Ann Oper Res
DOI 10.1007/s10479-008-0501-4

Metaheuristics for the multi-objective FJSP with sequence-dependent set-up times, auxiliary resources and machine down time

Jacomine Grobler · Andries P. Engelbrecht · Schalk Kok · Sarma Yadavalli

Abstract This paper investigates the application of particle swarm optimization (PSO) to the multi-objective flexible job shop scheduling problem with sequence-dependent set-up times, auxiliary resources and machine down time. To achieve this goal, alternative particle representations and problem mapping mechanisms were implemented within the PSO paradigm. This resulted in the development of four PSO-based heuristics. Benchmarking on real customer data indicated that using the priority-based representation resulted in a significant performance improvement over the existing rule-based algorithms commonly used to solve this problem. Additional investigation into algorithm scalability led to the development of a priority-based differential evolution algorithm. Apart from the academic significance of the paper, the benefit of an improved production schedule can be generalized to include cost reduction, customer satisfaction, improved profitability, and overall competitive advantage.

Keywords Particle swarm optimization · Differential evolution · Flexible job shop scheduling · Production scheduling

1 Introduction

Production scheduling plays an important role in the 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.

Optimatix is a South African-based company which specializes in providing customized software solutions. Recently, changing customer demands have led to an investigation into the use of more sophisticated production scheduling strategies. In addition to being a necessity for increasing the competitive advantage of *Optimatix*, research into improved production scheduling for complex job shop environments also have significant implications

J. Grobler (✉)
Department of Industrial and Systems Engineering, University of Pretoria, Pretoria 0002, South Africa
e-mail: jacomine.grobler@gmail.com

Published online: 10 January 2009

for production research in general. Hoitomt et al. (1993) justify the development of production scheduling algorithms geared for the job shop environment by the positive impact that improved production scheduling can have on production related problems, for example low machine utilization and excessive work in process. Addressing these problems through improved scheduling can have a significant impact on cost reduction, customer satisfaction, profitability and overall competitive advantage.

Scheduling problems in the low-volume-high-variety manufacturing environment have already received considerable attention in the Operations Research literature (Jain and Meeran 1999). The deterministic job shop scheduling problem has developed a reputation of being notoriously difficult to solve. However, the business requirements of *Optimatix* requires addressing a much more complex variation of this problem, namely the flexible job shop scheduling problem with additional constraints. Both the sequencing and allocation of operations to resources, where each operation represents a production process through which the parts to be manufactured have to be routed, are issues. The operations are categorized into jobs for which due dates are defined. Each operation can be performed on any machine from a set of primary resources. Tools and labor may be required and can be selected from a set of auxiliary resources. The processing time of an operation includes sequence-dependent set-up times and is dependent on the resource on which it is produced. The actual production time for each operation may also be affected by scheduled maintenance, machine breakdowns or production calendars. Finally, *Optimatix* is interested in simultaneously minimizing makespan, earliness/tardiness and queue time.

Since the proposed problem can be considered a direct derivation from the classical job shop scheduling problem, it is also classified as *NP*-hard and sufficient evidence exists to suggest that it cannot be solved optimally within a reasonable amount of computation time (Jain and Meeran 1999). Subsequently, particle swarm optimization (PSO) (Kennedy and Eberhart 1995) has been identified as the solution strategy of choice for a number of reasons. Firstly, placing emphasis on the concept of social versus individual learning, PSO is a robust algorithm which compares favorably with genetic algorithms (Kennedy et al. 2001) and tabu search, which are often utilized to solve the job shop scheduling problem (Jain and Meeran 1999). Secondly, PSO is one of the simplest optimization algorithms to implement. This inherent simplicity simplifies the structure and enhances the user-friendliness of the algorithm (Lian et al. 2006b).

Metaheuristic-based scheduling algorithms have to operate effectively within two separate search spaces. The purpose of this paper is to focus explicitly on alternative mapping mechanisms between these two search spaces, namely the particle space, $\boldsymbol{P}$, and the schedule allocation space, $\boldsymbol{S}$, as implemented within the PSO paradigm. Additionally, different particle representations from scheduling literature are also investigated. It is well known that these two elements have a significant impact on the performance of optimization algorithms (Norman and Bean 1999), thus justifying research into alternatives for the *Optimatix* problem.

As a direct result of this investigation, four PSO-based heuristics (the penalty-based PSO (Pen-PSO), the priority-based PSO (P-PSO), the random keys PSO (RKPSO) and the rule-based PSO (RBPSO)) were developed. When both computational time and accuracy were considered, the P-PSO algorithm was shown to outperform the other PSO-based heuristics. Benchmarking against currently used algorithms (the basic scheduling and earliest due date rules) and Norman and Bean's 1999 random keys genetic algorithm (RKGA) on real customer data showed that the priority-based representation could outperform all other tested algorithms over all three data sets. Scalability was also further investigated and by implementing a priority-based differential evolution algorithm, the best-performing priority-based algorithms were determined for different problem sizes.

This paper is considered significant because the authors are not aware of any other attempts in literature to solve the multi-objective FJSP with sequence-dependent set-up times, auxiliary resources and machine down time. The complex nature of the search space also results in the existence of a number of unique optimization challenges, the most critical of which is handling the production-specific constraints and multi-objective nature of the *Optimatix* environment.

The organization of the paper is as follows: Sect. 2 discusses previous scheduling work on which this paper builds and Sect. 3 introduces the PSO paradigm. Section 4 delves deeper into the use of PSO in scheduling before Sect. 5 introduces two alternative mapping mechanisms. Sect. 6 describes alternative particle representations. Sect. 7 presents the experimental results of the comparison between the four PSO-based heuristics, while Sect. 8 describes the benchmarking procedure in more detail. Finally, Sect. 9 concludes the paper.

2 An overview of applicable scheduling literature

Production scheduling has fascinated researchers since the 1950s and a large number of scheduling problems exist to address almost every scheduling need (Brucker 2004). This section sketches the context of the *Optimatix* problem with respect to existing literature and comments on solution strategies already employed successfully for complex job shop scheduling problems.

2.1 The problem context

Zandieh, Ghomi, and Husseini's (2006) classification of scheduling systems based on the associated resource environments is a good starting point to determine the context of the proposed problem. The models that are indicated in Fig. 1 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.

