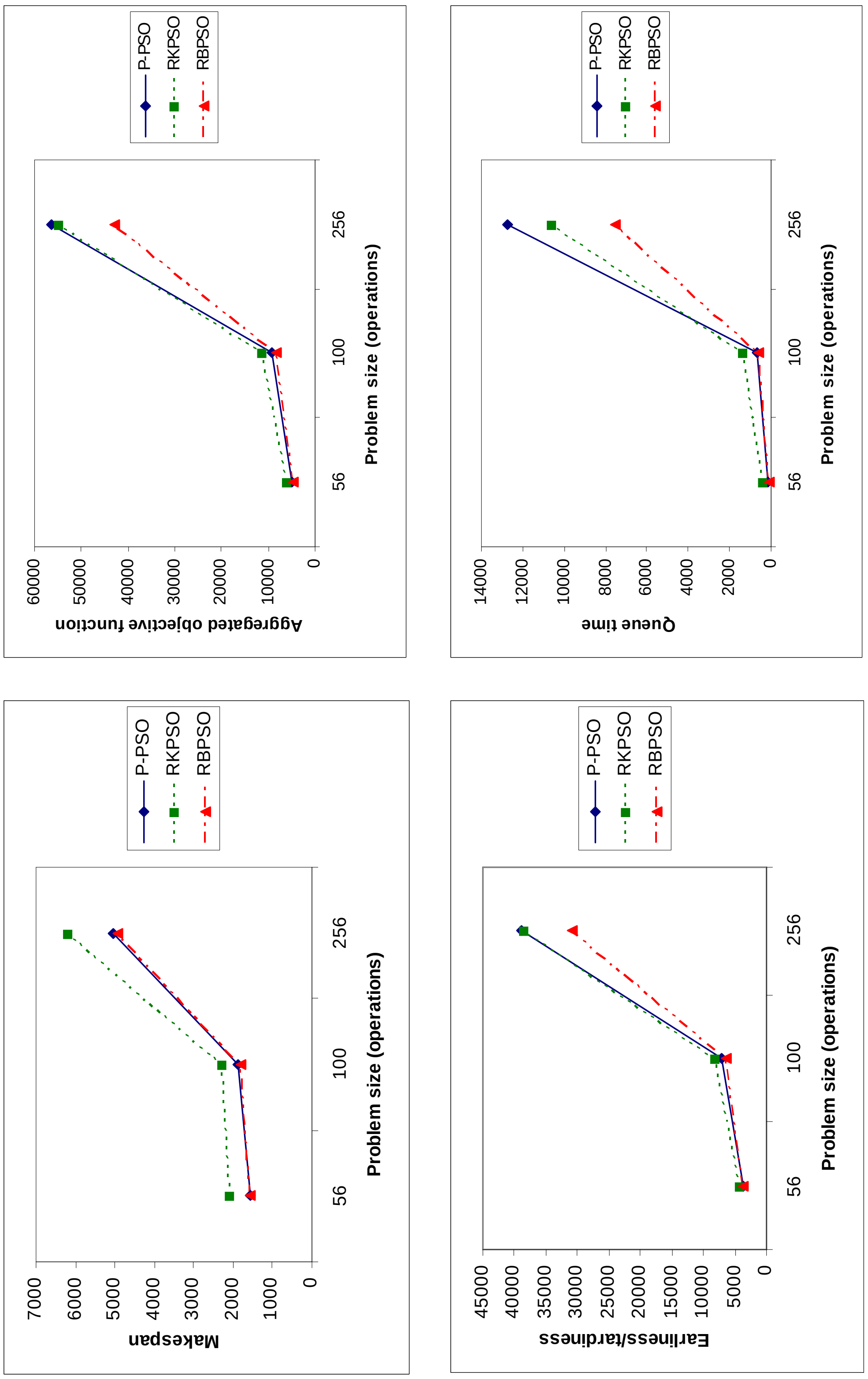


Figure 4.6: Results of the investigation into alternative particle representations

Table 4.3: Experimental results of alternative mapping strategies and particle representations with respect to queue time and the aggregated objective function

Pro-	Algo-	f_3			f_4		
blem	rithm	μ	σ	$CI_{0.05}$	μ	σ	$CI_{0.05}$
56-op	P-PSO	137.56	157.39	± 58.76	4788.46	390.82	± 145.92
	RKPSO	389.39	302.35	± 112.89	6042.93	609.69	± 227.64
	RBPSO	107.76	158.77	± 59.30	4727.16	429.11	± 160.21
100-op	P-PSO	685.03	254.60	± 95.06	9102.42	574.98	± 214.68
	RKPSO	1376.47	452.24	± 168.85	11184.20	1248.05	± 465.98
	RBPSO	629.22	388.59	± 145.09	8369.60	775.02	± 289.36
256-op	P-PSO	12760.31	2704.64	± 1009.82	56384.06	5223.65	± 1950.32
	RKPSO	10539.85	3118.69	± 1164.41	54668.49	7143.55	± 2667.15
	RBPSO	7504.28	2782.99	± 1039.07	42863.06	5982.83	± 2233.77

of order X and the complexity of determining the actual position where an operation should be inserted into the schedule is of order Y. If the actual insertion process of an operation is of order Z, then the complexity of the mapping mechanism of the P-PSO algorithm can be shown to be of order $nn_{Q_{max}} + X + nY + nZ$, where $n_{Q_{max}}$ is the maximum number of primary resources on which an operation may be scheduled. The complexity of the mapping mechanism associated with the RKPSO can then be shown to be of order $n + nn_{Q_{max}} + X + nY + nZ$.

It should be noted that both the RKPSO and P-PSO algorithms make use of a pre-optimization process where the search space is discretized for the purposes of resource allocation. However, this discretization, which is of order $nn_{Q_{max}}$, is only performed once per simulation. The mapping mechanism, on the other hand, is executed once for every individual during each iteration. The RBPSO does not make use of a pre-optimization process, however, the mapping mechanism is of order $X + nn_{Q_{max}}Y + nZ$, which is significantly more complex than the mapping mechanism of the P-PSO algorithm since $O(nn_{Q_{max}}Y) >> O(nY)$.

In summary, when both computational complexity and solution time is considered, P-PSO is more suited to the *Optimatix* requirements since it is computationally much less complex than the RBPSO algorithm, while still producing satisfactory results on most of the tested problems.

4.2 Investigating alternative PSO topologies and DE base vector selection strategies

From the previous section it became clear that the P-PSO was best suited to meeting the *Optimatix* requirements. This investigation, however, led to a number of additional questions. For example: (1) Would a priority-based DE (P-DE) algorithm perform similarly? (2) Which variation on the classic DE algorithm is best suited to the problem? and (3) Will alternative PSO topologies have a positive impact on algorithm performance? To answer these questions, this section compares the *gbest* guaranteed convergence priority-based PSO (*gbest* GCP-PSO) algorithm of the previous section to a Von Neumann guaranteed convergence priority-based PSO (Von Neumann GCP-PSO) algorithm. In addition, a P-DE algorithm is implemented and an investigation into alternative base vector selection strategies is performed. The base vector selection strategies which are investigated include P-DE/best/bin, P-DE/rand/bin, and P-DE/rand-to-best/bin (P-DE/R2B/bin) algorithms.

4.2.1 Comparative analysis of alternative PSO and DE strategies

The selection of suitable algorithm parameters for performance evaluation purposes becomes even more important when algorithms with different characteristics and structures are compared. In this study the algorithm parameters were first optimized for each algorithm-data set pair before attempting to comment on the relative performance of different PSO and DE strategies. For each algorithm-data set pair, 144 uniformly distributed parameter combinations were selected for evaluation. Algorithm performance was then recorded over 30 simulations for each unique parameter combination. Selecting the "best" parameter combination from the 144 combinations was the most problematic, since parameter optimization may be considered to be a multi-objective optimization problem.

The purpose of multi-objective optimization is to find a set of trade-off solutions referred to as the Pareto-optimal set, $\boldsymbol{PF}$, where

$$\boldsymbol{PF} = \{\boldsymbol{x}^* \in \boldsymbol{f} \mid \quad \nexists \quad \boldsymbol{x} \in \boldsymbol{f} : \boldsymbol{x}^* \prec \boldsymbol{x}\}. \tag{4.3}$$

The dominance relation, $\prec$, indicates that solution $\boldsymbol{x}^*$ dominates solution $\boldsymbol{x}$, i.e. $\boldsymbol{x}^*$ is not worse than $\boldsymbol{x}$ in all objectives and $\boldsymbol{x}^*$ is strictly better than $\boldsymbol{x}$ in at least one objective.

Parameter derivation can be considered as an optimization problem with the purpose of selecting suitable parameter values such that both solution quality and algorithm robustness, as determined by the standard deviation, is maximized. To solve this multi-objective optimization problem, all dominated solutions were first removed from the set of 144 parameter combinations. A Pareto front of possible parameter combinations which formed a trade-off curve between accuracy and robustness remained. With this trade-off curve as input, the impact of different parameter combinations on the accuracy and robustness of the different scheduling objectives could be considered and appropriate parameter values could be selected. The best parameter combination for each algorithm-data set pair is, subsequently, indicated in Table 4.4.

The next phase of the study compared each of the DE base vector selection strategies and PSO topologies on the three *Optimatix* data sets. Apart from the parameter optimization, the same experimental conditions, as described in the previous section, were used. The results are recorded in Tables 4.5 and 4.6 and Figure 4.7.

A number of interesting observations could be made from the results. Firstly, no statistically significant difference could be identified between the results obtained by the alternative DE-based strategies. This was true for all three problems and all four objective functions. Although it was disappointing that the use of alternative base vector selection mechanisms did not lead to performance improvement, this is actually a desirable algorithm characteristic. It indicates that, as long as appropriate algorithm parameters are used, the priority-based DE algorithm is robust with respect to the use

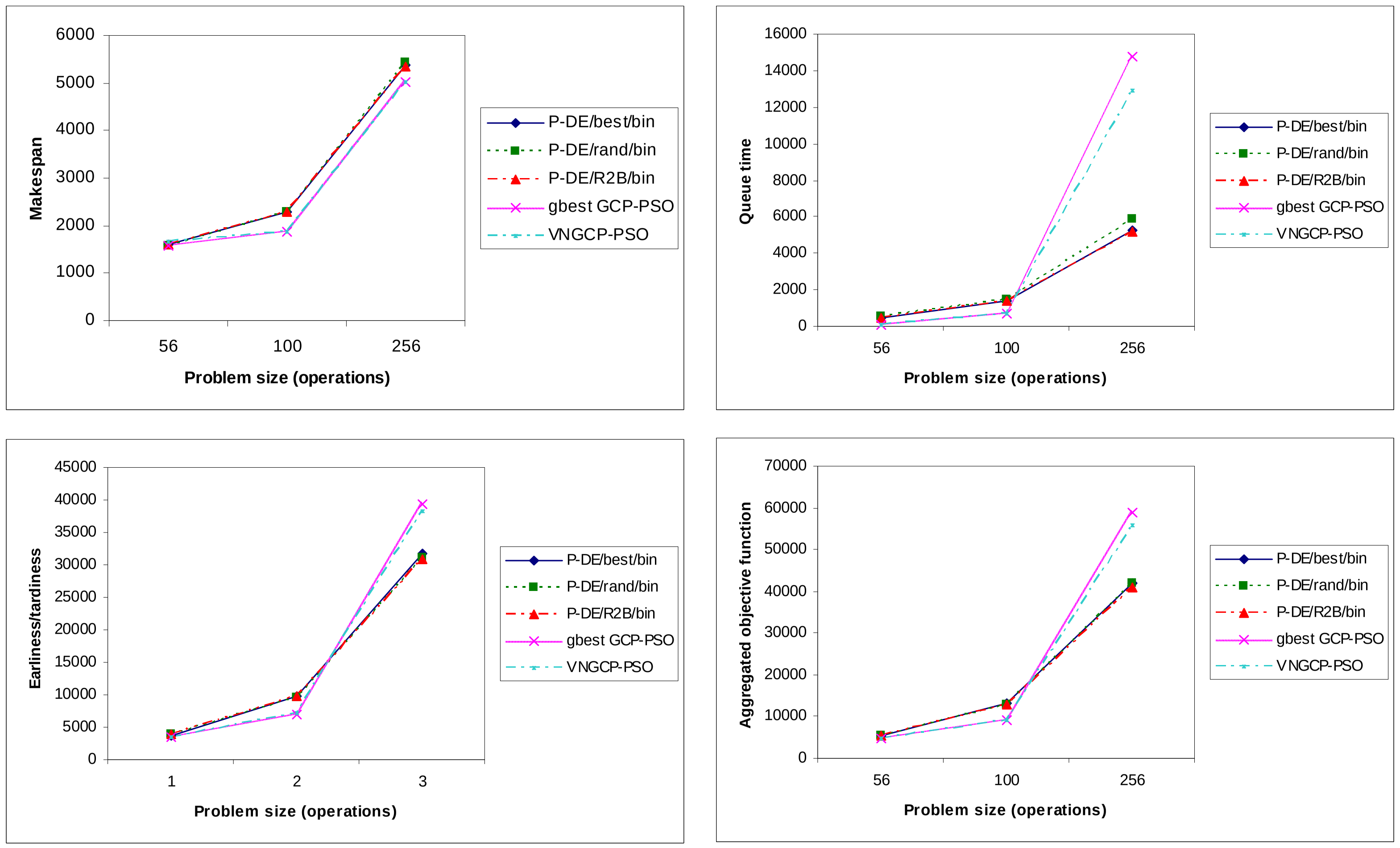


Figure 4.7: Results of the investigation into alternative PSO topologies and DE base vector selection strategies

Table 4.4: PSO and DE parameter derivation results

Problem	**Algorithm**	p_c/w	F/c_1	c_2
56-op	P-DE/best/bin	$0.5 \longrightarrow 0.1$	$0.9 \longrightarrow 0.1$	-
	P-DE/rand/bin	$0.5 \longrightarrow 0.3$	$0.7 \longrightarrow 0.1$	-
	P-DE/R2B/bin	$0.5 \longrightarrow 0.3$	$0.7 \longrightarrow 0.1$	-
100-op	P-DE/best/bin	$0.5 \longrightarrow 0.1$	$0.9 \longrightarrow 0.1$	-
	P-DE/rand/bin	0.5	$0.5 \longrightarrow 0.3$	-
	P-DE/R2B/bin	$0.5 \longrightarrow 0.3$	$0.7 \longrightarrow 0.5$	-
256-op	P-DE/best/bin	$0.7 \longrightarrow 0.5$	$0.7 \longrightarrow 0.5$	-
	P-DE/rand/bin	$0.5 \longrightarrow 0.1$	$0.7 \longrightarrow 0.3$	-
	P-DE/R2B/bin	$0.7 \longrightarrow 0.1$	0.5	-
56-op	*gbest* GCP-PSO	$0.9 \longrightarrow 0.5$	$2.2 \longrightarrow 1.0$	$1.0 \longrightarrow 2.2$
	VNGCP-PSO	$1.1 \longrightarrow 0.72$	$1.8 \longrightarrow 0.2$	$0.2 \longrightarrow 1.8$
100-op	*gbest* GCP-PSO	$0.9 \longrightarrow 0.3$	$2.6 \longrightarrow 1.4$	$1.4 \longrightarrow 2.6$
	VNGCP-PSO	$0.9 \longrightarrow 0.3$	$2.6 \longrightarrow 0.2$	$0.2 \longrightarrow 2.6$
256-op	*gbest* GCP-PSO	$1.1 \longrightarrow 0.5$	$1.4 \longrightarrow 0.6$	$0.6 \longrightarrow 1.4$
	VNGCP-PSO	$1.1 \longrightarrow 0.3$	$1.4 \longrightarrow 0.2$	$0.2 \longrightarrow 1.4$

of alternative base-vector selection strategies. This finding holds significant implications for the effort required for algorithm customization.

The P-PSO algorithms were slightly more sensitive to the use of alternative topologies. The *gbest* GCP-PSO algorithm outperformed the VNGCP-PSO algorithm on the 56 operation problem. This difference in performance with respect to the aggregated objective function could be traced back to a difference in queue time obtained by the two algorithms. Fortunately, no other statistically significant performance differences could be identified indicating that the P-PSO algorithms also seem to be relatively robust with respect to alternative PSO topologies.

The final phase of the analysis focused on directly comparing the use of P-PSO ver-

Table 4.5: A comparison of the performance of various DE base vector selection strategies and PSO topologies with respect to makespan and earliness/tardiness

Pro-	**Algo-**	f_1			f_2		
blem	**rithm**	μ	σ	$CI_{0.05}$	μ	σ	$CI_{0.05}$
56	P-DE/best/bin	1592.88	22.38	± 8.36	3796.47	259.22	± 96.78
	P-DE/rand/bin	1588.30	4.56	± 1.70	3907.23	192.41	± 71.84
	P-DE/R2B/bin	1594.19	24.71	± 9.23	3861.50	231.19	± 86.32
100-op	P-DE/best/bin	2293.15	58.17	± 21.72	9823.74	301.93	± 112.73
	P-DE/rand/bin	2282.50	54.65	± 20.40	9666.55	248.23	± 92.68
	P-DE/R2B/bin	2276.37	66.72	± 24.91	9754.20	227.96	± 85.11
256-op	P-DE/best/bin	5371.55	199.58	± 74.52	31705.76	1493.80	± 557.73
	P-DE/rand/bin	5426.02	158.52	± 59.19	31099.89	1082.29	± 404.09
	P-DE/R2B/bin	5343.74	143.92	± 53.73	30931.67	1018.09	± 380.12
56-op	*gbest* GCP-PSO	1581.92	17.55	± 6.55	3494.64	213.23	± 79.61
	VNGCP-PSO	1653.71	44.20	± 16.50	3579.17	231.81	± 86.55
100-op	*gbest* GCP-PSO	1867.66	13.53	± 5.05	6957.78	236.82	± 88.42
	VNGCP-PSO	1858.57	11.29	± 4.22	7151.72	235.12	± 87.79
256-op	*gbest* GCP-PSO	5007.81	50.17	± 18.73	39453.53	2618.41	± 977.62
	VNGCP-PSO	5010.93	45.57	± 17.01	38169.64	3065.49	± 1144.54

sus P-DE in the *Optimatix* environment. The best performing DE strategy and PSO topology (defined as the algorithm with the lowest mean) was subsequently identified for each of the problems and the two resulting algorithms were compared. The results obtained from this comparison indicated that the *gbest* GCP-PSO algorithm was the best performing algorithm for the two smaller problems, but the P-DE/R2B/bin outperformed the Von Neumann P-PSO on the 256-operation problem. It was thus obvious that no single DE or PSO strategy is superior for all three problems tested. Although the actual version of PSO or DE which were used does not seem to be that important,

Table 4.6: A comparison of the performance of various DE base vector selection strategies and PSO topologies with respect to queue time and the aggregated objective function

Pro-	**Algo-**	f_3			f_4		
blem	**rithm**	μ	σ	$CI_{0.05}$	μ	σ	$CI_{0.05}$
56-op	P-DE/best/bin	477.77	143.69	± 53.65	5372.99	390.66	± 145.86
	P-DE/rand/bin	525.40	107.15	± 40.01	5526.95	298.15	± 111.32
	P-DE/R2B/bin	497.38	122.14	± 45.60	5459.02	328.29	± 122.57
100-op	P-DE/best/bin	1422.32	257.26	± 96.05	13052.75	549.07	± 205.00
	P-DE/rand/bin	1467.37	228.83	± 85.44	12929.83	453.37	± 169.27
	P-DE/R2B/bin	1408.47	214.59	± 80.12	12952.48	418.30	± 156.18
256-op	P-DE/best/bin	5279.40	747.91	± 279.24	41899.06	2008.35	± 749.85
	P-DE/rand/bin	5882.99	1089.47	± 406.77	41951.31	1749.61	± 653.24
	P-DE/R2B/bin	5143.53	811.32	± 302.92	40960.35	1392.98	± 520.09
56-op	*gbest* GCP-PSO	74.46	127.93	± 47.76	4656.17	245.96	± 91.83
	VNGCP-PSO	90.64	129.21	± 48.24	4828.84	202.77	± 75.71
100-op	*gbest* GCP-PSO	675.02	211.25	± 78.87	9009.96	406.74	± 151.86
	VNGCP-PSO	675.70	214.73	± 80.17	9195.67	367.53	± 137.22
256-op	*gbest* GCP-PSO	14764.70	2093.57	± 781.66	58785.26	3927.42	± 1466.36
	VNGCP-PSO	12893.70	1788.01	± 667.58	55630.34	4088.77	± 1526.60

the choice of metaheuristic plays a more important role.

These findings makes sense from what is known about the problem-specific nature of metaheuristics. It is well known that the same metaheuristics's performance can fluctuate significantly over different problem instances and algorithms customized for specific problems usually tend to outperform more generally applicable algorithms. This knowledge is helpful in explaining the relative performance of the PSO and DE-based algorithms.

Deciding when to use which algorithm now becomes more important. The difference in performance between the P-PSO and P-DE algorithms could be either due to poor scalability on the part of one of the P-PSO algorithms or it could be that one algorithm is more suited to solving problems with specific characteristics. This issue is addressed in more detail in the next section. The algorithms are first benchmarked against existing solution strategies before a more detailed scalability analysis is done.

4.3 Benchmarking the priority-based algorithm against alternative solution strategies

To evaluate the significance of the previous results, it is important to compare the priority-based algorithms against existing benchmark algorithms. In this section the best P-PSO and P-DE algorithms for each data set, as determined in the previous section, are compared against three benchmark algorithms. The first two algorithms are existing *Optimatix* algorithms which are currently in use. Norman and Bean's random keys genetic algorithm (RKGA) [99] was identified as the most promising algorithm from existing literature, and was thus selected as the third benchmark algorithm. The rest of this section describes each of the benchmark algorithms in more detail before the results of the comparison are stated and discussed.

4.3.1 The existing *Optimatix* algorithms

Two of the existing *Optimatix* algorithms were selected as benchmark algorithms: the idea being to evaluate the improvements resulting from the use of the PSO-based heuristics instead of the existing algorithms. Both of these rule-based algorithms function on a very similar premise to the RBPSO discussed in the previous section, the only differences being the assignment of priorities to operations and the allocation of operations to the first available resource. While the operation priorities are evolved over time by a PSO-based algorithm in the RBPSO, the basic scheduling rule assigns priority values randomly to operations. The earliest due date rule (EDD) assigns operation priorities according to the earliest due date of the jobs corresponding to the operations under consideration. For the sake of completeness, the pseudocode for these algorithms is provided in Algorithm 4.6.

These rule-based benchmarking algorithms are based on the idea that a set, $\boldsymbol{I} \subset \boldsymbol{\Theta}$, of high priority operations can be identified at any stage during schedule construction. If, for example, the earliest due date rule (EDD) is used, this set includes all operations which need to be completed in time for the first delivery date. For each operation k, where $k \in \boldsymbol{I}$ the primary resource allocation is performed by selecting the first available primary resource, l_{min}, from the set of primary resources, $\boldsymbol{Q}_k$. The starting time, t_{kl}, associated with the allocation of operation k to resource l_{min} is then compared for all operations $k \in \boldsymbol{I}$. The operation which can be scheduled the soonest, denoted by k_{min}, is then scheduled first. The rest of the algorithm functions on a similar basis to the P-PSO and RBPSO algorithms.

