Swarm Intelligence

A multiobjective production scheduling application

Jacomine Grobler

October 13, 2006

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 project a generic production scheduling framework suitable for implementation with all popular metaheuristics is designed and evaluated for *Optimatix*, a Gauteng-based company specializing in supply chain optimization. To address the complex requirements of the customer, various additional constraints were added to the Classical Job Shop Scheduling Problem, including production down-time, scheduled maintenance, machine breakdowns, sequence-dependent set-up times, release dates and multiple predecessors per job. Differentiation between primary resources (machines) and auxiliary resources (labour, tools and jigs) were also achieved. To accommodate the complex variant of the problem, principles of Particle Swarm Optimization (PSO), a stochastic population based optimization technique originating from the study of social behavior of birds and fish, were employed to test the scheduling framework. The algorithm was tested against a comprehensive data set derived from actual data provided by *Optimatix*. The optimization technique showed promising results when compared with other traditional job shop scheduling solution strategies, being strongly dependent on the concepts of social intelligence and emergence. The benefit of an improved production schedule can be generalized to include cost reduction, customer satisfaction, improved profitability and overall competitive advantage.

Keywords: Job shop scheduling problem; Particle Swarm Optimization (PSO)

Acronyms

PSO Particle Swarm Optimization

SA Simulated Annealing

TS Tabu Search

GA Genetic Algorithms

SA-PSO Simulated Annealing-Particle Swarm Optimization

PSO-GA Particle Swarm Optimization-Genetic Algorithm

GCPSO Guaranteed Convergence Particle Swarm Optimization

CLPSO Convergent Linear Particle Swarm Optimization

MOO Multiobjective Optimization

VEPSO Vector Evaluated Particle Swarm Optimization

TSP Travelling Salesperson Problem

MOP Multiobjective Optimization Problem

JSSP Job Shop Scheduling Problem

FJSSP Flexible Job Shop Scheduling Problem

RJSSP Reentrant Job Shop Scheduling Problem

EJSSP Expanded Job Shop Scheduling Problem

EA Evolutionary Algorithms

PFSSP Permutation Flow Shop Scheduling Problem

FSSP Flow Shop Scheduling Problem

SMP Single Machine Scheduling Problem

Contents

List of Figures

Chapter 1

Introduction

1.1 Background and rationale

Impacting on efficiency, customer satisfaction and overall competitive advantage, production scheduling plays an important role in the current business environment. Customers increasingly expect to receive the right product, at the right price, at the right time. In order to meet these requirements, manufacturing companies need to improve their production scheduling performance. The aim of this project is to design a generic production scheduling algorithm for specific implementation in the job shop environment.

Optimatix is a Gauteng-based company specializing in supply chain optimization. Their primary business activities consist of the design and implementation of the *Tactix* product suite. The product suite consists of a number of inter-related modules, which can be implemented independently if required. Providing a customized supply and demand planning solution on a strategic, tactical and operational level, they boast a number of successful implementations in the low-volume-high-variety manufacturing industry. The scheduling module, *Tactix Scheduling*, can be classified as a forwards finite capacity production scheduling tool. Currently, the underlying algorithms of this module consists of rules-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 Blazewicz et al. (1996). However, the use of a more sophisticated approximation technique should be investigated.

Before embarking on a project it is important to evaluate the potential benefits and costs associated with its execution. Hoitomt et al. (1993) consider a number of factors to justify the development of a production scheduling algorithm for the job-shop environment: production scheduling being one of the most important issues in the planning and operation of manufacturing systems. 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 have a significant impact on cost reduction, customer satisfaction, profitability and overall competitive advantage.

Recent customer demand for higher variety products have also contributed to an increase in product complexity. The number of parts being produced in job shop environments has dramatically increased during the past decade.

Production scheduling in the low-volume-high-variety manufacturing environment has already received considerable attention in Operations Research literature. The deterministic job shop scheduling problem has developed a reputation of being notoriously difficult to solve. However, the business requirements of *Optimatix* requires that a much more complex variation of the classical job shop scheduling problem should be addressed. Since this problem can be considered a direct derivation from the classical job shop scheduling problem, it is also classified as *NP*-hard. The sheer magnitude of the resources and operations which the algorithm should be able to schedule, further adds to the complexity of the problem. Therefore, sufficient evidence exist to suggest that this problem can not be solved optimally (Jain and Meeran, 1999).

An extensive analysis of solution strategies, which is documented in Chapter 2, provides the basis for selecting a metaheuristic to solve the identified problem. The allowance of non-improving moves to escape local optima, along with the ability to integrate local search techniques for exploitation purposes ensure that metaheuristics are considered to be the most effective solution strategy (Jain and Meeran, 1999). Particle Swarm Optimization (PSO) is the metaheuristic of choice for a number of reasons.

- Placing emphasis on the concept of social versus individual learning, PSO is a robust algorithm which compares favourably with Genetic Algorithms and Tabu Searches, which are often utilized to solve the job shop scheduling problem (Kennedy et al., 2001).

- PSO is one of the simplest metaheuristics to implement. This inherent simplicity simplifies the design and enhances the user-friendliness of the algorithm (Lian et al., 2006).

- Even though PSO has been the focus of many a research study since its development, it does not have a rich literature with respect to scheduling problems. Therefore, there is a definite research opportunity associated with this optimization technique (Lian et al., 2006).

1.2 Problem definition

Optimatix has identified the need for an in-depth investigation into their production scheduling algorithms. Opportunities for improvement exist in terms of solution quality and model formulation. Furthermore, management is curious with respect to the potential use of metaheuristics to obtain improved schedules. However, any improvements made should retain the functionality of the existing production scheduling model. A compromise can be made with respect to the time required to obtain a solution.

There are a number of other factors affecting the problem formulation that should be considered.

- Definite variations exists between the different production environments serviced by *Optimatix*. It would be beneficial to meet the production scheduling requirements of all of these clients by the implementation of a single algorithm. This implies that the metaheuristic algorithm should be relatively generic.
- In Operations Research literature there is a tendency to focus on solving unrealistically simplified problems. This implies that a very limited amount of information is available with respect to model formulation of complex job shops. This complicates the formulation of a JSSP which is intricate enough to be of value on the production floor.

The research question for the project can thus be formulated as:

> *"Can the use of Particle Swarm Optimization add value to the production scheduling process in complex job-shop environments?"*

1.3 Research design

Using the research question as a point of departure, the research design is analogous to a blueprint of the desired end results of the project (Moutton, 2001). Specifying the research design is helpful in determining the type of study or activities that should be performed to achieve the desired end results.

The purpose of this project is to answer the research question through designing and testing a generic job shop scheduling algorithm geared towards implementation in *Tactix Scheduling*. The design phase of the algorithm mainly consists of defining the structure of the problem and the development of an effective solution strategy. The algorithm's structure makes provision for the following features:

- The scheduling of 500 operations on 216 resources.

- The scheduling of machines, manpower, as well as complementary resources such as jigs and tools.
- Alternative routing and resource allocation.
- The evaluation of quality of schedule with respect to a number of predefined objectives.
- The incorporation of release dates and production calendars. a frozen horizon.
- Determining the effects of sequence dependent set-up times on the schedule.
- Processing time of fifteen minutes per 100 operations.

There are a number of factors which affect the model formulation and solution process. The scope of the model is dependent on the underlying modeling assumptions made. The inherent characteristics of the JSSP result in the model to only be applicable to the low volume-high variety and mid-volume-mid-variety industries (Hoitomt et al., 1993). Furthermore, the algorithm assumes sufficient raw material being available at all times and is based on the principles of PSO as defined in Kennedy et al. (2001).

In order to use PSO, or any other metaheuristic for that matter, to solve the *Optimatix* problem a specific structure is required to transfer solutions between the problem space in which the metaheuristic functions and the space in which the objective function values are to be evaluated. The generic production scheduling framework developed to address this issue has the attractive feature that it is common to all popular metaheuristics. Since the generic production scheduling algorithm is based exclusively on this framework, a large degree of flexibility in terms of alternative metaheuristic-based solution strategies and varieties of scheduling problems, is obtained.

One of the project's main requirements is to ensure that the metaheuristic algorithm is well-tested. The algorithm evaluation phase consists of a comparative analysis between the designed algorithm and the various other algorithms currently implemented at *Optimatix*. *Tactix Scheduling* currently implements priority dispatching rules, iterative schedule generation and random schedule generation methods. The PSO algorithm is also evaluated with respect to both standard benchmark problems defined in literature and benchmark data developed specifically for *Optimatix*.

All required input data for testing the performance on real-world problems was provided by *Optimatix*. Pilot implementation may occur at the discretion of the client to aid further algorithm evaluation. Full implementation feasibility, however, will only be evaluated at a later stage.

In terms of the rest of the document, Chapter 2 consists of a discussion of the applicable scheduling literature. An extensive discussion of the generic production scheduling framework is provided in Chapter 3, while Chapter 4 introduces PSO within the larger framework of Swarm Intelligence and addresses its application to the *Optimatix* problem. Chapter 5 provides a discussion around the various algorithm development activities. All algorithm testing activities are discussed in Chapter 6 along with the results obtained during algorithm development. Finally, Chapter 7 concludes the document and provides insight into future research opportunities.

Chapter 2

The *Optimatix* problem — a literature review

A review of the existing literature is of paramount importance. The purpose of this review is to become aware of the most recent theories, results, definitions, tools and techniques in the field of job shop scheduling. This prevents duplication and provides valuable insight towards a point of departure (Moutton, 2001).

2.1 Introduction to production scheduling

2.1.1 Definition of production scheduling

The definition of production scheduling, as quoted by Graves (1981), is specifically applicable to this project. He widely defines production scheduling as "the allocation of available production resources over time, to best satisfy some criteria".

He elaborates on the definition by stipulating the differences between production scheduling and two other related concepts: inventory management and production planning. Where production planning focuses partly on the determination of the production resource level, that is determined exogenously to the production scheduling process. Where production scheduling is largely concerned with the allocation decisions of the production resources, the allocation of production resources to jobs is used merely as input into traditional inventory management models.

2.1.2 Generic classification of production scheduling models

Production scheduling has been fascinating researchers since the 1950s (Jain and Meeran, 1999). The numerous articles and research papers resulting from this fascination led to the development

of a large number of production scheduling models. Models in literature are capable of addressing almost every possible production scenario. The most popular method of classifying production scheduling models is the four-field notation (A/B/C/D). Problems are classified according to: the number of jobs (A), the number of machines (B), the flow pattern within the machine shop (C) and the performance measure by which a schedule is evaluated (D).

Although the four-field notation is well-known among scheduling researchers, the recent trend towards consideration of more complex scheduling problems have shown it to be largely inadequate to describe nonbasic models (Xia and Wu, 2005). The inclusion of certain problem specific requirements, which varies from sequence-dependent set-up times to release dates, results in models which can be more effectively classified by means of the three-field notation ($\alpha/\beta/\gamma$) developed by Graham et al. (1979). The three-field notation classifies models according to the flow pattern and number of machines (α), the constraints placed on the jobs (β) and the scheduling criteria (γ).

2.2 Zandieh's classification of scheduling models

Recently Zandieh et al. (2006) has developed a classification for scheduling systems based on the associated resource environments. Figure 2.1 indicates the classification of various, yet similar, types of models, each consisting of one or more operations to be processed on a number of resources. The models are primarily classified according to the following criteria:

- The characteristics of the routings of each of the jobs.
- The number of operations for each of the jobs.
- The number of resources available to perform the required operations.

The models range from more generic formulations, for example the job shop scheduling problem with duplicate machines, to more specific formulations, for example, the single machine shop problem.

In most cases, operations are performed according to a predefined sequence, 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.

The deterministic Job Shop Scheduling Problem (JSSP) is considered to be the best known of the classical scheduling problems. Jain and Meeran (1999) describes the problem as consisting of a finite set $\boldsymbol{J}$ of n jobs $\left\{J_i\right\}_{i=1}^{n}$ to be processed on a finite set $\boldsymbol{M}$ of m machines

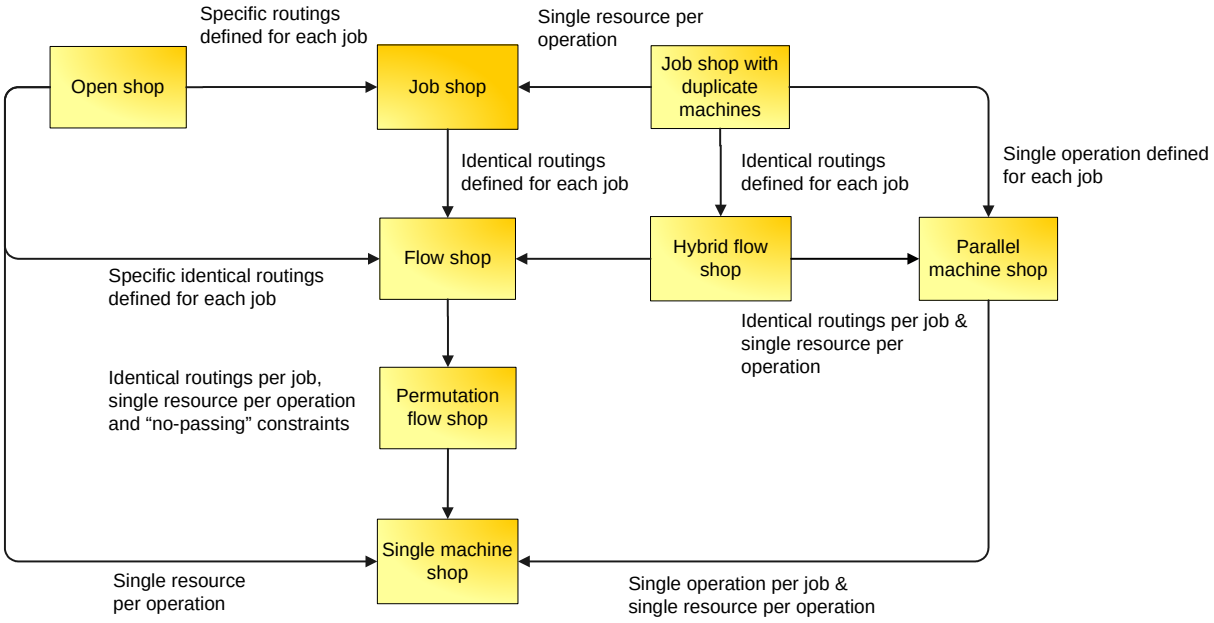

Figure 2.1: Classification of scheduling systems based on resource environments.

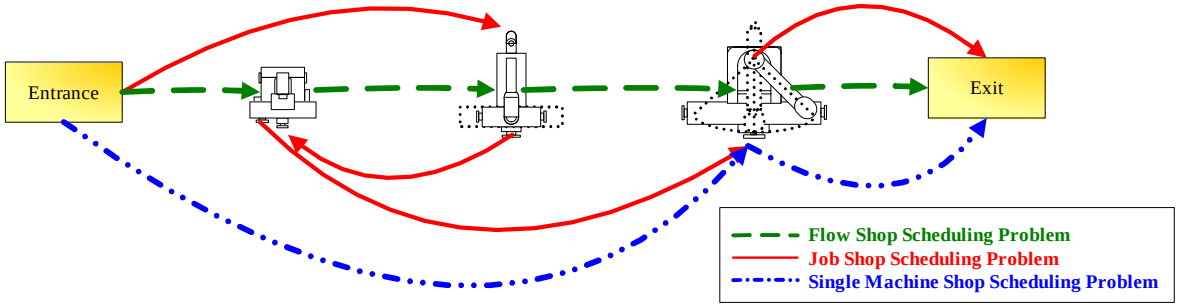

Figure 2.2a: Flow patterns of "single resource per operation" models.

$\{M_k\}_{k=1}^{m}$. Each job J_i must be processed on every machine and consists of a chain of m_i operations $O_{i1}, O_{i2}, ..., O_{im_i}$, which have to be scheduled in a predetermined sequence. There are N operations in total, where $N = \sum_{i=1}^{n} m_i$. O_{ik} is the operation of job J_i which has to be processed on machine M_k for an uninterrupted processing time period τ_{ik}. No operation may be pre-empted. 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.

Figures 2.2a and 2.2b provide visual indications of the flow patterns, relative to the JSSP, supported by each of the classified models, since most of the models in Figure 2.1 are merely specific applications of the job shop scheduling problem. For example, 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.

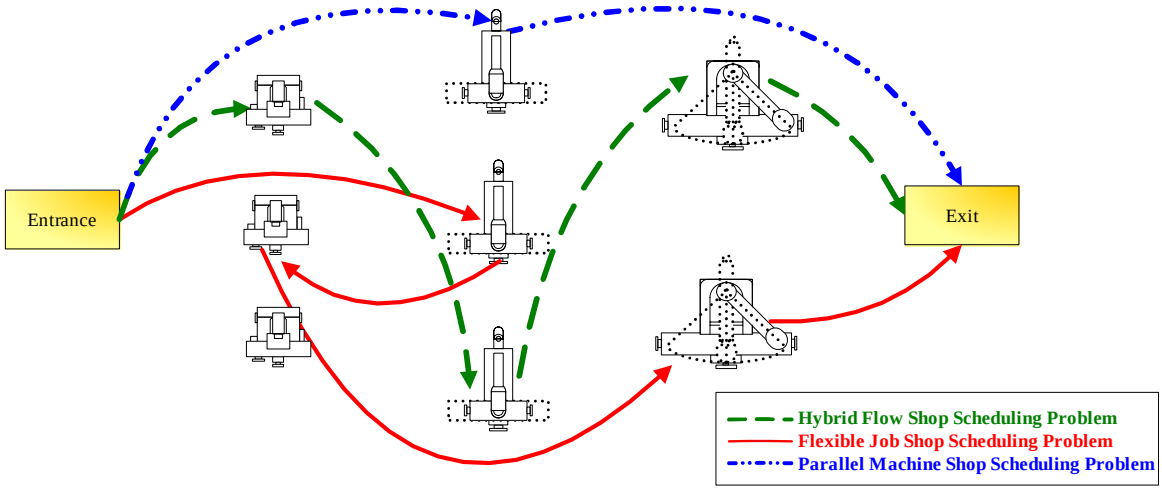

Figure 2.2b: Flow patterns of "multiple resources per operation" models.

The Flow Shop Scheduling Problem (FSSP) 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 results in the same job sequence being followed for each of the resources. Yet another close relation of the FSSP, the hybrid flow shop scheduling problem allows for identical routings along with multiple resources per operation.

The Single Machine Scheduling Problem (SMP) is merely a JSSP with only one operation per job. 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.

An analysis of the business requirements of *Optimatix* resulted in the job shop with duplicate machines problem, the parallel machine scheduling problem and the single machine scheduling problem identified as suitable candidates to use as points of departure for the design of the algorithm. The job shop with duplicate machines becomes an appropriate choice to consider as the parallel machine and single machine scheduling problems can easily be addressed through judicial selection of the input parameters of the job shop with duplicate machines.