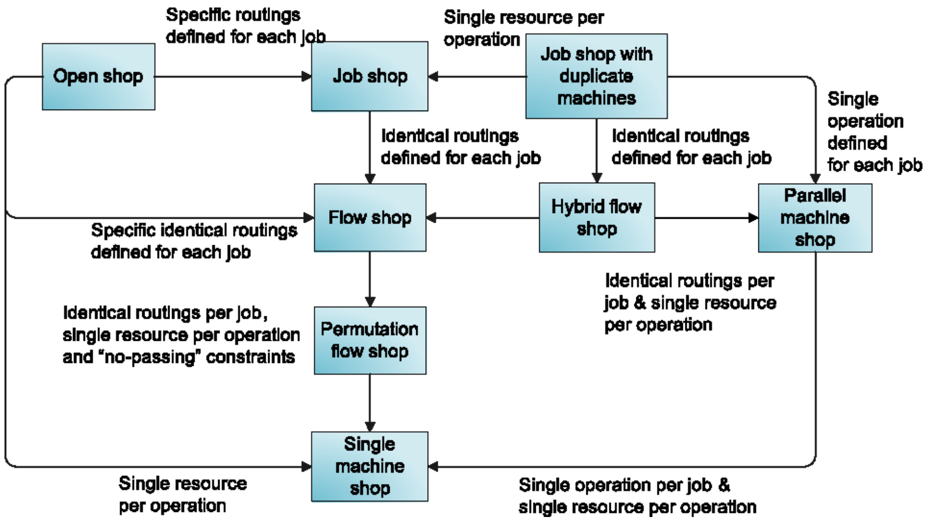

Specific routings defined for each job
Single resource per operation
Open shop
Job shop
Job shop with duplicate machines
Identical routings defined for each job
Identical routings defined for each job
Single operation defined for each job
Flow shop
Hybrid flow shop
Parallel machine shop
Specific identical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

Fig. 1 A classification of scheduling models based on their associated resource environments (Zandieh et al. 2006)

The job shop with duplicate machines problem, the parallel machine scheduling problem, and the single machine scheduling problem were identified as suitable points of departure for addressing the business requirements of *Optimatix*. By solving a problem belonging to the class of job shop scheduling problems, all three of the identified problem instances can be addressed by means of judicial selection of the input parameters.

The classical job shop scheduling problem (JSSP) is one of the most well-known production scheduling problems. Due to its extreme intractability, it has been used extensively to test the performance of a wide range of solution strategies, ranging from neural networks to mixed integer linear programming. Extensive reviews have been performed by Blazewicz et al. (1996) and Jain and Meeran (1999).

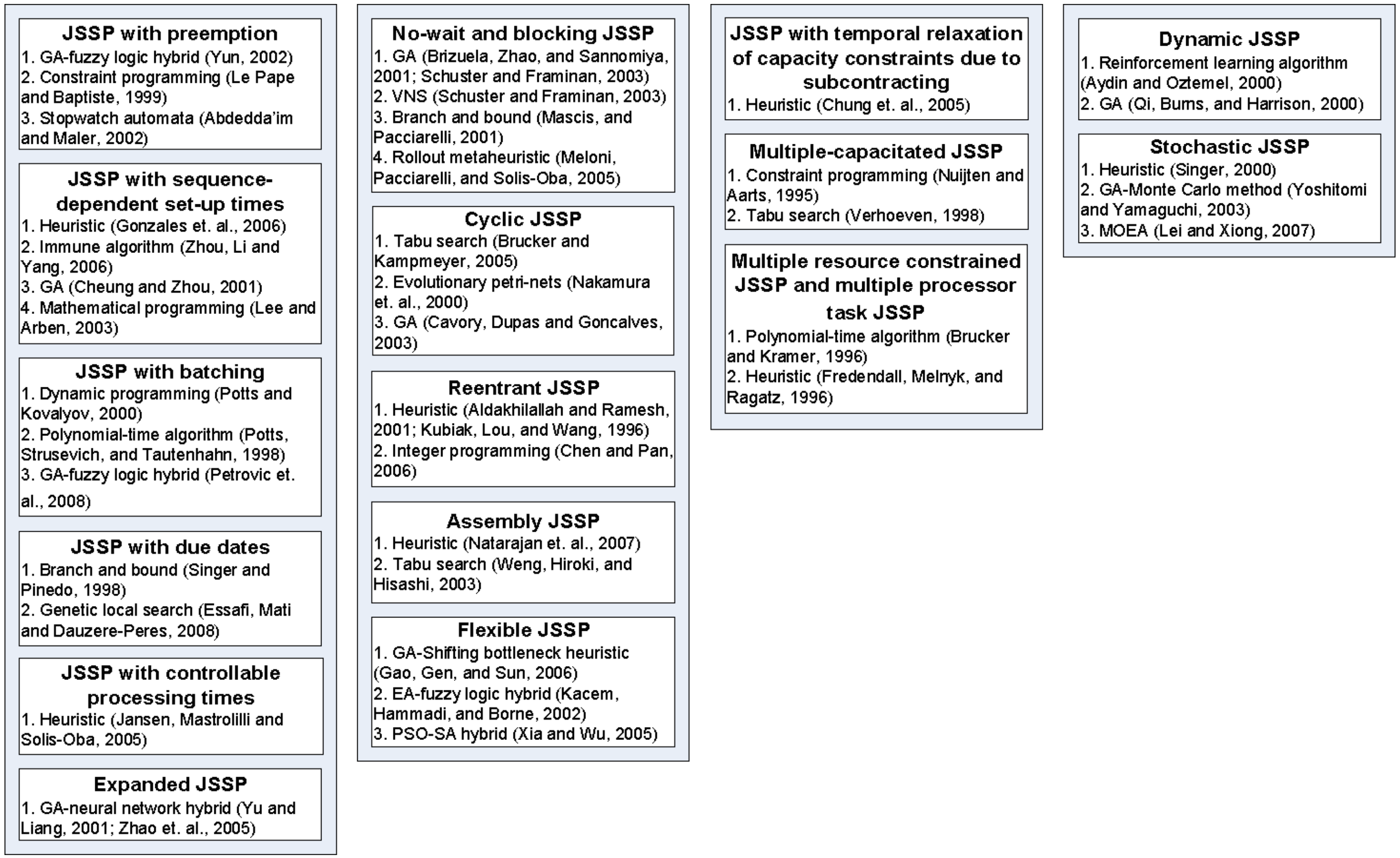
It is important, however, to note that a number of specific problem assumptions need to be met before a production scheduling problem can be modeled as a classical JSSP. For example only one operation may be processed on a single resource at a time. So even though the *Optimatix* problem belongs to the general class of job shop scheduling problems, a number of extensions need to be made to ensure that the problem-specific scheduling requirements can be addressed effectively.

Fortunately, a number of researchers have already made extensions to the classical JSSP to allow for the more realistic modeling of complex scheduling environments. The purpose of Fig. 2 is to provide the reader with a glimpse into some of the most common variations on the classical JSSP. For each variation, just two or three of the solution strategies which have already been effectively employed to solve the specific problem, are mentioned. The variations are further organized into four groups depending on the implications of the extension.