It should be noted that the inclusion of the more complex customer requirements defined and discussed in Section 4.1.4 required that certain changes had to be made to the existing algorithms. If t_k, p_k, and s_k, respectively denote the starting time, processing time and set-up time of operation k, the actual finishing time is currently calculated using

$$z_k = t_k + p_k + s_k \tag{4.4}$$

The new requirements resulted in the total processing time calculation changing to

$$z_k = t_k + p_{kd_k} + s_k \tag{4.5}$$

where

$$s_k = \begin{cases} S_{\kappa_k k} & \text{if } S_{\kappa_k k} > 0 \\ h_{kd_k} & \text{otherwise} \end{cases} \tag{4.6}$$

as can be seen in Algorithm 4.1. Here p_{kd_k} denotes the resource-dependent processing time of operation k on resource d_k. The set-up time, s_k, can be either sequence-dependent, $S_{\kappa_k k}$, where $S_{\kappa_k k}$ denotes the set-up time of operation k if processed immediately after operation κ_k on the same primary resource, or resource-dependent, h_{kd_k}, where h_{kd_k} denotes the set-up time of operation k when processed on resource d_k. These changes resulted in the existing algorithms being suitable almost “as-is” for benchmarking purposes and no additional structural changes to the existing *Optimatix* algorithms were required.

```
Initialize $\boldsymbol{\Phi} = \emptyset$
Initialize $\boldsymbol{\Theta}$ to contain all operations without any predecessors
while $\boldsymbol{\Theta} \neq \emptyset$ do
    Select $\boldsymbol{I}$ from $\boldsymbol{\Theta}$ as the set of operations with the highest priority
    for All operations $k \in \boldsymbol{I}$ do
        for All resources $l \in \boldsymbol{Q}_k$ on which operation $k$ may be scheduled do
            Determine $\boldsymbol{\Psi}_k$ (the set of possible starting times of operation $k$)
            while $k \in \boldsymbol{\Theta}$ do
                Set $t_{kl} = min\{\boldsymbol{\Psi}_k\}$
                Calculate the finishing time associated with $t_k$: $z_k$ = Finish($t_k$)
                (Refer to Algorithm 4.2)
                Determine feasibility of schedule with respect to auxiliary resources
                (Refer to Algorithm 4.3)
                if $t_{kl}$ results in a feasible schedule then
                    Delete $k$ from $\boldsymbol{\Theta}$
                else
                    Delete $t_{kl}$ from $\boldsymbol{\Psi}_k$
                end
            end
            Insert $k$ into $\boldsymbol{\Theta}$
        end
        $t_{kl_{min}} \leftarrow min_{l \in \boldsymbol{Q}_k}\{t_{kl}\}$
    end
    $t_{k_{min}l_{min}} \leftarrow min_{k \in \boldsymbol{I}}\{t_{kl_{min}}\}$
    Delete $k_{min}$ from $\boldsymbol{\Theta}$
    Insert $k_{min}$ into $\boldsymbol{\Phi}$
    for All successors $m$ of $k_{min}$ do
        if All other predecessors of operation $m \in \boldsymbol{\Phi}$ then
            Insert $m$ into $\boldsymbol{\Theta}$
        end
    end
end
```

Algorithm 4.6: The rule-based benchmarking algorithms

4.3.2 Norman and Bean's random keys genetic algorithm

When selecting a benchmark algorithm from literature it is important to select an algorithm that addresses both the sequencing of operations and their allocation to resources. By considering the classical FJSP literature, a number of potential benchmark algorithms can be identified. These include the work of Kacem *et al.* [62, 63], Gao *et al.* [45], and Xia and Wu [145]. However, these algorithms specialize in solving classical FJSPs and it is the authors' opinion that they cannot be extended easily to address more constrained problems without significantly changing the structure of the algorithms, even when sequence-dependent set-up times and resource-dependent processing times are excluded from the problem. Furthermore, most of these algorithms are hybridizations of two or more solution strategies [63] or employ complex local search mechanisms specific to the problem being solved [45]. These complexities greatly increase the time to solution, while a fast solution time is considered to be an important requirement in the South African manufacturing industry. For these reasons, the random keys genetic algorithm (RKGA) [99] was identified as a more suitable alternative for benchmarking purposes. This algorithm was developed for a more highly constrained problem and does not make use of a local search mechanism.

As the name implies, the RKGA applies a genetic algorithm to the random keys representation used for the RKPSO in the previous section. Elitism is incorporated by automatically including the ι best individuals from the parent population into the new population. The exploration ability of the algorithm is improved through the use of immigration, i.e. ξ solutions of the new population are randomly re-initialized. After the first $\iota + \xi$ individuals of the new population have been obtained by means of elitism and immigration, each of the remaining individuals are obtained through the application of crossover and selection operators. For each individual i, where $i \in \{1, \ldots, (n_s - \iota - \xi)\}$ random selection is used to select two individuals from the current population, namely $\boldsymbol{x}_{i_1}(t)$ and $\boldsymbol{x}_{i_2}(t)$, where $x_{ij}(t)$ denotes the j^{th} dimension of individual i of generation t. Then, Bernoulli crossover [99] is applied such that for all dimensions, n_x, if $r \sim U(0,1) \leq$

p_c,

$$c_{i_1j}(t) = x_{i_1j}(t) \tag{4.7}$$

$$c_{i_2j}(t) = x_{i_2j}(t). \tag{4.8}$$

Otherwise $c_{i_1j}(t) = x_{i_2j}(t)$ and $c_{i_2j}(t) = x_{i_1j}(t)$, where p_c is the crossover probability. The better of the two candidate solutions, $c_{i_1j}(t)$ and $c_{i_2j}(t)$, is subsequently incorporated into the new population.

4.3.3 Comparative analysis of alternative benchmark algorithms

For each of the three data sets, the RKGA and P-DE results were evaluated over 30 independent simulation runs. However, due to the deterministic nature of the EDD rule, only one simulation run was needed for each data set. Finally, in order to comply with current *Optimatix* scheduling practice, the basic scheduling rule was used to construct 100 schedules of which the best was selected as the result of the simulation. To ensure consistency between all stochastic algorithms this process was repeated 30 times. The parameters of the RKGA were also optimized by means of the procedure described in Section 4.2. The resulting parameters are indicated in Table 4.7.

Table 4.7: RKGA algorithm parameters as selected for benchmarking purposes

Problem	ι	ξ	p_c
56-op	14	1	$0.5 \longrightarrow 0.1$
100-op	10	5	$0.5 \longrightarrow 0.3$
256-op	10	1	$0.5 \longrightarrow 0.5$

The results in Tables 4.8 and 4.9 and Figure 4.8 show that the priority-based algorithms performed significantly better than all the rule-based heuristics on every problem instance indicating that a definite performance improvement can be attributed to the priority-based metaheuristics over the existing algorithms. The P-PSO outperformed all other algorithms for the first two problems. For the last problem the RKGA obtained a lower fitness value, which was improved upon by the P-DE.

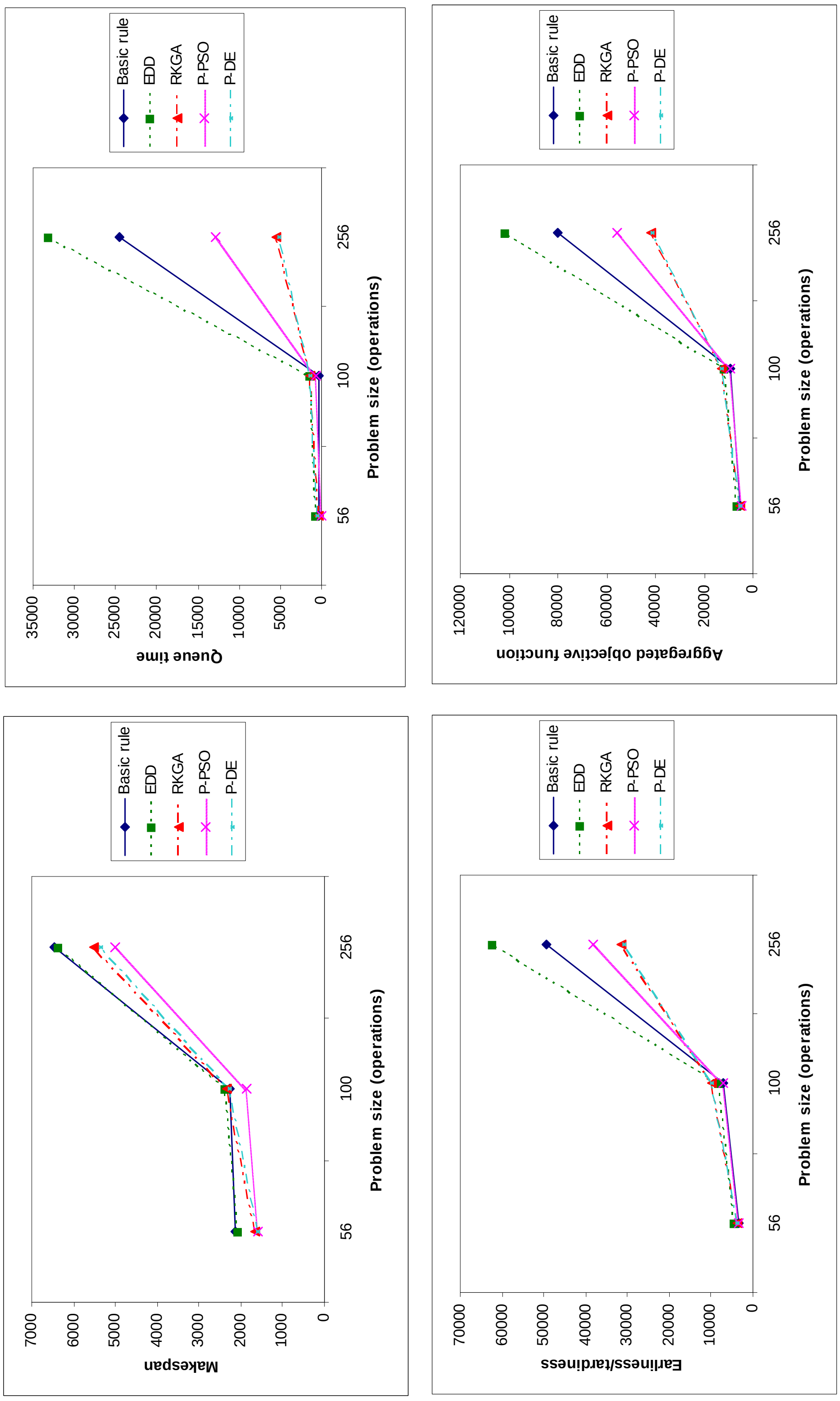


Figure 4.8: Benchmarking results

Table 4.8: Experimental comparison of alternative solution strategies with respect to makespan and earliness/tardiness

Pro-	**Algo-**	f_1			f_2		
blem	**rithm**	μ	σ	$CI_{0.05}$	μ	σ	$CI_{0.05}$
56-op	Basic rule	2124.62	43.48	± 16.23	3362.32	142.64	± 53.26
	EDD	2071.1	—	—	4242.8	—	—
	RKGA	1652.57	59.80	± 22.33	3578.04	331.49	± 123.77
	P-PSO	1581.92	17.55	± 6.55	3494.64	213.23	± 79.61
	P-DE	1594.19	24.71	± 9.23	3861.50	231.19	± 86.32
100-op	Basic rule	2266.31	51.20	± 19.12	6969.04	188.56	± 70.40
	EDD	2381.34	—	—	8168.33	—	—
	RKGA	2317.02	160.65	± 59.98	9770.28	372.64	± 139.13
	P-PSO	1867.66	13.53	± 5.05	6957.78	236.82	± 88.42
	P-DE	2276.37	66.72	± 24.91	9754.20	227.96	± 85.11
256-op	Basic rule	6446.67	304.80	± 113.80	49446	1329.76	± 496.48
	EDD	6360	—	—	62380	—	—
	RKGA	5514.86	189.35	± 70.70	31444.15	1240.96	± 463.33
	P-PSO	5010.93	45.57	± 17.01	38169.64	3065.49	± 1144.54
	P-DE	5343.74	143.92	± 53.73	30931.67	1018.09	± 380.12

Due to the poor performance of the P-PSO on the 256-operation problem, a closer investigation into the scalability of the RKGA, P-DE and P-PSO was performed. Algorithm performance was investigated on two additional problems, namely a 146 operation and a 200 operation problem. The results obtained in Figure 4.9 indicate that the VNGCP-PSO is the best performing algorithm for problems containing approximately 125 operations or less. Since the average number of operations which need to be scheduled by *Optimatix* is 100, the P-PSO will be the most suitable alternative most of the time. For larger problems the P-DE is the best performing algorithm of the three algo-

Table 4.9: Experimental comparison of alternative solution strategies with respect to queue time and the aggregated objective function

Pro-	**Algo-**	f_3			f_4		
blem	**rithm**	μ	σ	$CI_{0.05}$	μ	σ	$CI_{0.05}$
56-op	Basic rule	328.47	136.77	± 51.07	5321.22	80.04	± 29.88
	EDD	753.8	–	—	6574.8	–	—
	RKGA	300.84	190.52	± 71.13	5036.98	495.28	± 184.92
	P-PSO	74.46	127.93	± 47.76	4656.17	245.96	± 91.83
	P-DE	497.38	122.14	± 45.60	5459.02	328.29	± 122.57
100-op	Basic rule	323.13	155.44	± 58.04	9522.51	378.58	± 141.35
	EDD	1330.92	–	—	11392.47	--	—
	RKGA	1379.68	321.07	± 119.88	12980.46	724.83	± 270.63
	P-PSO	675.02	211.25	± 78.87	9009.96	406.74	± 151.86
	P-DE	1408.47	214.59	± 80.12	12952.48	418.30	± 156.18
256-op	Basic rule	24589.83	1304.41	± 487.02	80062.98	1870.93	± 698.54
	EDD	33090	–	—	101430	–	—
	RKGA	5475.59	873.54	± 326.15	41977.04	1889.94	± 705.64
	P-PSO	12893.70	1788.01	± 667.58	55630.34	4088.77	± 1526.60
	P-DE	5143.53	811.32	± 302.92	40960.35	1392.98	± 520.09

rithms tested. It is interesting to note that the poor scalability of the P-PSO algorithm is mostly due to poor performance with respect to queue time and earliness/tardiness. With respect to the makespan objective function, the P-PSO is always superior.

Finally, although the performance of the RKGA is very similar, though worse, than the P-DE algorithm over all problem sizes, the fact that it does not ever outperform the priority-based algorithms highlight the contribution made by the development of the priority-based representation for the multi-objective flexible job shop scheduling problem with sequence-dependent set-up times, auxiliary resources, and machine down time.

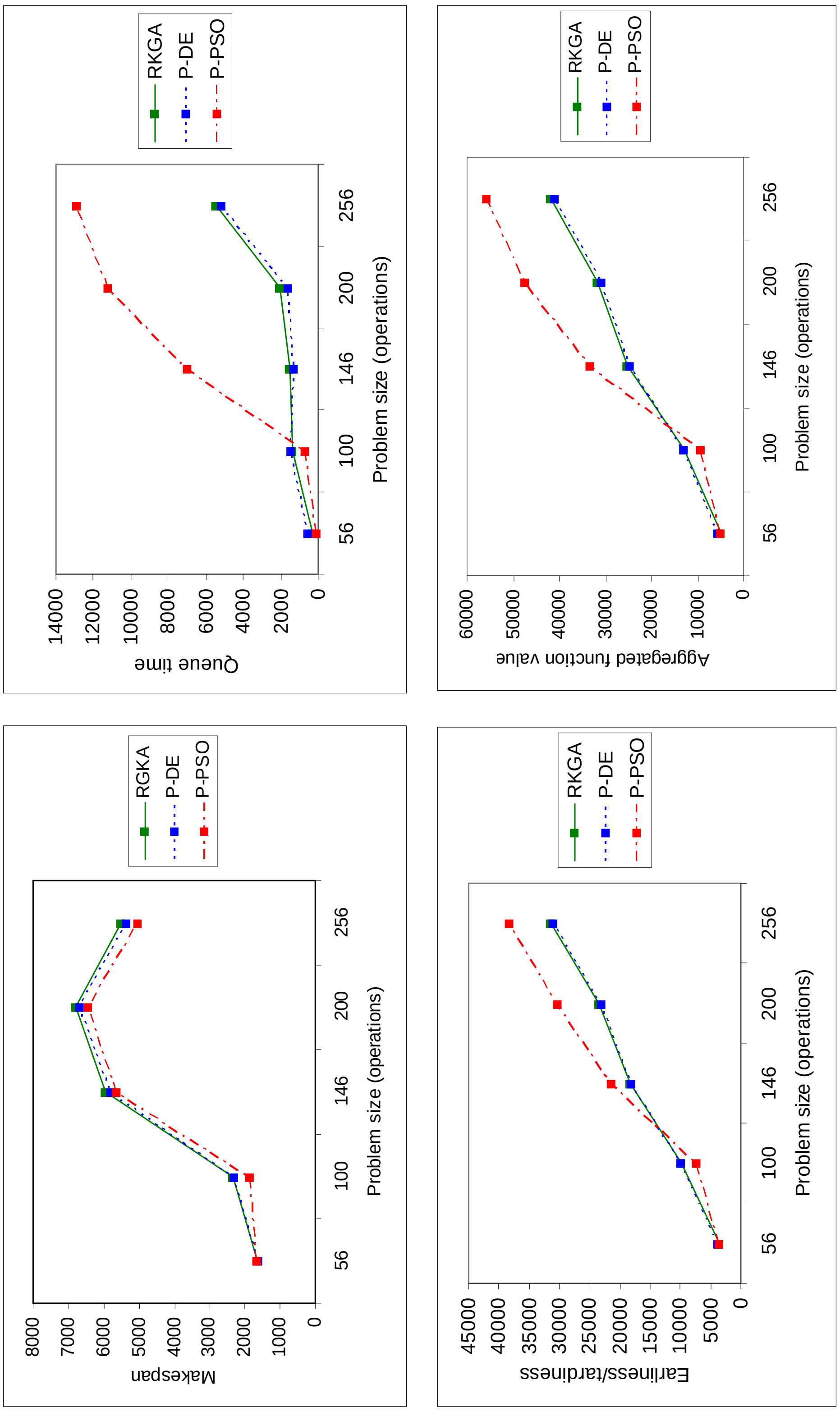


Figure 4.9: Investigating scalability

4.4 Summary

Three PSO-based heuristics, differing in terms of particle representation and problem mapping mechanism, were developed in this chapter. The priority-based PSO (P-PSO) algorithm was found to be the best performing PSO algorithm when both solution quality and computational complexity were considered. The priority-based algorithm was also extended to the differential evolution domain and alternative PSO topologies and DE base vector selection mechanisms were investigated. Further benchmarks against existing rule-based algorithms and Norman and Bean's RKGA [99] indicated that the P-PSO algorithm outperformed all other tested benchmark algorithms when 125 operations or less are to be scheduled. The P-DE algorithm was the best performing algorithm for larger problems. These results are encouraging, since a well-designed single objective optimization algorithm is a good starting point for improved multi-objective optimization, which is the focus of the next chapter.

Chapter 5

The multi-objective priority-based algorithm

The multi-objective aspect of the production environment is an important, yet frequently overlooked issue in production scheduling literature. This chapter focuses on extending the "single objective" algorithms developed in the previous chapter to multi-objective optimization algorithms capable of simultaneously optimizing a number of conflicting objective functions. The first section introduces the basic concepts of multi-objective optimization and provides an overview of existing multi-objective multiple machine literature. Section 5.2 is concerned with the analysis of various multi-objective strategies for the *Optimatix* problem. Two vector evaluated approaches are compared to a modified goal programming approach. Sections 5.3 and 5.4 respectively focus on comparing different variations of the vector evaluated PSO and DE approaches developed in the previous section. Finally, the main findings of the chapter is summarized in Section 5.5.

5.1 Introductory concepts and related literature

Multi-objective optimization, in general, has been receiving increasing attention over the last few years. This section provides a brief introduction into the field of multi-objective optimization before considering a number of existing multi-objective multiple machine scheduling algorithms.

5.1.1 Multi-objective optimization

A multi-objective optimization (MOO) problem can be formally defined as follows:

$$\begin{aligned} &\text{minimize } \boldsymbol{f}(\boldsymbol{x}) \\ &\text{subject to } \varsigma_q(\boldsymbol{x}) \leq 0, \qquad q = 1, \ldots, n_\varsigma \\ &\qquad\qquad \varrho_q(\boldsymbol{x}) = 0, \qquad q = n_\varsigma + 1, \ldots, n_\varsigma + n_\varrho \\ &\qquad\qquad \boldsymbol{x} \in [\boldsymbol{x}_{min}, \boldsymbol{x}_{max}]^{n_x} \end{aligned} \tag{5.1}$$

where $\boldsymbol{f}(\boldsymbol{x})$ denotes the vector of objective functions to be minimized, ς_q and ϱ_q are respectively the inequality and equality constraints and $\boldsymbol{x} \in [\boldsymbol{x}_{min}, \boldsymbol{x}_{max}]^{n_x}$ represent the boundary constraints. A solution to a MOO problem can thus be defined as a vector $\boldsymbol{x}$ that satisfies the constraints and optimizes the vector function $\boldsymbol{f}(\boldsymbol{x})$ [160].

A large number of approaches have already been used in literature to optimize multiple conflicting objectives. Fortunately, T'Kindt and Billaut [136] provide a generic classification which differentiates between different MOO approaches depending on the stage of the optimization process at which the decision makers' preferences are incorporated (refer to Figure 5.1).

A priori MOO techniques incorporate the decision makers' preferences at the start of the optimization process. Examples of preference information include relative importance or targets for each objective function. This information is then incorporated into the optimization process. Since most *a priori* techniques transform the multi-objective problem into a single-objective problem, only one solution to the scheduling problem can be obtained at a time. One of the most popular examples of this strategy include weighted aggregation where all objective functions are combined into a linear combination of criteria [119].

Interactive MOO methods involve the decision makers throughout the optimization process. Decision makers who specify weight changes at intermittent stages of the optimization process is an example of an interactive approach. This strategy is useful in steering the algorithm to more desirable regions of the objective space. This disserta-

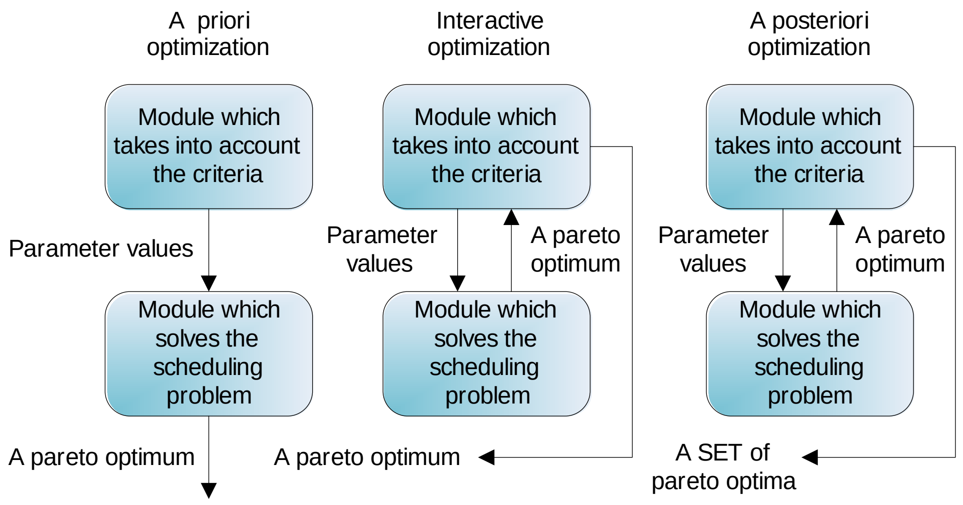

A priori optimization
Interactive optimization
A posteriori optimization
Module which takes into account the criteria
Module which takes into account the criteria
Module which takes into account the criteria
Parameter values
Parameter values
A pareto optimum
Parameter values
A pareto optimum
Module which solves the scheduling problem
Module which solves the scheduling problem
Module which solves the scheduling problem
A pareto optimum
A pareto optimum
A SET of pareto optima

Figure 5.1: T'Kindt and Billaut [136] define three main strategies for addressing multiple conflicting objectives

tion is more concerned with *a priori* versus *a posteriori* optimization, however, a more detailed study of *interactive* methods can be found in Vanderpooten [139].

No user preferences are taken into account before or during the optimization process in *a posteriori* MOO. The focus is on providing the decision makers with as diverse a set of solutions as possible to facilitate the selection of the most suitable solution from the set [136]. The purpose of *a posteriori* MOO can thus be summarized as finding a set of trade-off solutions referred to as the Pareto optimal set, $\boldsymbol{PF}$, which was defined in equation (4.3).

5.1.2 Multi-objective multiple machine scheduling

The importance of multi-objective scheduling problems have long since been recognized in literature and good overall reviews can be found in the works of Loukil and Teghem [84] and T'Kindt and Billaut [136]. However, due to the extreme complexity of multiple machine, multi-objective scheduling problems, little attention have been paid to these combinatorial problems. Nonetheless a number of notable exceptions exist in recent flexible job shop scheduling literature.