However, it is noteworthy that Zandieh et al. (2006)'s classification only addresses problems on a relatively general level. Various detailed problem-specific characteristics are omitted. In order to meet the distinct scheduling requirements of *Optimatix*, a further analysis is performed to identify a problem capable of addressing these specialized scheduling requirements. These requirements include, amongst others, the inclusion of sequence-dependent set-up times and release dates into the problem description.

The next two sections provide a more detailed discussion of a logical sequence of problem variations, which are closely related to the job shop scheduling problem with duplicate machines. For comparison purposes, the most generic model in the class of job shop scheduling problems, the classical JSSP, is first presented. An evaluation of the assumptions of this model leads to the identification of specific problem variations able to address the the proposed problem as it is currently defined.

2.3 The classical job shop scheduling problem

The classical JSSP has received considerable attention in scheduling literature (Zhang et al., 2006). A thorough review is provided by both Blazewicz et al. (1996) and Jain and Meeran (1999). The problem is classified as a combinatorial optimization problem and is one of the most difficult *NP*-complete problems to solve.

2.3.1 Problem formulation

Blazewicz et al. (1996) describes the problem as follows:

$n \triangleq$ The total number of operations.

$m \triangleq$ The total number of machines.

$A \triangleq$ The set of ordered pairs of operations constrained by the precedence relations for each job.

$E_k \triangleq$ The set of all pairs of operations to be performed on machine k, where $k = \{1, 2, \ldots, m\}$.

$p_i \triangleq$ The processing time of operation i, where $i = \{1, 2, \ldots, n\}$.

$t_i \triangleq$ The earliest possible starting time of operation i, where $i = \{1, 2, \ldots, n\}$.

$$\min z = t_n \tag{2.1}$$

subject to

$$t_j - t_i \geq p_i \qquad \forall (i,j) \in A \tag{2.2}$$

$$t_j - t_i \geq p_i \qquad \forall (i,j) \in E_k, \forall k \in \{1, \ldots, m\} \qquad \text{or}$$

$$t_i - t_j \geq p_j \qquad \forall (i,j) \in E_k, \forall k \in \{1, \ldots, m\} \tag{2.3}$$

$$t_i \geq 0 \qquad \forall i \in \{1, \ldots, n\} \tag{2.4}$$

The objective function (2.1) minimizes the maximum makespan. Constraint (2.2) ensures that the precedence constraints between operations within the same job are respected. Constraints (2.3) ensures that only one job may be scheduled on a machine at a time, and (2.4) ensures completion of all jobs.

2.3.2 Problem representation

Traditionally there are two main problem representations traditionally associated with the classical JSSP: the Gantt chart (more frequently used to display the schedule than solve the schedule) and the disjunctive graph model (Jain and Meeran, 1999). The latter is explained with the aid of Figure 2.3.

The disjunctive graph formulation 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 denoted by **C** of directed arcs. Capacity constraints ensure that two jobs which require the same machine can not 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 (Brucker, 2004). Shortest path algorithms are traditionally used to find the optimal solution when solving a makespan minimization problem. An example of a disjunctive graph formulation for a 3 machine 4 job problem is illustrated in Figure 2.3.

Although the disjunctive graph representation is robust and useful for solving real life job shop problems, there are a number of difficulties associated with this representation. Concurrent or parallel processing and indefinite cyclical process flows can not be modelled directly (White and Rogers, 1990). Feasibility of solutions can also be a problem since an acyclic graph is required for schedule feasibility.

2.3.3 Assumptions

There are a number of key assumptions inherent to the formulation of the classical JSSP. Some of these include:

- Only one operation may be scheduled on any one resource at a time.
- All jobs require the same set of resources.

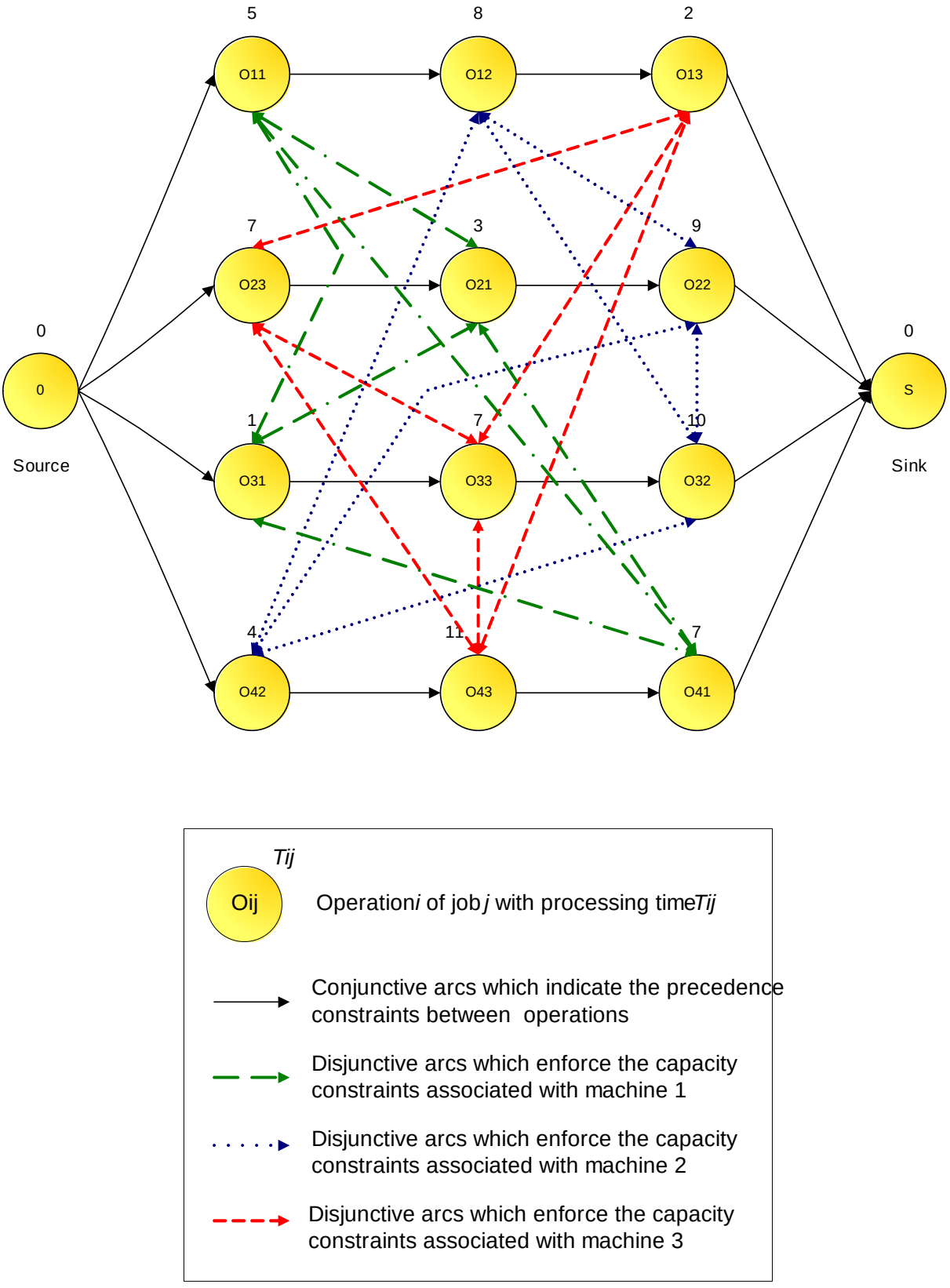

Figure 2.3: Disjunctive graph formulation (Jain and Meeran, 1999).

- Two or more operations from the same job may not be processed on the same resource.
- Each resource type consists of only one resource.
- All jobs are available for scheduling at time zero.

These assumptions result in the problem, as it is formulated in this document, not being suitable to address the requirements of *Optimatix*. However, there are a large number of variations to the classical JSSP and various other algorithmic approaches, which are extensively utilized in literature to meet more specific scheduling requirements. Figure 2.4 provides a visual indication of the various approaches to scheduling encompassed within job shop scheduling. They are classified according to Graham et al.'s 1979 three-field notation.

Table 2.1 lists the various assumptions and the appropriate approaches which should be investigated to overcome the limitations of the classical JSSP with respect to the specific business requirements.

2.4 Variations on the classical job shop scheduling problem

This section discusses the various different approaches to complex job shop scheduling which were identified in the previous section. It is noteworthy that all the variations on the classical job shop scheduling problem are in themselves classified as *NP*-hard problems, due to their direct derivation from the classical problem instance.

2.4.1 The JSSP with precedence constraints

Additional to the precedence constraints occurring between the various operations of each of the jobs, it is also possible to have precedence relations between jobs (Figure 2.5) — a very useful attribute in modeling assembly processes. Brucker (2004) describes the precedence relations by means of an acyclic directed graph $\mathbf{G} = (\mathbf{V}, \mathbf{B})$ where $(i, k) \in \mathbf{V}$ corresponds with the jobs, and $(i, k) \in \mathbf{B}$ if and only if J_i must be completed before J_k starts. The problem can be modeled analytically by adding the following constraint to the classical job shop scheduling problem:

$$t_j - t_i \geq p_i \qquad \forall (i, j) \in B \tag{2.5}$$

where $\mathbf{B}$ refers to the set of ordered pairs of operations constrained by the precedence relations between each job.

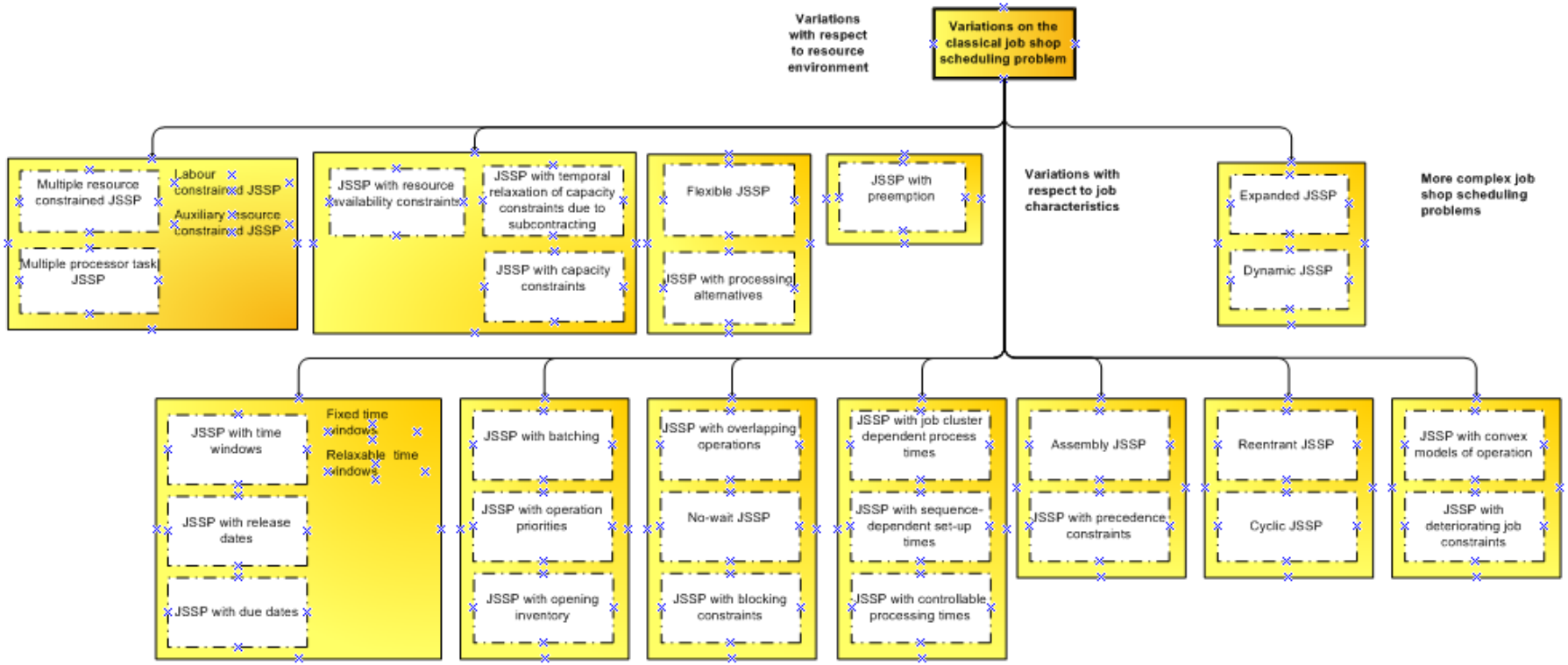

Figure 2.4: Variations on the classical JSSP.

Table 2.1: *Optimatix* business requirements and possible approaches.

Business requirement	Approaches in Literature
Scheduling of machines, manpower and other complementary resources	Multiple resource constrained JSSP
	JSSP with multiprocessor task systems
	Expanded Job Shop Scheduling Problem (EJSSP)
Alternative routing and resource allocation	Flexible Job Shop Scheduling Problem (FJSSP)
	EJSSP
The incorporation of release dates	EJSSP
	JSSP with release dates
The incorporation of production calendars	EJSSP
Scheduling of more than one operations from the same job	Reentrant Job Shop Scheduling Problem (RJSSP)
on the same resource	FJSSP
	EJSSP
The incorporation of sequence-dependent set-up times	JSSP with sequence-dependent set-up times
The incorporation of machine breakdowns	JSSP with machine availability constraints
	Dynamic JSSP
	EJSSP
The incorporation of precedences between jobs	JSSP with precedence constraints

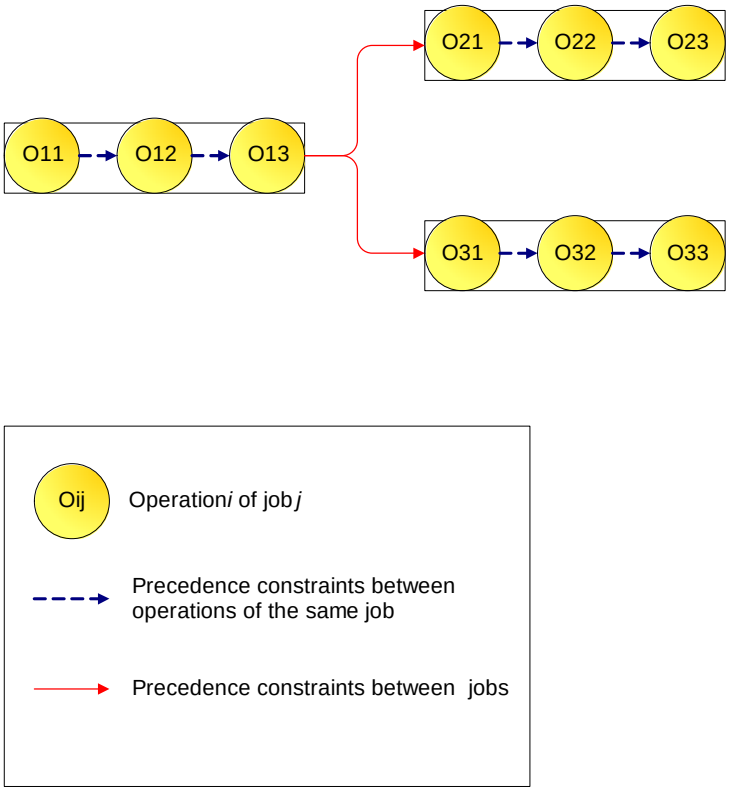

Figure 2.5: Precedence relationships between jobs.

2.4.2 The JSSP with opening inventory

Inventory is a common occurrence in almost every production system. Companies often generate schedules in intermittent intervals during the production run. Due to the continuous nature of most production systems, it is possible to have jobs which are half complete at schedule generation.

Taking into consideration the effect of beginning inventory during schedule generation is relatively trivial, but has a definite impact on the realism of the schedule. The impact of opening inventory is only apparent during the calculation of the processing time. However, beginning inventory of a certain product affects the processing time of all operations of all jobs associated with the specific product. This implies the requirement of additional data: the unit processing times associated with each operation.

The problem can be formulated by adding Constraint (2.6) to the classical JSSP where d_i denotes the demand in number of units which requires processing as part of operation i, g_i denotes the number of units in inventory which have completed processing up to a point past operation i, and u_i denotes the unit processing time associated with operation i:

$$p_i = (d_i - g_i)u_i \qquad \forall i \in \{1, 2, \ldots, n\} \tag{2.6}$$

Ivens and Lambrecht (1996) incorporated initial work-in-process into a disjunctive graph job shop scheduling formulation. The opening inventory was incorporated by only generating the remaining part of the network. In other words, the node weights are calculated similarly to p_i in constraint (2.5). The problem was solved successfully by means of the Shifting Bottleneck Heuristic.

2.4.3 The JSSP with sequence-dependent set-up times

Set-up times are defined in literature as the time intervals between the completion of one operation and the start of the next operation. The inclusion of set-up times is one of the most frequent additional complications in scheduling and incorporation into traditional scheduling models has already been attempted as early as the 1970s (Lockett and Muhlemann, 1972). The JSSP with sequence-dependent set-up times is useful in situations where cleaning operations and tool changes play an important role in production. A typical example is the manufacturing of different colours of paint.

Over the years, two distinct approaches have been developed. The first differentiates between sequence-dependent and sequence-independent set-up times. The second approach only includes set-up times when two jobs, which belong to two different predefined subsets of operations, are processed sequentially (Noivo and Ramalhinho-Loureno, 2006).

In the most complicated case, sequence-dependent setup times, where the setup 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 m. The set-up time of operation j is defined by s_{ijk} where i is the index of the previous operation scheduled on machine k. If the set-up times are machine independent, s_{ijk} can be simplified to s_{ij}. Sequence-independent set-up times are denoted by s_i and constant set-up times only by s (Brucker, 2004).

In terms of the solution strategies followed, Noivo and Ramalhinho-Loureno (2006) addressed the JSSP with sequence-dependent set-up times by means of a number of priority dispatch rules. Lockett and Muhlemann (1972) and Asano and Ohta (1991) developed branch-and-bound algorithms for single machine scheduling problems.

2.4.4 The JSSP with machine availability constraints

In real-life production systems, the advent of production calendars, holidays, preventative maintenance and unexpected breakdowns have a significant effect on machine availability and subsequently the scheduling of production resources. If no preemption is allowed, the inclusion of machine availability constraints results in dividing the planning horizon into a number of disconnected time windows (Asano and Ohta, 1991).

White and Rogers (1990) addressed the problem by means of the disjunctive graph formulation. Scheduled maintenance is regarded similarly to any other operation and is assigned a node with a weight proportional to the time required for maintenance.

2.4.5 The flexible job shop scheduling problem

Xia and Wu (2005) describes the recent trend which exists in the research domain to solve a much more complex version of the classical job shop scheduling problem. Being an extension of the classical JSSP, the FJSSP incorporates all of the complexities and challenges associated with classical job shop scheduling. The FJSSP is the most common variation on the job shop scheduling problem and it is also referred to in literature as the job shop scheduling problem with duplicate machines, the hybrid job shop scheduling problem and the job shop with multi-purpose machines.