Variations which affect the length of the processing time of operations include the preemptive JSSP, where operations may be interrupted and continued at a later stage (Abdedda'im and Maler 2002; Le Pape and Baptiste 1999; and Yun 2002) and the JSSP with sequence-dependent set-up times, where the set-up time of at least one operation is dependent on the operation which was processed immediately before the operation in question, on the same resource (Cheung and Zhou 2001; Gonzalez et al. 2006; and Zhou et al. 2006). The JSSP with controllable processing times allow the assignment of different processing times to operations, where a pre-determined cost is associated with each process time-reduction alternative (Jansen et al. 2005). Processing times in the JSSP with batching is determined by the number of similar products which are produced simultaneously as part of the same operation (Petrovic et al. 2005; Potts et al. 1998; and Potts and Kovalyov 2000). Minimizing a function of job due dates, as in the JSSP with due dates, indirectly places restrictions on the processing times of jobs (Essafi et al. 2008 and Singer and Pinedo 1998). Operation starting and completion times in an expanded JSSP are restricted by release dates, due dates and technological enabling constraints (Yu and Liang 2001 and Zhao et al. 2005).

The second group of variations affect the flow of jobs on the shop floor. The no-wait and blocking JSSP consists of a JSSP where at least two operations are constrained such that the second operation has to start immediately upon completion of the first operation (Brizuela et al. 2001; Mascis and Pacciarelli 2002; Meloni et al. 2004; and Schuster and Framinan 2003). The reentrant JSSP allows the processing of two operations from the same job on the same machine (Aldakhilallah and Ramesh 2001 and Chen and Pan 2006) while the cyclic job shop consists of operations which are repeated in a cyclic fashion (Brucker and Kampmeyer 2005; Cavory et al. 2005; and Nakamura et al. 2000). When two or more jobs need to be completed before a third may be scheduled, the system can be considered as an example of an assembly JSSP (Natarajan et al. 2007 and Weng et al. 2003).

In terms of changes made to the resource requirements of operations, the multiple-capacitated JSSP (Nuijtne and Aarts 1996 and Verhoeven 1998) and JSSP with temporal relaxation of capacity constraints due to subcontracting (Chung et al. 2005), allow the



JSSP with preemption
1. GA-fuzzy logic hybrid (Yun, 2002)
2. Constraint programming (Le Pape and Baptiste, 1999)
3. Stopwatch automata (Abdedda'im and Maler, 2002)

JSSP with sequence-dependent set-up times
1. Heuristic (Gonzales et. al., 2006)
2. Immune algorithm (Zhou, Li and Yang, 2006)
3. GA (Cheung and Zhou, 2001)
4. Mathematical programming (Lee and Arben, 2003)

JSSP with batching
1. Dynamic programming (Potts and Kovalyov, 2000)
2. Polynomial-time algorithm (Potts, Strusevich, and Tautenhahn, 1998)
3. GA-fuzzy logic hybrid (Petrovic et. al., 2008)

JSSP with due dates
1. Branch and bound (Singer and Pinedo, 1998)
2. Genetic local search (Essafi, Mati and Dauzere-Peres, 2008)

JSSP with controllable processing times
1. Heuristic (Jansen, Mastrolilli and Solis-Oba, 2005)

Expanded JSSP
1. GA-neural network hybrid (Yu and Liang, 2001; Zhao et. al., 2005)

No-wait and blocking JSSP
1. GA (Brizuela, Zhao, and Sannomiya, 2001; Schuster and Framinan, 2003)
2. VNS (Schuster and Framinan, 2003)
3. Branch and bound (Mascis, and Pacciarelli, 2001)
4. Rollout metaheuristic (Meloni, Pacciarelli, and Solis-Oba, 2005)

Cyclic JSSP
1. Tabu search (Brucker and Kampmeyer, 2005)
2. Evolutionary petri-nets (Nakamura et. al., 2000)
3. GA (Cavory, Dupas and Goncalves, 2003)

Reentrant JSSP
1. Heuristic (Aldakhilallah and Ramesh, 2001; Kubiak, Lou, and Wang, 1996)
2. Integer programming (Chen and Pan, 2006)

Assembly JSSP
1. Heuristic (Natarajan et. al., 2007)
2. Tabu search (Weng, Hiroki, and Hisashi, 2003)

Flexible JSSP
1. GA-Shifting bottleneck heuristic (Gao, Gen, and Sun, 2006)
2. EA-fuzzy logic hybrid (Kacem, Hammadi, and Borne, 2002)
3. PSO-SA hybrid (Xia and Wu, 2005)

JSSP with temporal relaxation of capacity constraints due to subcontracting
1. Heuristic (Chung et. al., 2005)

Multiple-capacitated JSSP
1. Constraint programming (Nuijten and Aarts, 1995)
2. Tabu search (Verhoeven, 1998)

Multiple resource constrained JSSP and multiple processor task JSSP
1. Polynomial-time algorithm (Brucker and Kramer, 1996)
2. Heuristic (Fredendall, Melnyk, and Ragatz, 1996)

Dynamic JSSP
1. Reinforcement learning algorithm (Aydin and Oztemel, 2000)
2. GA (Qi, Bums, and Harrison, 2000)

Stochastic JSSP
1. Heuristic (Singer, 2000)
2. GA-Monte Carlo method (Yoshitomi and Yamaguchi, 2003)
3. MOEA (Lei and Xiong, 2007)

Fig. 2 A classification of common variations on the classical JSSP

processing of more than one operation at a time on a single resource. The multiple resource constrained JSSP or multiple processor task JSSP, in turn, requires that an operation be processed simultaneously on two or more resources (Brucker and Kramer 1996 and Fredendall et al. 1996). Finally, the flexible job shop scheduling problem (FJSP) incorporates the allocation of operations to resources into the optimization process (Gao et al. 2006; Kacem et al. 2002a; and Xia and Wu 2005).

The dynamic JSSP and stochastic JSSP belong to the final group since both of these variations allow the scheduler to take into account a certain amount of uncertainty in the scheduling process. The dynamic JSSP incorporates uncertainty with respect to the number of jobs and the release dates associated with the jobs which are to be scheduled (Aydin and Oztemel 2000 and Qi et al. 2000), while the stochastic JSSP focuses on incorporating uncertainty into the process time estimates (Lei and Xiong 2007; Singer 2000; and Yoshitomi and Yamaguchi 2003).

From the classification in Fig. 2, a combination of variations can be selected which may be used as points of departure for addressing the *Optimatix* problem. Careful consideration and discussion with *Optimatix* resulted in the JSSP with sequence-dependent set-up times, the multiple resource constrained JSSP, the flexible JSSP, and the expanded JSSP being selected. In addition, a more accurate portrayal of the *Optimatix* environment could be obtained by including resumable nonavailability intervals. Although the authors are not aware of any previous work where nonavailability intervals are considered in a job shop environment, Lee (2004) refers to a number of applications in the one machine, parallel machine and flow shop environments.