From the algorithms listed in Table 5.1, two important conclusions can be drawn with

Table 5.1: Multiple machine, multi-objective scheduling literature

Algorithm	**Reference**
Multi-objective simulated annealing (MOSA)	Loukil *et al.* [85]
Tabu search (TS)	Dauzère–Pérès and Paulli [35]
Localization and evolutionary approach	Kacem *et al.* [63]
Hybrid evolutionary algorithm and fuzzy logic	Kacem *et al.* [62]
Simulated annealing-PSO algorithm	Xia and Wu [145]
Multi-stage genetic algorithm (GA)	Zhang and Gen [156]
GA-bottleneck shifting hybrid	Gao *et al.* [45]

respect to the use of multi-objective optimization techniques in multiple machine, multi-objective scheduling problems. Firstly, the use of the weighted aggregation approach is quite common [45], and secondly, the more sophisticated MOO techniques are difficult to understand and computationally complex [85]. There seems to be a research opportunity in developing a multi-objective multiple machine scheduling algorithm which overcomes the problems associated with weighted aggregation [39] while still being relatively simple to understand and implement.

5.2 Alternative multi-objective PSO and DE-based scheduling strategies

Over the years a number of papers have addressed the extension of both particle swarm optimization (PSO) and differential evolution (DE) to multiple objectives [2, 86, 120, 148]. A comprehensive review of multi-objective PSO algorithms can be found in [120]. Three popular multi-objective DE algorithms, from the past few years, include the

Pareto-based multi-objective DE algorithms of Xue *et al.* [148], Abbas *et al.* [2], and Madavan [86].

Goal programming (GP) [119], which attempts to minimize the deviation between the objective function values and specific targets set for each function, was used in the previous chapter to address the multi-objective aspects of the *Optimatix* problem. This *a priori* strategy allowed the problem to be solved by means of single-objective PSO and DE algorithms. However, in the *Optimatix* environment it can happen that target values for each objective function are not available. The exploratory nature of an *a posteriori* approach then seems to be more suitable, since valuable information with respect to the objective space can be obtained from the set of Pareto optimal solutions. However, the solution quality of an *a priori* optimization approach will in most cases be superior to that of an *a posteriori* approach since a significantly simpler single objective optimization problem simply needs to be solved repeatedly in the *a priori* case.

One of the key objectives of this chapter is to investigate the solution quality-complexity trade-off of *a posteriori* versus *a priori* optimization in the *Optimatix* environment. Vector evaluated MOO was selected for comparison purposes against the already developed GP approach, due to the high level of simplicity and subsequent reduced computational complexity of vector evaluated algorithms when compared to more sophisticated *a posteriori* MOO techniques [109]. The two vector evaluated approaches, as well as the modified goal programming approach, is described in more detail in Sections 5.2.1 and 5.2.2. The investigation into the relative performance of these three algorithms is presented in Section 5.2.3.

5.2.1 The vector evaluated approach

The vector evaluated approach can be classified as a criterion-based multi-objective strategy, where different stages of the optimization process consider different objectives [39]. The actual implementation involves assigning each objective function to one of multiple populations for optimization. Information with respect to the different populations are exchanged in an algorithm-dependent fashion resulting in the simultaneous optimization of the various objective functions. As previously stated, the advantage of this approach lies in reduced computational complexity, which is a desirable property when solving a complex combinatorial problem where the fitness function evaluations are in themselves computationally expensive.

The first vector evaluated algorithm was Schaffer's vector evaluated GA (VEGA) [125]. A number of years later the concept was transferred to the PSO and DE domains by Parsopoulos *et al.* [108, 109]. Their work focused on comparing the vector evaluated PSO (VEPSO) algorithm to various aggregation-based approaches including bang-bang and dynamic weighted aggregation. The vector evaluated DE (VEDE) algorithm proved to compare well with Schaeffer's VEGA. These studies used two-objective problems from the well-known MOO benchmark set of Zitzler *et al.* [160].

The VEDE and VEPSO algorithms differ only with respect to the information exchange mechanism used. VEPSO (refer to Algorithm 5.1) makes use of an alternative velocity update equation where (considering two swarms) the global best position of the first swarm is used in the velocity equation of the second swarm and vice versa. Thus,

$$v_{ijs}(t+1) = wv_{ijs}(t) + c_1 r_{1j}(t)[\hat{x}_{ijs}(t) - x_{ijs}(t)] + c_2 r_{2j}(t)[x^*_{jm_s}(t) - x_{ijs}(t)] \qquad (5.2)$$

where $v_{ijs}(t)$, $x_{ijs}(t)$ and $\hat{x}_{ijs}(t)$, respectively denote the velocity, position and personal best position of the j^{th} dimension of the i^{th} particle of swarm s at time t, and $x^*_{jm_s}(t)$ is the j^{th} dimension of the global best position of swarm m_s at time t. Although the VEPSO was originally developed for optimizing bi-objective optimization problems, this concept can be easily extended to the *Optimatix* problem which requires the simultaneous minimization of three objective functions.

VEDE (refer to Algorithm 5.2) functions on a slightly different basis. The best individual is migrated between different populations and an additional dominance-based selection mechanism is used, i.e. an individual can only be replaced by a dominating offspring.

5.2.2 Modified goal programming

Goal programming [119] is based on the assumption that a user has sufficient knowledge of the problem domain to specify suitable target values for each of the objective functions to be optimized. An aggregate objective function consisting of the sum of the deviations between the actual values obtained and the specified targets is then minimized [119].

```
Initialize three swarms, $\boldsymbol{X}_1, \boldsymbol{X}_2$ and $\boldsymbol{X}_3$
Initialize the external archive as an empty set
$t = 1$
while $t < I_{max}$ do
    for All swarms $s$ do
        Evaluate the fitness function of each swarm w.r.t. its allocated objective:
        $f(\hat{\boldsymbol{x}}_{\boldsymbol{s}}(t)) = \min_i\{f_s(\boldsymbol{x}_{\boldsymbol{s}}(t))\}$
        for All individuals $i$ in swarm $\boldsymbol{X}_s$ do
            if $f_s(\boldsymbol{x}_{\boldsymbol{is}}(t)) < f_s(\hat{\boldsymbol{x}}_{\boldsymbol{is}}(t))$ then
                Set the personal best position:
                $\hat{\boldsymbol{x}}_{\boldsymbol{is}}(t) = \boldsymbol{x}_{\boldsymbol{is}}(t)$
            end
        end
        if $\min_i\{f_s(\hat{\boldsymbol{x}}_{\boldsymbol{s}}(t))\} < f_s(\boldsymbol{x}^*_{\boldsymbol{s}}(t))$ then
            Set the global best position of each swarm w.r.t. its allocated objective:
            $f(\boldsymbol{x}^*_{\boldsymbol{s}}(t)) = \min_i\{f_s(\hat{\boldsymbol{x}}_{\boldsymbol{s}}(t))\}$
        end
        Update the external archive to include all non-dominated solutions in $\boldsymbol{X}_s$
    end
    for All swarms $s$ do
        for The gbest particle $\tau$ do
            for All dimensions $j$ do
                Update the gbest particle of each swarm:
                $v_{\tau js}(t) = -x_{\tau js}(t) + x^*_{jm_s}(t) + wv_{\tau js}(t) + \rho(1 - 2r_{2j})$
                $x_{\tau js}(t) = x^*_{js}(t) + wv_{\tau js}(t) + \rho(1 - 2r_{2j})$
            end
        end
        for All particles $i$ such that $i \neq \tau$ do
            for All dimensions $j$ do
                $v_{ijs}(t) = wv_{ijs}(t) + c_1r_{1j}(\hat{x}_{ijs}(t) - x_{ijs}(t)) + c_2r_{2j}(x^*_{jm_s}(t) - x_{ijs}(t))$
                $x_{ijs}(t) = x_{ijs}(t) + v_{ijs}(t)$
            end
        end
    end
    $t = t + 1$
end
```

Algorithm 5.1: The vector evaluated particle swarm optimization (VEPSO) algorithm

```
Initialize three populations, $\boldsymbol{X}_1, \boldsymbol{X}_2$ and $\boldsymbol{X}_3$
Initialize the external archive as an empty set
$t = 1$
while $t < I_{max}$ do
    for All populations $s$ do
        Evaluate the fitness function of each population w.r.t. its allocated objective:
        $f(\boldsymbol{x}^*_{\boldsymbol{s}}(t)) = \min_i\{f_s(\boldsymbol{x}_{\boldsymbol{s}}(t))\}$
        Update the external archive to include all non-dominated solutions in $\boldsymbol{X}_s$
    end
    for All populations $s$ do
        Move the best individual of population $m_s$ to population $s$:
        $\boldsymbol{x}^*_{\boldsymbol{s}}(t) = \boldsymbol{x}^*_{\boldsymbol{m_s}}(t)$
    end
    for All populations $s$ do
        for All individuals $i$ do
            Randomly select 3 individuals from population $s$, $\boldsymbol{x}_{\boldsymbol{i_1 s}}(t)$, $\boldsymbol{x}_{\boldsymbol{i_2 s}}(t)$ and $\boldsymbol{x}_{\boldsymbol{i_3 s}}(t)$,
            such that $i_1 \neq i_2 \neq i_3 \neq i$
            Randomly select one of the dimensions $\nu$
            for All dimensions $j$ do
                if $r \sim U(0,1) \leq p_c$ or $j = \nu$ then
                    $c_{ijs} = x_{i_3 js}(t) + F(x_{i_1 js}(t) - x_{i_3 js}(t))$
                else
                    $c_{ijs} = x_{ijs}(t)$
                end
            end
            Only replace $\boldsymbol{x}_{\boldsymbol{is}}(t)$ by a dominating individual:
            if $\boldsymbol{c}_{\boldsymbol{is}} \prec \boldsymbol{x}_{\boldsymbol{is}}(t)$ then
                $\boldsymbol{x}_{\boldsymbol{is}}(t) = \boldsymbol{c}_{\boldsymbol{is}}$
            end
        end
    end
    $t = t + 1$
end
```

Algorithm 5.2: The vector evaluated differential evolution (VEDE) algorithm

The user, in fact, attempts to direct a single objective optimization algorithm towards a pre-specified point on the Pareto front. Due to the complexity of MOO problems it can happen that the user defines targets which denote a position in the objective space which is dominated by the Pareto front. In this case, it is desirable for the algorithm to continue optimizing until convergence is obtained on a nearby solution located on the actual Pareto front. The aggregated fitness function, f_4, is thus given as

$$f_4 = \sum_{s=1}^{n_p} |(f_s - g_s)| + \upsilon f_s, \tag{5.3}$$

where f_s denotes the s^{th} fitness function, g_s is the target value of the s^{th} fitness function and υ is a sufficiently small number.

For the sake of completeness the GP algorithm is described in Algorithm 5.3. Appropriate minimum and maximum target values, denoted by g^s_{ref} and f^s_{ref}, are obtained for each of the s objectives by minimizing each objective function individually. A user defined required number of target points, $\jmath$, is used to divide the objective space into a uniform grid. Here $\triangle_s$ denotes the size of the discretization intervals of dimension s of the objective space. Finally, the set $\boldsymbol{PF}$ is populated by optimizing the aggregated objective function f_4 at each of the target points on the grid.

As previously mentioned, goal programming has the advantage that the algorithm focuses explicitly on a single objective throughout the optimization process. However, this algorithm characteristic results in only one solution being obtained at a time as output to the scheduling algorithm. Obtaining the same number of solutions as can be obtained during a single iteration of a vector evaluated algorithm is indeed a time-consuming process. Furthermore, the performance of a GP-based MOO algorithm is highly dependent on the type of single objective optimization algorithm used.

5.2.3 Comparative analysis of alternative MOO strategies

Measuring the performance of the vector evaluated algorithms to the modified goal programming approach is significantly more complex than simply considering accuracy or speed of convergence, which are commonly considered to be suitable measures of performance for single objective algorithms. Instead, three aspects need to be considered,

```
Obtain the minimum and maximum target values of each objective:
for All criteria $s$ do
    $g^s_{ref} = \min_{\forall s'}\{f_s(\boldsymbol{x}^*_{s'})\}$
    $f^s_{ref} = \max_{\forall s'}\{f_s(\boldsymbol{x}^*_{s'})\}$
    $\triangle_s = \frac{f^s_{ref} - g^s_{ref}}{J}$
end
Divide the objective space into a uniform grid:
$g_1 = g^1_{ref} - \triangle_1$
for All points $i_1$ do
    $g_1 = g_1 + \triangle_1$
    $g_2 = g^2_{ref} - \triangle_2$
    for All points $i_2$ do
        $g_2 = g_2 + \triangle_2$
        $g_3 = g^3_{ref} - \triangle_3$
        for All points $i_3$ do
            $g_3 = g_3 + \triangle_3$
            Optimize the aggregated objective function $f_4$ at each target point:
            $\boldsymbol{PF} = \boldsymbol{PF} \cup \{f_4(\boldsymbol{x}^*)\}$
        end
    end
end
```

Algorithm 5.3: The modified GP approach

namely the minimization of the distance between the solutions obtained and the actual Pareto front, the maximization of the diversity of the Pareto front, and the ability to maintain already found non-dominated solutions [39]. Comprehensive reviews of MOO performance measures can be found in Zitzler *et al.* [161] and Engelbrecht [39].

Three performance measures were used in this chapter to compare the different MOO strategies. The S metric [67, 161] measures the size of the region dominated by the Pareto front based on a reference vector consisting of the maximum value in each objective. For replication purposes the reference vectors which were used in this section is listed in Table 5.2. The S metric provides information with respect to both solution quality and Pareto front diversity. More specifically, the S metric of the set, $\boldsymbol{PF}$, with respect to $\boldsymbol{f}_{ref}$ can be described as the Lebesgue integral of the set $R(\boldsymbol{PF}, \boldsymbol{f}_{ref})$, where

$$R(\boldsymbol{PF}, \boldsymbol{f}_{ref}) = \cup_{\forall \boldsymbol{f} \in \boldsymbol{PF}} R(\boldsymbol{f}, \boldsymbol{f}_{ref}) \tag{5.4}$$

and

$$R(\boldsymbol{f}, \boldsymbol{f}_{ref}) = \{\boldsymbol{f}' | \boldsymbol{f}' \prec \boldsymbol{f}_{ref} \text{ and } \boldsymbol{f} \prec \boldsymbol{f}', \boldsymbol{f}' \in \mathbb{R}^{n_p}\} \tag{5.5}$$

Table 5.2: The reference vectors used for each of the problems

Problem	**Vector**
56-operation problem	{1000, 4000, 1000}
100-operation problem	{2000, 5000, 2000}
256-operation problem	{2500, 12000, 5000}

The other two measures include the size of the approximated Pareto front, $N(\boldsymbol{PF})$, as well as the extent of the Pareto front, $\chi(\boldsymbol{PF})$ [39], where

$$\chi(\boldsymbol{PF}) = \sqrt{\sum_{s=1}^{N(\boldsymbol{PF})} \max\{|f_s(\boldsymbol{x}) - f_s'(\boldsymbol{x})| : \boldsymbol{f}, \boldsymbol{f}' \in \boldsymbol{PF}\}}. \tag{5.6}$$

A DE/rand/bin algorithm was used as basis for the modified GP algorithm. Initial experimentation with different algorithm parameter settings resulted in the probability of reproduction, p_r, being decreased linearly from 0.75 to 0.25 over 95% of the total number of iterations, I_{max}. Similarly, the scaling factor, F, was decreased from 0.75 to 0.125. A population size, n_s, of 27 individuals and a maximum number of iterations, I_{max}, of 200 was used.

For the VEDE and VEPSO algorithms, the "best" parameter settings and PSO topology or DE base vector selection strategy was retained from Section 4.2 for each problem-algorithm pair. However, the vector evaluated algorithms each consists of n_p

swarms or populations each consisting of n_s candidate solutions. For both the VEPSO and VEDE algorithms an external archive of unlimited size [61] was used to store all non-dominated solutions. The actual results of this investigation into alternative multi-objective optimization strategies are recorded in Tables 5.3 through 5.5 and Figure 5.2.

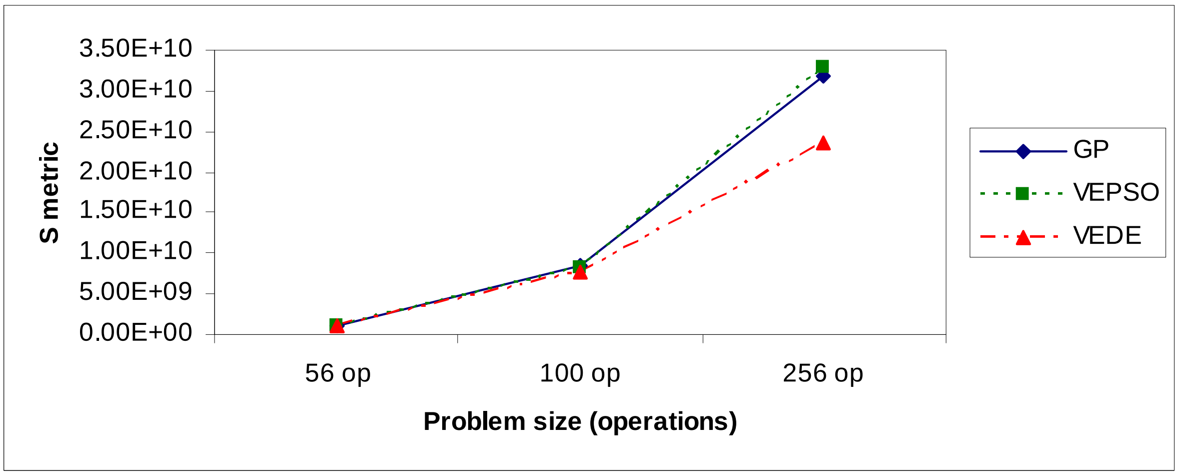
Table 5.3: Investigating alternative multi-objective optimization strategies for the 56-operation problem

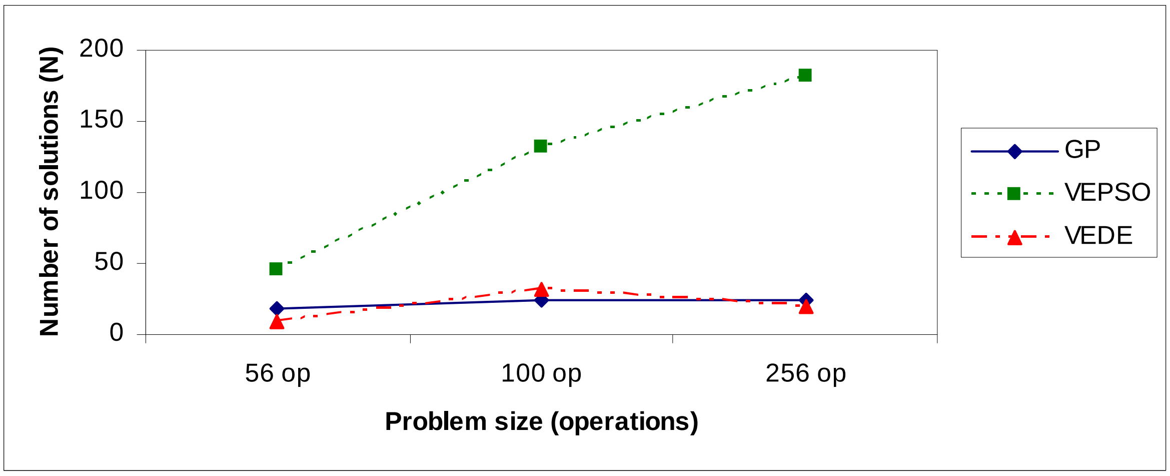
	Strategy	VEPSO	VEDE	GP
$S(\boldsymbol{PF})$	μ	9.63×10^8	9.47×10^8	1.04×10^9
	σ	2.72×10^7	3.12×10^7	1.00×10^7
	$\boldsymbol{CI}_{0.05}$	$\pm 1.02 \times 10^7$	$\pm 1.16 \times 10^7$	$\pm 3.73 \times 10^6$
$N(\boldsymbol{PF})$	μ	45.53	8.97	17.47
	σ	12.46	3.12	2.08
	$\boldsymbol{CI}_{0.05}$	± 4.65	± 1.16	± 0.78
$\chi(\boldsymbol{PF})$	μ	28.33	19.51	47.83
	σ	2.36	2.45	2.47
	$\boldsymbol{CI}_{0.05}$	± 0.88	±0.91	±0.92

For two out of the three problems, the GP approach outperformed the vector evaluated algorithms with respect to the S metric. The GP approach also outperformed the two vector evaluated algorithms with reference to the extent of the obtained Pareto fronts. This was the case for all three data sets. These findings provide a good indication of GP's ability to generate diverse Pareto fronts consisting of high quality solutions.

However, the modified GP approach is extremely computationally expensive due to the fact that a single objective optimization algorithm is, in fact, executed repeatedly. In addition, a concerning aspect is evident from the results: For each simulation the GP algorithm was executed 27 times, where each "execution" attempted to zoom in on a different uniformly distributed point on the Pareto front. It can thus be expected that 27 non-dominated solutions, $N(\boldsymbol{PF})$, would be obtained for each simulation. However, when

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA





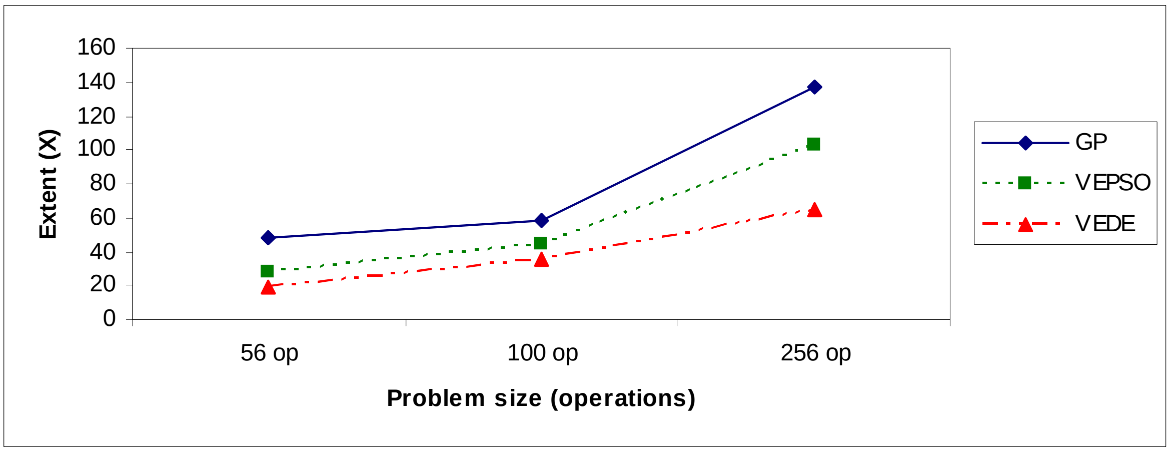


Figure 5.2: Results of the investigation into alternative MOO strategies

Table 5.4: Investigating alternative multi-objective optimization strategies for the 100-operation problem

	Strategy	VEPSO	VEDE	GP
$S(\boldsymbol{PF})$	μ	8.33×10^9	7.82×10^9	8.44×10^9
	σ	7.36×10^7	1.63×10^8	8.01×10^7
	$\boldsymbol{CI}_{0.05}$	$\pm 2.75 \times 10^7$	$\pm 6.09 \times 10^7$	$\pm 2.99 \times 10^7$
$N(\boldsymbol{PF})$	μ	132.27	31.13	24.70
	σ	15.62	4.42	1.32
	$\boldsymbol{CI}_{0.05}$	± 5.83	± 1.65	±0.49
$\chi(\boldsymbol{PF})$	μ	44.70	35.73	58.52
	σ	1.42	2.36	1.85
	$\boldsymbol{CI}_{0.05}$	± 0.53	± 0.88	± 0.69

Table 5.5: Investigating alternative multi-objective optimization strategies for the 256-operation problem

	Strategy	VEPSO	VEDE	GP
$S(\boldsymbol{PF})$	μ	3.28×10^{10}	2.36×10^{10}	3.17×10^{10}
	σ	7.82×10^9	1.02×10^9	5.90×10^8
	$\boldsymbol{CI}_{0.05}$	$\pm 2.92 \times 10^9$	$\pm 3.81 \times 10^8$	$\pm 2.20 \times 10^8$
$N(\boldsymbol{PF})$	μ	181.73	20.07	23.93
	σ	36.54	5.31	1.93
	$\boldsymbol{CI}_{0.05}$	± 13.64	± 1.98	± 0.72
$\chi(\boldsymbol{PF})$	μ	102.35	65.07	137.01
	σ	4.57	5.64	3.70
	$\boldsymbol{CI}_{0.05}$	± 1.71	± 2.11	± 1.38

all dominated solutions were removed from the resulting Pareto fronts, only 17.47, 24.70 and 23.93 non-dominated solutions were obtained on average over the 30 simulation runs for the three problems tested. These figures indicate that on average, 9.53, 2.3 and 3.07 solutions which were expected to be non-dominated solutions, were actually dominated by one or more other solutions in the Pareto front. This realization holds dramatic implications for algorithm efficiency indicating that up to 35% of computational time is "wasted" by the modified GP approach during optimization. This is probably due to the the inability of GP to use information regarding previous non-dominated solutions during the search for new non-dominated solutions.