The aim of the FJSSP is twofold: the assignment of all operations to machines and the subsequent sequencing of these operations on each of the assigned machines. For solving any realistically sized problem, either a hierarchical or integrated approach can be followed. The former results in decomposition of the problem into two simpler problems which are solved independently of one another, whereas the latter does not differentiate between allocation and sequencing (Xia and Wu, 2005). The most notable solution strategies include Evolutionary Algorithms (Kacem et al., 2002) and Tabu Searches (Dauzre-Prs and Paulli, 1997).

The problem is defined formally by Kacem et al. (2002). It consists of a finite set $\mathbf{J}$ of N jobs $\left\{J_i\right\}_{i=1}^{N}$ to be processed on a finite set $\mathbf{U}$ of M machines $\left\{M_k\right\}_{k=1}^{M}$. Each job represents a number (m_i) of nonpreemptable ordered operations $O_{i1}, O_{i2}, \ldots, O_{im_i}$. The execution of each operation of a job requires one resource or machine selected from a set of available machines called $U_{ij} \subseteq \mathbf{U}$. The assignment of the operation O_{ij} to the machine $M_k (M_k \in U_{ij})$ entails the occupation of this machine during a processing time called d_{ijk}.

The uniqueness of the FJSSP lies in that each operation does not have to be performed on a machine specifically dedicated to the specific operation. 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 FJSSP is useful for scheduling in a wider range of production systems, including flexible manufacturing systems and parallel machine shops (Dauzre-Prs and Paulli, 1997).

In the case of total flexibility $U_{ij} = \mathbf{U}$. In other words, all operations can be executed on all machines. If a FJSSP is classified as a partial flexibility problem, there exist at least one operation $O_{i_o j_o}$ such that $U_{i_o j_o} \subset \mathbf{U}$. This implies that operations can only be performed on machines with which they are compatible.

Brucker (2004) further differentiates between FJSSPs with identical machines, uniform machines and unrelated machines. If U_{ij} consists of identical machines the processing time (d_{ij}) of operation O_{ij} is the same regardless of which machine is used for processing. If U_{ij} consists of uniform machines, the processing time (d_{ijk}) of operation O_{ij} can be calculated by the formula:

$$d_{ijk} = \frac{d_{ij}}{s_k} \tag{2.7}$$

where s_k is the processing speed of machine k. Finally, if U_{ij} consists of unrelated machines, the processing time (d_{ijk}) varies from machine to machine and the processing time is dependent on both the characteristics associated with the operation and the characteristics of the machine on which it is performed.

2.4.6 Multiple resource constrained job shop scheduling

In real life production systems, the scheduling of operations are not only constrained by one type of resource. 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 indicates that typically 16% of scheduled production cannot be met because tooling is 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 (1996) describes 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 solved in literature are the labour constrained job shop (for scheduling workers and machines) and the auxiliary resource constrained job shop (for scheduling tools and machines).

Brucker (2004) defines a class of problem 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 be linked to each operation. Each operation O_{ij} requires during a processing period p_{ij} all resources belonging to the set $\mu_{ij} \subseteq \{M_1, \ldots, M_m\}$. 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. In his book, Brucker (2004) converted a JSSP with multiprocessor tasks, unit processing times, release dates and precedence constraints between jobs into a shortest path problem. Patel et al. (1999) followed a genetic algorithm approach for solving a dual resource constrained scheduling problem, labour being the secondary resource.

2.4.7 The reentrant job shop scheduling problem

The classical JSSP assumption that each job may visit a specific machine only once, is often violated in practice. This resulted in the development of the RJSSP, where two different operations belonging to the same job can be scheduled on one machine. The RJSSP is especially useful to model production scheduling scenarios in high-tech industries. For example, in semiconductor manufacturing, wafers usually make multiple visits to the same machine as successive circuit layers are added.

Pan and Chen (2005) formulated the RJSSP as an integer programming model.

$n \triangleq$ The total number of jobs available for processing at time 0.

$m \triangleq$ The total number of machines.

$N_i \triangleq$ The number of operations of job i, where $i = \{1, 2, \ldots, n\}$.

$p_{ij} \triangleq$ The processing time of operation j of job i, where $i = \{1, 2, \ldots, n\}$, $j = \{1, 2, \ldots, N_i\}$.

$$r_{ijk} \triangleq \begin{cases} 1 & \text{if operation } j \text{ of job } i \text{ requires machine } k, \text{ where } i = \{1, 2, \ldots, n\}, \\ & j = \{1, 2, \ldots, N_i\}, k = \{1, 2, \ldots, m\} \ . \\ 0 & \text{otherwise.} \end{cases}$$

$$Z_{iji'j'} \triangleq \begin{cases} 1 & \text{if operation } j \text{ of job } i \text{ precedes operation } j' \text{ of job } i', \text{ where } i, i' = \{1, 2, \ldots, n\}, \\ & j, j' = \{1, 2, \ldots, N_i\}. \\ 0 & \text{otherwise.} \end{cases}$$

$H \triangleq$ A very large positive number.

$C_i \triangleq$ Completion time of job i, where $i = \{1, 2, \ldots, n\}$.

$C_{max} \triangleq$ Maximum completion time or makespan.

$t_{ij} \triangleq$ The starting time of operation j of job i, where $i = \{1, 2, \ldots, n\}$, $j = \{1, 2, \ldots, N_i\}$.

$$\min C_{max} = \max\{C_i\}_{i=1}^{n} \tag{2.8}$$

subject to

$$\sum_{k=1}^{m} r_{ijk}(t_{ij} + p_{ij}) \leq \sum_{k=1}^{m} r_{i(j+1)k} t_{i(j+1)}$$
$$\forall i \in \{1, 2, \ldots, n\}, \forall j \in \{1, 2, \ldots, N_i - 1\} \tag{2.9}$$

$$p_{ij} \leq H(2 - r_{ijk} - r_{i'j'k}) + H(1 - Z_{iji'j'}) + (t_{i'j'} - t_{ij})$$
$$\forall i \geq 1, \forall i < i' \leq n, \forall j \in \{1, 2, \ldots, N_i\},$$
$$\forall j' \in \{1, 2, \ldots, N_{i'}\}, \forall k \in \{1, 2, \ldots, m\} \tag{2.10}$$

$$p_{i'j'} \leq H(2 - r_{ijk} - r_{i'j'k}) + HZ_{iji'j'}) + (t_{ij} - t_{i'j'})$$
$$\forall i \geq 1 \forall i < i' \leq n, \forall j \in \{1, 2, \ldots, N_i\},$$
$$\forall j' \in \{1, 2, \ldots, N_{i'}\}, \forall k \in \{1, 2, \ldots, m\} \tag{2.11}$$

$$C_{max} \geq \sum_{k=1}^{m} r_{iN_ik}(t_{iN_i} + p_{iN_i})$$
$$\forall i \in \{1, 2, \ldots, n\} \tag{2.12}$$

$$C_{max} \geq 0, t_{ij} \geq 0$$
$$\forall i \in \{1, 2, \ldots, n\}, \forall j \in \{1, 2, \ldots, N_i - 1\} \tag{2.13}$$

$$Z_{iji'j'} = 0 \text{ or } 1$$
$$\forall i \geq 1 \forall i < i' \leq n, \forall j \in \{1, 2, \ldots, N_i\},$$
$$\forall j' \in \{1, 2, \ldots, N_{i'}\} \tag{2.14}$$

Constraint (2.9) ensures that the starting time of operation $j+1$ of job i on machine k is no earlier than the finish time of operation j of job i. Constraints (2.10) and (2.11) ensure that only one job may be processed on a machine at any one time. If both operation j of job i and operation j' of job i' are processed on machine k, i.e., $r_{ijk} = r_{i'j'k} = 1$, then either $s_{i'j'} - s_{ij} \geq p_{ij}$ or $s_{ij} - s_{i'j'} \geq p_{i'j'}$ holds (H is a very large positive number). The either-or construct is incorporated into the model by means of constraints (2.10) and (2.11). Constraint (2.12) is used to calculate C_{max} which is minimized in the objective function (2.8). Constraint (2.14) ensures that C_{max}, s_{ij}, and the binary integer variable $Z_{iji'j'}$, are greater than zero.

2.4.8 The expanded job shop scheduling problem

The EJSSP is more general than the classical JSSP. The EJSSP incorporates both release dates and due dates. Job starting times are also restricted by what Yu and Liang (2001) define as technological planning constraints or operation enabling conditions. Simply put, all cutting tools, machines and other resources required for processing an operation has to be available before the processing of the operation can start. This has positive implications for the scheduling of labour and auxiliary resources. It also implies that alternative resources can be specified for each operation. The EJSSP has already been solved successfully by means of a hybrid approach involving neural networks and genetic algorithms (Yu and Liang, 2001).

The model can be formulated as follows:

$n \triangleq$ The total number of jobs.

$m \triangleq$ Type number of various resources.

$N_i \triangleq$ The number of operations of J_i, where $i = \{1, 2, \ldots, n\}$.

$r_s \triangleq$ Number of resources of type s, where $s = \{1, 2, \ldots, m\}$.

$R_i \triangleq$ Set of pairs of operations $\{k, l\}$ belonging to job i, where operation k precedes operation l, where $i = \{1, 2, \ldots, n\}$.

$Q_i \triangleq$ Set of pairs of operations $\{k, l\}$ belonging to job i, for any operation k and operation l, where $i = \{1, 2, \ldots, n\}$.

$N_q \triangleq$ Set of operations requiring resource k, where $k = \{1, 2, \ldots, r\}$.

$H \triangleq$ A very large positive number.

$p_{il} \triangleq$ The processing time of operation l of job i, where $i = \{1, 2, \ldots, n\}$, $l = \{1, 2, \ldots, N_i\}$.

$t_{il} \triangleq$ The starting time of operation k of job i, where $i = \{1, 2, \ldots, n\}$, $k = \{1, 2, \ldots, N_i\}$.

$t_{\wr i} \triangleq$ The starting time of the first (or free) operation of job i, where $i = \{1, 2, \ldots, n\}$.

$t_{i\wr} \triangleq$ The completion time of the last (or free) operation of job i, where $i = \{1, 2, \ldots, n\}$.

$a_i \triangleq$ The availability time of job i, where $i = \{1, 2, \ldots, n\}$.

$d_i \triangleq$ The delivery due date of job i, where $i = \{1, 2, \ldots, n\}$.

$[i, k] \triangleq$ The k^{th} operation of job i, where $i = \{1, 2, \ldots, n\}$.

$$z_{ij} \triangleq \begin{cases} 1 & \text{if operation } i \text{ precedes operation } j, \text{ where } \{i,j\} \in N_q, q = \{1,2,\ldots,r_s\}, \\ & s = \{1,2,\ldots,m\}. \\ 0 & \text{otherwise.} \end{cases}$$

$$y_{kl} \triangleq \begin{cases} 1 & \text{if operation } k \text{ precedes operation } l, \text{ where } \{k,l\} \in Q_i, i = \{1,2,\ldots,n\}. \\ 0 & \text{otherwise.} \end{cases}$$

$t_{ij} \triangleq$ The starting time of operation O_{ij}, where $i = \{1,2,\ldots,n\}$, $j = \{1,2,\ldots,N_i\}$.

$$\min C_{max} = \max_i (t_{i\wr} + p_{i\wr}) \tag{2.15}$$

subject to

$$t_{il} - t_{ik} - p_{ik} \geq 0 \qquad \forall i \in \{1,2,\ldots,n\}, \text{ if } \{k,l\} \in R_i \tag{2.16}$$

$$t_{jl} - t_{ik} - p_{ik} + H(1 - z_{kl}) \geq 0 \qquad \forall q \in \{1,2,\ldots,r\}, \text{ if } \{k,l\} \in N_q \tag{2.17}$$

$$t_{ik} - t_{jl} - p_{jl} + H z_{kl} \geq 0 \qquad \forall q \in \{1,2,\ldots,r\}, \text{ if } \{k,l\} \in N_q \tag{2.18}$$

$$t_{il} - t_{ik} - p_{ik} + H(1 - y_{kl}) \geq 0 \qquad \forall i \in \{1,2,\ldots,n\}, \text{ if } \{k,l\} \in Q_i \tag{2.19}$$

$$t_{ik} - t_{il} - p_{il} + H y_{kl} \geq 0 \qquad \forall i \in \{1,2,\ldots,n\}, \text{ if } \{k,l\} \in Q_i \tag{2.20}$$

$$t_{\wr i} \geq 0 \qquad \forall i \in \{1,2,\ldots,n\} \tag{2.21}$$

$$t_{i\wr} \leq d_i - p_{i\wr} \qquad \forall i \in \{1,2,\ldots,n\} \tag{2.22}$$

$$t_{ik}, t_{\wr i}, t_{i\wr} \geq 0 \qquad \forall i \in \{1,2,\ldots,n\}, \tag{2.23}$$

$$z_{kl}, y_{kl} = 0 \text{ or } 1 \tag{2.24}$$

The objective function expressed in (2.15) minimizes the completion time of the last completed job. Constraint (2.16) ensures that the precedence constraints between operations belonging to the same job are not violated. Constraints (2.17) and (2.18) ensures that a resource only processes one operation at a time, while constraints (2.19) and (2.20) are referred to as hidden job constraints. These hidden job constraints ensure that two operations belonging to the same job are not processed simultaneously. The starting and completion time constraints ((2.21) and (2.22)) restricts the processing time of the job to start and end in the interval between the job available time (release date) and the due date. Constraint (2.23) ensures that the decision variables remain positive and constraint (2.24) defines the binary variables z_{kl} and y_{kl}.

2.4.9 Selection of suitable variations

There are a number of factors which were considered during the evaluation of the variations. Initially the algorithm will be kept as simple as possible, therefore any unnecessary complexity will be eliminated. However, it is still important to ensure that the metaheuristic algorithm will have the same functionality as the algorithm currently implemented by *Optimatix*. Similar functionality will ensure that the comparison between the two algorithms will be meaningful. The variations were also selected on the basis of efficiency: those variations incorporating the most functional characteristics receive preference.

The selection process resulted in three variations deemed suitable to use as points of departure for the design of the algorithm. Therefore the algorithm should be designed as such that its structure will enable it to have the same functionality as the following variations:

- The expanded job shop scheduling problem.
- The JSSP with sequence-dependent set-up times.
- The JSSP with precedence constraints.

2.5 Objective function variations on the classical job shop scheduling problem

Apart from cost functions, a large number of objective functions have been formulated to evaluate the quality of schedules. Brucker (2004) provides a list of the most common measurements which can be used to formulate objective functions. Any of the measures in Table 2.2 can be used to formulate at least four different objective functions of the form: $\max\{m_i\} \mid i = \{1 \ldots, n\}$, $\sum_{i=1}^{n} m_i$, $\sum_{i=1}^{n} w_i m_i$ and $\max\{w_i m_i\} \mid i = 1 \ldots, n\}$, where m_i denotes the measurement associated with job i, w_i the numerical weight associated with job i and d_i the due date of job i. For example, the following four objective functions can be formulated for job completion time, where C_i denotes the completion time of job i: makespan ($\max\{C_i\} \mid i = 1 \ldots, n\}$), total flow time ($\sum_{i=1}^{n} C_i$) and weighted total flow time ($\sum_{i=1}^{n} w_i C_i$).

2.6 Solution strategies

The design of an efficient algorithm requires the identification of the most appropriate solution strategy for the given implementation environment. Being the most general of the classical scheduling problems, the classical job shop scheduling problem has a rich history. An indication

Table 2.2: Commonly used JSSP measurements.

JSSP measurement	Formulation
Lateness	$L_i = C_i - d_i$
Earliness	$E_i = \max\{0, d_i - C_i\}$
Tardiness	$T_i = \max\{0, C_i - d_i\}$
Absolute deviation	$D_i = \mid C_i - d_i \mid$
Squared deviation	$D_i = (C_i - d_i)^2$

of all the major solution strategies which have been applied to the JSSP in the past fifty years, along with the benchmark problems on which they were tested, is indicated in Figure 2.6 (Jain and Meeran, 1999).

2.6.1 Optimal solution strategies

During the 1960s a lot of emphasis was placed on finding exact solutions by means of elaborate and sophisticated mathematical constructs. However, in their article, Jain and Meeran (1999) refer to research from the 1970s which clearly highlights the extreme intractability of the job shop scheduling problem. The problem can be classified as strongly *NP*-hard. Therefore, only a small number of special instances of the problem are solvable within polynomial time. This implies that these techniques are of limited practical use, since the majority of these techniques are unable to achieve feasible solutions to many problems.

The most widely used enumerative strategy is the Branch-and-Bound algorithm entailing implicitly searching a tree structure which represents the solution space. A number of procedures have been developed to exclude large portions of the tree to speed up the searching 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 (Jain and Meeran, 1999).

2.6.2 Heuristic methods

The use of approximation methods has more and more become a viable alternative. Even though the optimality of the solutions can not be guaranteed, larger problems can be solved more efficiently. 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 other priority based rules are often applied to the JSSP.

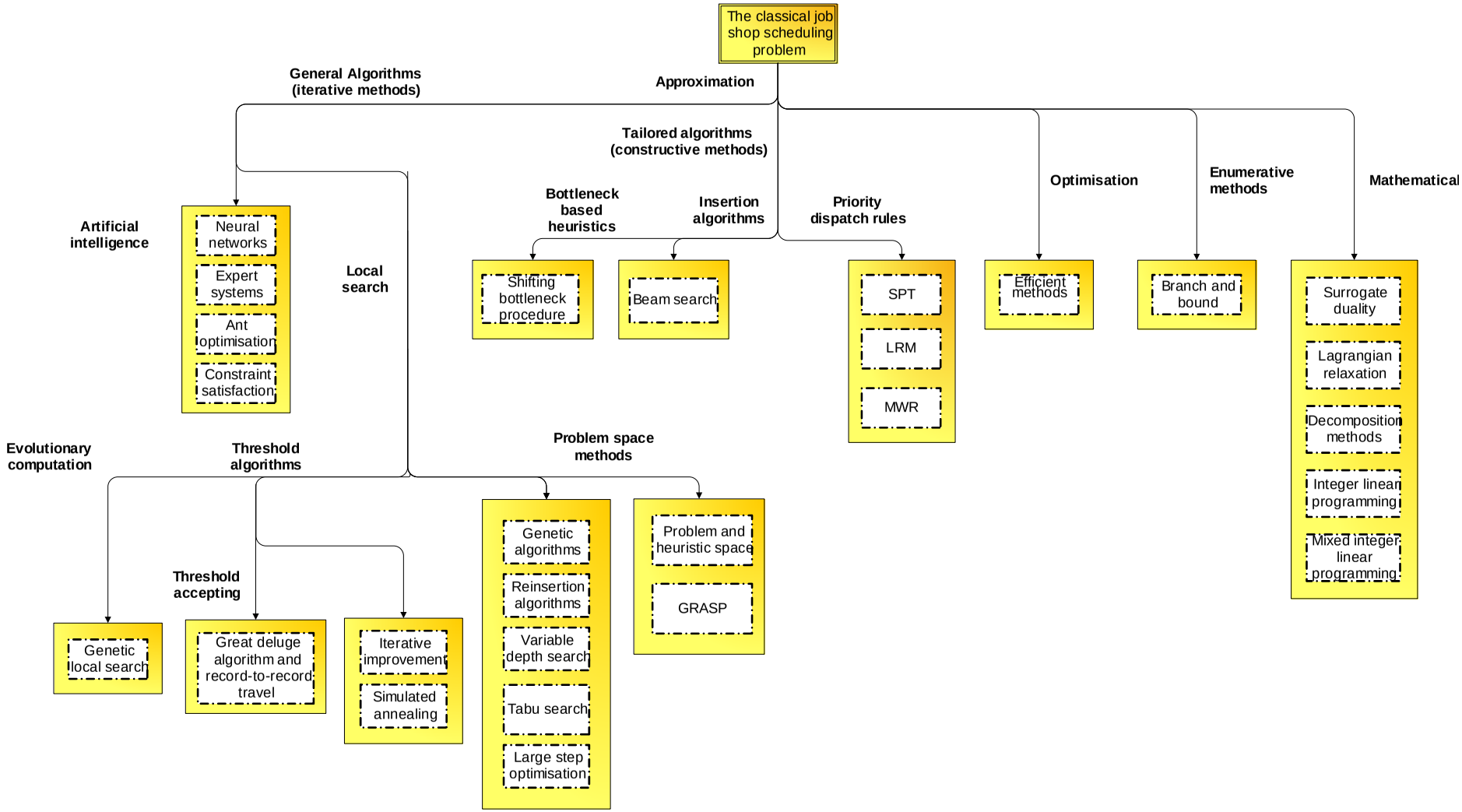

Figure 2.6: Solution strategies of the classical JSSP (Jain and Meeran, 1999).