Combining a number of JSSP variations to meet specific scheduling requirements is not, however, new. In fact, as scheduling models have become more and more complex, this practice is quite common. Hoitomt et al. (1993) solved a JSSP with a number of additional constraints by means of an augmented Lagrangian formulation. Bertel and Billaut (2004) developed both a greedy algorithm and a genetic algorithm for a hybrid flow shop scheduling problem with re-entrance and release dates. Hwang and Sun (1997) used a dynamic programming formulation for a re-entrant JSSP with sequence-dependent set-up times.

However, incorporation of auxiliary resources along with a relatively large number of additional constraints and problem features, as is the case with the proposed problem, is not commonly found in literature. One notable exception is the work done by Norman and Bean (1999) in the application of a random keys genetic algorithm to a complex production problem, which is in many respects similar to the problem faced by *Optimatix*. Multiple machines, ready times, sequence-dependent set-up times, machine down time and scarce tools are addressed. However, only one objective (total tardiness) is considered, where the *Optimatix* problem requires the simultaneous minimization of three objective functions.

It is important to note that most of the existing algorithms mentioned in Fig. 2 specializes in solving only the specific variation for which it was developed. This paper follows a similar strategy and the algorithms developed should subsequently be considered as specialized algorithms for the multi-objective FJSP with sequence-dependent set-up times, auxiliary resources and machine down time.

3 Particle swarm optimization

PSO can be classified as a stochastic population-based optimization technique (Kennedy and Eberhart 1995), which was developed as a model of the flocking behavior of birds. Since its development, the algorithm has established itself as a competitive solution strategy for a wide range of real-world problems. This section briefly discusses the basics of the algorithm and provides details regarding the specific variation on the standard *gbest* algorithm which was implemented.

3.1 Introductory concepts

In the *gbest* PSO algorithm each potential problem solution is represented by the position of a particle in multi-dimensional hyperspace. Throughout the optimization process velocity and displacement updates are applied to each particle to move it to a different position and therefore a different solution in the search space. The velocity of particle i in dimension j at time $t+1$ is given by

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[Y_j(t) - x_{ij}(t)] \tag{1}$$

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

The displacement of particle i at time t is defined as

$$\boldsymbol{x}_i(t+1) = \boldsymbol{x}_i(t) + \boldsymbol{v}_i(t+1). \tag{2}$$

This simultaneous movement of particles towards their own previous best solutions (*pbest*) and the best solution found by the entire swarm (*gbest*), results in the particles converging to one or more good solutions in the search space.

3.2 The guaranteed convergence PSO algorithm

Unfortunately, the *gbest* PSO developed by Kennedy and Eberhart (1995) has the potential to stagnate on a solution which is not necessarily even a local optimum (Van den Bergh and Engelbrecht 2002). The guaranteed convergence particle swarm optimization (GCPSO) algorithm of Van den Bergh and Engelbrecht (2002) has been shown to address this problem effectively and have thus been used in this paper. This algorithm requires that different velocity and displacement updates, respectively indicated by (3) and (4), are applied to the global best particle:

$$v_{\tau j}(t+1) = -x_{\tau j}(t) + Y_j(t) + wv_{\tau j}(t) + \rho(t)(1 - 2r_j(t)), \tag{3}$$

$$x_{\tau j}(t+1) = Y_j(t) + wv_{\tau j}(t) + \rho(t)(1 - 2r_j(t)). \tag{4}$$

This forces the *gbest* particle 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. It should be noted that throughout the optimization run, only the *gbest* particle is treated differently. All other particles utilize the velocity update defined in (1).

3.3 The Von Neumann PSO architecture

In recent years, the degree of social interaction between particles in the PSO algorithm has received increasingly more attention. Engelbrecht (2005) provides a brief overview of this research. A number of alternative social network structures have been developed to explore different information exchange mechanisms between the particles within a swarm. A number of these social network structures, including the *gbest*, *lbest*, *pyramid*, *star* and Von Neumann structures, have been tested empirically by Kennedy and Mendes (2002).

It is well known in PSO literature that the *gbest* PSO algorithm converges fairly quickly (Kennedy et al. 2001). This is due to the fact that all particles are partially attracted to the best position found by the entire swarm since the beginning of the optimization run. Depending on the problem, this relatively fast loss of diversity can result in the algorithm finding a suboptimal solution within relatively few iterations.

The Von Neumann PSO organizes the particles into a lattice according to the particle indices. Each particle belongs to a neighborhood consisting of its nearest neighbors in the cubic structure. Instead of being partially attracted to *gbest*, the velocity of a particle is influenced by the best solution found thus far by the other particles in the same neighborhood. Since these neighborhoods overlap, information about good solutions is eventually propagated throughout the swarm, but at a much slower rate. This results in more diversity being maintained and subsequent slower convergence. This, in turn, is thought to significantly improve the algorithm's chances of finding a good solution.

During initial experimentation on *Optimatix* test data, it was verified that a significant performance improvement was indeed observed upon implementation of the Von Neumann social network structure. The resulting Von Neumann GCPSO algorithm was subsequently used for all variations presented in this paper.

4 A brief analysis of existing PSO-based scheduling algorithms

The number of papers where PSO is applied to scheduling have dramatically increased over the past few years. This section provides an overview of PSO machine scheduling literature before making a number of interesting observations.

4.1 General observations

Effective production scheduling requires solving a complex combinatorial and discrete optimization problem. Subsequently, two main approaches for solving scheduling problems by means of continuous optimization algorithms, such as PSO and DE, can be identified from literature (Lei 2008). Firstly, the standard velocity and displacement operators of the classical PSO algorithm may be redefined, allowing the algorithm to function in a discrete domain. Secondly, the scheduling problem may be converted into a continuous problem which can be solved easily by means of PSO.

According to Lei (2008), the redefinition of the standard PSO operators often lead to poor performance in scheduling. Although the structure of the problem can be more easily exploited, it is quite difficult to retain the trade-off between social and individual learning which is largely responsible for the success of the continuous PSO. The inherent structure of the information exchange between particles is, after all, changed. Nonetheless, discrete PSO algorithms have already been applied successfully to single machine scheduling (Anghinolfi and Paolucci 2009 and Pan et al. 2006), flow shop scheduling (Lian et al. 2006b; Pan et al. 2008; and Pan and Wang 2008) and job shop scheduling (Lian et al. 2006a).

In this paper, the second approach will be followed where the scheduling problem is converted to a continuous problem which can be solved directly by the PSO algorithm. Following this approach requires the PSO algorithm to be able to operate effectively within two separate search spaces: (1) The schedule allocation space, $\boldsymbol{S}$, consists of all the feasible schedules associated with the problem to be addressed. (2) The particle space, $\boldsymbol{P}$, consists of all the possible positions of the particles within the search space. To evaluate the fitness of a particle, the solution $\boldsymbol{x} \in \boldsymbol{P}$ must first be mapped to $\boldsymbol{y} \in \boldsymbol{S}$ before the fitness function value $\boldsymbol{F} \in \mathcal{F}$ (where $\mathcal{F}$ denotes the objective space) can be calculated. This approach has already been applied successfully to various flow shop (Liu et al. 2005, 2007a, 2008; and Tasgetiren et al. 2007), job shop (Sha and Hsu 2006; Xia et al. 2004; and Xia and Wu 2006) and multiprocessor task scheduling problems (Ercan and Fung 2007).