For the purposes of this dissertation, a more computationally efficient approach is desired. The VEPSO algorithm outperformed both of the VEDE and modified goal programming algorithms in terms of the number of solutions obtained for each of the three test problems. The VEPSO also performed well with respect to the S metric and the extent of the obtained Pareto fronts. The VEPSO algorithm even outperformed GP in terms of the S metric on the 256 operation problem, indicating that the algorithm generally scales better than the VEDE and modified GP approaches. A closer inspection of the actual scheduling solutions further supports the fact that VEPSO is an acceptable alternative to the computationally complex modified goal programming approach.

Surprisingly, the VEDE algorithm performed relatively poor on the three data sets. The Pareto fronts which were obtained consist of a small number of low quality solutions. These results are interesting, since the single objective version of the DE algorithm performed well in the experiments described in the previous chapter.

5.3 A further investigation of the VEPSO algorithm

Since the same algorithm can perform drastically different on different instances of the same problem, it is desirable to reduce the effort associated with customizing an algorithm for a specific data set. The use of alternative information exchange strategies is thought to have an important impact on VEPSO and VEDE algorithm performance [109]. The next two sections are aimed at investigating exactly how profound this impact is in the *Optimatix* environment. To achieve this objective, four additional

VEPSO algorithms were developed. The rest of this section provides a brief description of each of the algorithms before an empirical analysis of algorithm performance is conducted.

5.3.1 Variations on the basic VEPSO algorithm

The four VEPSO-based algorithms can be described effectively with reference to Parsopoulos *et al.*'s VEPSO algorithm [109], which is denoted by VEPSO(1) in this section. VEPSO(1) makes use of a ring structure for information exchange purposes. A specific swarm, $\boldsymbol{X}_s$, always uses information regarding the best solution from one specific neighbouring swarm, $\boldsymbol{X}_{(s+1 \mod s)}$, to update its particle velocities. Since each swarm is associated with the optimization of one specific criterion, this results in each swarm only considering two criteria at a time. Fortunately, the connectedness of the ring structure ensures that information with respect to the other criteria is eventually propagated to all swarms. However, the fact that information exchange always occurs between the same pairs of swarms could have a definite negative effect on the pace of this propagation.

VEPSO(2): The VEPSO(2) algorithm was developed to investigate algorithm performance when the speed of the information exchange between different swarms is increased. In this algorithm, each swarm randomly selects the second swarm, which may be itself, from which *gbest* information is used to update the particle velocities. Equation (5.2) is thus updated such that $m_s \sim U(1, ..., n_p)$ where n_p is the total number of swarms. Even though only two criteria are still optimized at a time, the criteria change during each iteration, leading to faster information propagation. There is a now a nonzero probability that each swarm will be directly influenced by all objective functions.

A graphical comparison of VEPSO(1) and VEPSO(2) is provided in Figure 5.3. Here the solid lines represent the possible sources from which *gbest* information can be obtained by swarm $\boldsymbol{X}_s$ in a single iteration. The dotted lines indicate possible paths of information flow between the other swarms.

VEPSO(3): One of the disadvantages of VEPSO(1) is that the *gbest* particle of a swarm is not incorporated into the velocity update equations of the particles of the same swarm. This issue results in information regarding good solutions with respect to the current "swarm objective" being lost in the optimization process. VEPSO(3) attempts

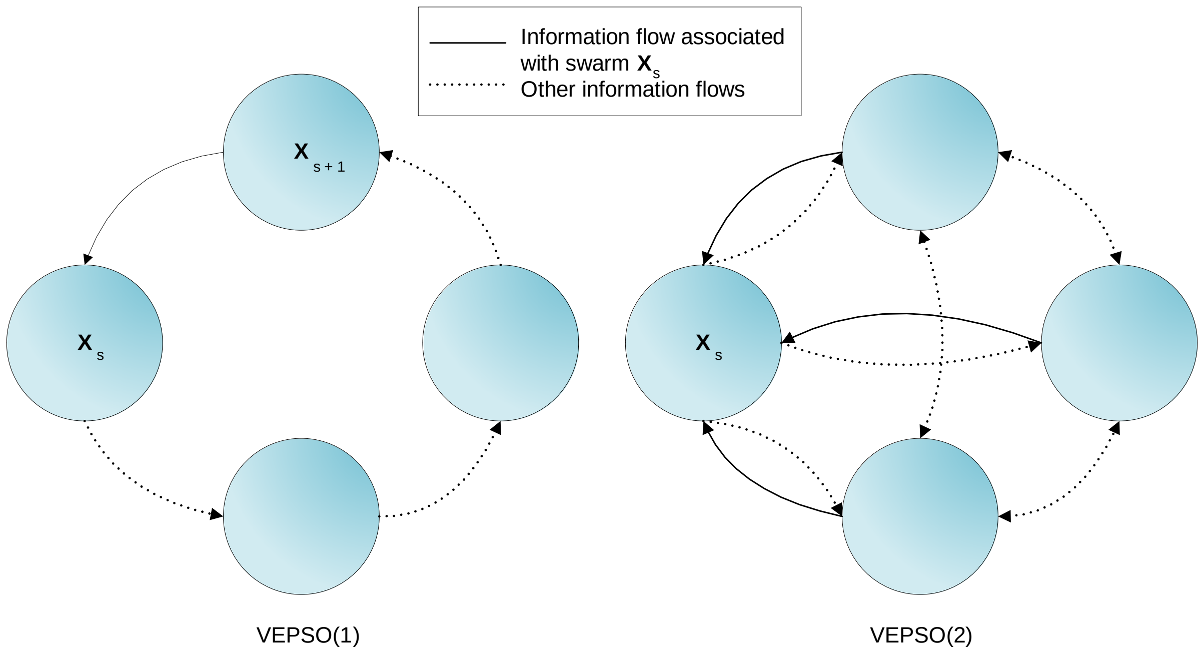


Figure 5.3: A graphical comparison of the VEPSO(1) and VEPSO(2) algorithms

to improve the exploitation ability of the VEPSO(1) algorithm by defining an additional term in the velocity update equation:

$$\begin{aligned} v_{ijs}(t+1) = wv_{ijs}(t) + c_1r_{1j}(t)[\hat{x}_{ijs}(t) - x_{ijs}(t)] + c_2r_{2j}(t)[x^*_{jm_s}(t) - x_{ijs}(t)] \\ + c_2r_{2j}(t)[x^*_{js}(t) - x_{ijs}(t)] \end{aligned} \tag{5.7}$$

VEPSO(4): VEPSO(4) uses alternating velocity update equations. During each even numbered iteration the standard *gbest* PSO velocity update is used to exploit promising areas of the search space with reference to the assigned objective function. Each odd numbered iteration makes use of the VEPSO(1) velocity update.

The aim of a swarm in a VEPSO-based algorithm is to obtain good solutions with respect to its assigned objective function while simultaneously optimizing the objective function of a neighbouring swarm. VEPSO(4) uses principles of criterion-based MOO, where different objectives are optimized during different phases of the optimization process, to simultaneously achieve these conflicting aims. Since VEPSO(1) does not make use of a swarm's "own" *gbest* information, the exploitation ability of the algorithm with respect to its assigned objective is adversely affected. VEPSO(4) addresses this issue by alternating between the optimization of only the assigned objective and simultaneously optimizing the assigned objective along with a neighbouring objective function. The disadvantages associated with VEPSO(1) is, however, still applicable since the ring structure is used in this implementation of VEPSO(4). This results in only two criteria being directly optimized at a time.

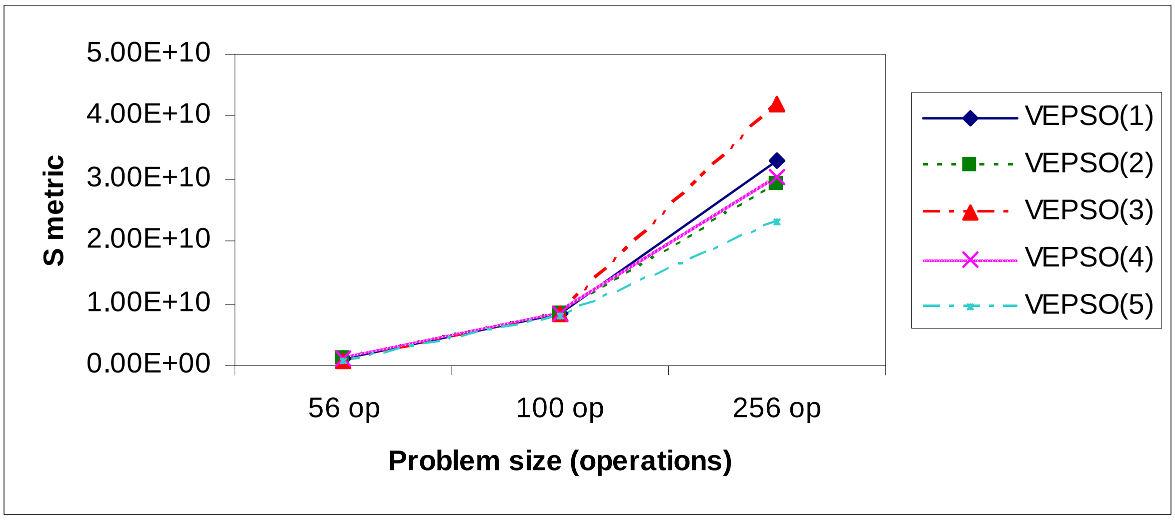
VEPSO(5): Parsopolous *et al.*'s VEDE algorithm [108] inspired the development of VEPSO(5). The algorithm can be considered as a VEPSO(1) algorithm with the addition of a dominance-based selection mechanism. To facilitate the inclusion of a selection mechanism in a PSO-based algorithm, the conditions under which the *pbest* of a particle is updated can be redefined. In this initial investigation, the *pbest* of each particle was only updated if a dominating solution existed.

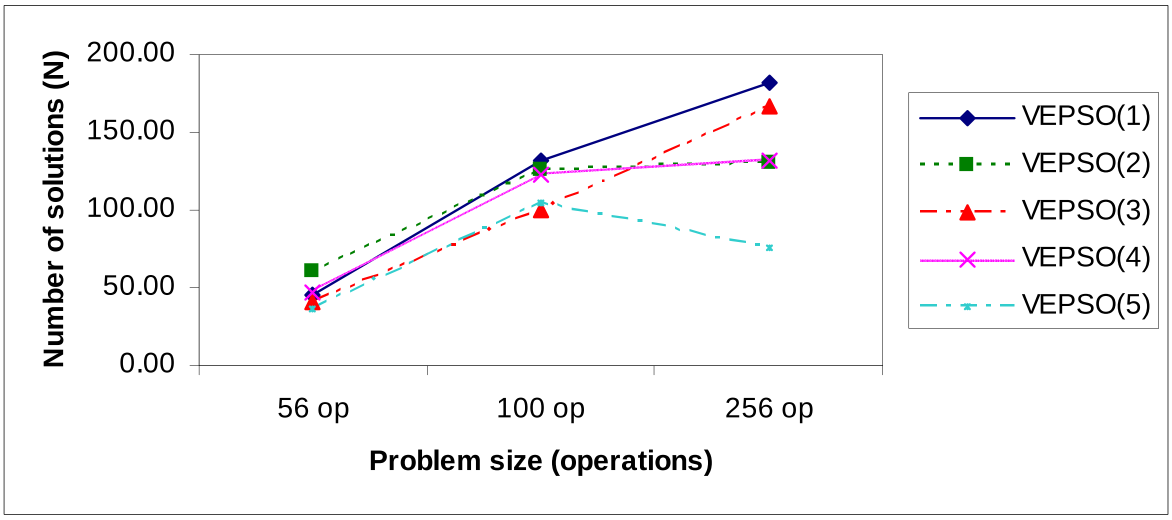
5.3.2 Comparative analysis of alternative VEPSO algorithms

The optimized parameters and PSO topologies as determined for each algorithm-data set pair in Section 4.2, was again used for the purposes of comparing the variations on the VEPSO(1) algorithm. The results of this investigation are recorded in Tables 5.6 through 5.8 and Figure 5.4.

No VEPSO-based algorithm could be identified as superior with respect to all criteria over all three problem sizes. The VEPSO(2) algorithm significantly outperformed the other four VEPSO variations with respect to the S metric and the number of solutions obtained when the 56 operation problem was considered. No statistically significant difference in performance could be identified between any of the algorithms tested on the 100 operation problem. VEPSO(3) was found to be the best performing algorithm for the largest sized problem when the S metric and extent of the obtained Pareto fronts were considered. VEPSO(3) also performed well with respect to the number of non-dominated solutions obtained since no difference in performance could be identified at an alpha of 0.05 between the $N(\boldsymbol{PF})$ values obtained by VEPSO(1) and VEPSO(3).

It is important to note that the different VEPSO variations differ largely with respect to the emphasis placed on different objective functions during the optimization process and the tempo of information propagation between the different swarms. It is quite possible that different VEPSO-based algorithms is more suited to solving different





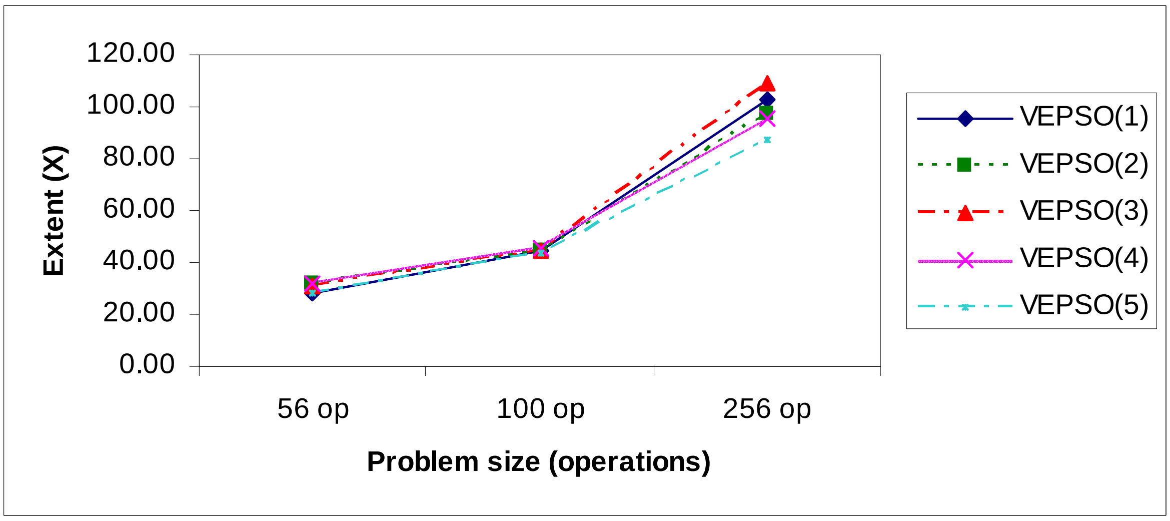


Figure 5.4: Results of the investigation into alternative VEPSO-based strategies

Table 5.6: Investigating alternative VEPSO-based strategies for the 56-operation problem

	Strategy	VEPSO(1)	VEPSO(2)	VEPSO(3)	VEPSO(4)	VEPSO(5)
$S(\boldsymbol{PF})$	μ	9.63×10^8	1.01×10^9	9.05×10^8	9.59×10^8	9.17×10^8
	σ	2.72×10^7	2.34×10^7	2.53×10^7	3.06×10^7	2.78×10^7
	$\boldsymbol{CI}_{0.05}$	$\pm 1.02 \times 10^7$	$\pm 8.74 \times 10^6$	$\pm 9.45 \times 10^6$	$\pm 1.14 \times 10^7$	$\pm 1.04 \times 10^7$
$N(\boldsymbol{PF})$	μ	45.53	60.17	41.20	47.20	35.63
	σ	12.46	16.29	11.18	11.71	9.98
	$\boldsymbol{CI}_{0.05}$	$\pm$ 4.65	$\pm$ 6.08	$\pm$ 4.17	$\pm$ 4.37	$\pm$ 3.73
$\chi(\boldsymbol{PF})$	μ	28.33	31.89	30.70	31.60	28.58
	σ	2.36	2.40	1.98	2.61	2.46
	$\boldsymbol{CI}_{0.05}$	$\pm$ 0.88	$\pm$ 0.90	$\pm$ 0.74	$\pm$ 0.97	$\pm$ 0.92

Table 5.7: Investigating alternative VEPSO-based strategies for the 100-operation problem

	Strategy	VEPSO(1)	VEPSO(2)	VEPSO(3)	VEPSO(4)	VEPSO(5)
$S(\boldsymbol{PF})$	μ	8.33×10^9	8.31×10^9	8.15×10^9	8.30×10^9	8.10×10^9
	σ	7.36×10^7	9.99×10^7	5.38×10^7	7.14×10^7	8.66×10^7
	$\boldsymbol{CI}_{0.05}$	$\pm 2.75 \times 10^7$	$\pm 3.73 \times 10^7$	$\pm 2.01 \times 10^7$	$\pm 2.67 \times 10^7$	$\pm 3.23 \times 10^7$
$N(\boldsymbol{PF})$	μ	132.27	125.93	99.77	122.97	104.53
	σ	15.62	22.37	12.71	20.74	13.61
	$\boldsymbol{CI}_{0.05}$	$\pm$ 5.83	$\pm$ 8.35	$\pm$ 4.75	$\pm$ 7.74	$\pm$ 5.08
$\chi(\boldsymbol{PF})$	μ	44.70	44.64	44.18	45.28	43.54
	σ	1.42	1.50	1.16	1.37	1.25
	$\boldsymbol{CI}_{0.05}$	$\pm$0.53	$\pm$0.56	$\pm$0.43	$\pm$ 0.51	$\pm$ 0.47

classes of problems, depending on the size and complexity of the problems considered. Interesting future research could focus on combining the promising characteristics of VEPSO(1), VEPSO(2), VEPSO(3), and VEPSO(4) into an improved vector evaluated

Table 5.8: Investigating alternative VEPSO-based strategies for the 256-operation problem

	Strategy	**VEPSO(1)**	**VEPSO(2)**	**VEPSO(3)**	**VEPSO(4)**	**VEPSO(5)**
$S(\boldsymbol{PF})$	μ	3.28×10^{10}	2.93×10^{10}	4.21×10^{10}	3.02×10^{10}	2.31×10^{10}
	σ	7.82×10^{9}	6.39×10^{9}	1.05×10^{10}	6.08×10^{9}	3.06×10^{9}
	$\boldsymbol{CI}_{0.05}$	$\pm 2.92 \times 10^{9}$	$\pm 2.38 \times 10^{9}$	$\pm 3.92 \times 10^{9}$	$\pm 2.27 \times 10^{10}$	$\pm 1.14 \times 10^{9}$
$N(\boldsymbol{PF})$	μ	181.73	130.93	166.67	131.97	76.20
	σ	36.54	23.12	37.00	39.07	24.11
	$\boldsymbol{CI}_{0.05}$	± 13.64	± 8.63	±13.81	±14.59	±9.00
$\chi(\boldsymbol{PF})$	μ	102.35	97.51	108.96	95.07	87.27
	σ	4.57	5.79	4.39	6.10	5.48
	$\boldsymbol{CI}_{0.05}$	± 1.71	± 2.16	± 1.64	± 2.28	± 2.05

PSO algorithm.

The results further indicated that the VEPSO(5) algorithm was unsuitable for solving the *Optimatix* problem, since this algorithm performed significantly worse than the other four VEPSO algorithms over all three test problems. This poor performance is most probably due to the dominance-based selection mechanism used to update the *pbest* values of the particles. Only updating *pbest* with dominating solutions could result in infrequent updates and good solutions with respect to individual objective functions not being used during the optimization process.

When scalability is considered, all algorithms generally scale well with respect to the extent of the Pareto fronts obtained. More work is, however, required to improve the scalability with respect to the S metric and number of non-dominated solutions obtained.

5.4 A further investigation of the VEDE algorithm

Similar to the analysis conducted in the previous section with respect to the VEPSO algorithm, the effect of alternative information exchange strategies on the performance of the VEDE algorithm should also be investigated. This section introduces two variations on Parsopoulos *et al.*'s VEDE algorithm [108] before presenting the results of the empirical comparison.

5.4.1 Variations on the basic VEDE algorithm

In this section, the basic VEDE algorithm is denoted by VEDE(1) and is used for reference purposes throughout the rest of this section.

VEDE(2): Since the best solution of each population is migrated to a neighbouring population in the VEDE(1) algorithm, information with respect to good solutions is physically removed from the current population. This could result in important information being lost. The VEDE(2) algorithm addresses this issue by retaining the best solutions in their respective swarms. Alternative reproduction operators are defined instead and DE/best/bin and DE/rand-to-best/bin (DE/R2B/bin) may be redefined as DE/best$_{m_s}$/bin and DE/R2B$_{m_s}$/bin , where best$_{m_s}$ refers to the best individual from population m_s.

VEDE(3): This algorithm employs the same strategy as the VEPSO(2) algorithm to speed up the propagation of information throughout all populations. The "no-move strategy" of VEDE(2) is retained and each population randomly selects another population to use as basis for differential mutation. In other words, m_s, is sampled from a random uniform distribution, $U(1, \ldots, n_p)$, where n_p denotes the total number of populations.

5.4.2 Comparative analysis of alternative VEDE strategies

For comparison purposes the same experimental set-up as defined in Section 4.2, in terms of parameters and base vector selection strategies, was used. The results of the subsequent empirical evaluation is recorded in Tables 5.9 through 5.11 and Figure 5.5.