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 steepest gradient descent, conjugate gradients or Newton methods (Engelbrecht, 2005).

Priority Dispatch Rules

One of the earliest heuristic methods developed for scheduling applications, Priority Dispatch Rules (PDRs), functions by assigning 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 (Jain and Meeran, 1999).

The Shifting Bottleneck Heuristic

The Shifting Bottleneck Heuristic (SBP) is commonly considered to have had the greatest influence on approximation methods for production scheduling applications. This is 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 values. The schedule for the more complex problem is generated by sequentially scheduling each machine based on its rank. The SBP does, however, have a problem with the generation of infeasible solutions (Jain and Meeran, 1999).

2.6.3 Metaheuristics

The inability of heuristic methods to escape local optima have resulted in the development of metaheuristics. The allowance of nonimproving feasible moves allows the search to continue in regions where progress can resume (Rardin, 1998). The motivation for using a metaheuristic to solve the job-shop scheduling algorithm is imbedded in the inherent complexity of the problem. Simulated Annealing (SA), Tabu Search (TS) and Genetic Algorithms are most commonly applied to the JSSP (Gongalves et al., 2005).

Tabu Search

Rardin (1998) defines a TS as an algorithm which deals with cycling by temporarily forbidding moves that would return to a solution recently visited. 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 on the list) and frequency-based memory (where the number of times a solution occurs has an influence). Additionally, an incumbent solution (Zhang et al., 2006) 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.

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 change. 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 (Engelbrecht, 2005).

$$P_{ij} \triangleq \begin{cases} 1 & \text{if } f(x_j) < f(x_i) \\ \exp^{\frac{f(x_j)-f(x_i)}{c_b T}} & \text{otherwise} \end{cases} \tag{2.25}$$

where P_{ij} is the probability of moving from point x_i to x_j, $c_b > 0$ and T is the temperature of the system. Jain and Meeran (1999) classifies SA as a generic technique unable to achieve good solutions quickly and requires excessive computational effort. However, the hybridization of SA with other solution strategies, including Genetic Algorithms, has greatly improved its competitiveness.

Genetic Algorithms

Genetic Algorithms (GA)s attempt to parallel the process of biological evolution to find better and better solutions (Rardin, 1998). A number of operators (for example selection, crossover, mutation and cloning) act upon on a population of randomly initialized individuals to trans-

form these individuals into better solutions. This technique has been the most unsuccessful at addressing the classical job shop scheduling problem. Jain and Meeran (1999) states that GAs are unable to successfully represent the classical job shop scheduling problem due to the fact that crossover operators cannot generate feasible schedules without losing efficiency.

2.7 Conclusion

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*. Much success has been achieved in the field of job shop scheduling, yet room for improvement still exists. Numerous research papers propose solution strategies ranging from complex metaheuristic implementations to simple rules-based approaches. The next chapter discusses in greater detail the development of a suitable solution strategy for the *Optimatix* problem. Every solution strategy has its merits — the challenge is to find the best one for the purpose at hand.

Chapter 3

A generic production scheduling framework

Jain and Meeran (1999) states that metaheuristics is the most successful class of solution strategies for solving the job shop scheduling problem. In order to use a metaheuristic-based approach to solving the *Optimatix* problem, the proposed solution strategy should fulfill a number of problem-dependent requirements, the first of which is an effective problem-mapping mechanism.

The problem-mapping mechanism has to ensure that effective communication occurs between the various domains the *Optimatix* problem consists of. The problem can be divided into three domains: the domain of the metaheuristic, consisting of for example, chromosomes or particles, the decision space, consisting of all possible decision variable values and the objective space which consists of all possible objective function values. This state of events is largely due to the fact that not all the information required to derive an objective function value associated with a specific scheduling solution can be stored inside the schedule representation.

A useful byproduct of effective communication between the various domains is that the inherent structure of such a framework results in it being generic to all common metaheuristics. *Optimatix* has only recently started to consider metaheuristics as a potential solution strategy and a large number of alternative metaheuristics exist that can in some way or other meet the unique scheduling requirements of *Optimatix*. The framework will enable *Optimatix* to employ population-based techniques when the computational cost is justified or to select a technique which is relatively insensitive to the quality of the initial solution, when it is inevitable to use a poor starting solution, this structure can add immense value to the production scheduling efforts of *Optimatix*. After all, metaheuristics are largely problem-dependent, therefore the more alternatives available, the higher the quality of the final solution and the faster the most suitable metaheuristic can be found.

3.1 Structural requirements of the framework

In order to create such a generic production scheduling framework for *Optimatix* there are a number of important structural requirements which should be addressed. The first step in using a metaheuristic is to define the schedule representation. Critical to the success of the algorithm, this representation can be used to implicitly enforce a number of problem constraints. Because the framework should be a fully functional solution strategy, special emphasis should be placed on addressing the various problem-specific constraints, multiple objectives and it should be ensured that effective functionality is obtained in the dynamic scheduling environment of *Optimatix*. For the sake of simplicity PSO terminology will be used throughout the rest of the chapter.

3.1.1 The schedule representation

The schedule representation can be loosely defined as the structure in which the metaheuristic stores each scheduling solution and thus it is extremely important for this representation to address all aspects of the problem to be solved. For example, the *Optimatix* problem requires the representation to address both the allocation of operations to primary resources as well as the subsequent sequencing of each of the operations on their allocated resources. In this framework, the allocation of operations are denoted by the decision variables x_{ij} which takes on a value of 1 if operation i is performed on resource j and is 0 otherwise. The sequencing decision is denoted by t_i (the starting time of operation i).

Selecting a suitable representation has an enormous impact on the algorithm's efficiency and is partially responsible for incorporating the required constraints into the algorithm's structure. The more constraints which are implicitly incorporated into the particle representation, the less constraints remain which have to be enforced by other methods. Probably one of the most important decisions to be made during algorithm development, a brief review of scheduling representations is provided in Section 3.1.1 before a decision is made regarding the most suitable option for the *Optimatix* problem.

Review of existing schedule representations

The most common particle representation for the allocation of operations uses a vector of integer values corresponding to the unique resource index of the machine on which each operation is to be processed (Tamaki et al., 2001). This approach can, however, result in infeasible resource allocations in partially Flexible Job Shop Scheduling Problem (FJSSP)s, where a specific operation can not simply be processed on any available resource. Traditional scheduling constraints, for example the constraint that ensures no two operations are processed simultaneously on the

same resource, may also be violated. Attempting to incorporate more constraints implicitly into the particle representation, Tay and Wibowo (2004) discusses a representation which only assigns operations to those resources which are available at the time of the assignment.

Bit strings indicating the satisfaction of proposed precedences between operations and sorted vectors consisting of the correct sequence of operation indices have been used to sequence operations. Other representations make provision for the starting times of all operations or assigns priority values (Zhang et al., 2004). However, the assignment of priorities to all operations should be accompanied by a relatively complex scheduling heuristic which translates the particle representation into a feasible schedule.

Most scheduling algorithms, which address both allocation and sequencing, use a two-vector particle representation, but an interesting three-vector representation can be found in Tay and Wibowo (2004) and there are a number of authors who have used matrix representations (Hsu et al., 2002) (Mesghouni et al., 1997).

Schedule representation selection

The final selection opts for a two-vector representation consisting of primary resource indices and starting times of operations (Figure 3.1). Consisting of one continuous-valued and one discreet-valued vector, both continuous and discreet optimization algorithms can be used, provided that a discretization-mechanism is incorporated for continuous algorithms. This mechanism simply ensures that the continuous-valued output of the metaheuristic can be converted to its associated discreet value for interpretation as a production schedule.

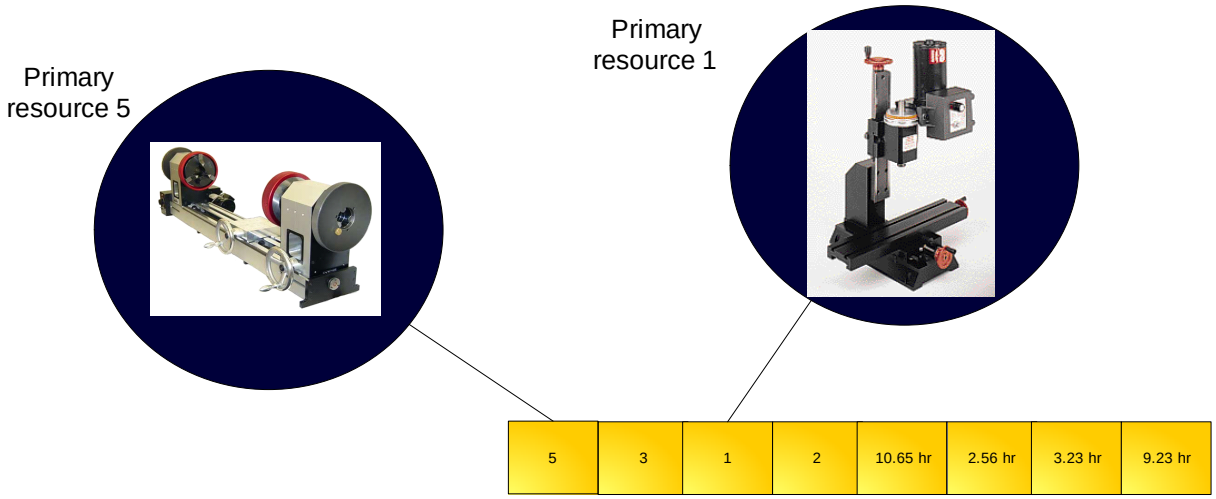

Figure 3.1: The selected schedule representation.

Since the sequencing decision variables ($t_i \ \forall i \in \{1, \ldots, I\}$) are already used in the representation, the only other required conversion to decision variables is relatively simple. If we denote

d_i as the resource index of operation i, then $x_{(d_i)j}$ takes on the value of 1 $\forall i \in \{1, \ldots, I\}$ and $j \in \{1, \ldots, J\}$, where I denotes the total number of operations and J denotes the total number of primary resources.

3.1.2 Constraints

In any scheduling environment, factors exist which prevent the complete optimization of the various objectives. A number of these constraints have been identified for the *Optimatix* problem. For example, precedence relationships, release dates and production calendars are just some of the constraining factors in the *Optimatix* problem.

Consider the scenario illustrated in Figure 3.2. A particle p flies through a search space S, which consists of an infeasible area I and a number of disjointed feasible areas denoted by F_i, where $i \in \{1, \ldots, 3\}$. The particle has to search S to find the optimal solution which can be located in any of the feasible areas.

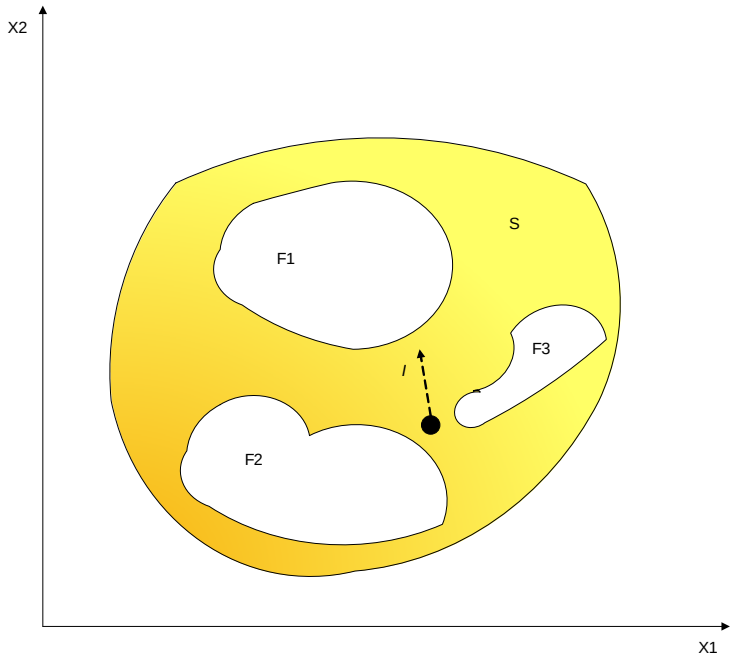

Figure 3.2: Constrained search space.

Various approaches, which range from randomly initializing infeasible particles to repairing infeasible particles by converting them to their associated feasible solutions, have already been used to address constrained optimization problems. Unfortunately many of these methods are not suitable for use in the *Optimatix* problem because of the highly constrained nature of the search space.

The use of a penalty function is the most common way of addressing all constraints which are not implicitly incorporated into the particle representation in cases where feasible solutions

can not be initialized easily. This approach involves calculating a penalty associated with each particle in accordance with the number of constraints which are violated as well as the extent of the violations. Subsequent minimization over the sum of all penalty values attracts the particles towards feasible areas of the search space and repels them from infeasible areas. For example, if two operations intersect on the same primary resource, the penalty is calculated as the number of time units of unavailable capacity.

It should be noted that the inclusion of a penalty function involves an additional objective over which the problem is to be minimized and therefore the *Optimatix* problem becomes a multiobjective optimization problem. However, these multiple objectives can easily be addressed and moreover this allows for the incorporation of additional standard production scheduling objectives so that the schedule can be optimized simultaneously with respect to more than one objective functions. This situation more realistically portrays reality as the "final solution" of many real world problems is often a trade-off between multiple objectives.

3.1.3 Multiple objectives

Currently *Optimatix* aims to evaluate the schedules obtained by the algorithm simultaneously in terms of three scheduling objectives: minimum makespan (Equation (3.1)), minimum lateness/earliness (Equation (3.2)) and minimum total queue time (Equation (3.3)). In Equations (3.1) to (3.3) f_i denotes the finishing time, t_i the starting time and d_i the due date of operation i and L is the set containing the last operations of each job j where J denotes the total number of jobs and I_j the number of operations in job j.

$$\min \max_i \{f_i\} \tag{3.1}$$

$$\min \sum_{i \in L} |d_i - f_i| \tag{3.2}$$

$$\min \sum_{j=1}^{J} \sum_{i \in I_j, i \notin L} (t_{i+1} - f_i) \tag{3.3}$$

In order to address all three objectives simultaneously, a number of concepts in Multiobjective Optimization (MOO) theory will first be discussed.

The use of MOO somewhat complicates the concept of an "optimal" solution. The purpose of MOO is to obtain a set of good compromises rather than a single solution. A non-dominated solution is a good example of an appropriate compromise that will suffice as a solution to a Multiobjective Optimization Problem (MOP). Rardin (1998) defines a non-dominated solution to be a feasible solution to a MOP so that no other feasible solution performs at least as well in

all objective functions and strictly better in one. Assuming a minimization problem:

$$f_k(\boldsymbol{x_1}) \leq f_k(\boldsymbol{x_2}) \qquad \forall k \in \{1, 2, \ldots, n_k\} \tag{3.4}$$

and there exists a $k \in \{1, 2, \ldots, n_k\}$ such that

$$f_k(\boldsymbol{x_1}) < f_k(\boldsymbol{x_2}) \tag{3.5}$$

where $f_k(\boldsymbol{x_1})$ denotes the objective function value of solution vector $\boldsymbol{x_1}$ with respect to the k_{th} objective.

The set of non-dominated solutions of a MOP, is referred to as the Pareto-optimal set and the set of objective function values associated with each point in the Pareto-optimal set, is referred to as the Pareto front. The estimation of the Pareto front is an integral part of MOO and is often used to evaluate algorithm performance.

Engelbrecht (2005) discusses a number of different approaches to MOO which includes, amongst others, the aggregation-based approch to MOO. This approach is considered to be the most simple and involves converting all functions which are to be optimized into a single aggregated objective function of the form $\sum_{k=1}^{n_k} w_k f_k(\boldsymbol{x})$, where $f_k(\boldsymbol{x})$ again refers to the objective function value of solution vector x with respect to the k^{th} objective. The weight associated with the k_{th} objective function is denoted by w_k and n_k is the total number of objectives over which the problem is optimized.

Unfortunately, this method does have a number of drawbacks. All w_k values are problem-dependent and the algorithm needs to be solved repeatedly to obtain an estimate of the Pareto front because in actual fact, one aggregated function is solved with a single-solution metaheuristic. Furthermore, if the Pareto front is concave, in other words, if there are conflicting objectives or if objectives are measured in incompatible units, this method is not suitable at all. In the light of these facts, it is clear that a more suitable approach to MOO will need to be utilized.

Goal programming is intuitive, yet effective and involves the assignment of target values to each of the objective functions (Rardin, 1998). Deficiency variables are also assigned to each objective function value and determines the extent to which the target values are not met. Minimizing over a weighted sum of all deficiency variables and repeated solving of the algorithm with different target values, provides a reasonable estimate of the Pareto front.

3.2 The production scheduling framework

By being able to meet the structural requirements defined in the previous section the scheduling framework is capable of addressing all the various classes of scheduling problems which form part

of the *Optimatix* scheduling environment, as well as a large number of additional scenarios. This was made possible by using two different reference frames as basis for the framework: Zandieh et al.'s scheduling classification, discussed in Section 2.2 and the variations on the job shop scheduling problem, presented in Figure 2.4.

The actual framework developed from these two classifications is indicated in Figure 3.3. The generic production scheduling framework consists of five interrelated components: an initialization procedure, the schedule representation, a conversion mechanism, the penalty function and finally, the metaheuristic algorithm.