4.2 Addressing more complex scheduling problems

Although PSO is starting to establish itself as a solution strategy of note for simpler scheduling problems, the applications of PSO to more complex production scheduling problems are still considered to be relatively sparse. This is especially true for scheduling problems where both the sequencing of operations and the allocation of these operations to resources need to be addressed.

All the PSO scheduling applications discussed in Sect. 4.1 can be reduced effectively to the problem of finding an "optimal" sequence of operations subject to a number of problem-specific constraints. However, when the processing of operations on alternative resources can lead to a reduction in the overall processing time, for example as in the case of a flexible job shop scheduling problem, the allocation of operations to resources becomes an important part of the optimization problem. This additional complexity understandably creates a number of additional challenges for the scheduling algorithm.

The authors are only aware of five papers addressing flexible job shop scheduling problems by means of PSO:

- In Xia and Wu (2005), a simulated annealing-PSO-based hybrid solution strategy is developed. It is notable that only the allocation of operations to resources is done by means of PSO. The actual sequencing of the assigned operations is performed by a simulated annealing (SA) algorithm and the multiple objectives are addressed by combining all relevant objectives into a single weighted sum objective.
- Liu et al. (2006, 2007b) have solved the muli-objective FJSP with minimum makespan and flowtime by means of a variable neighborhood PSO algorithm, which employs a variable neighborhood-based local search mechanism to enhance the exploitation ability of the swarm. Dynamic weighted aggregation is used to simultaneously minimize the two objective functions.
- Jia et al. (2007a) have minimized makespan, total workload and maximum workload by means of a PSO algorithm employing a chaotic local search around the gbest particle. The multiple objectives are addressed by further minimizing the objective with the smallest fitness value at each function evaluation.
- Jia et al. (2007b) used a fully informed Pareto-based PSO algorithm to minimize the makespan and maximum lateness in a FJSP environment. A problem-specific mutation operator was also defined to improve the diversity of the swarm.

A number of interesting observations can be made from this brief analysis. Firstly, apart from the PSO-SA hybrid of Xia and Wu (2005), all the algorithms make use of a two-part particle representation and resource allocation is addressed by means of rounding off the continuous PSO particle dimensions to the nearest integer value representing a resource index. Secondly, the largest problem attempted by any of these PSO-based algorithms only consider the scheduling of 56 operations on at most 15 resources, which is currently considered to be a problem of "great size" in FJSP scheduling literature (Kacem et al. 2002a).

This paper, on the other hand considers a multi-objective FJSP where the scheduling of up to 256 operations on 216 resources need to be considered. Thirdly, and probably most importantly, no additional constraints are considered in the listed examples. In addition, some of the algorithms (Liu et al. 2006, 2007b; and Jia et al. 2007a) employ complex local search mechanisms and extending these algorithms to include sequence-dependent set-up times, auxiliary resources and production downtime, as is required in the proposed problem, would require that major structural changes be made to the existing algorithms.

5 Investigating effective mapping strategies

An effective mapping mechanism between $\boldsymbol{P}$ and $\boldsymbol{S}$ is so critical to the success of a scheduling algorithm that research into alternative mapping mechanisms can be easily justified. This section investigates two alternative mapping mechanisms, each with their own advantages and limitations.

For illustrative purposes, a small example problem, similar in complexity to the proposed problem, was selected. This problem requires the scheduling of four operations on four resources and can be illustrated by means of the network diagram in Fig. 3. This network diagram can be converted to various different particle representations depending on the algorithm utilized. The final solution (before the inclusion of production downtime) can be converted into a feasible schedule indicated by means of the Gantt chart in Fig. 3. For each of the algorithms described in this and the next section, an example particle representation is provided which, when decoded, results in the schedule illustrated in Fig. 3.

5.1 The penalty-based PSO algorithm

The $\boldsymbol{S}$ space of the penalty-based PSO (Pen-PSO) algorithm consists of both feasible and infeasible schedules. The algorithm determines the infeasibility of all scheduling solutions by calculating penalty values corresponding to the extent of infeasibility of each of the scheduling-specific constraints. These penalty values are summed and incorporated into a penalty function which is minimized simultaneously with the other scheduling-specific objective functions.

Each scheduling solution is represented by $2n$ dimensions, where n denotes the number of operations to be scheduled. Dimensions 1 to n is denoted by t_i (the starting time of operation i) and dimensions $n+1$ to $2n$ are used to represent the allocation of operations to resources. This is done by discretizing the search space as follows: For each operation i, the i^{th} dimension of the $\boldsymbol{P}$ space is divided into M_i intervals, where M_i denotes the number of primary resources on which operation i can be processed. Since each interval is associated with a unique integer number or resource index, dimensions $n+1$ to $2n$ of the position vector can easily be interpreted as resource allocation variables. In the example two-part particle representation provided in Fig. 4, M_1, M_2 and M_4 is equal to two, M_3 equals one and $x_i \in \{-1500, 1500\}$, where x_i denotes the i^{th} dimension of the particle representation and $i \in \{n+1, \ldots, 2n\}$.

One notable complexity of calculating the fitness function values is the inclusion of production down time. Comprising of scheduled maintenance, machine breakdowns and production calendars, this is the single most complicating factor in the proposed problem since not all resources are necessarily affected simultaneously. The production downtime intervals 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