Similar to the results obtained by the investigation into alternative VEPSO algorithms, the results of this section indicate that no statistically significant difference in performance could be identified between any of the VEDE variations on any of the test problems. These findings further support the conclusion of Section 4.2 that the priority-

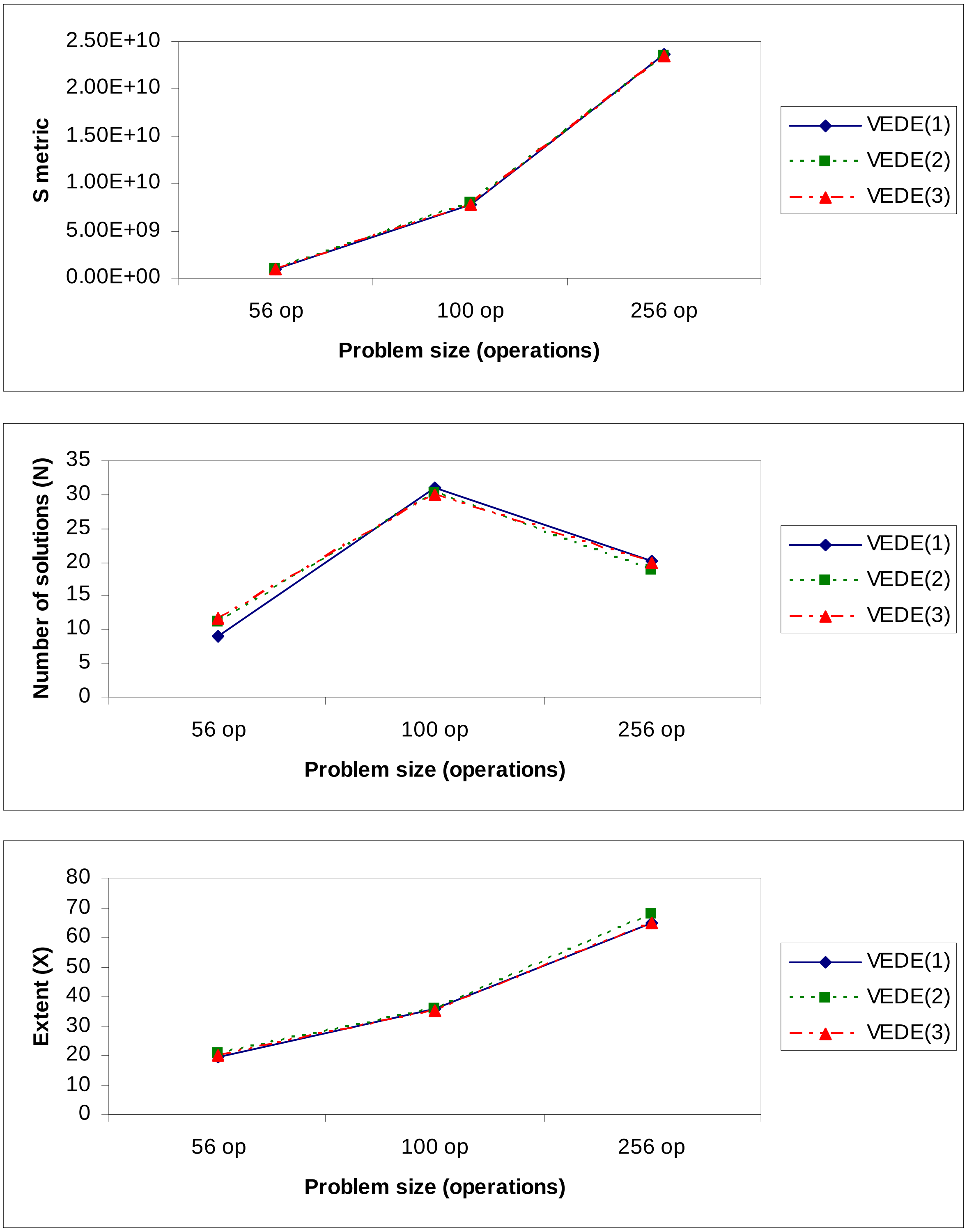


Figure 5.5: Results of the investigation into alternative VEDE-based strategies

Table 5.9: Investigating alternative VEDE-based strategies for the 56-operation problem

	Strategy	**VEDE(1)**	**VEDE(2)**	**VEDE(3)**
$S(\boldsymbol{PF})$	μ	9.47×10^8	9.31×10^8	9.60×10^8
	σ	3.12×10^7	3.87×10^7	2.44×10^7
	$\boldsymbol{CI}_{0.05}$	$\pm 1.16 \times 10^7$	$\pm 1.44 \times 10^7$	$\pm 9.11 \times 10^6$
$N(\boldsymbol{PF})$	μ	8.97	11.10	11.70
	σ	3.12	2.95	3.99
	$\boldsymbol{CI}_{0.05}$	$\pm$ 1.16	$\pm$ 1.10	$\pm$ 1.49
$\chi(\boldsymbol{PF})$	μ	19.51	20.54	19.91
	σ	2.45	2.86	2.70
	$\boldsymbol{CI}_{0.05}$	$\pm$ 0.91	$\pm$ 1.07	$\pm$ 1.01

Table 5.10: Investigating alternative VEDE-based strategies for the 100-operation problem

	Strategy	**VEDE(1)**	**VEDE(2)**	**VEDE(3)**
$S(\boldsymbol{PF})$	μ	7.82×10^9	7.86×10^9	7.82×10^9
	σ	1.63×10^8	1.06×10^8	1.22×10^8
	$\boldsymbol{CI}_{0.05}$	$\pm 6.09 \times 10^7$	$\pm 3.96 \times 10^7$	$\pm 4.56 \times 10^7$
$N(\boldsymbol{PF})$	μ	31.13	30.27	30.03
	σ	4.42	4.32	4.72
	$\boldsymbol{CI}_{0.05}$	$\pm$ 1.65	$\pm$ 1.61	$\pm$ 1.76
$\chi(\boldsymbol{PF})$	μ	35.73	35.49	35.16
	σ	2.36	1.85	2.06
	$\boldsymbol{CI}_{0.05}$	$\pm$ 0.88	$\pm$ 0.69	$\pm$ 0.77

based DE algorithm with optimized control parameters is relatively robust. This implies that no extensive customization is required to optimize the optimization exchange strategy of VEDE-based algorithms.

Table 5.11: Investigating alternative VEDE-based strategies for the 256-operation problem

	Strategy	VEDE(1)	VEDE(2)	VEDE(3)
$S(\boldsymbol{PF})$	μ	2.36×10^{10}	2.35×10^{10}	2.34×10^{10}
	σ	1.02×10^{9}	1.05×10^{9}	1.07×10^{9}
	$\boldsymbol{CI}_{0.05}$	$\pm 3.81 \times 10^{8}$	$\pm 3.92 \times 10^{8}$	$\pm 3.99 \times 10^{8}$
$N(\boldsymbol{PF})$	μ	20.07	18.77	19.97
	σ	5.31	4.32	3.88
	$\boldsymbol{CI}_{0.05}$	± 1.98	± 1.61	± 1.45
$\chi(\boldsymbol{PF})$	μ	65.07	67.76	65.13
	σ	5.64	6.45	6.83
	$\boldsymbol{CI}_{0.05}$	± 2.11	± 2.41	± 2.55

5.5 Summary

The aim of this chapter was to extend the single-objective algorithms developed in Chapter 4 to multi-objective algorithms capable of simultaneously addressing a number of conflicting objective functions. Two vector evaluated algorithms, namely the VEPSO and VEDE, were compared to a DE-based modified goal programming approach. From this empirical evaluation it was shown that the VEPSO algorithm is a suitable alternative to the computationally complex modified GP approach. A further investigation into the effect of alternative information exchange strategies in the context of VEPSO and VEDE was also conducted. This investigation highlighted the robustness of the vector evaluated approach to multi-objective optimization. The next chapter focuses on further reducing the effort associated with customization of optimization algorithms for different production environments and the attainment of a suitable level of algorithm generality.

Chapter 6

Investigating the generality of priority-based metaheuristic algorithms

In recent years, much has been said about the problem dependent nature of metaheuristic algorithms [16]. Algorithm performance often fluctuates dramatically between different problem types and even different instances of the same problem. For example, consider the experiments performed in the past two chapters. The priority-based particle swarm optimization (PSO) algorithm was identified to be the best performing algorithm for the 56 and 100 operation problems while the priority-based differential evolution (DE) algorithm significantly outperformed the PSO algorithm on the 256 operation problem. In fact, throughout the dissertation no single algorithm or algorithm variation could be identified as being the best performing algorithm with respect to all performance measures over all tested data sets.

As a direct result of the no free lunch theorem [144], this problem-dependence is not unexpected. On the other hand, it is commonly thought that certain algorithms are more suited to specific classes of problems. From a "general use" point of view it certainly makes sense to reduce the amount of customization required when the same algorithm is applied to different instances of the same problem. Moreover, the use of a single algorithm in different, yet related, problem environments is an interesting emerging research area.

The purpose of this chapter is to investigate the extent of algorithm customization required to be able to solve a completely different production scheduling problem by means of the same vector evaluated algorithms developed in the previous chapter. The first section introduces the problem environment before Section 6.2 provides a brief synopsis of related literature. The application of the vector evaluated PSO (VEPSO) and vector evaluated DE (VEDE) algorithms to the proposed problem are described in Section 6.3. Finally, in Section 6.4 a self-adaptive DE algorithm is investigated as a mechanism for reducing the effort associated with algorithm customization.

6.1 The *Centurion Ice Cream* scheduling problem

Centurion Ice Cream and Sweets CC is a small family-owned business specializing in the packaging and distribution of ice cream products. The company services more than 400 South African schools and the annual turnover is well in excess of \$500 000 per year. The establishment is thus the ideal size for investigating the performance of the priority-based vector evaluated algorithms in a different, yet related environment.

The production process at *Centurion Ice Cream* consists of a number of simple steps which are indicated in Figure 6.1. Firstly, ice cream in liquid form is mixed and transformed into various ice-cream products ranging from 100ml cups of soft serve ice cream to ice cream cones. After a 12 hour freezing period the dipping process, where ice cream cones are dipped by hand, can commence. After dipping, the products are frozen again before being individually packaged and packed into cardboard boxes, ready for distribution.

An investigation of the process resulted in the ice cream mixing process being identified as the bottleneck [54, 52]. Focusing the scheduling effort on this segment of the process drastically reduces the computational requirements involved. Internal due dates can be defined such that the final product due dates can be met easily by the downstream processes where resources are not fully utilized.

The mixing process can occur on any one of four different ice cream machines, each

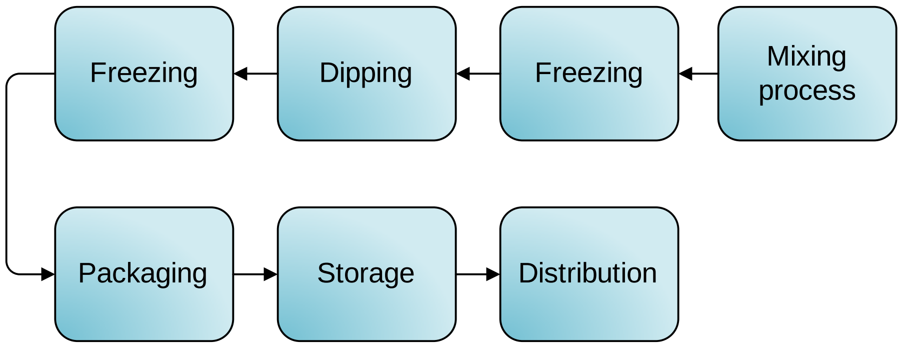


Figure 6.1: The *Centurion Ice Cream* production process

operated by a different operator with a unique work rate. The differing work rates only affect the processing times of the various products since the washing of the machines between processing of two products are not constrained by the work rate of each individual employee but rather by machine capacity, which is equal for all four machines. These set-up times are sequence-dependent since the amount of cleaning required for, for example, processing *vanilla* flavoured ice cream immediately after *chocolate* flavoured ice cream, is significantly less than the cleaning time required for processing *vanilla* immediately after *choc-toffee*. The inclusion of production down time is also an important requirement for this problem since the facility operates in fixed shifts. From a managerial perspective it is important to minimize the deviation between product deliveries and customer requirements, while simultaneously minimizing production lead time.

6.2 An overview of related scheduling literature

From the information provided in the previous section, the *Centurion Ice Cream* problem can be described as a parallel machine scheduling problem. However, similar to the *Optimatix* problem, a number of variations on the parallel machine scheduling problem needs to be combined in order to address the specific complexities of the *Centurion Ice Cream* problem. The two most critical additional complexities include the existence of sequence-dependent set-up times and production down time. This section provides a formal definition of the parallel machine scheduling problem, before the inclusion of sequence-dependent set-up times and unavailability constraints in a parallel machine environment are discussed in more detail.

6.2.1 The classical parallel machine scheduling problem

The classical parallel machine scheduling problem requires that a finite set, $\mathbf{J}$, of n_j jobs, $\{J_v\}_{v=1}^{n_j}$, be processed on a finite set, $\mathbf{M}$, of n_m machines, $\{M_l\}_{l=1}^{n_m}$. All jobs consist of only one operation which may be scheduled on any of the machines in $\boldsymbol{M}$, all jobs are available for scheduling at time zero and no preemption is allowed. The purpose is, firstly, to allocate all operations from set $\boldsymbol{J}$ to set $\boldsymbol{M}$ and secondly, to determine a processing sequence for each machine such that a pre-defined objective function can be optimized [93]. A large number of papers have already been written about this class of scheduling problems and comprehensive reviews can be found in Mokotoff [93] and Cheng and Sin [31].

Three different types of parallel machine scheduling problems have been defined in literature, namely parallel machine problems with identical, uniform, and unrelated machines. All operation process times are equal regardless of machine allocation if the machines are identical. Each machine has a different speed which affects the processing time of the allocated operations when the machines are defined as uniform and in the unrelated parallel machine scheduling problem machine speed also affects the processing times of jobs, but no relationship between the processing times of the same job on different machines can be determined [93]. Based on these definitions, the *Centurion Ice Cream* scheduling problem can be defined as a uniform parallel machine scheduling problem.

6.2.2 The parallel machine scheduling problem with sequence-dependent set-up times

The addition of set-up times as an additional job constraint is discussed in detail in Section 2.2. The parallel machine scheduling problem with sequence-dependent set-up times also requires the addition of a set-up constraint. By defining $S_{\kappa_k k}$ as the sequence-dependent set-up time associated with operation k when processed immediately after operation κ_k on the same resource, the finishing time, z_k, of operation k can be calculated as

$$z_k = t_k + p_{kl} + S_{\kappa_k k} \tag{6.1}$$

where t_k is the starting time of operation k and p_{kl} is the processing time of operation k on resource l.

A detailed review of scheduling literature with set-up times can be found in Allahverdi *et al.* [8], which discusses different variations of parallel machine scheduling problems with sequence-dependent set-up times. This problem variation has already been solved by heuristic methods, branch and bound algorithms, tabu search, simulated annealing, and genetic algorithms [83].

6.2.3 The parallel machine scheduling problem with unavailability constraints

The parallel machine problem with unavailability constraints is closely related to the job shop scheduling problem with unavailability constraints which was discussed in Section 2.2. Production calendars, holidays, preventative maintenance and unexpected breakdowns have a significant effect on machine availability. These resource unavailability intervals may be either fixed before the scheduling effort, thus adding additional constraints to be considered, or can be considered as decision variables where the start and end times of these intervals need to be determined by the scheduling algorithm.

The literature on parallel machine scheduling with unavailability constraints is discussed in detail in Lee [70]. It is notable that, although a number of researchers have considered unavailability intervals as constraints, only one example is listed where non-availability intervals are considered as decision variables. Lee and Chen [71] have used a branch and bound algorithm based on the column generation approach to schedule both jobs and maintenance activities.

The *Centurion Ice Cream* scheduling environment is characterized by fixed production down time intervals which affect all resources as well as unavailability intervals due to scheduled maintenance which may be performed at any time during the planning horizon. This work thus focuses on determining an “optimal” sequence and allocation of operations and scheduled maintenance activities subject to unavailability constraints due to production down time. All unavailability intervals are semiresumable type II intervals [70] where an additional set-up is incurred at each job interruption.

6.2.4 Summary

Using the literature discussed in this section as a point of departure, Graham’s three field notation [51] can be used to describe the problem as a multi-objective uniform parallel machine scheduling problem with sequence-dependent set-up times and unavailability intervals. Makespan and total earliness/tardiness may be selected as suitable objective functions to minimize simultaneously.

The selection of a suitable solution strategy should, in most cases, be driven by the the complexity of the problem. The proposed problem can be considered a generalization of the uniform parallel machine scheduling problem, which is considered to be NP hard. The *Centurion Ice Cream* problem is thus also NP hard. Sufficient motivation subsequently exists for applying approximation methods, such as the priority-based multi-objective algorithms of the previous chapter, to the proposed problem. The next section focuses on the actual application of these algorithms in the *Centurion Ice Cream* environment.

6.3 Vector evaluated algorithms for the *Centurion Ice Cream* problem

Very little adaptation to the priority-based algorithms of Chapter 4 is required to allow for the scheduling of maintenance intervals — an important requirement for the *Centurion Ice Cream* environment. The rest of this section provides a brief description of the changes required with respect to problem representation before the performance of the VEPSO and VEDE algorithms are investigated.

The particle representation of the modified priority-based algorithm consists of a $2n+\varpi$-dimensional vector. Here n denotes the number of manufacturing operations and ϖ is the number of maintenance sessions to be scheduled. The total number of tasks to be scheduled is then $n + \varpi$. The sequencing variables of dimensions 1 to n denote the priority values of each of the tasks. The existing priority-based algorithm described in Section 4.1 can subsequently be used “as-is” to address these priorities. The same can be said for the resource allocation variables of dimensions $n + 1$ to $2n + \varpi$.

Of the vector evaluated algorithms of the previous chapter, the VEPSO(1) and VEPSO(2) algorithms as well as the VEDE(1) algorithm were selected for comparison purposes. Since the VEDE(2) and VEDE(3) algorithms did not seem to perform significantly better than the VEDE(1) algorithm in the experiments conducted in Chapter 5, an additional variation on the VEDE algorithm, namely VEDE(4), was used for the purposes of this investigation. VEDE(4) is simply a combination of VEDE(1) and VEDE(3), where the best individual of each population is moved to a neighbouring population m_s, where m_s is sampled from a random uniform distribution, $U(1, \ldots, n_p)$ and n_p denotes the total number of populations.

All experiments recorded in this section were performed on real data sets, which were obtained through observation and time studies of the production processes at *Centurion Ice Cream*. All three data sets, which are available from the author, require the scheduling of 30 customer orders and two scheduled maintenance intervals. However, different order sizes corresponding to the slow winter months, average production rates, and peak summer months, were used.

A preliminary algorithm analysis resulted in the parameter values listed in Table 6.1 being defined as suitable for initial comparison purposes. In support of the objectives of this chapter, very little effort was made to optimize the algorithm parameters at this stage of the investigation. Using the notation of the previous chapters, the number of particles in the swarm is denoted by n_s and the size of the discretization intervals, δ_k, of operation k is dependent on λ, where

$$\delta_k = \frac{2\lambda}{|\boldsymbol{Q}_k|} \tag{6.2}$$

λ is the number of discretization intervals which are allocated to a single operation-resource pair, $|\boldsymbol{Q}_k|$ is the number of resources on which operation k may be scheduled, and $a \longrightarrow b$ indicates that the associated parameter is decreased linearly from a to b over 95% of the total number of iterations, I_{max}. The sequencing and resource allocation variables were initialized randomly within the intervals $\{-500, 500\}$ and $\{-1500, 1500\}$.

Table 6.1: VEDE and VEPSO algorithm parameters

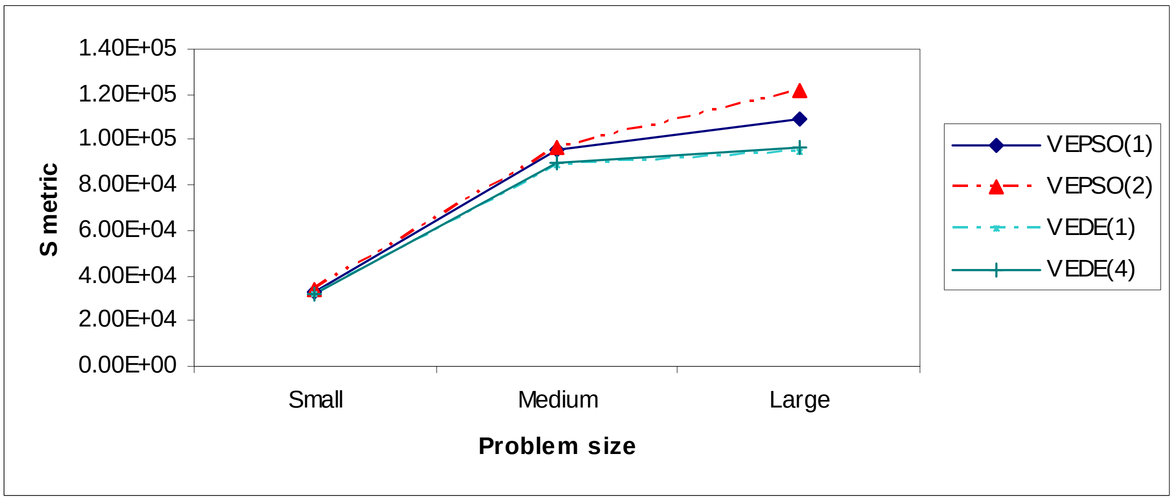
Parameter	**Value used**
General parameters	
n_s	30
I_{max}	200
PSO-specific parameters	
c_1	$2.0 \longrightarrow 1.0$
c_2	$2.8 - c_1$
w	$0.8 \longrightarrow 0.4$
DE-specific parameters	
p_r	$0.75 \longrightarrow 0.25$
F	$0.75 \longrightarrow 0.125$

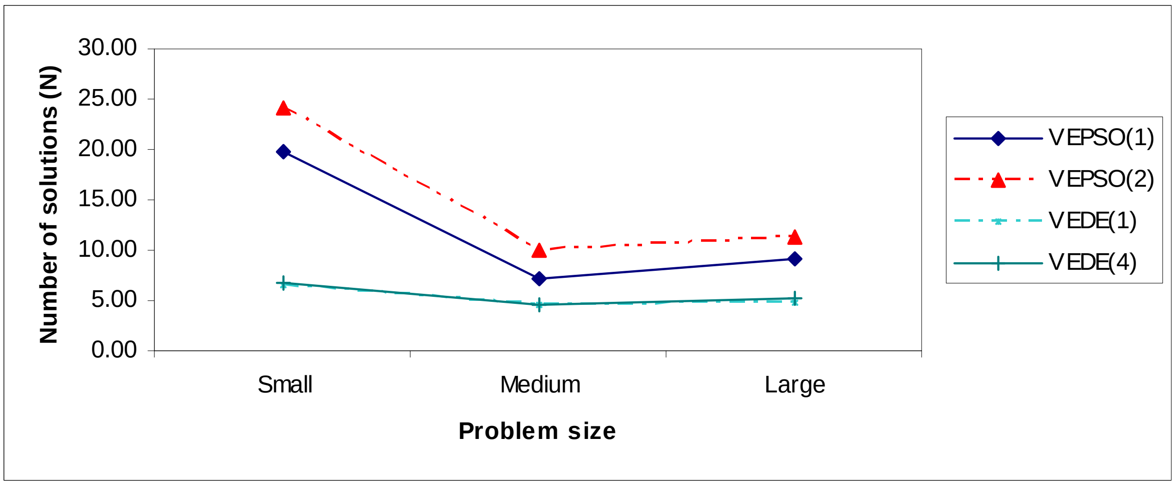
A DE/rand-to-best/bin (DE/R2B/bin) algorithm formed the basis for the VEDE algorithms. Similarly, a *gbest* PSO algorithm was used for the VEPSO algorithms. Both of the algorithms made use of an external archive of unlimited size [61] to store all non-dominated solutions. This archive was initialized as an empty set.

The same three performance measures which were used in the previous chapter, were again used to compare the relative performance of the VEPSO and VEDE algorithms on the *Centurion Ice Cream* problem. The S metric [67, 161], $S(\boldsymbol{PF})$, measures the size of the region dominated by the Pareto front based on a predefined reference vector, $\boldsymbol{f}_{ref}$. The reference vectors used for the three data sets are indicated in Table 6.2. The other two measures include the size of the approximated Pareto front or number of non-dominated solutions obtained, $N(\boldsymbol{PF})$, as well as the extent of the Pareto front, $\chi(\boldsymbol{PF})$ [39].

The VEPSO and VEDE algorithm results are recorded in Tables 6.3 through 6.5 and Figure 6.2. The mean and standard deviation over 30 replications, associated with each of the corresponding performance measurements is denoted by μ and σ. $\boldsymbol{CI}_{0.05}$ is a 95% confidence interval on μ.