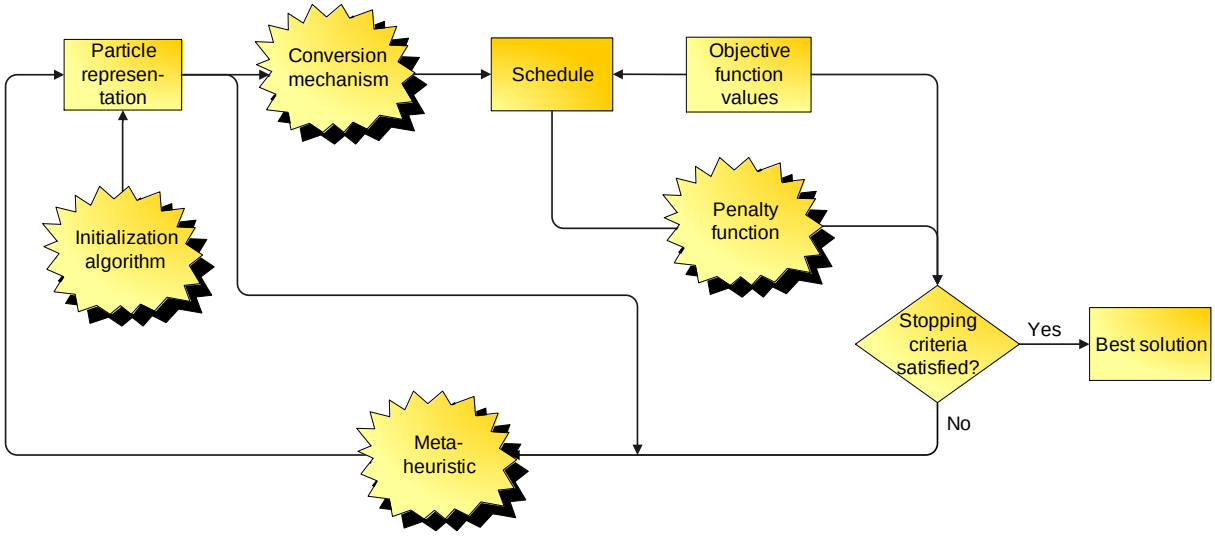

Figure 3.3: The generic production scheduling framework.

The framework is structured around the schedule representation which is continuously updated and improved by being moved through the search space. In order to optimize the schedule, fitness evaluations should be performed for each of the potential solutions. The objective function value of the associated scheduling solution of each particle can be obtained by means of the conversion mechanism between the domain of the metaheuristic and the decision space. This mechanism also incorporates machine breakdowns, scheduled maintenance and production downtime into the schedule and provides as output the decision variables which can be used to evaluate the objective function values. The information obtained from this conversion mechanism is fed back to the metaheuristic algorithm to use during the optimization process. Completing the cycle, the standard objective function values and penalties associated with each particle is used to intelligently update the particle representation until a stopping criteria is satisfied.

Although the overview in the previous paragraph was helpful in describing the internal workings of the framework, the rest of this chapter is dedicated to a more in-depth discussion

of each of the three remaining components of the structure.

3.2.1 The initialization procedure

During algorithm development, it became clear that PSO was unable to obtain feasible solutions from random generated initial solutions. The rationale behind using an initialization algorithm is to start the algorithm off with a feasible initial solution to reduce the search space which the metaheuristic is to cover. Due to the added computational complexity, the framework was developed to not require completely feasible initial solutions. However, experimental results indicated that immense improvement in algorithm performance was obtained by using semi-feasible particles when implementing the framework by means of PSO. For this framework, a semi-feasible initial particle is defined as a particle which does not violate any of the precedence constraints of the problem.

The procedure used to initialize the starting-time variables consists of sorting the various operations according to the number of predecessors associated with each operation. The starting times of all operations with the same number of predecessors are then randomly initialized within the same interval. These initialization intervals are chronologically sequenced to ensure that the initialized solutions satisfy all precedence constraints.

The procedure, illustrated by means of an example in Figure 3.4, reduces the optimization problem to the problem of sequencing sets of operations independently based on the precedence constraints between operations and jobs. However, it should be noted that the inherent structure of the optimization framework allows the metaheuristic to change the sequence of any of the operations in any way it sees fit to obtain an improved solution. After initialization, the penalty function is responsible throughout the rest of the optimization process for recording and also minimizing the extent to which these subsequent sequencing changes violate the problem constraints.

In contrast, allocation variables are initialized randomly. These random numbers are subsequently discretized to obtain resource indices. For each operation, unique intervals are defined for each resource on which the operation may be scheduled, such that a resource index can be assigned to the operation depending on where the allocation variable is initialized.

3.2.2 The conversion mechanism

The conversion mechanism produces as output the actual finishing time associated with each operation given that the starting time of operation i processed on resource d_i is given by t_i. The total processing time of an operation in the *Optimatix* problem depends on a number of factors

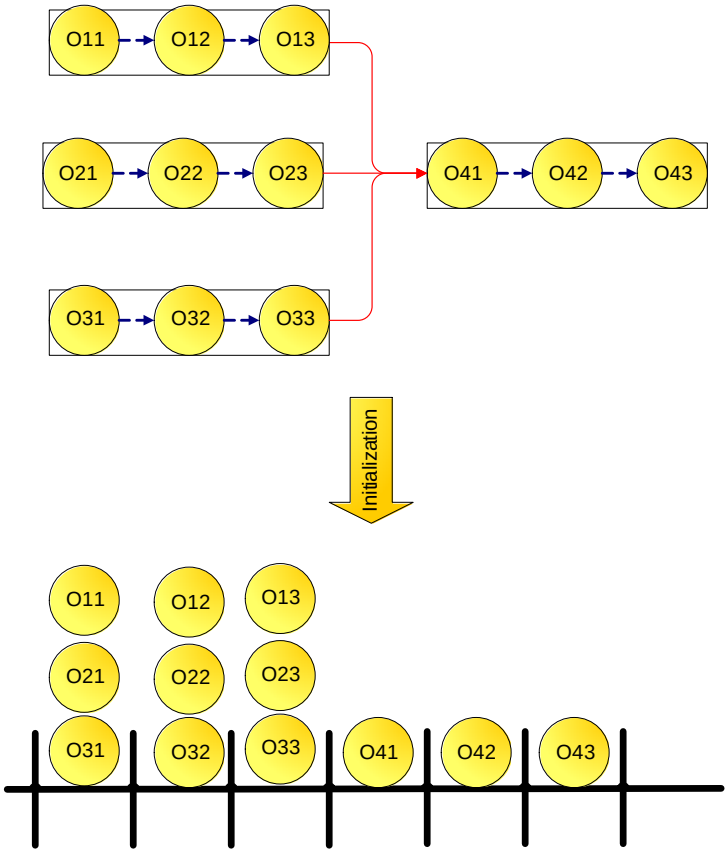

Figure 3.4: A network diagram of the precedence constraints between operations is used as input to the initialization procedure. The operations initialized into their respective intervals is obtained as output.

including the resource on which it is processed, the setup-time associated with the operation and the effect of machine breakdowns, scheduled maintenance and production calendars on available processing time. These factors are incorporated into the processing time of the operations by distinguishing between a proposed finishing time and an actual finishing time for each operation. The proposed finishing time ignores the time intervals where the required resources are not available. If we define g_i to be the proposed finishing time and s_i to be the setup-time of operation i

$$g_i = t_i + x_{id_i} p_{id_i} + s_i \tag{3.6}$$

and

$$s_i = \begin{cases} y_{ji} u_{ji} & \text{if } u_{ji} > 0. \\ x_{id_i} v_{id_i} & \text{otherwise.} \end{cases}$$

where

$$x_{id_i} = \begin{cases} 1 & \text{if operation } i \text{ is performed on resource } d_i. \\ 0 & \text{otherwise.} \end{cases}$$

$$y_{ji} = \begin{cases} 1 & \text{if operation } j \text{ is performed before operation } i \text{ on resource } d_i. \\ 0 & \text{otherwise.} \end{cases}$$

and p_{id_i} denotes the processing time and v_{id_i} the default set-up time of operation i on resource d_i and u_{ji} is the sequence-dependent setup-time of operation i if processed immediately after operation j. Basically, Equation (3.6) allows the framework to make use of default set-up time values when no sequence-dependent set-up times are defined in the problem data as well as for the first operation processed on each resource.

As soon as the proposed finishing time of each operation is determined, production down time can be taken into account to determine the actual finishing time of each operation. The inclusion of production down time is the single most complicating factor within the *Optimatix* problem and this also ensures that the problem can not be modelled analytically and can therefore not be solved to optimality.

There are three factors which influence the available production time in the *Optimatix* environment. For each unique production calendar a number of intervals are defined during which no production on any of the auxiliary or primary resources is allowed. Scheduled maintenance defines intervals which only affects the specified resources and for the sake of simplicity machine breakdowns are treated as prolonged scheduled maintenance. After all production down time intervals are defined, this data is concatenated per primary and auxiliary resource.

Incorporating the down time intervals into the production time of each operation requires an analysis of the relationship between the current starting time of each operation to the down time intervals associated with the primary and auxiliary resources on which it is to be scheduled. If the starting time (q_j) and the finishing time (r_j) is given for each of the j production down time intervals, the actual finishing time of operation i, denoted by f_i, can be determined by the procedure described in Algorithm 1 and the diagram provided in Figure 3.5.

3.2.3 The penalty function

The *Optimatix* scheduling environment is characterized by a large number of constraints, which range from the enforcement of precedence constraints between jobs and operations within the same job to the inclusion of release dates into the schedule. This section attempts to provide an analysis of those constraints enforced by the penalty function as well as the calculation of the penalty values associated with their violation.

Algorithm 1: The conversion mechanism for the incorporation of production down time into the schedule.

for *All operations* i **do**
  for *All downtime intervals* j **do**
    if $f_i \leq q_j$ *or* $t_i \geq r_j$ **then**
      Interval j is an intersected down time interval of operation i
    end
  end
  for *Intersected down time intervals* k *of operation* i *to* K **do**
    if $q_k \leq f_i$ *and* $t_i \leq q_k$ **then**
      $f_i = f_i + r_k - q_k$
    end
    if $q_k < t_i$ *and* $t_i \leq r_k$ **then**
      $f_i = f_i + r_k - t_i$
    end
  end
  for *All downtime intervals* j *from* k *to* J **do**
    if $f_i > q_j$ **then**
      $f_i = f_i + r_k - q_k$
    else
      Break to operation $i + 1$
    end
  end
end

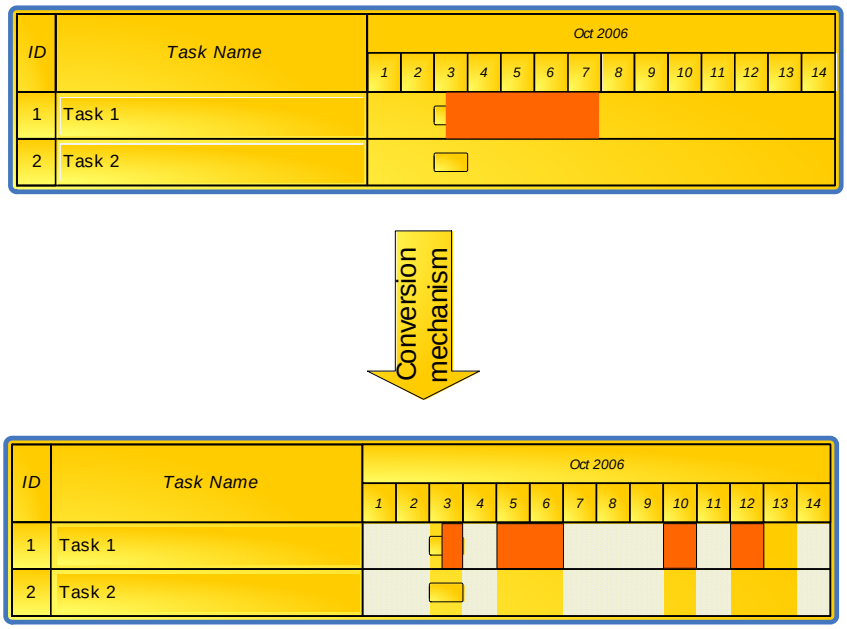

Figure 3.5: The schedule before and after the inclusion of production down time intervals.

The penalty function makes use of two important decision variables, obtained as output from the particle representation and the conversion mechanism, namely the starting time t_i of operation i as well as the actual finishing time f_i, referred to hereafter as simply the finishing time, of operation i. Both t_i and f_i may only take on positive values.

The precedence relationships are enforced by means of Constraint (3.7) where the set A contains all precedence relationships. For each relationship, the starting time of the second operation (operation i) in the precedence relationship has to be larger than the finishing time of the first operation (operation j) in the precedence relationship. In case of violation of Constraint (3.7) a penalty corresponding to the number of time units the precedence constraint is violated can be calculated for each operation according to Equation (3.8). These penalty values are then summed over all precedence relationships (Equation (3.9)) to obtain the precedence relationships penalty value (p1a) for each schedule.

$$f_i \leq t_j \qquad \forall (i,j) \in A \tag{3.7}$$

$$p1_{(i,j)} = |\min(0, (t_j - f_i))| \qquad \forall (i,j) \in A \tag{3.8}$$

$$p1a = \sum_{(i,j)\in A} p1_{(i,j)} \tag{3.9}$$

Since release dates are defined per job and not per operation, Constraint (3.10) ensures that the first operation of job k, defined in set F, is only released on the production floor after the arrival of the job release date r_k. The release date penalty value of each schedule can be

calculated according to Equations (3.11) and (3.12).

$$t_k \geq r_k \qquad \forall k \in F \tag{3.10}$$

$$p2_k = |\min(0, (t_k - r_k))| \qquad \forall k \in F \tag{3.11}$$

$$p2a = \sum_{k \in F} p2_k \tag{3.12}$$

The most critical production scheduling constraints are those that prevent two operations being scheduled at the same time on the same resource. The constraints are relatively simple to formulate, but the calculation of the penalty, defined as the actual intersection time of two operations on the same primary resource, is more complex. If operation i and operation j is performed on the same finite capacity primary resource, the relationship between f_i, f_j, t_i and t_j determines the value of the penalty assigned. The four mutually exclusive scenarios which can occur is incorporated into the calculation of $p3_{(i,j)}$, where J_{d_i} consists of the set of all operations performed on resource d_i. The total penalty value ($p3a$) associated with these "intersection constraints" can be calculated according to Equation (3.14).

$$p3_{(i,j)} = \begin{cases} f_i - t_i & \text{if } t_i \geq t_j,\ f_i < f_j \text{ and } w_{ij} = 0\ \forall (i,j) \in J_{d_i}. \\ f_j - t_i & \text{if } t_i \geq t_j,\ f_j \leq f_i \text{ and } w_{ij} = 0\ \forall (i,j) \in J_{d_i}. \\ f_j - t_j & \text{if } t_j > t_i,\ f_j < f_i \text{ and } w_{ij} = 0\ \forall (i,j) \in J_{d_i}. \\ f_i - t_j & \text{if } t_j > t_i,\ f_i \leq f_j \text{ and } w_{ij} = 0\ \forall (i,j) \in J_{d_i}. \end{cases}$$

where

$$w_{ij} = \begin{cases} 0 & \text{if } f_i \leq t_j \text{ or } f_j \leq t_j\ \forall (i,j) \in J_{d_i}. \\ 1 & \text{otherwise.} \end{cases}$$

$$p^3_{(i,j)} = \begin{cases} f_i - t_i & \text{if } t_i \geq t_j,\ f_i < f_j \text{ and } w_{ij} = 0\ \forall (i,j) \in J_{d_i}. \\ f_j - t_i & \text{if } t_i \geq t_j,\ f_j \leq f_i \text{ and } w_{ij} = 0\ \forall (i,j) \in J_{d_i}. \\ f_j - t_j & \text{if } t_j > t_i,\ f_j < f_i \text{ and } w_{ij} = 0\ \forall (i,j) \in J_{d_i}. \\ f_i - t_j & \text{if } t_j > t_i,\ f_i \leq f_j \text{ and } w_{ij} = 0\ \forall (i,j) \in J_{d_i}. \end{cases} \tag{3.13}$$

$$p3a = \sum_{(i,j) \in J_{d_i}} p3_{(i,j)} \tag{3.14}$$

The operation processing time is independent of the auxiliary resource allocation. This results in the auxiliary resource allocation not having a direct influence on schedule optimization. Instead the auxiliary resources is simply incorporated as constraints into the penalty function. However, before the penalty values can be calculated, the algorithm first attempts to obtain feasible auxiliary resource allocations for all the operations to one auxiliary resource from each set of auxiliary resources required. An allocation is considered feasible if all the specified auxiliary resources are available throughout the time period that the operation is scheduled on the primary resource.

This allocation procedure, indicated in Algorithm 2, provides as output a list of operations for which no feasible auxiliary resource allocation can be obtained. Since this implies that insufficient capacity exist to fulfill the processing requirements for these infeasible operations, the penalties are calculated in Equation (3.15) as the operation production times and are summed to obtain the total penalty value associated with auxiliary resource allocation ($p4a$).

$$p4a = \sum_i a_i(f_i - t_i) \tag{3.15}$$

where

$$a_i = \begin{cases} 1 & \text{if operation } i \text{ is infeasible.} \\ 0 & \text{otherwise.} \end{cases}$$

The total penalty function value (P) for each schedule can be calculated according to Equation (3.16). It should be noted that due to the total per schedule penalty function value being calculated as the sum of the penalty values associated with each set of constraints, the penalization of the objective function is directly proportional to the extent of infeasibility.

$$P = p1a + p2a + p3a + p4a \tag{3.16}$$

3.3 Conclusion

At this point it should be remembered that an algorithm is only as good as the results it produces and in order for appropriate testing and benchmarking to be done, a specific metaheuristic has to be selected within which the framework can be implemented. In compliance with the research question, the next chapter will provide an extensive introduction into Swarm Intelligence and more specifically PSO, before the application of PSO to the framework developed in this chapter, is discussed.

Algorithm 2: Allocation of operations to auxiliary resources.

```
1.1  for All operations i do
1.2      for All resource sets j do
1.3          if A resource is required from resource set j then
1.4              for All resources k in set j do
1.5                  for All scheduled intervals l do
1.6                      if f_i ≤ q_l or t_i ≥ r_l then
1.7                          Operation i will overlap interval l
1.8                      end
1.9                  end
1.10                 if Operation i overlaps any intervals then
1.11                     Operation i cannot be scheduled on resource k of set j
1.12                     if k = K_j then
1.13                         Operation i is infeasible
1.14                         Break to operation i + 1
1.15                     else
1.16                         Break to resource k + 1
1.17                     end
1.18                 else
1.19                     Schedule operation i on resource k of set j
1.20                 end
1.21             end
1.22         end
1.23     end
1.24 end
```

Chapter 4

Particle Swarm Optimization

> An insect may only have a number of brain cells, but insect organizations are capable of architectural marvels, elaborate communication systems, and terrific resistance to the threats of nature (Kennedy et al., 2001).

4.1 Optimization through Swarm intelligence

Over the past twenty years, a number of studies have addressed the collective behaviour of animals in a fixed social hierarchy. The more interesting cases, however, are those where no clear leader can be identified, yet the individuals manage to self-organize in order to meet a specified objective. This objective can range from predator avoidance to searching for food.