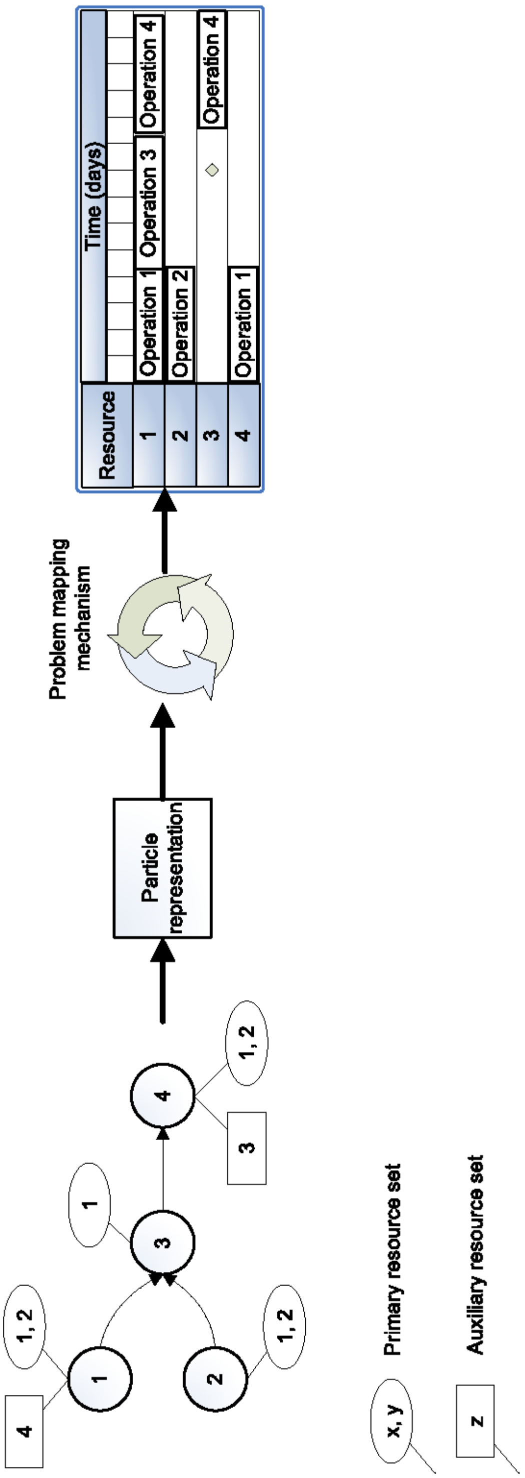


Fig. 3 A small example problem and possible solution

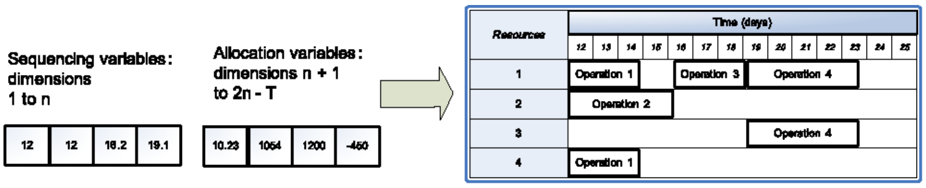


Fig. 4 An example Pen-PSO particle representation and corresponding solution to the example problem

time ignores the time intervals where the required resources are not available. If g_i is defined as the proposed finishing time, s_i is the set-up time of operation i, and d_i is the index of the primary resource on which operation i is to be scheduled, then

$$g_i = t_i + p_{id_i} + s_i, \tag{5}$$

where

$$s_i = \begin{cases} u_{ji} & \text{if } u_{ji} > 0, \\ h_{id_i} & \text{otherwise.} \end{cases} \tag{6}$$

Here p_{id_i} denotes the processing time and h_{id_i} the default set-up time of operation i on resource d_i, and u_{ji} is the sequence-dependent set-up time of operation i if processed immediately after operation j. Equation (5) allows the algorithm to make use of default set-up times for the first operation processed on each resource or when no sequence-dependent set-up time data is available.

Incorporating the down time intervals into the total processing 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, u_j, is given for each of the j production downtime intervals, the actual finishing time of operation i, denoted by f_i, can be determined by the procedure described in Algorithm 1.

In terms of the actual penalty function, there are four main types of constraints which need to be taken into account. These include, precedence constraints, release dates, the enforcement of finite capacity resource constraints, and the inclusion of auxiliary resources.

– The precedence constraints between operations and jobs can be modeled as

$$f_i \le t_j \quad \forall (i, j) \in \boldsymbol{A}, \tag{7}$$

where the set $\boldsymbol{A}$ contains all precedence relationships. For each violated precedence constraint a corresponding penalty can be calculated using

$$p_{1(i,j)} = |\min(0, (t_j - f_i))| \quad \forall (i, j) \in \boldsymbol{A}. \tag{8}$$

Summing over all precedence relationships gives the total penalty value associated with the violation of precedence relationships as

$$p_{1a} = \sum_{(i,j) \in \boldsymbol{A}} p_{1(i,j)}. \tag{9}$$

Algorithm 1: Including production downtime into the schedule.

```
1  for All operations $i$ do
2      for All downtime intervals $j$ do
3          if $f_i \leq q_j$ or $t_i \geq u_j$ then
4              Interval $j$ is an intersected downtime interval of operation $i$
5          end
6      end
7      for Intersected downtime intervals $k$ of operation $i$ to $K$ do
8          if $q_k \leq f_i$ and $t_i \leq q_k$ then
9              $f_i = f_i + u_k - q_k$
10         end
11         if $q_k < t_i$ and $t_i \leq u_k$ then
12             $f_i = f_i + u_k - t_i$
13         end
14     end
15     for All downtime intervals $j$ from $k$ to $J$ do
16         if $f_i > q_j$ then
17             $f_i = f_i + u_k - q_k$
18         else
19             Break to operation $i + 1$
20         end
21     end
22 end
```

– Release dates can be modeled by

$$t_k \geq R_k \quad \forall k \in \boldsymbol{K}, \tag{10}$$

where (10) ensures that the first operation of job k, defined in set $\boldsymbol{K}$, is only released on the production floor after the arrival of the job release date R_k. The associated penalties are calculated by

$$p_{2k} = |\min(0, (t_k - R_k))| \quad \forall k \in \boldsymbol{K} \tag{11}$$

and

$$p_{2a} = \sum_{k \in \boldsymbol{K}} p_{2k}. \tag{12}$$

– The third problem constraint ensures that no intersecting operations are allowed. 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 can be directly incorporated into the calculation of $p_{3(i,j)}$ using

$$p_{3(i,j)} = \begin{cases} f_i - t_i & \text{if } t_i \geq t_j,\ f_i < f_j \text{ and } w_{ij} = 0\ \forall (i,j) \in \boldsymbol{J}_{d_i}, \\ f_j - t_i & \text{if } t_i \geq t_j,\ f_j \leq f_i \text{ and } w_{ij} = 0\ \forall (i,j) \in \boldsymbol{J}_{d_i}, \\ f_j - t_j & \text{if } t_j > t_i,\ f_j < f_i \text{ and } w_{ij} = 0\ \forall (i,j) \in \boldsymbol{J}_{d_i}, \\ f_i - t_j & \text{if } t_j > t_i,\ f_i \leq f_j \text{ and } w_{ij} = 0\ \forall (i,j) \in \boldsymbol{J}_{d_i}, \end{cases} \tag{13}$$

where $\boldsymbol{J}_{d_i}$ consists of the set of all operations performed on resource d_i. The total penalty value can thus be calculated as

$$p_{3a} = \sum_{(i,j)\in \boldsymbol{J}_{d_i}} p_{3(i,j)}. \tag{14}$$

– The final aspect to be included in the penalty function relates to the assignment of operations to auxiliary resources. The operation processing time, which to a certain extent drives the optimization process, is independent of the auxiliary resource allocation. Therefore the auxiliary resources do not affect the objective function and can simply be incorporated as constraints. However, before the penalty values can be calculated, the algorithm attempts to obtain feasible auxiliary resource allocations for all operations. Each operation is allocated to one auxiliary resource from each set of auxiliary resources required. A feasible allocation is found if all the specified auxiliary resources are available throughout the required time period.