From the results it is clear that the VEPSO-based algorithms outperformed the





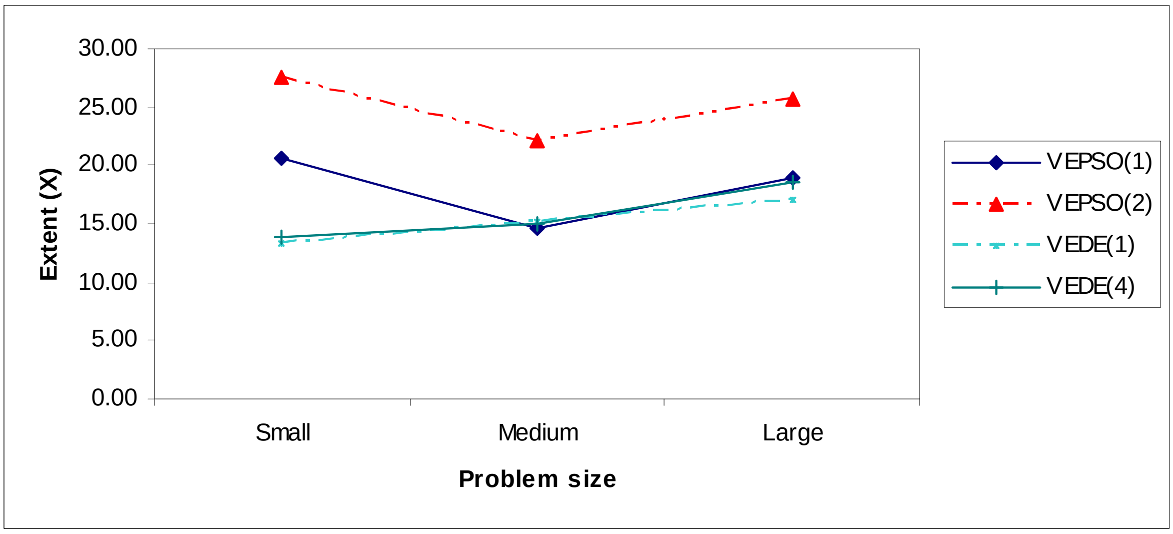


Figure 6.2: Results of the application of VEPSO and VEDE to the *Centurion Ice Cream* problem

Table 6.2: The reference vectors associated with each of the three data sets

Problem size	**Vector**
Small	{100, 1500}
Medium	{200, 1500}
High	{300, 1500}

Table 6.3: Investigating alternative VEPSO and VEDE-based strategies for the small-sized *Centurion Ice Cream* problem

Strategy		**VEPSO(1)**	**VEPSO(2)**	**VEDE(1)**	**VEDE(4)**
$S(\boldsymbol{PF})$	μ	3.26×10^4	3.40×10^4	3.21×10^4	3.22×10^4
	σ	2.08×10^3	2.43×10^3	1.23×10^3	1.08×10^3
	$\boldsymbol{CI}_{0.05}$	$\pm 7.78 \times 10^2$	$\pm 9.06 \times 10^2$	$\pm 4.58 \times 10^2$	$\pm 4.04 \times 10^2$
$N(\boldsymbol{PF})$	μ	19.87	24.20	6.60	6.83
	σ	4.01	5.08	2.03	2.17
	$\boldsymbol{CI}_{0.05}$	± 1.50	± 1.90	± 0.76	± 0.81
$\chi(\boldsymbol{PF})$	μ	20.71	27.54	13.28	13.85
	σ	2.57	3.93	2.91	3.38
	$\boldsymbol{CI}_{0.05}$	± 0.96	± 1.47	± 1.09	± 1.26

VEDE-based algorithms with respect to all performance measurements over all data sets. Similar to the results obtained on the *Optimatix* problem, the algorithms again seemed to be relatively robust with respect to alternative information exchange mechanisms. This was determined from the fact that no significant difference in performance could be identified between VEPSO(1) and VEPSO(2), or between VEDE(1) and VEDE(4), over any of the data sets.

In terms of algorithm scalability, the VEPSO(2), VEDE(1), and VEDE(4) algorithms tended to not scale well with respect to the S metric when compared to VEPSO(1). However, the difference in algorithm performance decreased as the problem size increased

Table 6.4: Investigating alternative VEPSO and VEDE-based strategies for the medium-sized *Centurion Ice Cream* problem

Strategy		**VEPSO(1)**	**VEPSO(2)**	**VEDE(1)**	**VEDE(4)**
$S(\boldsymbol{PF})$	μ	9.53×10^4	9.68×10^4	8.86×10^4	8.98×10^4
	σ	6.96×10^3	5.76×10^3	5.31×10^3	4.38×10^3
	$\boldsymbol{CI}_{0.05}$	$\pm 2.60 \times 10^3$	$\pm 2.15 \times 10^3$	$\pm 1.98 \times 10^3$	$\pm 1.64 \times 10^3$
$N(\boldsymbol{PF})$	μ	7.23	10.10	4.67	4.50
	σ	3.35	2.58	2.02	1.74
	$\boldsymbol{CI}_{0.05}$	± 1.25	± 0.96	± 0.75	± 0.65
$\chi(\boldsymbol{PF})$	μ	14.54	22.21	15.12	15.08
	σ	6.42	4.32	4.63	5.06
	$\boldsymbol{CI}_{0.05}$	± 2.40	± 1.61	± 1.73	± 1.89

Table 6.5: Investigating alternative VEPSO and VEDE-based strategies for the large-sized *Centurion Ice Cream* problem

Strategy		**VEPSO(1)**	**VEPSO(2)**	**VEDE(1)**	**VEDE(4)**
$S(\boldsymbol{PF})$	μ	1.09×10^5	1.22×10^5	9.47×10^4	9.67×10^4
	σ	1.66×10^4	2.39×10^4	7.84×10^3	7.30×10^3
	$\boldsymbol{CI}_{0.05}$	$\pm 6.18 \times 10^3$	$\pm 8.91 \times 10^3$	$\pm 2.93 \times 10^3$	$\pm 2.73 \times 10^3$
$N(\boldsymbol{PF})$	μ	9.03	11.33	4.73	5.30
	σ	2.93	3.02	1.76	1.90
	$\boldsymbol{CI}_{0.05}$	± 1.09	± 1.13	± 0.66	± 0.71
$\chi(\boldsymbol{PF})$	μ	18.93	25.68	17.04	18.65
	σ	6.83	4.57	6.26	5.52
	$\boldsymbol{CI}_{0.05}$	± 2.55	± 1.71	± 2.34	± 2.06

with respect to the number of solutions and extent of the obtained Pareto fronts.

It is interesting to note that the relative rankings of the algorithms seemed to remain almost identical for both the *Optimatix* and the *Centurion Ice Cream* problems. For example, on both the *Optimatix* and the *Centurion Ice Cream* data sets, the VEPSO-based algorithms outperformed the VEDE-based algorithms by a significant margin. These findings imply that the success of the VEPSO-based algorithms is not specific to the *Optimatix* problem only, but can be generalized to a related multiple machine multi-objective scheduling problem. The reason for the similar performance probably lies in the similar structures of the vector evaluated *Optimatix* and vector evaluated *Centurion Ice Cream* algorithms.

The VEPSO algorithms do, however, suffer from a large number of algorithm-specific parameters which need to be optimized separately for each specific data set and problem type. To address this issue, a number of researchers have developed self-adaptive or parameter-free algorithms [2, 24, 34, 82, 153]. The next section investigates the use of a self-adaptive differential evolution algorithm in a multi-objective multiple machine environment.

6.4 Self-adaptive algorithms for the *Centurion Ice Cream* problem

Parameter-free or self-adaptive algorithms are specifically useful for minimizing the computational effort associated with the optimization of control parameters. The effort can be minimized by, for example, defining one or more algorithm parameters as decision variables which is then evolved over time [101]. Due to the large number of self-adaptive DE algorithms which have already been developed [2, 24, 82, 153] and the fewer parameters associated with classical DE algorithms when compared to classical PSO algorithms, this section focuses solely on investigating the use of self-adaptive DE algorithms in the *Centurion Ice Cream* environment. The rest of this section describes the selected algorithm in more detail before the performance of the self-adaptive DE algorithm is compared to the standard VEDE algorithms of the previous section.

6.4.1 Self-adaptive differential evolution

Recently an empirical analysis of the performance of a number of well-known adaptive DE approaches has been performed. The self-adaptive DE algorithm of Omran *et al.* [101] generally outperformed the other DE algorithms on all the benchmark functions. This algorithm was thus considered to be the best point of departure for this investigation.

The self-adaptive DE algorithm [101] is based on two main ideas. Firstly, the reproduction probability, p_r, is sampled from a normal distribution for each dimension of each individual during each iteration. Secondly, each individual i is assigned a unique scaling factor, F_i, which is evolved as a decision variable over the total number of iterations, I_{max}, of the algorithm. If $x_{ij}(t)$ is defined as the j^{th} dimension of individual i of generation t and $i \neq i_1 \neq i_2 \neq i_3$, the equation for the target vector, $\boldsymbol{T}_i$, is given as

$$T_{ij}(t) = x_{i_1j}(t) + F_i(x_{i_2j}(t) - x_{i_3j}(t)) \tag{6.3}$$

where

$$F_i(t) = F_{i_4}(t) + N(0, 0.5)(F_{i_5}(t) - F_{i_6}(t)), \tag{6.4}$$

$i_4 \neq i_5 \neq i_6$, and $i_4, i_5, i_6 \sim U(1, \ldots, n_s)$. Then, for all dimensions, n_x, if $r \sim U(0,1) \leq N(0.5, 0.15)$ or $j = \nu \sim U(1, ..., n_x)$

$$c_{ij}(t) = T_{ij}(t) \tag{6.5}$$

otherwise $c_{ij}(t) = x_{ij}(t)$, where n_x is the total number of dimensions, n_s is the number of individuals in the population, c_{ij} denotes the j^{th} dimension of the i^{th} trial vector and $N(\mu, \sigma^2)$ denotes a number sampled from a normal distribution with mean μ and standard deviation σ.

6.4.2 A comparative analysis of VEDE and S-VEDE

In this section, the two existing DE-based algorithms, namely VEDE(1) and VEDE(4), are compared against self-adaptive VEDE(1) and VEDE(4) algorithms. The self-adaptive VEDE algorithms are respectively referred to as S-VEDE(1) and S-VEDE(4). To facilitate the comparison, the same experimental procedure and settings as defined in the previous section were used. As recommended by Omran *et al.* [101], the scaling factor, F_i, associated with each particle, was initialized as a random number sampled from a normal distribution with mean 0.5 and variance 0.15.

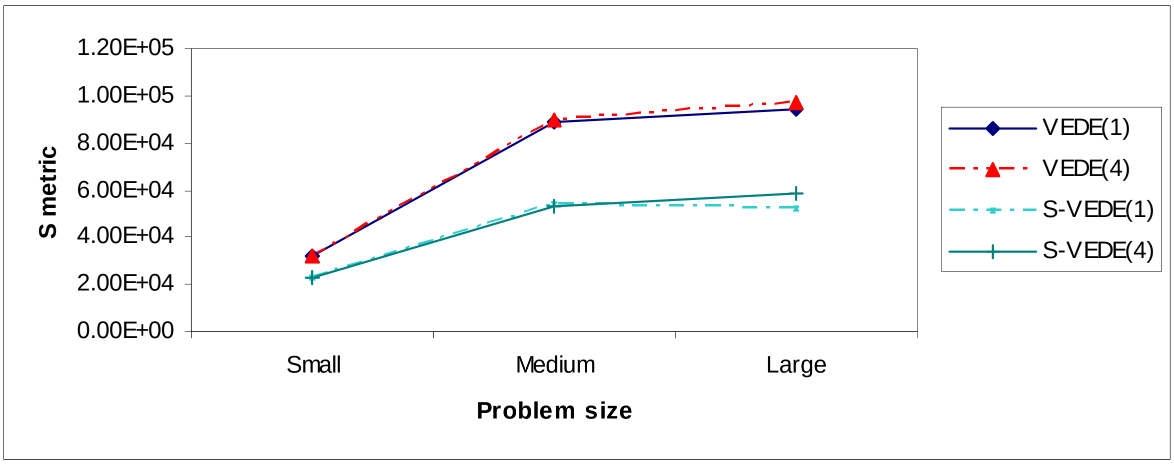
The results are recorded in Tables 6.6 through 6.8 and Figure 6.3. With respect to the S metric, the VEDE(1) and VEDE(4) algorithms outperformed the S-VEDE(1) and S-VEDE(4) algorithms on all three tested data sets. Furthermore, the performance of the self-adaptive algorithms could also be shown to degrade significantly as problem size increased.

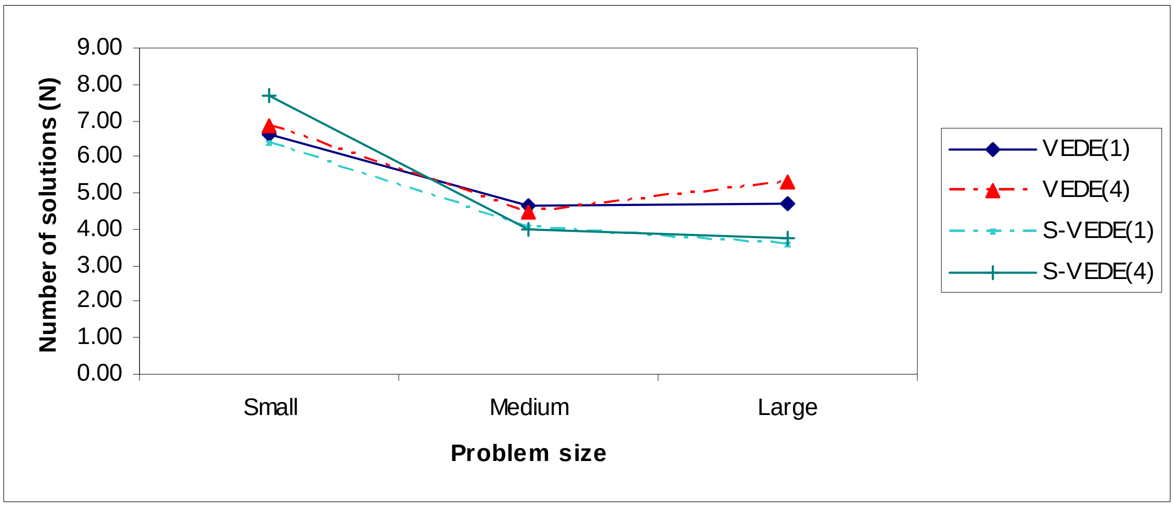
Table 6.6: Investigating self-adaptive differential evolution for the small-sized *Centurion Ice Cream* problem

Strategy		**VEDE(1)**	**VEDE(4)**	**S-VEDE(1)**	**S-VEDE(4)**
$S(\boldsymbol{PF})$	μ	3.21×10^4	3.22×10^4	2.31×10^4	2.29×10^4
	σ	1.23×10^3	1.08×10^3	1.59×10^3	1.41×10^3
	$\boldsymbol{CI}_{0.05}$	$\pm 4.58 \times 10^2$	$\pm 4.04 \times 10^2$	$\pm 5.92 \times 10^2$	$\pm 5.28 \times 10^2$
$N(\boldsymbol{PF})$	μ	6.60	6.83	6.37	7.67
	σ	2.03	2.17	1.59	1.67
	$\boldsymbol{CI}_{0.05}$	± 0.76	± 0.81	± 0.59	±0.62
$\chi(\boldsymbol{PF})$	μ	13.28	13.85	17.88	19.45
	σ	2.91	3.38	3.38	3.00
	$\boldsymbol{CI}_{0.05}$	± 1.09	± 1.26	± 1.26	± 1.12

The poor performance of the S-VEDE algorithms can be explained by considering the changes made to the size of the search space. With the addition of n_s extra decision variables used for evolving the scaling factor, F_i, of each individual, the size of the search space of the S-VEDE algorithms is $2n + \varpi + n_s$. In contrast to the $2n + \varpi$ dimensional search space of the VEDE algorithms, the n_s additional variables result in a more difficult problem which needs to be solved under the same experimental conditions.

With respect to the extent of the Pareto fronts as well as the number of non-dominated solutions obtained, the S-VEDE algorithms performed significantly better.





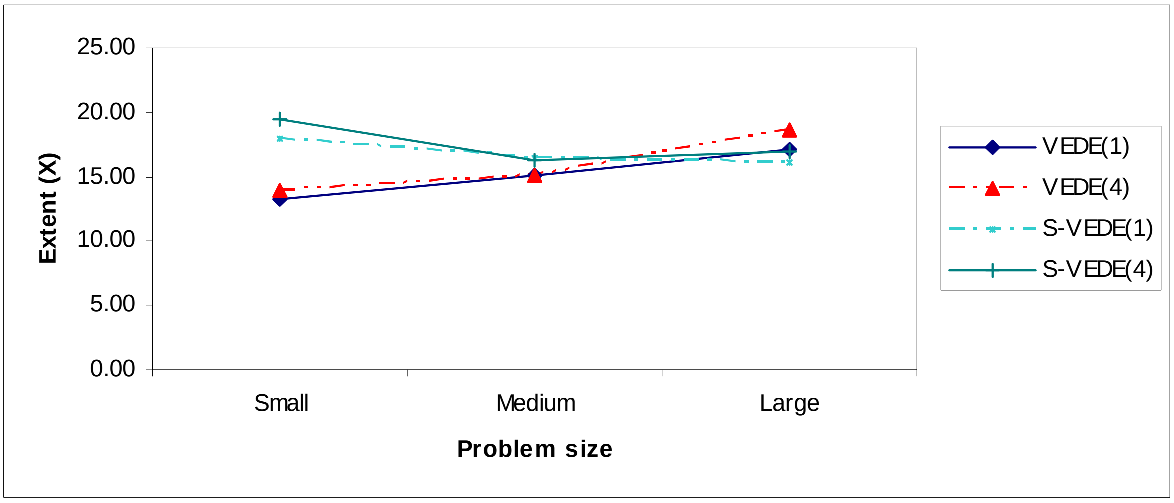


Figure 6.3: Results of the investigation into the use of self-adaptive differential evolution

Table 6.7: Investigating self-adaptive differential evolution for the medium-sized *Centurion Ice Cream* problem

Strategy		**VEDE(1)**	**VEDE(4)**	**S-VEDE(1)**	**S-VEDE(4)**
$S(\boldsymbol{PF})$	μ	8.86×10^4	8.98×10^4	5.43×10^4	5.33×10^4
	σ	5.31×10^3	4.38×10^3	4.72×10^3	4.13×10^3
	$\boldsymbol{CI}_{0.05}$	$\pm 1.98 \times 10^3$	$\pm 1.64 \times 10^3$	$\pm 1.76 \times 10^3$	$\pm 1.54 \times 10^3$
$N(\boldsymbol{PF})$	μ	4.67	4.50	4.03	4.00
	σ	2.02	1.74	1.43	1.49
	$\boldsymbol{CI}_{0.05}$	$\pm$ 0.75	$\pm$ 0.65	$\pm$ 0.53	$\pm$ 0.56
$\chi(\boldsymbol{PF})$	μ	15.12	15.08	16.46	16.34
	σ	4.63	5.06	4.91	5.68
	$\boldsymbol{CI}_{0.05}$	$\pm$ 1.73	$\pm$ 1.89	$\pm$ 1.83	$\pm$ 2.12

Table 6.8: Investigating self-adaptive differential evolution for the large-sized *Centurion Ice Cream* problem

Strategy		**VEDE(1)**	**VEDE(4)**	**S-VEDE(1)**	**S-VEDE(4)**
$S(\boldsymbol{PF})$	μ	9.47×10^4	9.67×10^4	5.23×10^4	5.90×10^4
	σ	7.84×10^3	7.30×10^3	2.51×10^4	3.48×10^4
	$\boldsymbol{CI}_{0.05}$	$\pm 2.93 \times 10^3$	$\pm 2.73 \times 10^3$	$\pm 9.38 \times 10^3$	$\pm 1.30 \times 10^4$
$N(\boldsymbol{PF})$	μ	4.73	5.30	3.57	3.73
	σ	1.76	1.90	1.38	1.76
	$\boldsymbol{CI}_{0.05}$	$\pm$ 0.66	$\pm$ 0.71	$\pm$ 0.52	$\pm$ 0.66
$\chi(\boldsymbol{PF})$	μ	17.04	18.65	16.17	16.90
	σ	6.26	5.52	6.11	7.35
	$\boldsymbol{CI}_{0.05}$	$\pm$ 2.34	$\pm$ 2.06	$\pm$ 2.28	$\pm$ 2.74

The S-VEDE algorithms outperformed the VEDE algorithms on the smallest problem. Performance again decreased as problem-size increased. However, the extent of and number of solutions in the Pareto fronts obtained by the S-VEDE algorithms on the largest problem are not significantly different from those obtained by the VEDE algorithms.

The improved performance of the S-VEDE algorithms with respect to the extent and size of the obtained Pareto fronts provide a good indication that the S-VEDE algorithms are able to obtain Pareto fronts with a relatively high level of diversity. These findings could be expected since the scaling factor of each individual is "optimized" individually. These individually "optimized" F_i values allow individuals to better balance exploration and exploitation depending on their relative location in the objective space. This may in turn lead to more individuals converging to unexplored regions of the objective space.

It should be noted, however, that the purpose of using self-adaptive algorithms in this dissertation was to reduce the effort associated with algorithm customization. For the *Centurion Ice Cream* problem the VEDE algorithms with arbitrarily selected parameters outperformed the S-VEDE algorithms. Although these results reflect positively on the robustness of the VEDE algorithms, it indicates that the S-VEDE algorithms were not successful under the experimental conditions of this dissertation at maintaining acceptable solution quality while reducing the required customization effort. In future, further refinement might show otherwise, but currently well-tuned static parameters seem to be a more suitable alternative for the multi-objective multiple machine environments of this dissertation.

The usability of the VEDE-based algorithms can be further illustrated by an example Pareto Front indicated in Figure 6.4. By taking into account additional qualitative factors which were not incorporated earlier in the optimization process, management is able to select the most suitable solution.

6.5 Summary

The aim of this chapter was to investigate the generality of the priority-based vector evaluated algorithms developed in the previous chapters. Four promising VEDE and VEPSO algorithms of Chapter 5 were applied to the *Centurion Ice Cream* production environment. The robustness of the vector evaluated algorithms was emphasized by the ability of the priority-based VEPSO and VEDE algorithms to solve a completely different, yet related multi-objective multiple machine scheduling problem without the need for significant customization. The relative performance of the various algorithms were shown to support the results obtained for the *Optimatix* problem. Finally, the performance of a self-adaptive VEDE algorithm was investigated. However, the performance and generality of the VEDE algorithms could not be further improved.

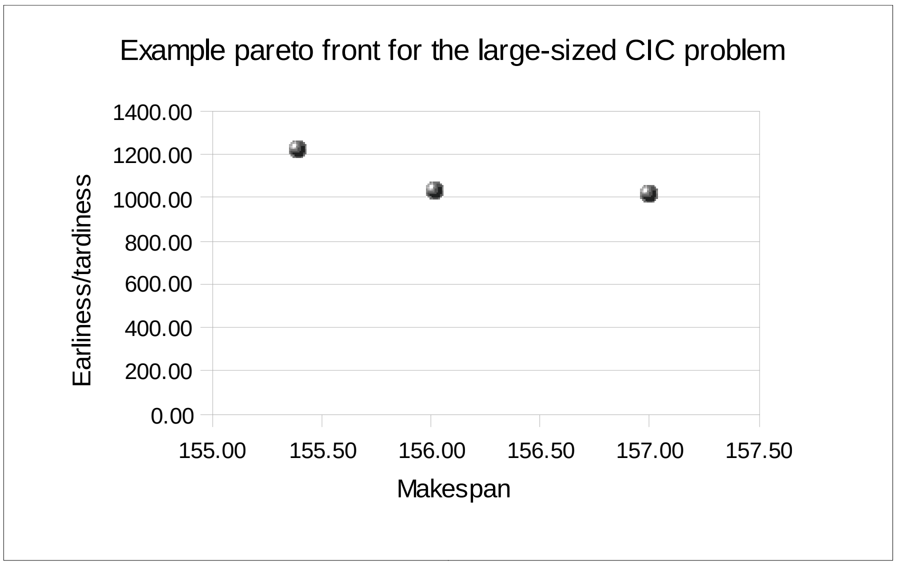


Figure 6.4: An example Pareto front for the large-sized *Centurion Ice Cream* problem, as obtained by the VEDE(4) algorithm

Chapter 7

Conclusion

This dissertation touched on a number of issues of interest in multi-objective multiple machine scheduling environments ranging from alternative problem representations to improved multi-objective optimization strategies. This chapter summarizes the main findings of this study before opportunities for future research are identified and discussed.

7.1 Summary

The purpose of this dissertation was to investigate the use of particle swarm optimization (PSO) and differential evolution (DE) in multiple machine multi-objective production scheduling environments. To achieve this aim, the scheduling problem faced by *Optimatix*, a South African software company specializing in supply chain optimization, was considered. An analysis of the variations on the classical job shop scheduling problem resulted in the *Optimatix* problem being modeled as a multi-objective flexible job shop scheduling problem with sequence-dependent set-up times, auxiliary resources, and production down time. Three objective functions, namely makespan, earliness/tardiness and queue time had to be minimized simultaneously.

The selection of an appropriate problem representation was considered to be one of the most important design choices in the development of a PSO or DE-based scheduling algorithm for the *Optimatix* problem. Subsequently, three PSO-based scheduling algorithms were developed: the priority-based PSO (P-PSO), the random keys PSO (RKPSO), and the rule-based PSO (RBPSO) algorithms. An empirical comparison of the three algorithms resulted in the RBPSO outperforming the other two algorithms for all three customer data sets. However, based on solution time and an analysis of algorithm complexity, the P-PSO algorithm was considered to be more suitable for the purposes of *Optimatix*.