These early studies have led to the development of various distributive collective problem solving strategies. More formally, Engelbrecht (2005) defines Swarm Intelligence (SI) as the property of a system whereby the collective behaviours of unsophisticated agents, which interact locally with their environment, cause coherent functional global patterns to emerge. This definition addresses four interrelated concepts, commonly believed to be the secret behind the success of SI as an optimization strategy. These concepts include emergence, social intelligence, stigmergy and adaptation.

Consider the example of a flock of birds which have a single objective — finding a source of food within a predefined solution space. The ability of the flock to find the food source can be best explained by the concept of emergence: simple local interactions which lead to global effects. A single bird possesses a relatively limited amount of intelligence, but the complex behaviour of the entire flock allow the birds to settle on the food source. This flocking behaviour results from simple social interactions between individual birds. The knowledge that a single bird obtains from interacting with and adapting to its environment is propagated through the entire flock by

indirect communication or stigmergy between individuals. In other words, even though a single bird is not aware of the location of the source of food, the local interactions and information-exchange mechanism of the flock leads to convergence of the entire flock to the intended location.

The purpose of Computational Swarm Intelligence (CSI) is to model the simple local interactions of individuals with the environment and neighbouring individuals, in order to obtain more complex behaviours that can be used to solve optimization problems (Engelbrecht, 2005). To aid this objective, two CSI paradigms have been developed: Ant Algorithms, which are developed from behaviours observed in ant and termite colonies and Particle Swarm Optimization (PSO) — the focus of this project.

4.2 Introduction to Particle Swarm Optimization (PSO)

4.2.1 Origin and applications

Many researchers trace the origins of the PSO algorithm back to Reynold's "boid" simulations. The initial objectives of this and the other collective behaviour studies of the late 80s was to simulate the graceful, unpredictable choreography of collision-proof birds in a flock (Eberhart and Shi, 2001). 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 et al. 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 like evolving artificial neural networks and optimizing various planning and scheduling models to more specific applications, like the design of aircraft wings and the generation of interactive, improvised music.

4.2.2 Schedule-specific applications

One of the most complex production scheduling applications of PSO, was performed by Xia and Wu (2005), who developed a Simulated Annealing-Particle Swarm Optimization (SA-PSO) based hybrid solution strategy for the Flexible Job Shop Scheduling Problem (FJSSP). The algorithm addresses FJSSPs with both total and partial flexibility. In this specific implementation, only the allocation of operations to resources is done by means of PSO. The actual sequencing of the assigned operations are performed by a Simulated Annealing (SA) algorithm. Additionally, multiple objectives are addressed by combining the relevant objectives into a single weighted sum objective. Finally, although the SA-PSO algorithm improved on current best known FJSSP

benchmark values, the algorithm found a 56 operation problem to be challenging.

The most interesting aspect of Xia and Wu's article is the problem mapping of PSO particles to schedules. All the machines on which a specific operation can be processed, is sorted according to their shortest processing time. One particle is represented by a discreet value corresponding to each operation. This value refers to the position of the machine to which the operation is allocated, with respect to the other sorted machines. The continuous optimization problem is converted to its corresponding discreet optimization problem by simply rounding the continuous PSO variables to discreet resource indices.

Due to its extreme intractability, the basic job shop scheduling problem has been used a number of times to test the performance of scheduling algorithms. A number of authors have managed to solve basic JSSPs with hybrid PSO algorithms (Xia Weijun and Genke, 2004). Other scheduling applications include schedule optimization in a flexible manufacturing system (Jerald et al., 2004), various Permutation Flow Shop Scheduling Problem (PFSSP)s (Lian et al., 2006) and a resource-constrained project scheduling problem (RCPSP) (Zhang et al., 2004).

4.3 The Basic PSO algorithm

In order to apply a PSO-based approach to the proposed problem, a working knowledge of PSO techniques is required. A good place to start is with the simplest of the lot — the basic PSO algorithm.

4.3.1 Algorithm structure

All Swarm Intelligence-based solution strategies, and therefore also the PSO algorithm, have four distinct characteristics:

- **A group of individuals performs space and time calculations.** In the PSO algorithm, each potential solution to the scheduling problem is represented by the position of a particle in a multi-dimensional hyperspace (Figure 4.1). Velocity and displacement updates are applied over time to each particle to move it to a different position and therefore a different solution in the search space.

- **The individuals respond to quality factors in the environment.** Information about the environment is stored by each particle in the form of the personal best position obtained by each particle (*pbest*) and the global best position obtained by the entire swarm (*gbest*). Both the *pbest* and *gbest* values are incorporated into the velocity updates and therefore have a significant influence on particle trajectories.

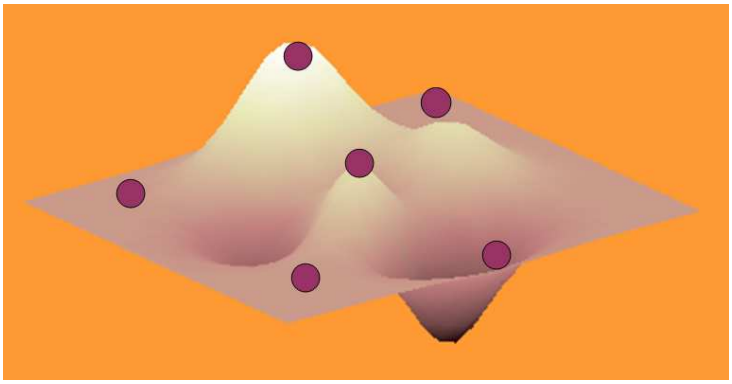

Figure 4.1: Particle positions that represent potential solutions.

- **The behaviours of the individuals should only be changed when the computational cost is justified, which implies that a minimum level of stability should be maintained.** Although the state of the swarm is changed every time the *pbest* and *gbest* values are updated, these updates do not occur at each iteration, but only occur if a particle happens to stray onto a better solution.
- **A specific method of exploration should be incorporated into the algorithm.** Their are numerous methods of ensuring that exploration into a PSO algorithm. Due to the importance of adequate exploration, this is discussed in more detail in Section 4.3.2.

Emergence and social intelligence is incorporated into the velocity update which is considered to be the most critical component of the PSO algorithm. Figure 4.2 illustrates that the magnitude and direction of a particle's velocity at time step 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 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 velocity of particle i in dimension j at time step $t+1$ is given by:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_j(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_j(t)[Y_j(t) - x_{ij}(t)] \tag{4.1}$$

where $v_{ij}(t)$ represents the velocity of particle i in dimension j at time step t, c_1 and c_2 are the cognitive and social acceleration constants, $y_{ij}(t)$ and $x_{ij}(t)$ respectively denotes the personal best position (*pbest*) and the position of particle i in dimension j during time step t. $Y_j(t)$ denotes the global best position (*gbest*) in dimension j and $r(t)$ is a random number sampled from a uniform distribution during time step t.

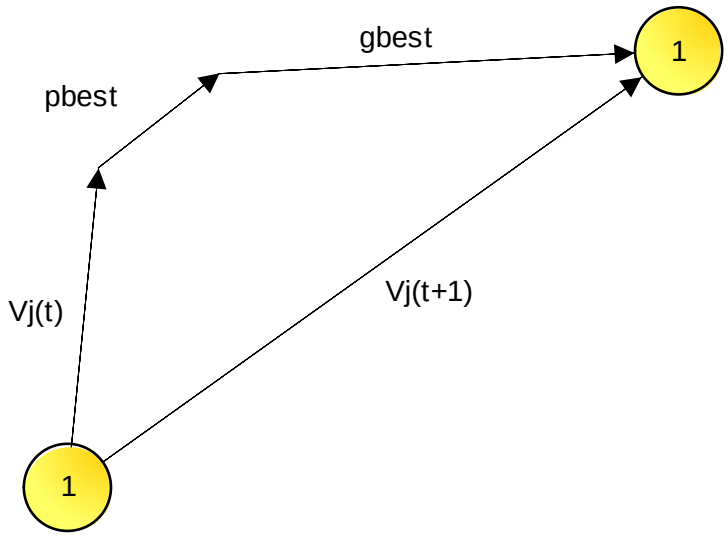

Figure 4.2: Particle velocity as resultant of three components.

The displacement of particle i at time t is simply derived from the calculation of $v_{ij}(t+1)$ in Equation (4.6):

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{4.2}$$

For the sake of completeness a flowchart of the basic PSO algorithm is provided in Figure 4.3, where the *pbest* and *gbest* updates referred to in Figure 4.3 are enforced by Equations (4.3) and (4.4).

$$\text{if } f(\boldsymbol{x_i}) < f(\boldsymbol{y_i}) \text{ then}$$
$$\boldsymbol{y_i} = \boldsymbol{x_i} \qquad \forall i \in \{1, \ldots, n_s\} \tag{4.3}$$
$$\text{if } f(\boldsymbol{y_i}) < f(\boldsymbol{Y}) \text{ then}$$
$$\boldsymbol{Y} = \boldsymbol{y_i} \qquad \forall i \in \{1, \ldots, n_s\} \tag{4.4}$$

4.3.2 Algorithm requirements and limitations

A number of factors exist which results in the basic PSO algorithm not being suitable for use in the *Optimatix* scheduling framework. Some of these limitations include: the algorithm has a tendency to stagnate, has problem-dependent parameters and there exists significant potential for improvement in the algorithm's use of exploration and exploitation. Fortunately a number of strategies exist to overcome these limitations and there is significant proof that a variation on the basic PSO algorithm may be more than adequate (Engelbrecht, 2005).

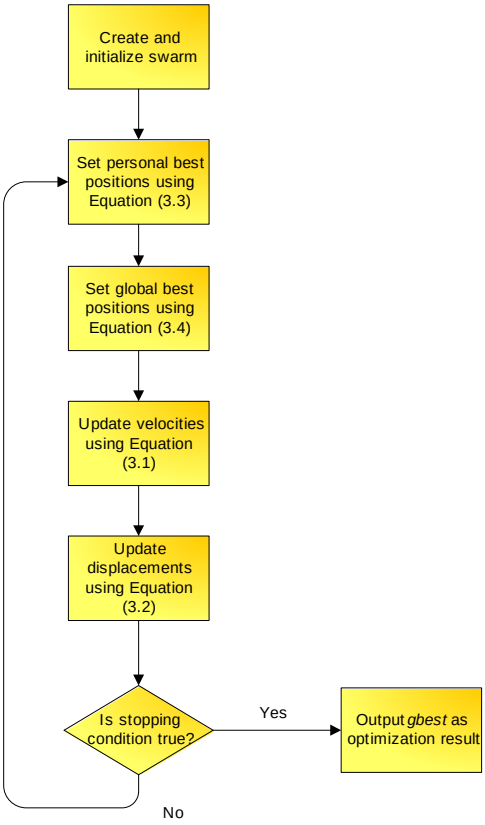

Figure 4.3: The basic PSO algorithm.

The tendency to stagnate

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. However, if any of the particles reaches a position in the search space where

$$y_{ij}(t) = x_{ij}(t) = Y_j(t) \tag{4.5}$$

there is no driving force for the particle to continue exploring the rest of the search space. This property can result in the swarm converging to a point which cannot necessarily be classified as a local optima. It can simply be said that the swarm has reached an equilibrium state.

Fortunately a number of different methods have been developed to force the *gbest* particle out of its stagnation point. The first method, the GCPSO will be used in this project. In effect the *gbest* particle is forced into a random search around the global best position. The size of the search space is 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.

The second approach to prevent algorithm stagnation involves the use of the Convergent Linear Particle Swarm Optimization (CLPSO) algorithm. Originally developed to address con-

strained optimization problems, the algorithm requires that all particle adjustments be linear combinations of the initial conditions (Engelbrecht, 2005).

The problem-dependent nature of input parameters

Engelbrecht (2005) states that the performance of PSO is sensitive to control parameter choices and inadequate selection of parameter values can lead to divergent or cyclic behaviour. This limitation is often addressed by means of comprehensive parameter derivation studies.

Exploration versus exploitation

The use of approximation algorithms results in a trade-off required between exploration and exploitation. Where exploration refers to the algorithm's ability to cover large areas of the search space, exploitation refers to the refinement of a potential solution. In most applications, exploration is called for in the initial stages of algorithm execution and exploitation is utilized near the end, to fine-tune potential solutions. The easiest way of addressing this trade-off is again by judicial selection of required input parameters.

4.4 Existing variations on the basic PSO

In order to address the inherent limitations and requirements of the basic PSO, a number of variations on the PSO algorithm has been developed. These variations, indicated in Figure 4.4, are organized into six main categories:

- **Social based algorithms** use different social typologies or network structures. The three most common network structures implemented in PSO are indicated in Figure 4.5. The black lines between particles represent the existence of a communication mechanism between the linked particles. Algorithms using a different *pbest* and *gbest* calculations are also classified as social based algorithms.

- **Hybrid algorithms** refer to all PSO variations that combine Evolutionary Algorithms (EA)-based concepts like selection, reproduction and mutation, as well as all algorithms which consist of more than one metaheuristic.

- **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.

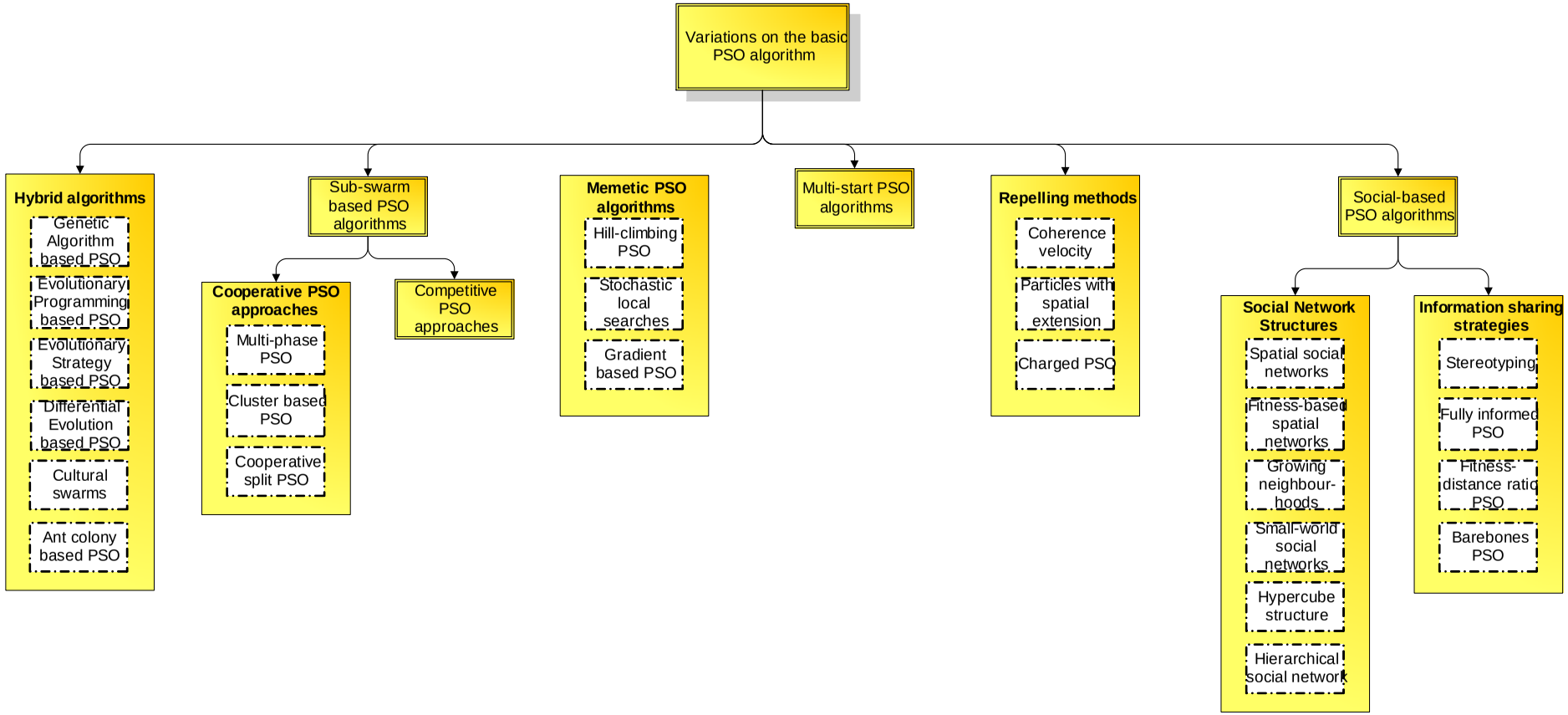

Figure 4.4: Variations on the basic PSO algorithm (Engelbrecht, 2005).

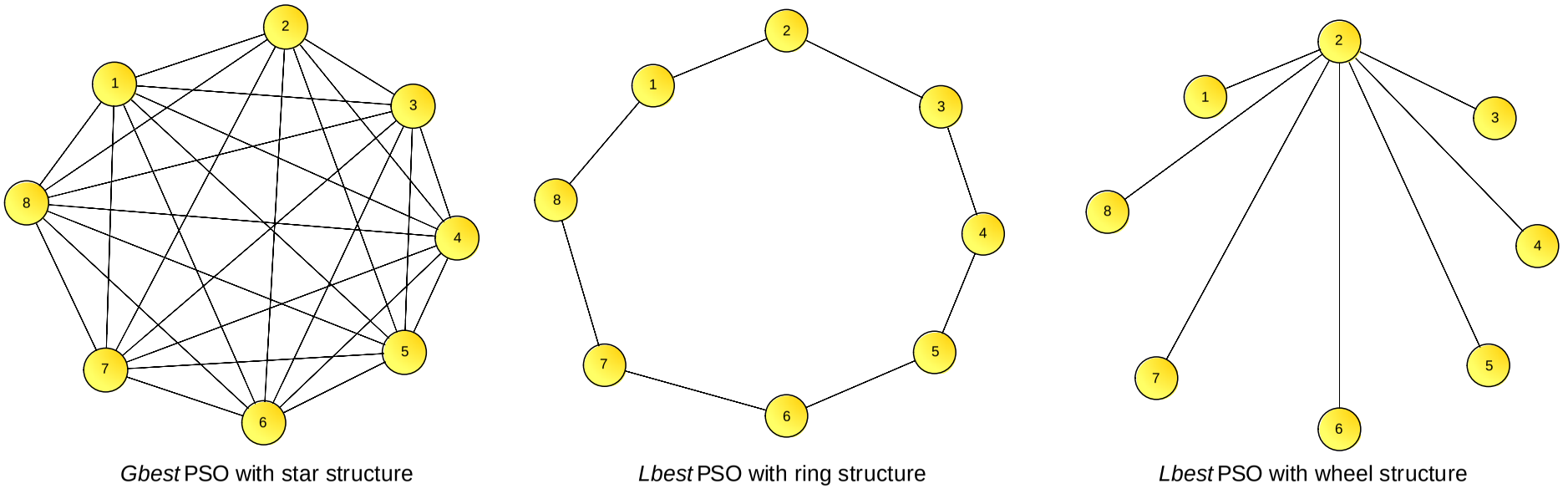

Figure 4.5: The three most popular social network structures. The black lines between particles indicate the propagation of information throughout the swarm (Kennedy et al., 2001).

- **Memetic algorithms** incorporates local search procedures between iterations of 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 has 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 to repel adjacent particles.