The allocation procedure in Algorithm 2, where n_l and m_l respectively denote the start and end of scheduled interval l, 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 (15) as the operation production times of infeasible operations,

Algorithm 2: Allocation of operations to auxiliary resources.

```
for All operations i do
    for All resource sets j do
        if A resource is required from resource set j then
            for All resources k in set j do
                for All scheduled intervals l do
                    if $f_i \leq m_l$ or $t_i \geq n_l$ then
                        Operation $i$ will overlap interval $l$
                    end
                end
                if Operation i overlaps any intervals then
                    Operation $i$ cannot be scheduled on resource $k$ of set $j$
                    if $k = K_j$ then
                        Operation $i$ is infeasible
                        Break to operation $i + 1$
                    else
                        Break to resource $k + 1$
                    end
                else
                    Schedule operation $i$ on resource $k$ of set $j$
                end
            end
        end
    end
end
```

i.e.

$$p_{4a} = \sum_i a_i (f_i - t_i) \tag{15}$$

where a_i is 1 if operation i is infeasible and 0 otherwise.

The total penalty function value, $\wp$, for each schedule can then be calculated using

$$\wp = \lambda_1 p_{1a} + \lambda_2 p_{2a} + \lambda_3 p_{3a} + \lambda_4 p_{4a}, \tag{16}$$

where λ_2 was taken to be 1000 and λ_1, λ_3 and λ_4 were each assigned a value of 1. 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.

Using a penalty function as mapping mechanism is simple and computationally inexpensive. However, the scheduling allocation space, $\boldsymbol{S}$, is massive and a large part of the optimization process is actually spent working towards a feasible solution, instead of improving an already feasible solution. However, due to the computational complexity of existing scheduling methods and the success obtained by penalty-based methods in the PSO community, investigating a penalty-based approach was deemed meaningful. Some of the computational load associated with minimizing the penalty function was reduced by initializing all particles to semi-feasible schedules. Judicial use of a local search mechanism towards the end of the optimization process was also found to be useful (Grobler et al. 2007).

5.2 The priority-based PSO algorithm

The priority-based PSO (P-PSO) algorithm utilizes a popular scheduling heuristic to decode the continuous problem representation into a feasible schedule. Priority values, which are evolved over time by the PSO algorithm, are assigned to each operation and these determine the sequence in which operations are scheduled. Although, the priority-based algorithm is computationally more complex than the Pen-PSO, the size of the search space is significantly reduced since the scheduling space consists of only feasible solutions.

The particle representation of the P-PSO consists of a $(2n - \tau)$-dimensional vector, where τ is the number of operations which may be processed on only one primary resource. The sequencing variables of dimensions 1 to n denote the priority values of each of the operations. These priorities are used as input to a schedule-building heuristic which attempts to schedule each operation at the earliest available time on its associated resource. The resource allocation variables are also addressed according to the procedure described in the previous section with reference to the Pen-PSO.

In this paper, Giffler and Thompson's (1960) heuristic (initially developed in 1960 and since then successfully used by Sha and Hsu (2006) and Gao et al. (2006)) was extended to include the unique problem characteristics of the *Optimatix* environment (Algorithm 3). Figure 5 provides an example of the procedure followed to obtain $\boldsymbol{Q}_i$, the set of possible starting times of operation i on resource d_i.

In the example indicated in Fig. 6, the priorities can be converted into the job permutation: $\{1, 4, 3, 2\}$. However, Giffler and Thompson's (1960) heuristic ensures that the precedence constraints between operations are satisfied and subsequently, the schedule in Fig. 6 is obtained.

Algorithm 3: The Priority-based PSO mapping mechanism.

```
1  Let $\boldsymbol{S}$ be the partial schedule which contains scheduled operations
2  Let $\boldsymbol{\Omega}$ be the set of schedulable operations
3  Initialize $\boldsymbol{S} = \emptyset$
4  Initialize $\boldsymbol{\Omega}$ to contain all operations without any predecessors
5  while $\boldsymbol{\Omega} \neq \emptyset$ do
6      Select $i$ from $\boldsymbol{\Omega}$ as operation with the highest priority
7      Determine $\boldsymbol{Q}_i$ (the set of possible starting times for operation $i$ on resource $d_i$)
8      while $i \in \boldsymbol{\Omega}$ do
9          Set $t_i = \min(\boldsymbol{Q}_i)$
10         Calculate $f_i$ by applying Algorithms 1 and 2
11         if $t_i = \min(\boldsymbol{Q}_i)$ results in a feasible schedule then
12             Delete $i$ from $\boldsymbol{\Omega}$
13         else
14             Delete $\min(\boldsymbol{Q}_i)$ from $\boldsymbol{Q}_i$
15         end
16     end
17     Insert $i$ into $\boldsymbol{S}$
18     for All successors $j$ of $i$ do
19         if All other predecessors of operation $j \in \boldsymbol{S}$ then
20             Insert $j$ into $\boldsymbol{\Omega}$
21         end
22     end
23 end
```

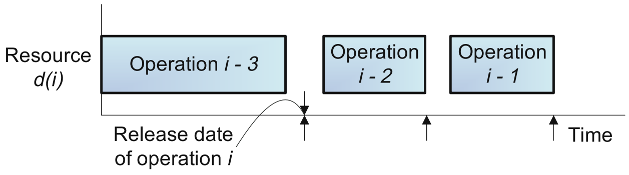


Fig. 5 When operations $i-3$, $i-2$ and $i-1$ are already scheduled on resource d_i, the possible starting times of operation i, $\boldsymbol{Q}_i$, are A, B and C

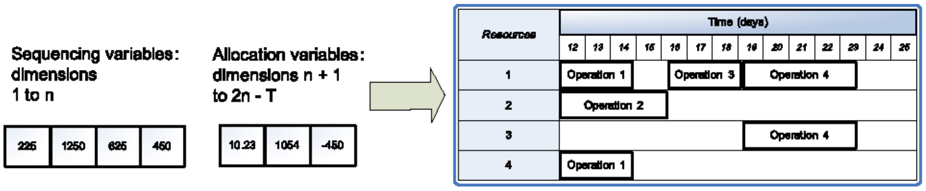


Fig. 6 An example PPSO particle representation and corresponding solution to the example problem

The P-PSO algorithm is considered useful since the continuous nature of the PSO algorithm can be exploited to solve a very complex, discrete combinatorial optimization problem (Sha and Hsu 2006). However, Sha and Hsu (2006) also mentions a concerning characteristic of scheduling algorithms which utilize a priority-based fitness function evaluation mechanism. A very small change in the position of the particle within the $\boldsymbol{P}$ space, may result in a very large change in the $\boldsymbol{S}$ space. The algorithm, additionally, has the property that many different solutions within $\boldsymbol{P}$ map to the same solution in $\boldsymbol{S}$.