After a suitable problem representation was selected, the P-PSO algorithm was extended to the DE domain and alternative PSO topologies and DE base vector selection strategies were investigated. The implemented variations include the Von Neumann guaranteed convergence PSO (GCPSO) [138], *gbest* GCPSO, DE/best/bin, DE/rand/bin and DE/rand-to-best/bin algorithms. It soon became clear that the priority-based algorithms were relatively robust with respect to alternative PSO and DE variations, but that the use of either PSO or DE was more problem-dependent. To ensure that an accurate evaluation of each algorithm was performed, the algorithm parameters were optimized for each of the five algorithms over each of the three customer data sets. Subsequently, the best performing algorithm, with its associated parameter values, could be identified for each of the three customer data sets. These optimized algorithm-data set pairs were then used throughout the rest of the dissertation to evaluate various other algorithmic aspects.

Benchmarking the priority-based PSO and DE algorithms against existing rule-based *Optimatix* algorithms showed that the priority-based algorithms could outperform the *Optimatix* algorithms with respect to all objective functions over all tested data sets. The priority-based algorithms were also benchmarked against existing literature. To the best of the author's knowledge, Norman and Bean's random keys genetic algorithm [99] is the only other algorithm specifically developed to address flexible job shop scheduling problems with constraints similar to those required by *Optimatix*. Benchmarking against the random keys genetic algorithm further emphasized the contribution of the priority-based problem representation. For all three data sets the priority-based representation outperformed all other tested algorithms.

A number of options exist for extending the single objective priority-based algorithms to multi-objective algorithms capable of simultaneously optimizing three objective functions. An investigation into alternative multi-objective optimization (MOO) strategies for the *Optimatix* problem was quite insightful. This investigation involved a comparison between an *a priori* modified goal programming approach and an *a posteriori* multiple population vector evaluated approach. The vector evaluated PSO (VEPSO) algorithm was identified as the most suitable MOO algorithm for *Optimatix* due to its reduced computational complexity and acceptable solution quality. An additional investigation into the effect of alternative information exchange strategies on the performance of the VEPSO and VEDE algorithms further highlighted the robustness of the vector evaluated approach.

The algorithms in this dissertation was developed with the aim of being generally applicable to multi-objective multiple machine scheduling problems. The generality of the priority-based modified vector evaluated algorithms was subsequently considered by investigating algorithm performance in the *Centurion Ice Cream* production scheduling environment. This completely different, yet related, multi-objective multiple machine problem was modeled as a uniform parallel machine scheduling problem with sequence-dependent set-up times and unavailability intervals. The relative performance of the vector evaluated algorithms was shown to be similar for both the *Optimatix* and the *Centurion Ice Cream* problems. The robustness of the vector evaluated algorithms was further emphasized by the inability of a self-adaptive VEDE algorithm to improve algorithm performance.

Overall, the modified priority-based VEPSO algorithm seems to be both a fast, effective, and generally applicable solution strategy for multiple machine multi-objective production scheduling problems. A number of opportunities for improvement does, however, exist and is listed in the next section.

7.2 Future research opportunities

The opportunities for future research can be categorized into eight main focus areas which are discussed in detail throughout the rest of this section.

Alternative metaheuristics:
The scope of this dissertation was limited to investigating the application of continuous algorithms such as PSO and DE to multi-objective multiple machine scheduling problems. An investigation into the use of alternative metaheuristic-based algorithms for addressing both the *Optimatix* and *Centurion Ice Cream* problems may be quite insightful. Tabu search is commonly thought to be well suited to job shop-based problems [57]. The use of the inherently discreet ant colony optimization algorithm or computationally simple evolutionary programming algorithm might also prove to be useful.

Alternative single-objective PSO and DE algorithms:
Since the development of the original PSO and DE algorithms in 1995, a number of improved variations on these standard algorithms have been proposed. In this study only the Von Neumann and *gbest* guaranteed convergence PSO algorithms [138] and three variations on the DE/y/bin algorithm [131] have been used. Other PSO and DE algorithm variations could potentially lead to improved performance when employed within the priority-based framework. Examples for further investigation include the barebones PSO and DE as well as the charged PSO and various other memetic PSO [39] and DE algorithms [1, 14, 98]. These algorithms have been shown to be superior to the classical PSO and DE algorithms when tested on existing benchmark data sets.

Further mention should also be made of the cooperative PSO algorithm of Van den Bergh *et al.* [137], which have been used for training product unit neural networks. The cooperative PSO algorithm is a multiple population technique which splits the original problem into several smaller problems which can be solved more effectively. This technique could be useful in addressing the scalability issues experienced by the priority-based PSO algorithm.

Alternative PSO and DE multi-objective optimization algorithms:
Multi-objective optimization is one of the fastest growing subject areas in the field of optimization and as a direct result a large number of multi-objective algorithms exist. This study made use of an *a priori* modified goal programming approach as well as an *a posteriori* vector evaluated MOO approach for addressing the multiple objective functions. A number of alternative PSO and DE *a posteriori* approaches exist. In Section 5.2, the work of Reyes-Sierra and Coello Coello [120], Xue *et al.* [148], Abbas

et al. [2], and Madavan [86] was briefly mentioned. An investigation into the use of these more advanced MOO strategies in a multiple machine environment might lead to significant performance improvement.

Alternative problem representations:
Three problem representations, namely the priority-based, random keys, and rule-based representations were investigated in this dissertation. However, alternative representations are obviously possible. Representing the problem by means of a disjunctive graph has already been used effectively in complex job shop environments [89, 90].

Defining discrete scheduling specific operators:
Two main approaches for solving discrete combinatorial problems by means of continuous optimization algorithms were discussed in Section 3.4. This dissertation focused on converting multiple machine scheduling problems into continuous optimization problems which can be solved easily by means of PSO and DE. However, the second approach, where the standard algorithm operators are redefined to exploit the inherent structure of the scheduling problem, may also hold significant performance improvement potential.

Improving algorithm generality:
Due to the large investment in time and resources required to customize optimization algorithms for specific applications, the generality of optimization algorithms have received increasing attention in the last decade. An initial attempt was made in Chapter 6 to investigate the applicability of the vector evaluated algorithms to an additional multiple machine multi-objective optimization problem. However, significant room for improvement still exists.

Hyper-heuristics can be loosely defined as an emerging search technology where a specific (meta)heuristic is used to intelligently select other (meta)heuristics for solving a specific optimization problem [26, 28]. Since a hyper-heuristic operates on a search space of meta(heuristic) algorithms, the search methodology is able to maintain a relatively high level of generality. In order to address a completely different type of optimization problem, only the fitness function and candidate meta(heuristics) need to be redefined. It could be interesting to compare a PSO or DE-based hyper-heuristic to the vector evaluated priority-based algorithms of this dissertation.

Improving the realism of the proposed algorithms:
The trade-off between realism and tractability becomes an important consideration when modeling a real world optimization problem. Ideally the model should be as realistic as possible. However, as the complexity of a problem formulation increases, it becomes more and more difficult to develop a quick and reliable solution strategy. Fortunately, as improved search methodologies are being developed, more and more complex problems can be solved efficiently.

If the *Centurion Ice Cream* problem is taken as an example, a number of opportunities for improving the realism of the model can be identified. Prioritizing customer orders, allowing for uncertain processing times, and explicitly considering the dynamic nature of the production scheduling environment, may all positively impact the realism of the model formulation. The requirement for real time rescheduling can also result in interesting future research since the PSO algorithm is well known for its ability to track and optimize dynamically changing objective functions and constraints. Research shows that PSO converges faster and obtains higher quality solutions when compared to a number of other dynamic evolutionary algorithms [37].

Investigating differentiating problem characteristics: Throughout the dissertation it was clear that no single algorithm could be shown to outperform all other investigated algorithms for all performance measurements over all data sets. An investigation into the problem characteristics which drive algorithm performance might shed light on which algorithms are more suitable for solving which types of multiple machine problems.

7.3 Last words

As technology is evolving and more advanced optimization techniques are becoming the norm, both the academic and business worlds are starting to realize the importance of addressing increasingly complex scheduling scenarios. This dissertation attempted to make a contribution towards the application of PSO and DE to multi-objective multiple machine scheduling problems. In so doing it also highlighted the numerous opportunities for future research in this area.

Bibliography

[1] H. A. Abbas. A memetic pareto evolutionary approach to artificial neural networks. *Lecture Notes in Computer Science*, 2256:113–152, 2001.

[2] H. A. Abbas, R. Sarker, and C. Newton. PDE: A pareto-frontier differential evolution approach for multi-objective optimization problems. *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 971–978, 2001.

[3] Y. Abdedda'im and O. Maler. Pre-emptive job shop scheduling problem using stopwatch automata. *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 113–126, 2002.

[4] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34:391–401, 1988.

[5] F. S. Al-Anzi and A. Allahverdi. A self-adaptive differential evolution heuristic for two-stage assembly scheduling problem to minimize maximum lateness with setup times. *European Journal of Operational Research*, 182:80–94, 2007.

[6] K. A. Aldakhilallah and R. Ramesh. Cyclic scheduling heuristics for a re-entrant job shop manufacturing environment. *International Journal of Production Research*, 39(12):2635–2657, 2001.

[7] A. Allahverdi, J. N. D. Gupta, and T. Aldowaisan. A review of scheduling research involving setup considerations. *Omega*, 1999:219–239, 1998.

[8] A. Allahverdi, C. T. Ng, T. C. E. Cheng, and M. Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187:985–1032, 2008.

[9] D. Anghinolfi and M. Paolucci. A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, 193(1):73–85, 2007.

[10] M. Asano and H. Ohta. Scheduling with shutdowns and sequence dependent set-up times. *International Journal of Production Research*, 37(7):1661–1676, 1991.

[11] M. E. Aydin and E. Oztemel. Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, 33:169–178, 2000.

[12] B. V. Babu and R. Angira. Optimization of non-linear functions using evolutionary computation. *Proceedings of* 12^{th} *ISME Conference*, pages 153–157, 2001.

[13] A. Bauer, B. Bullnheimer, R. F. Hartl, and C. Strauss. An ant colony optimization approach for the single machine totaltardiness problem. *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, 2:1445–1450, 1999.

[14] P. K. Bergey and C. Ragsdale. Modified differential evolution: a greedy random strategy for genetic recombination. *Omega*, 33(3):255–265, 2005.

[15] S. Bertel and J. C. Billaut. A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation. *European Journal of Operational Research*, 159:651–662, 2004.

[16] H. Beyer and K. Deb. On self-adaptive features in real-parameter evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 5(3):250–270, 2001.

[17] T. M. Blackwell and P. J. Bentley. Improvised music with swarms. *Proceedings of the IEEE Congress on Evolutionary Computation*, 2:1462–1467, 2002.

[18] J. Blazewicz, W Domschke, and E. Pesch. The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, 93:1–33, 1996.

[19] C. Blum and M. Sampels. An ant colony optimization algorithm for shop scheduling problems. *Journal of Mathematical Modelling and Algorithms*, 3:285–308, 2004.

[20] C. A. Brizuela, Y. Zhao, and N. Sannomiya. No-wait and blocking job-shops: challenging problems for GA's. *Proceedings of the 2001 International Conference on Systems, Man, and Cybernetics*, pages 2349–2354, 2001.

[21] P. Brucker. *Scheduling Algorithms.* Springer, 4th edition, 2004.

[22] P. Brucker and T. Kampmeyer. Tabu search algorithms for cyclic machine scheduling problems. *Journal of Scheduling*, 8:303–322, 2005.

[23] P. Brucker and A. Kramer. Polynomial algorithms for resource-constrained and multiprocessor task scheduling problems. *European Journal of Operational Research*, 90:214–226, 1996.

[24] L. Bui, Y. Shan, F. Qi, and H. Abbas. Comparing two versions of differential evolution in real parameter optimization. Technical Report TR-ALAR-200504009, The artificial life and adaptive robotics laboratory, School of IT and electrical engineering, University of New South Wales, 2005.

[25] R. L. Burdett and E. Kozan. Evolutionary algorithms for resource constrained non-serial mixed flow shops. *International Journal of Computational Intelligence Applications*, 3(4):411–435, 2003.

[26] E. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg. Hyper-heuristics: an emerging direction in modern search technology. *Chapter 16 in the Handbook of Meta-heuristics (F. Glover and G. Kochenberger)*, pages 457–474, 2003.

[27] E. K. Burke and G. Kendall. *Search methodologies, introductory tutorials in optimization and decision support techniques.* Springer, 2006.

[28] E. K. Burke, G. Kendall, and E. Soubeiga. A tabu search hyper-heuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470, 2003.

[29] G. Cavory, R. Dupas, and G. Goncalves. A genetic approach to solving the problem of cyclic job shop scheduling with linear constraints. *European Journal of Operational Research*, 161:73–85, 2005.

[30] J. Chen and J. C. Pan. Integer programming models for the re-entrant shop scheduling problem. *Engineering Optimization*, 38(5):577–592, 2006.

[31] T. Cheng and C. Sin. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47:271–292, 1990.

[32] W. Cheung and H. Zhou. Using genetic algorithms and heuristics for job shop scheduling with sequence-dependent setup times. *Annals of Operations Research*, 107:65–81, 2001.

[33] D. Chung, K. Lee, K. Shin, and J. Park. A new approach to jobshop scheduling problems with due date constraints considering operation subcontracts. *International Journal of Production Economics*, 98:238–250, 2005.

[34] M. Clerc. Tribes - un exemple d'optimisation par essaim particulaire sans parametres de contr. ole. *Optimisation par Essaim Particulaire (OEP 2003)*, 2003.

[35] S. Dauzère-Pérès and J. Paulli. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70:281–306, 1997.

[36] M. Dorigo and T. Stützle. *Ant colony optimization.* MIT Press, 2004.

[37] R. C. Eberhart and Y. Shi. Particle swarm optimization: Developments, applications, and resources. *Proceedings of the 2001 IEEE Congress on Evolutionary Computation*, pages 81–86, 2001.

[38] A. P. Engelbrecht. *Computational intelligence: an introduction.* John Wiley and Sons, Ltd, 2003.

[39] A. P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence.* Wiley, 2005.

[40] A. P. Engelbrecht and A. Ismail. Training product unit neural networks. *Stability and Control: Theory and Applications*, 2(1-2):59–74, 1999.

[41] M. F. Ercan and Y. Fung. Performance of particle swarm optimization in scheduling hybrid flow-shops with multiprocessor tasks. *Lecture Notes in Computer Science*, 4707:309–318, 2007.

[42] I. Essafi, Y. Mati, and Dauzere-Peres. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers and Operations Research*, 35:2599–2616, 2008.

[43] Feoktistov. *Differential evolution in search of solutions*, volume 5 of *Optimization and its Applications.* Springer, 2006.

[44] L. D. Fredendall, S. A. Melnyk, and G. Ragatz. Information and scheduling in a dual resource constrained job shop. *International Journal of Production Research*, 34(10):2783–2802, 1996.

[45] J. Gao, M. Gen, and L. Sun. A hybrid of genetic algorithm and bottleneck shifting for flexible job shop scheduling problem. *Proceedings of the* 8^{th} *Annual Conference on Genetic and Evolutionary Computation*, pages 1157–1164, 2006.

[46] V. B. Gargeya and R. H. Deane. Scheduling research in multiple resource constrained job shops: a review and critique. *International Journal of Production Research*, 34(8):2077–2097, 1996.

[47] J. Giffler and G. L. Thompson. Algorithms for solving production scheduling problems. *Operations Research*, 8:487–503, 1960.

[48] F. Glover. Tabu search—part 1. *ORSA Journal On Computing*, 1(3):190–206, 1989.

[49] J. F. Gongalves, J. J. de Magalh aes Mendes, and Resende M. G. C. A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167:77–95, 2005.

[50] M. A. Gonzalez, C. R. Vela, M. Sierra, I. Gonzales, and R. Varela. Comparing schedule generation schemes in memetic algorithms for the job shop scheduling problem with sequence dependent setup times. *Lecture Notes in Artificial Intelligence*, 4293:472–482, 2006.

[51] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[52] J. Grobler and A. P. Engelbrecht. A scheduling-specific modeling approach for real world scheduling. *Proceedings of the 2007 IEEE International Conference on Industrial Engineering and Engineering Management*, pages 85–89, 2007.

[53] J. Grobler, A. P. Engelbrecht, S. Kok, and V. S. S. Yadavalli. Metaheuristics for the multi-objective FJSP with sequence-dependent set-up times, auxiliary resources and machine down time. *Annals of Operations Research*. Submitted for review.

[54] J. Grobler and V. S. S. Yadavalli. Managing the cold chain: a case study at a South African ice cream company. *South African Journal of Industrial Engineering*, 2008. Accepted for publication.

[55] D. J. Hoitomt, P. B. Luh, and K. R. Pattipati. A practical approach to job-shop scheduling problems. *IEEE Transactions on Robotics and Automation*, 9(1):1–13, February 1993.

[56] H. Hwang and J. U. Sun. Production sequencing problem with reentrant work flows and sequence-dependent set-up times. *Computers and Industrial Engineering*, 33(3–4):773–776, 1997.

[57] A. S. Jain and S. Meeran. Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, 113:390–434, 1999.

[58] K. Jansen, M. Mastrolilli, and R. Solis-Oba. Approximation schemes for job shop scheduling problems with controllable processing times. *European Journal of Operational Research*, 167:297–319, 2005.

[59] Z. Jia, H. Chen, and J. Tang. An improved particle swarm optimization for multi-objective flexible job-shop scheduling problem. *Proceedings of the 2007 IEEE International Conference on Grey Systems and Intelligent Services*, pages 1587–1592, 2007.

[60] Z. Jia, H. Chen, and J. Tang. A new multi-objective fully-informed particle swarm algorithm for flexible job-shop scheduling problems. *Proceedings of the 2007 International Conference on Computational Intelligence and Security Workshops*, pages 191–194, 2007.

[61] Y. Jin, M. Olhofer, and B. Sendhoff. Dynamic weighted aggregation for evolutionary multi-objective optimization: why does it work and how? *Proceedings of 2001 Conference on Genetic and Evolutionary Computation*, 2001.

[62] I. Kacem, S. Hammadi, and P. Borne. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems, Man and Cybernetics - part C: Applications and Reviews*, 32(1):1–13, 2002.

[63] I. Kacem, S. Hammadi, and P. Borne. Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and Computers in Simulation*, 60:245–276, 2002.

[64] J. Kennedy and R. Eberhart. Particle swarm optimization. *Proceedings of the IEEE International Confererence on Neural Networks*, 4:1942–1948, 1995.

[65] J. Kennedy, R. C. Eberhart, and Y. Shi. *Swarm Intelligence*. Morgan Kaufmann Publishers, 2001.

[66] J. Kennedy and R. Mendes. Population structure and particle performance. *Proceedings of the IEEE Congress on Evolutionary Computation*, 2:1671–1676, 2002.

[67] J. D. Knowles. *Local-search and hybrid evolutionary algorithms for pareto optimization.* PhD thesis, University of Reading, 2002.

[68] A. H. Land and A. G. Doig. An automatic method for solving discrete programming problems. *Econometrics*, 28:83–97, 1960.

[69] C. Le Pape and P. Baptiste. Heuristic control of a constraint-based algorithm for the preemptive job-shop scheduling problem. *Journal of Heuristics*, 5:305–325, 1999.

[70] C. Lee. Machine scheduling with availability constraints. Chapter 22 in the Handbook of Scheduling: Algorithms, Models and Performance Analysis (J. Leung). Chapman and Hall/CRC., 2004.

[71] C. Lee and Z. Chen. Scheduling jobs and maintenance activities on parallel machines. *Naval Research Logistics*, 47(2):145–165, 2000.

[72] S. M. Lee and A. A. Asllani. Job scheduling with dual criteria and sequence-dependent setups: mathematical versus genetic programming. *Omega*, 32(2):145–153, 2004.

[73] D. Lei. A pareto archive particle swarm optimization for multi-objective job shop scheduling. *Computers and Industrial Engineering*, 54(4):960–971, 2008.

[74] D. Lei and H. Xiong. An efficient evolutionary algorithm for multi-objective stochastic job shop scheduling. *Proceedings of the* 6^{th} *International Conference on Machine Learning Cybernetics*, pages 867–872, 2007.

[75] Z. Lian, B. Jiao, and X. Gu. A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan. *Applied Mathematics and Computation*, 183:1008–1017, 2006.

[76] Z. Lian, B. Jiao, and X. Gu. A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan. *Applied Mathematics and Computation*, 175:773–785, 2006.

[77] B. Liu, L. Wang, and Y. Jin. Hybrid particle swarm optimization for flow shop scheduling with stochastic processing time. *Lecture Notes in Artificial Intelligence*, pages 630–637, 2005.

[78] B. Liu, L. Wang, and Y. Jin. An effective hybrid particle swarm optimization for no-wait flow shop scheduling. *International Journal of Advanced Manufacturing Technology*, 31:1001–1011, 2007.

[79] B. Liu, L. Wang, and Y. Jin. An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers. *Computers and Operations Research*, 35:2791–2806, 2008.

[80] H. Liu, A. Abraham, O. Choi, and S. H. Moon. Variable neighbourhood particle swarm optimization for multi-objective flexible job-shop scheduling problems. *Lecture Notes in Computer Science*, pages 197–204, 2006.

[81] H. Liu, A. Abraham, and C. Grosan. A novel variable neighbourhood particle swarm optimization for multi-objective flexible job-shop scheduling problems. *Proceedings of the* 2^{nd} *International Conference on Digital Information Management*, pages 138–145, 2007.

[82] J. Liu and J. Lampinen. A fuzzy adaptive differential evolution algorithm. *Proceedings of the IEEE International Region 10 Conference*, pages 606–611, 2002.

[83] A. G. Lockett and A. P. Muhlemann. A scheduling problem involving sequence dependent changeover times. *Operations Research*, 20(4):895–902, 1972.

[84] T. Loukil and J. Teghem. A review of multi-objective production scheduling. Technical report, Faculte Polytechnique de Mons, 2004.

[85] T. Loukil, J. Teghem, and P. Fortemps. A multi-objective production scheduling case study solved by simulated annealing. *European Journal of Operational Research*, 179:709–722, 2007.

[86] N. K. Madavan. Multi-objective optimization using a pareto differential evolution approach. *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1145–1150, 2002.

[87] A. Mascis and D. Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143:498–517, 2002.

[88] F. Mason. Computerized cutting-tool management. *American Machinists and Automated Manufacturing*, 130(5):105–132, 1986.

[89] S. J. Mason, J. W. Fowler, and W. M. Carlyle. A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops. *Journal of Scheduling*, 5(3):247–262, 2002.

[90] S. J. Mason and K. Oey. Scheduling complex job shops using disjunctive graphs: a cycle elimination procedure. *International Journal of Production Research*, 41(5):981–994, 2003.

[91] D. C. Mattfield. *Evolutionary search and the job shop: investigations on genetic algorithms for production scheduling*. Physica-Verlag, 1996.

[92] C. Meloni, D. Pacciarelli, and M. Pranzo. A rollout metaheuristic for job shop scheduling problems. *Annals of Operations Research*, 131:215–235, 2004.

[93] E. Mokotoff. Parallel machine scheduling problems: a survey. *Asia-Pacific Journal of Operational Research*, 18:193–242, 2001.

[94] M. Nakamura, H. Tome, K. Hachiman, B. M. Ombuki, and K. Onaga. Cyclic job-shop-scheduling based on evolutionary petri nets. *Proceedings of the 26th Annual Conference of the IEEE Industrial Electronics Society*, pages 2855–2860, 2000.

[95] K. Natarajan, K. M. Mohanasundaram, B. Shoban Babu, S. Suresh, K. Antony Arokia Durai Raj, and C. Rajendran. Performance evaluation of priority dispatching rules in multi-level assembly job shops with jobs having weights for flowtime and tardiness. *International Journal of Advanced Manufacturing Technology*, 31:751–761, 2007.

[96] A. C. Nearchou. A differential evolution approach for the common due date early/tardy job scheduling problem. *Computers and Operations Research*, 35:1329–1343, 2008.

[97] A. C. Nearchou and S. L. Omirou. Differential evolution for sequencing and scheduling optimization. *Journal of Heuristics*, 12:395–411, 2006.

[98] N. Noman and H. Iba. Enhancing differential evolution performance with local search for high dimensional function optimization. *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pages 967–974, 2005.

[99] B. A. Norman and J. C. Bean. A genetic algorithm methodology for complex scheduling problems. *Naval Research Logistics*, 46(2):199–211, 1999.

[100] W. P. M. Nuijtne and E. H. L. Aarts. A computational study of constraint satisfaction for multiple capacitated job shop scheduling. *European Journal of Operational Research*, 90:269–284, 1996.

[101] M. G. H. Omran, A. P. Engelbrecht, and A. Salman. Empirical analysis of self-adaptive differential evolution. *European Journal of Operational Research*, 183(2):785–804, 2007.