4.5 Applying PSO to the *Optimatix* problem

In order to apply PSO effectively to the production scheduling framework developed in the previous chapter, it is important to briefly review the context and functionality of the metaheuristic within the framework. Basically, the framework provides the PSO algorithm with a schedule representation to be intelligently updated. These updated schedules are continuously fed back to the framework which in turn provides the PSO algorithm with information of the fitness of the various solutions. This information is then used to further guide the optimization algorithm's search throughout the problem space.

Unfortunately the basic PSO, in its current form, is not able to meet these requirements and the algorithm has to be adapted accordingly. This adaptation takes the form of a discretization mechanism, the inclusion of two additional parameters and converting the basic PSO to a GCPSO.

4.5.1 Algorithm discretization

Due to the continuous nature of displacement and velocity values, all variables obtained as output from a PSO-based algorithm are elements of the set of real numbers. Because the particle representation of the *Optimatix* production scheduling framework consists of one discreet and one continuous-valued vector (Section 3.1.1), a discretization-mechanism has to be incorporated into the PSO algorithm. To interpret the resource allocation obtained as output from the PSO algorithm, the same initialization mechanism of Section 3.2.1 is used.

It should be noted that the discreet variables are converted back to their associated continuous values before being used by the PSO algorithm in the velocity and displacement updates of the next iteration.

4.5.2 The inclusion of inertia weight as an input parameter

In order to achieve greater control over the exploration and exploitation abilities of the swarm and to ensure that the particle trajectories converge, an inertia weight (w) was incorporated into the velocity update of the algorithm thereby changing it to Equation (**??**).

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_j(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_j(t)[Y_j(t) - x_{ij}(t)] \tag{4.6}$$

4.5.3 The Guaranteed Convergence Particle Swarm Optimization (GCPSO) algorithm

In order to prevent algorithm stagnation, the GCPSO algorithm applies different velocity and displacement updates, respectively indicated by Equations (4.7) and (4.8), to the global best particle.

$$v_{\tau j}(t+1) = -x_{\tau j}(t) + Y_j(t) + wv_{\tau j}(t) + \rho(t)(1 - 2r_j(t)) \tag{4.7}$$

$$x_{\tau j}(t+1) = Y_j(t) + wv_{\tau j}(t) + \rho(t)(1 - 2r_j(t)) \tag{4.8}$$

The GCPSO seemed to be effective to prevent algorithm stagnation as can be seen from Figure 4.6, where the GCPSO algorithm avoids swarm stagnation at iterations 22, 52 and 93.

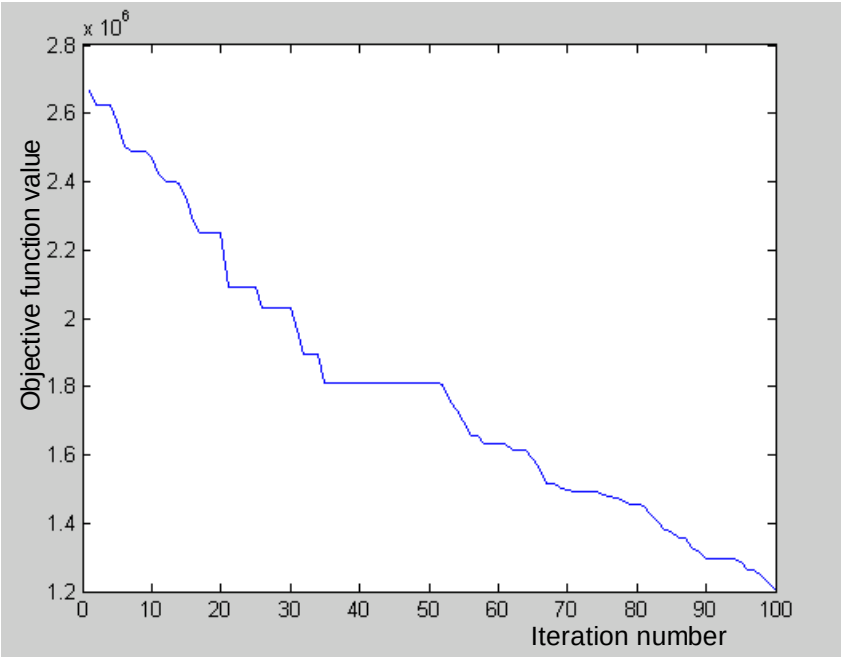

Figure 4.6: Convergence graph of the implementation of the GCPSO algorithm on the *Optimatix* problem. The plot is of the aggregated objective function which minimizes the deficiencies between the target values for the objective functions and the actual values obtained.

4.6 Conclusion

This implementation of the GCPSO in the generic production scheduling algorithm is the simplest algorithm structure which can be used to solve the *Optimatix* problem by means of PSO. As this structure results in a relatively computationally complex algorithm, the algorithm development process described in the next chapter serves as a detailed rationale.

The rationale also serves another purpose. After the design and coding of the basic algorithm requirements, the results obtained are simply not good enough for the client and in order to identify suitable strategies to improve algorithm performance, it is sometimes helpful to consider the path that has already been walked.

Chapter 5

Reflection on the algorithm development process

Algorithm development consisted of three distinct phases. The initial phase was simply concerned with obtaining a feasible solution. After the feasible solution was obtained, it became clear that the algorithm was very sensitive to the input parameter values. Subsequently the most critical parameters were identified and used in an experimental run. After suitable parameter values were obtained, the algorithm development process was concluded with the final phase which involved further experimentation with different parameters and the inclusion of a local search.

A number of performance measurements were used throughout the development process to evaluate algorithm performance. All algorithm improvements were evaluated with respect to the *Optimatix* benchmark data set, a set of benchmark data developed in Chapter 6 that requires the schedule to be evaluated simultaneously with respect to all three objective functions.

The following performance measurement criteria largely drove the improvement process (Engelbrecht, 2005):

- The accuracy $acc(\boldsymbol{Y_j})$ of a solution $\boldsymbol{Y_j}$ is defined by Equation (5.1), where $\mathbf{x}$ represents the theoretical optimum. This was the most important measure used during the initial phase of algorithm development. Due dates are not a binding constraint in the *Optimatix* problem. This implies that a large number of feasible solutions exist for any problem modelled as an *Optimatix* problem and that the global optima, when minimizing the penalty function, is always equal to zero.

$$acc(\boldsymbol{Y_j}) = |f(\boldsymbol{Y_j}) - f(\boldsymbol{x})| \tag{5.1}$$

- As soon as the algorithm started generating feasible solutions the most obvious measurement, the actual objective function values of each of the three objective functions, was used.

- Efficiency, in other words either the time taken or the number of iterations either to a specified solution or level of accuracy, became increasingly important as the computational complexity of the algorithm increased. As time to convergence increased it became more and more necessary to find a good answer within as few iterations as possible.

- After the algorithm proved competent in generating feasible solutions, reliability was incorporated as a measurement to determine the percentage of time the algorithm produced feasible solutions. The reliability (R) of an algorithm can be defined by Equation (5.2), where n_ϵ is the number of feasible solutions and N is the number of times the algorithm was executed.

$$R = \frac{n_\epsilon}{N} \times 100 \tag{5.2}$$

- At the final stages of algorithm development, the robustness of the algorithm was also measured. This measurement focuses on the stability of the solution obtained over a number of algorithm executions. The robustness (Rb) of a performance criteria can be calculated according to Equation (5.3), where $\overline{\theta}$ denotes the average and σ_θ the variance of the performance criteria over a fixed number of iterations.

$$Rb = [\overline{\theta} - \sigma_\theta, \overline{\theta} + \sigma_\theta] \tag{5.3}$$

5.1 Phase 1: working towards a feasible schedule

Obtaining a feasible solution turned out to be no mean feat as the procedure followed in Figure 5.2 testifies. The figure also serves as a rationale of why the complex structure described in Chapter 3 is required. Throughout this phase accuracy was used as the performance measurement of choice and the results obtained by the algorithm upon conclusion of Phase 1 is indicated in Table 5.1. The convergence graph of this first feasible answer of the algorithm is indicated in Figure 5.1. The graph was obtained under the conditions stipulated in Table 5.2.

Table 5.1: Results obtained at completion of the first development phase.

Performance measurement	Answer obtained
Makespan	$10564h$
Lateness/earliness	$29108h$
Queue time	$9879h$
Aggregated objective function	$49100h$
Penalty function	$0h$
Time to solution	$226s$

Table 5.2: Parameter values used after completion of the first development phase.

Parameter	c_1	c_2	w	δ	P	a	b	n_s
Value used	1.4	1.4	0.7	0.5	35	10000	700	30

5.2 Phase 2: parameter derivation

One of the major drawbacks of PSO is that algorithm performance is dependent on the values of a large number of input parameters. To add insult to injury the inclusion of the GCPSO resulted in a number of additional parameters being included in the algorithm.

Both the fields of Operations Research and Computational Intelligence, is increasingly moving towards parameter-free optimization. These algorithms are capable of continuously adapting the values of their input parameter according to information obtained about the search space and the current state of the optimization process. Parameter-free algorithms reduce the time and cost associated with experimental runs geared towards obtaining suitable values for all parameters each time a new problem is solved. Although this is the ideal situation, it is not required for optimization in the *Optimatix* environment, since algorithm implementation involves a large degree of customization which generates revenue for the company.

In order to determine the most effective input parameter values while taking into account the complex relationships between the parameters, the optimization algorithm was repeatedly executed and the effect of different parameter values on algorithm performance was monitored. Table 5.3 provides information regarding the parameters tested, as well as a brief explanation of their purpose and the values used as input to the experimental run. It should be noted that the inertia weight, penalty function coefficient and the size of the swarm was kept constant and that the parameter values tested complies with Clerc's parameter selection heuristics (Equation (5.4)) which guarantee convergent particle trajectories (Engelbrecht, 2005).

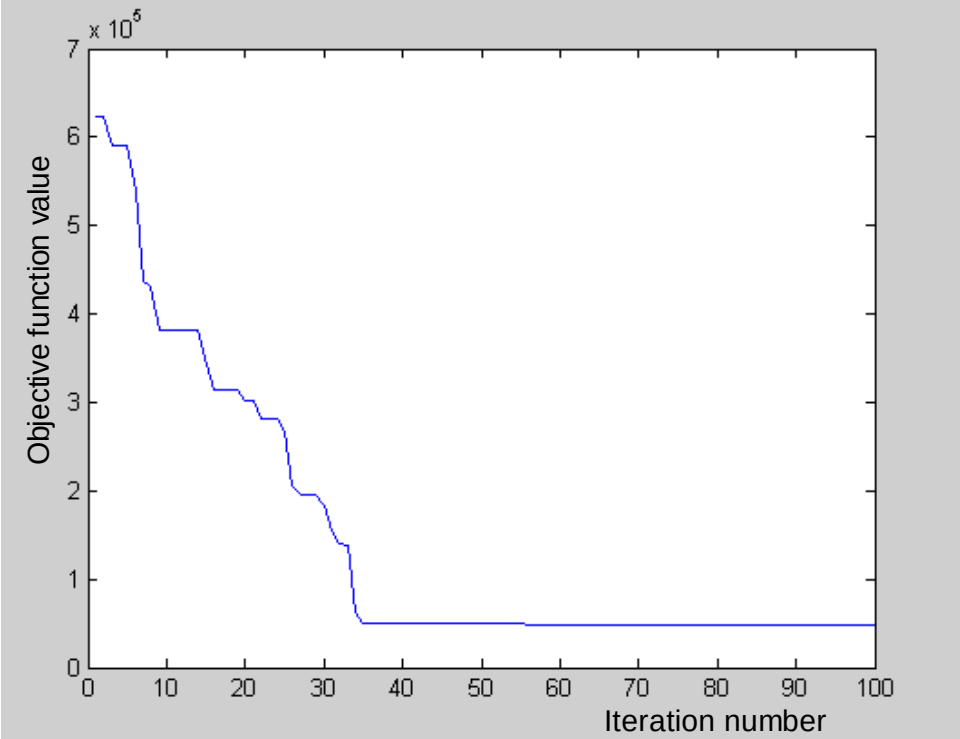

Figure 5.1: Convergence graph of the algorithm at the end of Phase 1 of the development process. The plot is of the aggregated objective function which minimizes the deficiencies between the target values for the objective functions and the actual values obtained.

$$1 > w > \frac{1}{2}(c_1 + c_2) - 1 \geq 0 \tag{5.4}$$

From the results of the experimental analysis illustrated in Figure 5.3, it became clear that a definite relationship exists between the input parameter values. The data was analyzed and suitable parameter values were selected based on three criteria: the feasibility of the schedule, the value of the aggregated objective function and the rate of convergence of the algorithm, where slower convergence is more desirable. Based on this analysis, the best parameter values for the *Optimatix* benchmark data is indicated in Table 5.4. The results obtained from this set of parameter values are indicated in Table 5.5 with the associated convergence graph in Figure 5.4.

5.3 Phase 3: performance improvement

The final phase of development again focused on improving the algorithm structure. From the results obtained after parameter derivation, it became clear that algorithm performance could be improved by incorporating a local search after the final iteration.

Rardin (1998) describes a local search as an algorithm that starts with a feasible solution and advances along a search path of feasible points with ever-improving function values. Concluding the third phase of algorithm improvement, this local search resulted in significant improvement in algorithm performance. This improvement can be attributed to the transferring of the global best particle to the nearest optimum in the search space.

5.4 Conclusion

After the final algorithm design and testing, the focus of performance measurement shifts from evaluation for the sole purpose of identifying and correcting weaknesses to a more thorough benchmark approach for algorithm evaluation against client expectations and academic standards.

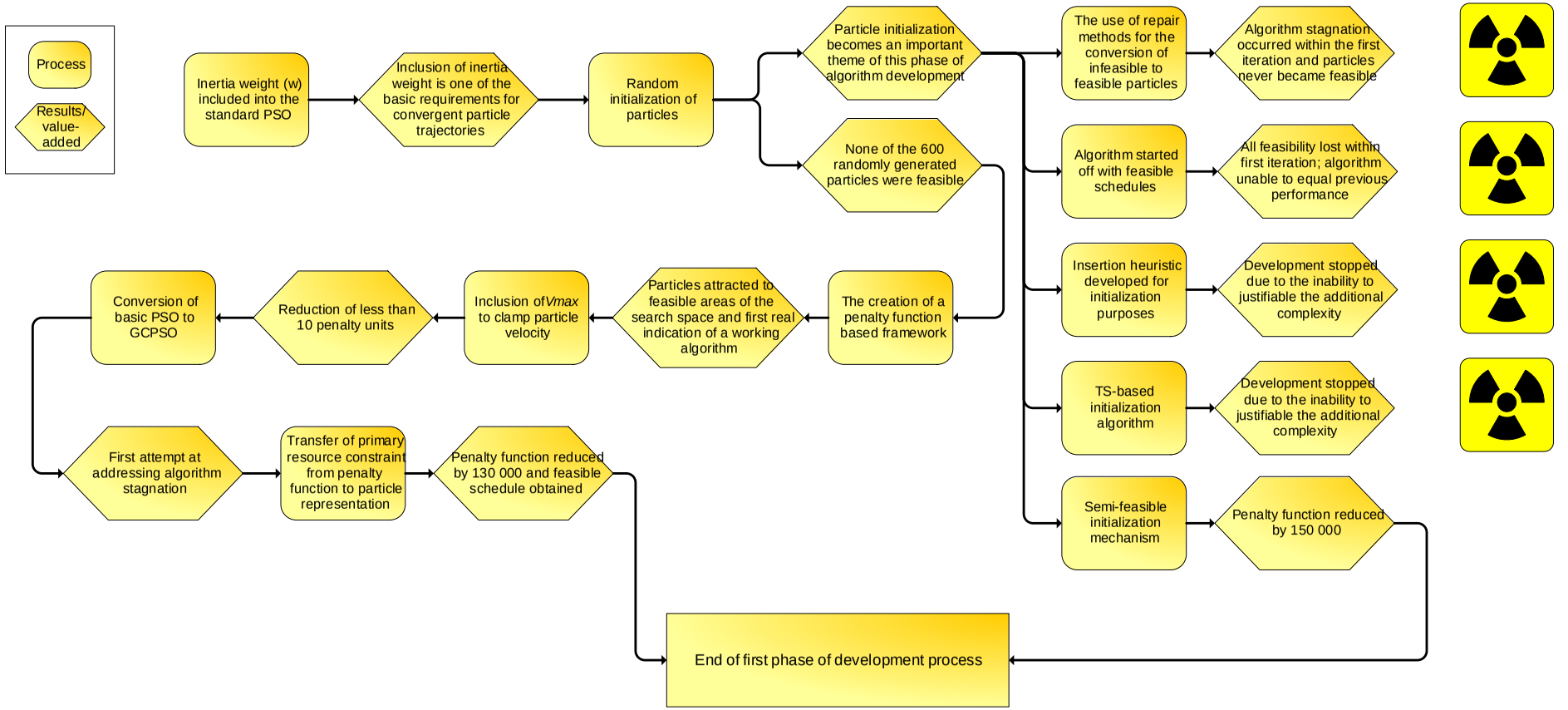

Figure 5.2: Process flow of the first phase of the algorithm development phase. The processes as well as the value-added is indicated.

Table 5.3: The PSO algorithm parameters and experimental run information.

Parameter	Symbol	Description	Suitable values (as stated in literature)	Experimental run i uts
Swarm size	n_s	The number of particles in the swarm	$n_s \in [10, 30]$	$n_s = 30$
Neighbourhood size	nN_i	Controls the extent of social interaction	$nN = 1$ for *gbest* PSO	$nN = 1$
Number of iterations	FE	The number of times the algorithm is executed	Problem dependent	Obtained as output
Cognitive acceleration constant	c_1	Controls the degree of cognitive stochastic influence	$c_1 = 1.4$	$c_1 = \{4, 2, 1.4\}$
Social acceleration constant	c_2	Controls the degree of social stochastic influence	$c_2 = 1.4$	$c_2 = \{0, 2, 1.4\}$
Inertia weight	w	Controls the exploration and exploitation abilities	$w = 0.7$	$w = 1.1$
Velocity clamping threshold	V_{max}	Controls granularity of search	$V_{max} = \delta(x_{max} - x_{min})$, where $\delta \in [0, 1]$	$\delta = \{0.25, 0.5, 1\}$
Allocation initialization parameter	a	Interval size in which allocation variables are initialized	Unknown	$a = \{20000, 15000, 10000\}$
Starting times initialization parameter	b	Interval size in which starting times are initialized	Unknown	$b = \{500, 700, 1000\}$
Threshold parameter 1	ϵ_s	GCPSO parameter determining influence of algorithm success on size of search space	$\epsilon_s = 15$	$\epsilon_s = 15$
Threshold parameter 2	ϵ_c	GCPSO parameter determining influence of algorithm failure on size of search space	$\epsilon_c = 5$	$\epsilon_c = 5$
Penalty function coefficient	P	The importance weighting of the penalty function with respect to the other three objective functions	Unknown	$P = 20$

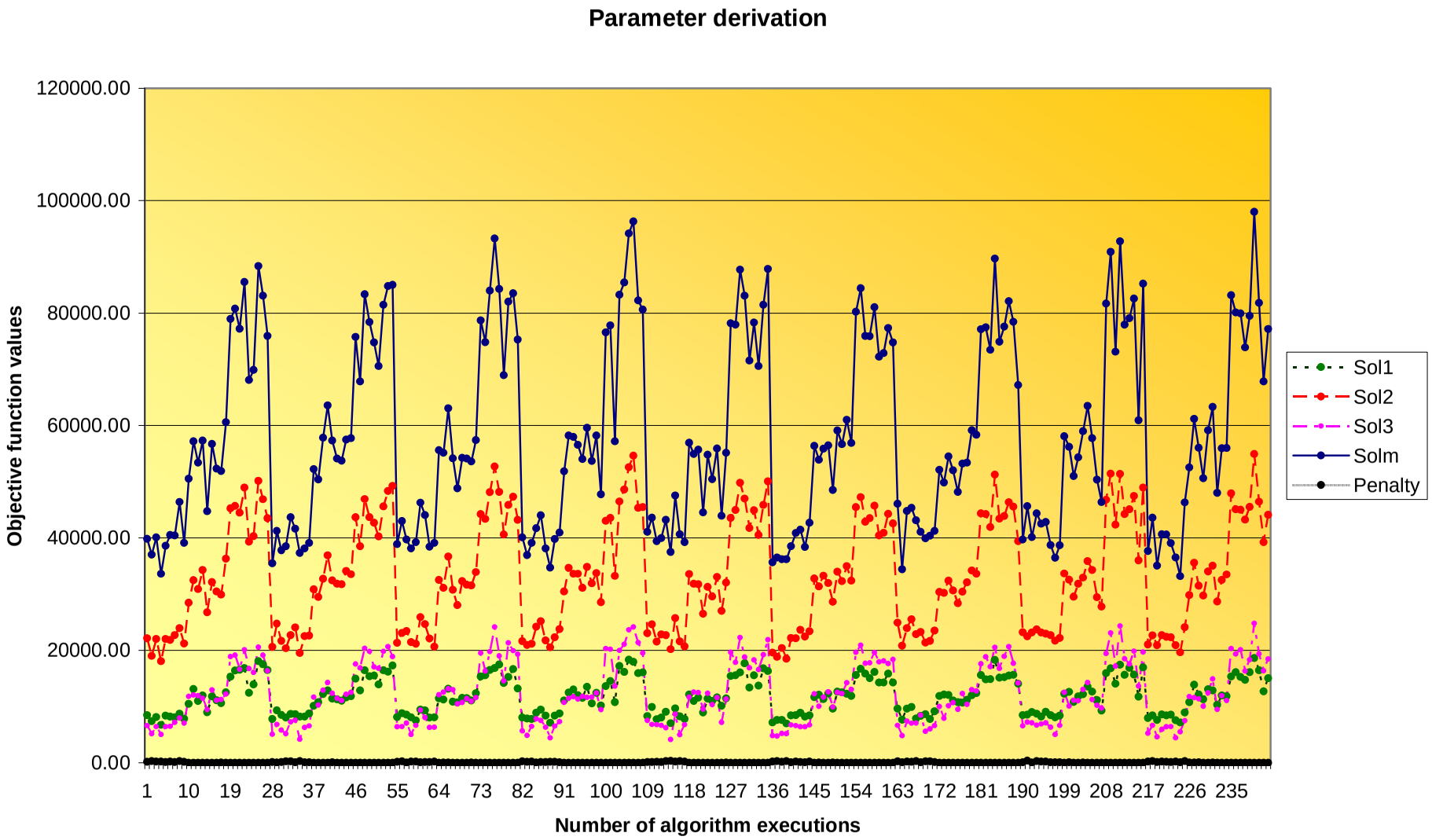

Figure 5.3: The relationship between the various input data parameters.

Table 5.4: Parameter values used after completion of the first development phase.

Parameter	c_1	c_2	w	δ	P	a	b	n_s
Value used	2	2	0.9	0.25	20	20000	700	30

Table 5.5: Results obtained at completion of the first development phase.

Performance measurement	Answer obtained
Makespan	$10229h$
Lateness/earliness	$27849h$
Queue time	$9304h$
Aggregated objective function	$46930h$
Penalty function	$0h$
Time to solution	$117.29s$

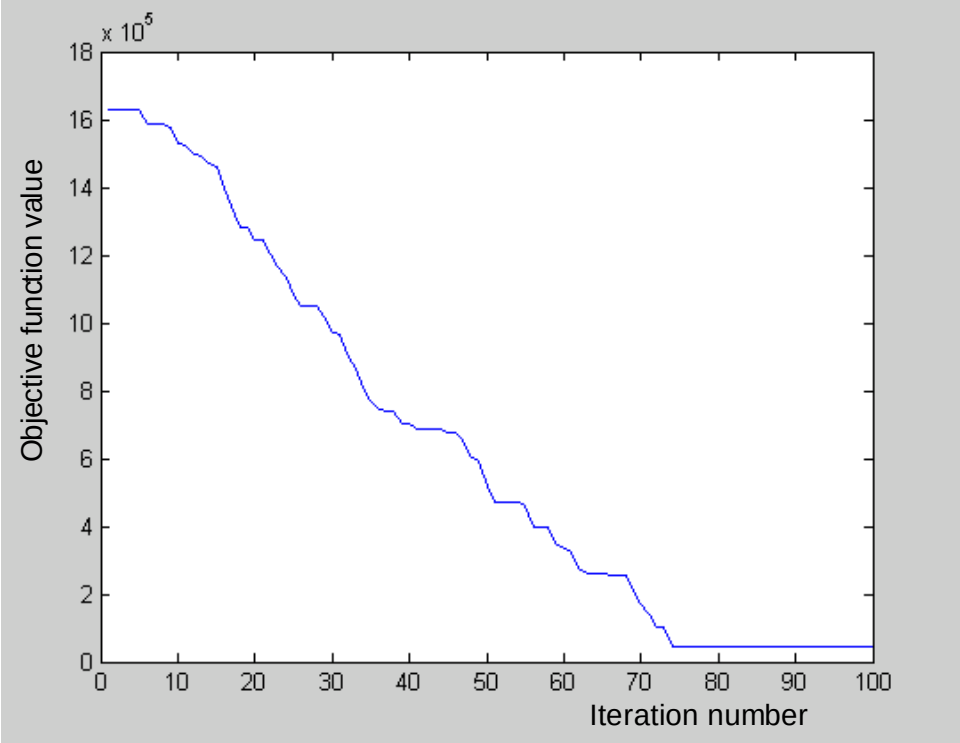

Figure 5.4: Convergence graph of the algorithm after parameter derivation. The plot is of the aggregated objective function which minimizes the deficiencies between the target values for the objective functions and the actual values obtained.

Chapter 6

Performance evaluation

Testing and validation makes up an integral part of the algorithm design process. The algorithm which was developed in Chapter 4 and improved in Chapter 5 has to be thoroughly verified and validated. Since the validation process was addressed in Chapter 2 this chapter will only focus on algorithm verification and evaluation with respect to both academic standards and client expectations. Unless otherwise specified, the final Guaranteed Convergence Particle Swarm Optimization (GCPSO) algorithm developed in the previous chapter is used under the conditions indicated in Table 5.4.

6.1 Algorithm verification

Verification can simply be defined as the process of evaluating the algorithm output against the expectations created by the conceptual model. In other words verification simply addresses the question: "Does this answer make sense?"

In order to facilitate algorithm verification the schedule was drawn in Excel. The schedule generated for the *Optimatix* benchmark data set does not have any serious anomalies, all the constraints are satisfied and in general, it appeared to be a good scheduling effort.

6.2 Benchmarking against industry standards

The easiest way to evaluate the academic significance of an algorithm is to test its performance with respect to standard benchmark problems provided in literature for this very purpose. Unfortunately, not a lot of benchmarking problems exist in literature for production scheduling algorithms which address both allocation and sequencing and those problems that do exist focus mainly on very small sized problems.

The most commonly used Flexible Job Shop Scheduling Problem (FJSSP) benchmark problem sets are the two sets developed by Kacem et al. (2002). The *T-FJSP* problem set requires the scheduling of 30 operations, each one on any one of 10 primary resources. The *P-FJSP* problem set consists of 27 operations to be scheduled on 8 primary resources. The second problem set falls within the class of Partial Flexible Job Shop Scheduling Problems, where all operations may not be processed on all resources. These problems are not as severely constrained as the *Optimatix* problem and are therefore in a completely different class to the problems which *Optimatix* deals with. Instead of spending a lot of time benchmarking against these industry standards this phase of the project will rather focus on the development of a benchmark data set catered specifically for the needs of *Optimatix*.

6.3 Design of a customized benchmark problem

The benchmark problem sets found in literature are clearly inadequate in evaluating the proposed algorithm. In order to overcome this problem a benchmark problem set was derived specifically for the *Optimatix* environment. Containing all constraints and features of the problems addressed by this project, a much better indication of algorithm performance can be obtained. Additionally, future production scheduling research can benefit from meaningful performance comparisons enabled by the existence of a benchmark data set which is generic enough to be applicable to a large number of scheduling scenarios.

The *Optimatix* benchmark problem is the same size as the problems which Kacem et al. (2002) describes to be great sized: namely 56 operations are to be scheduled on any operation from a resource set differing in size from one to 10 primary resources, selected from a total of 216 resources.

The benchmark data is based on real world production data sourced from one of the *Optimatix* data sets. However, the data set had to be extended to include sequence-dependent set-up times and resource-dependent processing and resource-dependent default set-up times. To ensure that this extension is based on realistic assumptions, the variation σ_B^2 of all operation process times processed on different resources were calculated from the Kacem et al. (2002) benchmark data set. Subsequently the *Optimatix* benchmark problem data was randomly generated within the interval $[\mu_B - \sigma_B, \mu_B + \sigma_B]$, where μ_B denotes the operation dependent data point, for example the resource-dependent default setup-time value (s_{id_i}) for operation i on resource d_i. For the sake of simplicity half of the sequence-dependent setup-times data was initialized to zero.

An analysis of algorithm performance with respect to the *Optimatix* benchmark problem is

provided in Table 6.1. The results are significantly better than those obtained earlier in the development process. The best indication that a schedule is sub-optimal is if operations have to wait on the production floor due to resources not being available. Therefore, the fact that the PSO algorithm is able to obtain a queue time of zero hours suggests that no room for improvement exists in terms of the sequencing of operations on resources. Further improvement efforts will therefore focus on minimizing the makespan and lateness/earliness functions through addressing the allocation decision.

Table 6.1: Final algorithm results.

Performance measurement	Answer obtained
Makespan	$2352h$
Lateness/earliness	$3895h$
Queue time	$0h$
Aggregated objective function	$5753h$
Penalty function	$0h$
Time to solution	$231s$

Unfortunately the data structures do not yet exist to evaluate these benchmark results against other algorithms currently in use by *Optimatix*, but this data set provides a solid basis for the extension of the algorithm to address larger and more complex problems.

6.4 Conclusion

This chapter was instrumental in showing that Particle Swarm Optimization (PSO) can definitely add value in the production scheduling environment. Obviously room for improvement does exist but the results that have been obtained thus far are very promising.

Chapter 7

Final remarks

7.1 Results and value added

This project achieved what it set out to do: to investigate the optimization potential of Particle Swarm Optimization (PSO) in complex job shop environments. A complex variation on the basic Job Shop Scheduling Problem (JSSP), now known as the *Optimatix* problem, has been analyzed extensively in terms of its position in existing production scheduling structures. This analysis resulted in the single most important contribution of this project: the development of a production scheduling framework generic to all common metaheuristics. Useful in solving a large number of variations on the Job Shop Scheduling Problem (JSSP), the framework also has important implications for evaluation of alternative metaheuristic-based strategies.

Important work was also done in the field of constrained and multiobjective optimization. Not many algorithms exist in the field of Computation Intelligence which are able to simultaneously address both these important aspects.

A number of important conclusions regarding the implementation of PSO to complex optimization problems and more specifically to production scheduling problems, was reached during the development process.

In problems where feasible solutions can not be obtained easily, the use of initialization algorithms or semi-feasible initialization procedures is of critical importance. Throughout the optimization process, a penalty function is required to allow particles to move through infeasible regions of the search space to obtain better solutions.

The uniform initialization of particles throughout the search space is absolutely critical for the implementation of PSO-based algorithms. This results in especially rules-based algorithms not always being suitable for initializing complex problems.

Finally, the judicial selection and incorporation of problem constraints have to be considered

very carefully during algorithm development to ensure acceptable performance.

7.2 Future research opportunities

The first thing that comes to mind, in terms of future research, is the implementation of alternative metaheuristics on the generic production scheduling framework. Tabu Search (TS) is commonly considered to be the best suited to complex production scheduling problems and could also prove to be less computationally expensive.

In terms of the actual problem formulation, it could potentially be valuable to determine how the structure of the problem could be exploited for optimization purposes. This project followed a more generic variable-based approach, but considering optimization in terms of, for example, swopping operations on the same resource to move from one iteration to the next could potentially add value and should be considered in the future.

Another option would be to adapt the disjunctive graph formulation to take production down time, scheduled maintenance and machine breakdowns into account. A binary optimization algorithm would be sufficient to solve this problem and the tradeoff between shorter execution time and computational complexity would definitely make for interesting research.

More specifically, a number of alternatives exist to address the structural requirements of the scheduling framework. Mostly considered to be too complex for the scope of this project they are prime candidates for future research in the field of Swarm Intelligence and production scheduling.

A more complex approach to algorithm discretization exists in the form of derivation of arithmetic operators. The standard operators, for example addition and multiplication, which forms the basis of PSO, is redefined to facilitate the solution process of various constraint satisfaction problems, amongst which the Travelling Salesperson Problem (TSP).

Another approach consists of addressing particle representations consisting of both discreet and continuous variables, by means of a Particle Swarm Optimization-Genetic Algorithm (PSO-GA) hybrid. Genetic Algorithms (GA)s are inherently discreet optimization techniques and could potentially add value where the continuous-valued PSO could not.

In terms of addressing constraints, Pareto-ranking methods uses concepts of MOO to rank solutions based on their degree of violation. These methods, along with conversion of the constrained problem into an unconstrained problem by means of La Grange multipliers was not investigated in this project due to the added complexity.

Multiobjective optimization is one of the fastest growing subject areas in the field of optimization. Dominance-based methods are the most complicated but also the most powerful class

of PSO MOO algorithms. Utilizing complex archiving mechanisms to store all non-dominated solutions, the distance to the true Pareto front can be minimized while sufficient diversification of the Pareto front is maintained. Due to the severe complexity and computational expensiveness of this class of algorithms, they were also not considered to fall within the scope of this project.

The production scheduling environment is notorious for its dynamic nature. There exists numerous approaches to incorporating stability between successive schedule generations. These range from the inclusion of an additional objective function to the use of Dynamic Job Shop Scheduling, where rescheduling is performed subject to certain predefined criteria. One of PSO's most important claims to fame is its tracking and optimizing ability of dynamic systems. Research showed that PSO converges faster and obtains a higher fitness value than any of the other evolutionary algorithms which were tested (Eberhart and Shi, 2001).

7.3 Conclusion

The concept of social learning and adaptation is responsible for a large number of natural phenomena ranging from the multiplication bacteria to the complex schooling behaviour of fish. This project discussed the significant potential for implementation of Particle Swarm Optimization (PSO) to production scheduling. In short, this project is all about the application of Particle Swarm Optimization — a simple algorithm with big implications.

Bibliography

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

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

Brucker, P. (2004). *Scheduling Algorithms*. Springer, 4$^{\text{th}}$ edition.

Dauzre-Prs, S. and Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70:281–306.

Eberhart, R. and Shi, Y. (2001). Particle swarm optimization: developments, applications and resources. *IEEE*, pages 81–86.

Engelbrecht, A. (2005). *Fundamentals of computational swarm intelligence*. Wiley.

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

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

Graham, R., Lawler, E., Lenstra, J., and Rinnooy Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326.

Graves, S. C. (1981). A review of production scheduling. *Operations Research*, 29(4):646–675.

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

Hsu, T., Dupas, R., Jolly, D., and Goncalves, G. (2002). Evaluation of mutation heuristics for the solving of multiobjective flexible job shop by an evolutionary algorithm. *IEEE*.

Ivens, P. and Lambrecht, M. (1996). Extending the shifting bottleneck procedure to real-life applications. *European Journal of Operational Research*, 90:252–268.

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

Jerald, J., Asokan, P., Prabaharan, G., and Saravanan, R. (2004). Scheduling optimisation of flexible manufacturing systems using particle swarm optimisation algorithm. *Springer-Verlag*, pages 964–971.

Kacem, I., Hammadi, S., and Borne, P. (2002). 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).

Kennedy, J., Eberhart, R., and Shi, Y. (2001). *Swarm Intelligence*. Morgan Kaufmann Publishers.

Lian, Z., Gu, X., and Jiao, B. (2006). A similar particle swarm optimization algorithm for permutation flowshop scheduling to minmize makespan. *Applied Mathematics and Computation*, 175(1):773–785.

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

Mesghouni, K., Hammadi, S., and Borne, P. (1997). Evolution programs for job-shop scheduling. *IEEE*, pages 720–725.

Moutton, J. (2001). *Succeeding in your Master's and Doctoral Studies A South African Guide and Resource Book*. Van Schaik Publishers, Pretoria, South Africa.

Noivo, J. and Ramalhinho-Loureno, H. (2006). Solving two production scheduling problems with sequence-dependent set-up times. Available online from `http://www.econ.upf.es/deenome/what/wpapers/postscripts/338.pdf`.

Pan, J. and Chen, J. (2005). Mixed binary integer programming formulations for the reentrant job shop scheduling problem. *Computers and Operations Research*, 32:1197–1212.

Patel, V., ElMaraghy, H., and Ben-Abdallah, I. (1999). Scheduling in dual-resource constrained manufacturing systems using genetic algorithms. *Emerging technologies and factory automation*, 2:1131–1139.

Rardin, R. L. (1998). *Optimization in Operations Research*. Prentice Hall.

Tamaki, H., Ono, T., Murao, H., and Kitamura, S. (2001). Modelling and genetic solution of a class of flexible job shop scheduling problems. *Proceedings of the IEEE international conference on emerging technologies and factory automation*, 2:343–350.

Tay, J. and Wibowo, D. (2004). An effective chromosome representation for evolving flexible job shop schedules. *GECCO*, pages 210–221.

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

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

Xia Weijun, Wu Zhiming, Z. W. and Genke, Y. (2004). A new hybrid optimization algorithm for the job-shop scheduling problem. *Proceeding of the 2004 American Control Conference*, pages 5552–5557.

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

Zandieh, M., Ghomi, S., and Husseini, S. (2006). An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times. *Applied Mathematics and Computation*. Forthcoming.

Zhang, C., Li, P., Guan, Z., and Rao, Y. (2006). A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers and Operations Research*. Forthcoming.

Zhang, H., Li, X., Li, H., and Huang, F. (2004). Particle swarm optimization-based schemes for resource-constrained project scheduling. *Automation in Construction*, 14(2005):393–404.