[102] G. Onwubolu and D. Davendra. Scheduling flow shops using differential evolution algorithm. *European Journal of Operational Research*, 171:674–692, 2006.

[103] Q. Pan, M. F. Tasgetiren, and Y. Liang. Minimizing total earliness and tardiness penalties with a common due date on a single-machine using a discrete particle swarm optimization algorithm. *Lecture Notes in Computer Science*, pages 460–467, 2006.

[104] Q. Pan, M. F. Tasgetiren, and Y. Liang. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Proceedings of the 2007 Conference on Genetic and Evolutionary Computation*, pages 126–133, 2007.

[105] Q. Pan and L. Wang. No-idle permutation flow shop scheduling based on a hybrid discrete particle swarm optimization algorithm. *International Journal of Advanced*

Manufacturing Technology, 2007. Article in Press (DOI 10.1007/s00170-007-1252-0).

[106] Q. Pan, L. Wang, M. F. Tasgetiren, and B. H. Zhao. A hybrid discrete particle swarm optimization algorithm for the no-wait flow shop scheduling problem with makespan criterion. *International Journal of Advanced Manufacturing Technology*, 2007. Article in Press (DOI 10.1007/s00170-007-1099-4).

[107] S. S. Panwalker and W. Iskander. A survey of scheduling rules. *Operations Research*, 25(1):45–61, 1977.

[108] K. E. Parsopoulos, D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis. Vector evaluated differential evolution for multiobjective optimization. *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 204–211, 2004.

[109] K. E. Parsopoulos and M. N. Vrahatis. Particle swarm optimization method in multiobjective problems. *Proceedings of the 2002 ACM Symposium on Applied Computing*, pages 603–607, 2002.

[110] V. Patel, H. A. ElMaraghy, and I. Ben-Abdallah. Scheduling in dual-resource constrained manufacturing systems using genetic algorithms. *Emerging Technologies and Factory Automation*, 2:1131–1139, 1999.

[111] S. Petrovic, F. Carole, and D. Petrovic. Job shop scheduling with lot-sizing and batching in an uncertain real-world environment. *Proceedings of the* 2^{nd} *Multidisciplinary International Conference on Scheduling*, pages 363–379, 2005.

[112] C. N. Potts and M. Y. Kovalyov. Scheduling with batching: A review. *European Journal of Operational Research*, 120:228–249, 2000.

[113] C. N. Potts, V. A. Strusevich, and T. Tautenhahn. Scheduling batches with simultaneous job processing for two-machine shop problems. Technical report, University of Southampton, 1998.

[114] K. V. Price, R. M. Storn, and J. A. Lampinen. *Differential evolution, a practical approach to global optimization*. Natural Computing Series. Springer, 2005.

[115] J. G. Qi, G. R. Burns, and D. K. Harrison. The application of parallel multi-population genetic algorithms to dynamic job-shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 16(8):609–615, 2000.

[116] B. Qian, L. Wang, R. Hu, W. Wang, D. Huang, and X. Wang. A hybrid differential evolution method for permutation flow-shop scheduling. *International Journal of Advanced Manufacturing Technology*, 38(7–8):757–777, 2007.

[117] B. Qian, L. Wang, D. Huang, and X. Wang. Multi-objective flow shop scheduling using differential evolution. *Lecture Notes in Computer Science*, 345:1125–1136, 2006.

[118] B. Qian, L. Wang, D. Huang, and X. Wang. Scheduling multi-objective job shops using a memetic algorithm based on differential evolution. *International Journal of Advanced Manufacturing Technology*, 35:1014–1027, 2008.

[119] R. L. Rardin. *Optimization in Operations Research*. Prentice Hall, 1998.

[120] M. Reyes-Sierra and C. A. C. Coello. Multi-objective particle swarm optimizers: a survey of the state-of-the-art. *International Journal of Computational Intelligence Research*, 2(3):287–308, 2006.

[121] C. W. Reynolds. Flocks, herds, and schools: a distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.

[122] M. Ruttgers. Differential evolution: A method for optimization of real scheduling problems. Technical report, Aachen University of Technology, 1997.

[123] B. Ruzek and M. Kvasnicka. Determination of the earthquake hypocenter: a challenge for the differential evolution algorithm. Section 7.5 in Differential evolution, a practical approach to global optimization (Price, K. V., Storn, R. M. and Lampinen, J. A.) Springer., 2005.

[124] A. Salman, I. Ahmad, and S. Al-Madani. Particle swarm optimization for task assignment problem. *Microprocessors and Microsystems*, 26(8):363–371, 2002.

[125] Schaffer. *Multiple objective optimization with Vector Evaluated Genetic Algorithms.* PhD thesis, Vanderbilt University, 1984.

[126] C. J. Schuster and J. M. Framinan. Approximative procedures for no-wait job shop scheduling. *Operations Research Letters*, 31:308–318, 2003.

[127] D. Y. Sha and C. Hsu. A hybrid particle swarm optimization for job shop scheduling problem. *Computers and Industrial Engineering*, 51:791–808, 2006.

[128] M. Singer. Forecasting policies for scheduling a stochastic due date job shop. *International Journal of Production Research*, 38(15):3623–3637, 2000.

[129] M. Singer and M. Pinedo. A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops. *IIE Transactions*, 30:109–118, 1998.

[130] R. Storn. On the usage of differential evolution for function optimization. *Biennial Conference of the North American Fuzzy Information Processing Society*, pages 519–523, 1996.

[131] R. Storn and K. Price. Differential evolution — a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[132] M. F. Tasgetiren, Q. Pan, Y. Liang, and P. Suganthan. A discrete differential evolution algorithm for the total earliness and tardiness penalties with a common due date on a single-machine. *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling*, pages 271–278, 2007.

[133] M. F. Tasgetiren, Q. Pan, P. Suganthan, and Y. Liang. A discrete differential evolution algorithm for the no-wait flowshop scheduling problem with total flowtime criterion. *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling*, pages 251—258, 2007.

[134] M. F. Tasgetiren, M. Sevkli, Y. Liang, and Gencyilmaz. Particle swarm optimization algorithm for single machine total weighted tardiness problem. *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 1412–1419, 2004.

[135] M. F. Tasgetiren, M. Sevkli, Y. Liang, and Gencyilmaz. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research*, 177:1930–1947, 2007.

[136] V. T'Kindt and J. C. Billaut. *Multicriteria Scheduling (Theory, Models and Algorithms)*. Springer, 2002.

[137] F. Van den Bergh and A. P. Engelbrecht. Cooperative learning in neural networks using particle swarm optimizers. *South African Computer Journal*, 26:84–90, 2000.

[138] F. Van den Bergh and A. P. Engelbrecht. A new locally convergent particle swarm optimiser. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 3:6–12, 2002.

[139] D. Vanderpooten. *L'approche interactive dans l'aide multicritere a la decision (in french)*. PhD thesis, University of Paris, 1990.

[140] G. Venter and J. Sobieszczanski-Sobieski. Multidisciplinary optimization of a transport aircraft wing using particle swarm optimization. *Proceedings of the* 9th *AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 2002.

[141] M. G. A. Verhoeven. Tabu search for resource-constrained scheduling. *European Journal of Operational Research*, 106:266–276, 1998.

[142] J. H. Weng, O. Hiroki, and O. Hisashi. An integrated algorithm based on tabu search for flexible assembly job-shop scheduling. *Journal of Japan Industrial Management Association*, 5(4):245–252, 2003.

[143] K. P. White and R. V. Rogers. Job-shop scheduling: Limits of the binary disjunctive formulation. *International Journal of Production Research*, 28(12):2187–2200, 1990.

[144] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[145] W. Xia and Z. Wu. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers and Industrial Engineering*, 48:409–425, 2005.

[146] W. Xia and Z. Wu. A hybrid particle swarm optimization approach for the job-shop scheduling problem. *International Journal of Advanced Manufacturing Technology*, 29:360–366, 2006.

[147] W. Xia, Z. Wu, and G. Yang. A new hybrid optimization algorithm for the job-shop scheduling problem. *Proceedings of the 2004 American Control Conference*, pages 5552–5557, 2004.

[148] F. Xue, A. C. Sanderson, and R. J. Graves. Pareto-based multi-objective differential evolution. *Proceedings of the 2003 IEEE Congress on Evolutionary Computation*, pages 862–869, 2003.

[149] W. Yang and C. Liao. Survey of scheduling research involving setup times. *International Journal of Systems Science*, 30(2):143–155, 1999.

[150] Y. Yoshitomi and R. Yamaguchi. A genetic algorithm and the monte carlo method for stochastic job-shop scheduling. *International Transactions in Operations Research*, 10:577–596, 2003.

[151] H. Yu and W. Liang. Neural network and genetic algorithm-based approach to expanded job-shop scheduling. *Computers and Industrial Engineering*, 39:337–356, 2001.

[152] S. Y. Yun. Genetic algorithm with fuzzy logic controller for preemptive and non-preemptive job-shop scheduling problems. *Computers and Industrial Engineering*, 43:623–644, 2002.

[153] D. Zaharie. Control of population diversity and adaptation in differential evolution algorithms. *Proceedings of the* 9th *International Conference on Soft Computing*, pages 41–46, 2003.

[154] M. Zandieh, S. M. T. F. Ghomi, and S. M. M. Husseini. An immune algorithm approach to hybrid flow shop scheduling with sequence-dependent setup times. *Applied Mathematics and Computation*, 180(1):111–127, 2006.

[155] C. Zhang, P. Li, Z. Guan, and Y. Rao. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers and Operations Research*, 34(11):3229–3242, 2006.

[156] H. Zhang and M. Gen. Multistage-based genetic algorithm for flexible job-shop scheduling problem. *Journal of Complexity International*, 11:223–232, 2005.

[157] J. Zhang and J. Xu. A new differential evolution for discontinuous optimization problems. *Proceedings of the* 3rd *International Conference on Natural Computation*, 2007.

[158] F. Zhao, Y. Hong, and D. Yu. A hybrid approach based on artificial neural network and genetic algorithm for job-shop scheduling problem. *Proceedings of the 2005 International Conference on Neural Networks and Brain*, pages 1687–1692, 2005.

[159] Y. Zhou, B. Li, and J. Yang. Study on job shop scheduling with sequence-dependent setup times using biological immune algorithm. *International Journal of Advanced Manufacturing Technology*, 30:105–111, 2006.

[160] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: empirical results. *Evolutionary Computation*, 8(2):173–195, 1999.

[161] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.

Appendix A

Acronyms

ACO	Ant colony optimization
CLPSO	Convergent linear particle swarm optimization algorithm
DE	Differential evolution
P-DE/R2B/bin	The priority-based DE base vector selection strategy: P-DE/rand-to-best/bin
EA	Evolutionary algorithms
EDD	Earliest due date rule
FJSP	Flexible job shop scheduling problem
FSSP	Flow shop scheduling problem
GA	Genetic algorithm
***gbest* GCP-PSO**	*gbest* guaranteed convergence priority-based PSO
GCPSO	Guaranteed convergence particle swarm optimization
GP	Goal programming
JSSP	Job shop scheduling problem
MOO	Multi-objective optimization

MOSA	Multi-objective simulated annealing
P-DE	Priority-based differential evolution
PDR	Priority dispatch rule
PFSSP	Permutation flow shop scheduling problem
P-PSO	Priority-based particle swarm optimization
PSO	Particle swarm optimization
PSO-GA	Particle swarm optimization-genetic algorithm
RBPSO	Rule-based particle swarm optimization
RKGA	Random keys genetic algorithm
RKPSO	Random keys particle swarm optimization
SA	Simulated annealing
SBH	Shifting bottleneck heuristic
SMP	Single machine scheduling problem
SPT	Shortest processing time rule
S-VEDE	Self adaptive vector evaluated differential evolution
TS	Tabu search
VEDE	Vector evaluated differential evolution
VEGA	Vector evaluated genetic algorithm
VEPSO	Vector evaluated particle swarm optimization
VNGCP-PSO	Von Neumann guaranteed convergence priority-based PSO

Appendix B

Symbols

$a \triangleq$	An arbitrary non-negative number
$\alpha \triangleq$	The first field of Graham's three field notation [51]
$\beta \triangleq$	The second field of Graham's three field notation [51]
$c_{ij}(t) \triangleq$	The j^{th} dimension of the i^{th} offspring at time t
$c_{ijs}(t) \triangleq$	The j^{th} dimension of the i^{th} offspring of population s at time t
$c_1, c_2 \triangleq$	Acceleration constants used in the PSO algorithm
$c_{1\delta}, c_{2\delta} \triangleq$	The per iteration change in the PSO acceleration constants
$\boldsymbol{C} \triangleq$	The set of directed arcs of a disjunctive graph
$C_v \triangleq$	The completion time of job v
$\boldsymbol{CI}_{0.05} \triangleq$	The 95% confidence interval on μ
$\chi_{pl} \triangleq$	The ending time of the p^{th} unavailability interval of resource l
$\chi(\boldsymbol{PF}) \triangleq$	The extent of the Pareto front $\boldsymbol{PF}$
$d_k \triangleq$	The index of the primary resource on which operation k is processed
$d_v \triangleq$	The due date of job v
$\delta_k \triangleq$	The size of the discretization intervals associated with operation k
$\boldsymbol{D} \triangleq$	The set of undirected arcs in a disjunctive graph

$D_{1v} \triangleq$	The absolute deviation between the completion time and the due date of job v
$D_{2v} \triangleq$	The squared deviation between the completion time and the due date of job v
$e_c \triangleq$	The threshold parameter which limits the number of consecutive moves where no improvement occurs in the fitness of the *gbest* particle of the GCPSO algorithm
$e_s \triangleq$	The threshold parameter which limits the number of consecutive improvements in the fitness of the *gbest* particle of the GCPSO algorithm
$\eta \triangleq$	The number of consecutive failures of the *gbest* particle of the GCPSO algorithm
$E_v \triangleq$	The earliness of job v with respect to the job due date
$F \triangleq$	The scaling factor used in the DE algorithm
$F_i \triangleq$	The scaling factor of individual i used in the self adaptive DE algorithm
$F_\delta \triangleq$	The per iteration change in the scaling factor, F
$f_s \triangleq$	The s^{th} objective function
$f^s_{ref} \triangleq$	An initial estimate of the maximum value of objective s
$\mathcal{F} \triangleq$	The objective space
$g_s \triangleq$	The target value of objective s used in the modified goal programming algorithm
$g^s_{ref} \triangleq$	An initial estimate, for reference purposes, of the minimum value of objective s used in the modified GP algorithm
$\gamma \triangleq$	The third field of Graham's three field notation [51]
$h_{kd_k} \triangleq$	The sequence-independent set-up time of operation k if processed on resource d_k
$\boldsymbol{I} \triangleq$	A set of high priority operations
$I_{max} \triangleq$	The maximum allowable number of iterations of an algorithm

$\iota \triangleq$	The number of elite individuals per iteration as used in the RKGA algorithm
$J_v \triangleq$	The v^{th} element in the set $\boldsymbol{J}$ which consists of all jobs to be scheduled
$\jmath \triangleq$	A positive constant used for identifying a user-defined number of target Pareto solutions used in the modified GP algorithm
$\kappa_k \triangleq$	The index of the operation completed immediately before operation k on resource d_k ($\kappa_k = 0$ if such an operation k does not exist)
$L_v \triangleq$	The lateness of job v with respect to the job due date
$\lambda \triangleq$	The number of discretization intervals which are allocated to a single operation-resource pair
$m_s \triangleq$	The index of the neighbouring population of population s for the purposes of information exchange
$\mu \triangleq$	The mean over 30 simulations of the associated performance measurement
$\mu_B \triangleq$	The operation-dependent data point as obtained from the Kacem *et al.* data set [62]
$M_l \triangleq$	The l^{th} element of the set $\boldsymbol{M}$ of all primary resources
$n \triangleq$	The total number of operations which are to be scheduled
$n_j \triangleq$	The total number of jobs to be scheduled
$n_m \triangleq$	The total number of primary resources
$n_p \triangleq$	The total number of swarms or populations
$n_{Q_{max}} \triangleq$	The size of the largest primary resource set
$n_s \triangleq$	The number of particles in a swarm or individuals in a population
$n_v \triangleq$	The number of operations associated with job J_v
$n_\varrho \triangleq$	The number of equality constraints in an optimization problem
$n_\varsigma \triangleq$	The number of inequality constraints in an optimization problem
$n_x \triangleq$	The number of dimensions of a particle or individual
$\nu \triangleq$	A randomly selected dimension

$N_{i_1} \triangleq$	The set of feasible nodes connected to node i_1 in the ACO algorithm
$N(\boldsymbol{PF}) \triangleq$	The number of non-dominated solutions in the Pareto front $\boldsymbol{PF}$
$N(\mu, \sigma^2) \triangleq$	A random number sampled from a normal distribution with mean μ and standard deviation σ
$o_{vl} \triangleq$	The l^{th} operation of job v
$\boldsymbol{\Omega_{d_k}} \triangleq$	The set of downtime intervals which affect resource d_k
$p_{kl} \triangleq$	The processing time of operation k on resource l
$p_{vl} \triangleq$	The processing time of operation l of job J_v on resource M_l as defined for the classical job shop scheduling problem
$p_c \triangleq$	The crossover probability used in a genetic algorithm
$p_{c\delta} \triangleq$	The per iteration change in the crossover probability, p_c
$p_r \triangleq$	The reproduction probability used in a DE algorithm
$P_{i_1 i_2}(t) \triangleq$	The probability of moving from solution i_1 to solution i_2 at time t
$\boldsymbol{P}_s \triangleq$	The particle space consisting of all the possible positions of the particles in the search space
$\boldsymbol{PF} \triangleq$	The set of Pareto-optimal vectors
$\pi_{i_1 i_2} \triangleq$	The pheromone concentration of the link between node i_1 and node i_2 used in the ACO algorithm
$\psi_{kl} \triangleq$	The starting time of the l^{th} unavailability interval of the k^{th} auxiliary resource
$\boldsymbol{\Phi} \triangleq$	The set of operations which have already been scheduled
$\boldsymbol{\Psi}_k \triangleq$	The set of possible starting times of operation k on resource d_k
$q_v \triangleq$	A generic measurement associated with job v
$Q_{kl} \triangleq$	The l^{th} element of the set of primary resources on which operation k may be scheduled
$r_k \triangleq$	The release date of operation k
$\rho(t) \triangleq$	The time-dependent GCPSO scaling factor

$\rho(t)_s \triangleq$	The time-dependent GCPSO scaling factor of swarm s
$s_k \triangleq$	The set-up time associated with operation k
$S_{\kappa_k k} \triangleq$	The sequence-dependent set-up time of operation k if processed immediately after operation κ_k on the same resource
$S_{\kappa_k k d_k}$	The resource-dependent sequence-dependent set-up time of operation k if processed immediately after operation κ_k on resource d_k
$S(\boldsymbol{PF}) \triangleq$	The hyper-volume between the reference vector, $\boldsymbol{f}_{ref}$, and the pareto front, $\boldsymbol{PF}$
$\boldsymbol{S}_s \triangleq$	The schedule allocation space consisting of all feasible schedules
$\sigma \triangleq$	The standard deviation over 30 simulations of the associated performance measurement
$\sigma_B \triangleq$	The variation of all operation process times as calculated from the Kacem *et al.* data set [62]
$t \triangleq$	Time step during the the PSO algorithm's progression
$t_k \triangleq$	The starting time of operation k
$t_{kl} \triangleq$	The starting time of operation k on resource l
$\tau \triangleq$	The index of the best solution in a swarm or population
$\triangle_1, \triangle_2, \triangle_3 \triangleq$	Positive constants used for identifying a user-defined number of target Pareto solutions used in the modified GP algorithm
$T_v \triangleq$	The tardiness of job v with respect to the job due date
$T_{ij}(t) \triangleq$	The j^{th} component of the i^{th} target vector at time t used in the DE algorithm
$\boldsymbol{\Theta} \triangleq$	The set of operations available for scheduling
$U(a,b) \triangleq$	A uniform random distribution with lower bound, a and upper bound, b
$\upsilon \triangleq$	A sufficiently small number
Υ	The temperature of the system used in the SA algorithm

$v_{ij}(t) \triangleq$	The j^{th} dimension of the velocity vector of the i^{th} particle time t used in the PSO algorithm
$v_{ijs}(t) \triangleq$	The j^{th} dimension of the velocity vector of the i^{th} particle of swarm s at time t used in the PSO algorithm
$\boldsymbol{V} \triangleq$	The set of nodes of a disjunctive graph
$V_{kl} \triangleq$	The l^{th} resource in the auxiliary resource set of operation k, where operation k can be processed on any one of the $\lvert\boldsymbol{V}_k\rvert$ resources in the set
$V_{max} \triangleq$	The maximum velocity of a particle used in the PSO algorithm
$\varphi \triangleq$	The number of operations which can only be processed on a single primary resource
$\varpi \triangleq$	The number of maintenance intervals which need to be scheduled
$\varrho_q \triangleq$	The q^{th} equality constraint of an optimization problem
$\varsigma_q \triangleq$	The q^{th} inequality constraint of an optimization problem
$w \triangleq$	The inertia weight used in the PSO algorithm
$w_\delta \triangleq$	The per iteration change in the inertia weight, w
$w_v \triangleq$	The weight associated with job v
$x_{ij}(t) \triangleq$	The j^{th} component of the position vector of the i^{th} individual at time t used in the PSO algorithm
$x^*_j(t) \triangleq$	The j^{th} dimension of the neighbourhood best individual at time t used in the PSO algorithm
$\hat{x}_{ij}(t) \triangleq$	The j^{th} dimension of the best previous position of individual i at time t used in the PSO algorithm
$x_{ijs}(t) \triangleq$	The j^{th} component of the position vector of the i^{th} individual of swarm s at time t used in the PSO algorithm. This definition assumes that swarm s is always assigned objective s for optimization.
$x^*_{js}(t) \triangleq$	The j^{th} dimension of the neighbourhood best individual from population s at time t used in the PSO algorithm. This definition assumes that swarm s is always assigned objective s for optimization.

$\hat{x}_{ijs}(t) \triangleq$	The j^{th} dimension of the best previous position of individual i of swarm s at time t used in the PSO algorithm. This definition assumes that swarm s is always assigned objective s for optimization.
$\boldsymbol{x}_{min} \triangleq$	The lower bound of the decision variables
$\boldsymbol{x}_{max} \triangleq$	The upper bound of the decision variables
$\boldsymbol{X}_s \triangleq$	The s^{th} swarm or population of individuals
$\xi \triangleq$	The number of immigrant individuals per iteration used in the RKGA algorithm
$z_k \triangleq$	The finishing time of operation k
$z_{kl} \triangleq$	The finishing time of operation k on resource l
$\zeta \triangleq$	The number of consecutive successes of the *gbest* particle of the GCPSO algorithm

Appendix C

Derived Publications

This appendix lists all the invited talks and papers that have been published, or are currently under review, that were derived from work done in this dissertation:

- J. Grobler, A.P. Engelbrecht, S. Kok, and V.S.S. Yadavalli. Metaheuristics for the multi-objective FJSP with sequence-dependent set-up times, auxiliary resources and machine down time. *Annals of Operations Research.* Accepted for publication, December 2008.
- J. Grobler, A.P. Engelbrecht, V.S.S. Yadavalli. Multi-objective DE and PSO strategies for production scheduling. *Proceedings of the 2008 IEEE Congress on Evolutionary Computation*, pages 1154–1161, 2008.
- J. Grobler and A.P. Engelbrecht. A scheduling-specific modeling approach for real world scheduling. *Proceedings of the 2007 IEEE International Conference on Industrial Engineering and Engineering Management*, pages 85–89, 2007.
- J. Grobler, A. P. Engelbrecht, J. W. Joubert, S. Kok. A starting time-based approach to production scheduling with Particle Swarm Optimization. *Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Scheduling*, pages 121–128, 2007.
- J. Grobler, A.P. Engelbrecht, V.S.S. Yadavalli. Multi-objective PSO and DE for parallel machine scheduling with setup times and unavailability intervals. In Ap-

plied Swarm Intelligence (A. P. Engelbrecht and M. Middendorf). Springer. Submitted for review, May 2008.

- J. Grobler, A.P. Engelbrecht, V.S.S. Yadavalli, and S. Kok. Multi-objective Particle Swarm Optimization for complex job shop scheduling. *Presented at the* 18th *Triennial Conference of the International Federation of Operational Research Societies (IFORS)*, 2008.