The last problematic aspect of a priority-based mapping mechanism is the effect which the schedule-building heuristic has on the simultaneous optimization of the specific set of multiple objectives required by *Optimatix*. Because the schedule-building mechanism attempts to position each operation at the earliest possible time, the algorithm is, in fact biased towards the minimization of makespan. However, this statement is also heavily dependent on the characteristics of the problem being solved.

As an example consider the scheduling of a single operation on a single resource while simultaneously minimizing earliness/tardiness and makespan. If the earliest starting time is larger than the due date of the job to which the operation belongs, minimizing makespan also minimizes the earliness/tardiness objective. However, if the earliest starting time is smaller than the due date of the associated job, the two objectives become conflicting, and the solution and subsequent fitness calculation is distorted.

6 Investigating alternative particle representations

One of the most important considerations in the application of PSO (or any other metaheuristic algorithm) is the selection of an appropriate solution representation. Affecting both the overall structure and performance of the algorithm, time spent in the evaluation of alternative representations may hold significant improvement opportunities. This section presents two alternative representations to the priority-based PSO algorithm of the previous section.

6.1 The random keys PSO algorithm

The random keys PSO (RKPSO) is a direct application of Norman and Bean's (1999) random keys genetic algorithm (RKGA) in the PSO paradigm. The gene representation of the RKGA consists of an n-dimensional vector in contrast with the $2n$ and $(2n - \tau)$ dimensions required for the two variations discussed in the previous section. A sorting mechanism (which is given in Algorithm 4) is used to decode the real-valued n-dimensional vector into its corresponding resource indices and priorities. Giffler and Thompson's (1960) schedule-building heuristic (Algorithm 3) can then be applied directly. From the example in Fig. 7, it can be seen that the resource allocation decision is again addressed through discretization of the search space. However, now $x_i \in \{0 \dots 1\}$, where x_i is the i^{th} dimension of the particle representation.

The RKPSO has the important advantage that the dimensionality of the $\boldsymbol{P}$ space is halved. However, the limitations of Giffler and Thompson's (1960) heuristic, as identified in the previous section, are still applicable.

Algorithm 4: The Random Keys PSO sorting mechanism.

```
1 Let $\boldsymbol{x}$ be the position vector to be sorted
2 for All dimensions $i$ do
3     $R \leftarrow \text{rem}(|\lfloor x_i \rfloor|, X_i) + 1$
4     Set $d_i$ as the $R^{th}$ resource on which operation $i$ can be processed
5     $t_i \leftarrow x_i - |\lfloor x_i \rfloor|$
6 end
```

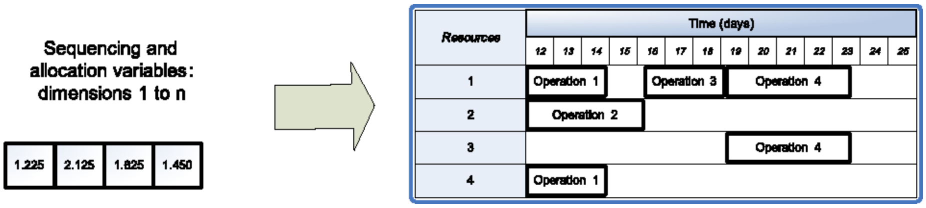


Fig. 7 An example RKPSO particle representation and corresponding solution to the example problem

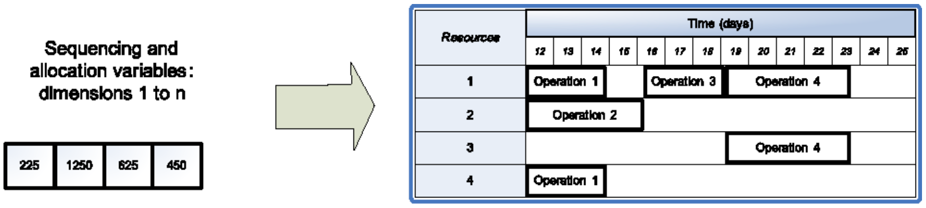


Fig. 8 An example RBPSO particle representation and corresponding solution to the example problem

Furthermore, in most traditional optimization applications one dimension is used to denote one unique and separate concept. The idea of representing two distinctly different concepts or decisions, namely the resource a specific operation should be allocated to and the priority of that operation with respect to the other operations, by one dimension cannot necessarily be applied with equal success in the context of particle swarm optimization.

6.2 The rule-based PSO algorithm

The rule-based PSO (RBPSO) algorithm is another attempt at reducing the $\boldsymbol{P}$ space. This strategy was inspired by both the rule-based algorithms currently used by *Optimatix*, as well as elements of Kacem, Hammadi, and Borne's (2002b) genetic algorithm-based approach to flexible job shop scheduling. Similar to the RKPSO, the particle representation consists of one n-dimensional vector which represents the sequencing variables. The resource allocation is performed within the schedule-building mechanism as described in Algorithm 5.

Even though the dimensionality of the $\boldsymbol{P}$ space is reduced, this is done at the cost of a more computationally complex algorithm since an explicit search of all possible resource allocations are performed for each particle during the schedule construction phase as can be seen from the example schedule obtained in Fig. 5. Due to the inclusion of sequence-dependent set-up times and production down times, actual finishing time was used as the determining factor for resource selection.

7 Comparison of the alternative PSO-based heuristics

For an investigation of algorithm performance to be most effective, it is important to conduct the experimental analysis under the same conditions under which the algorithms will eventually be used. It should be noted that this study is not only focused on improving the existing scheduling algorithms of *Optimatix*, but rather in identifying those requirements that if addressed will best meet the needs of *Optimatix*' clients and then to develop an effective solution which addresses all of these requirements. To achieve these objectives three

Algorithm 5: The Rule-based PSO mapping mechanism.

```
1  Let $\boldsymbol{S}$ be the partial schedule which contains scheduled operations
2  Let $\boldsymbol{\Omega}$ be the set of schedulable operations
3  Let $t_{id}$ be the starting time of operation $i$ on resource $d$
4  Let $f_{id}$ be the finishing time of operation $i$ on resource $d$
5  Initialize $\boldsymbol{S} = \emptyset$
6  Initialize $\boldsymbol{\Omega}$ to contain all operations without any predecessors
7  while $\boldsymbol{\Omega} \neq \emptyset$ do
8      Select $i$ from $\boldsymbol{\Omega}$ as operation with the highest priority
9      for All resources $d \in \boldsymbol{D_i}$ on which operation $i$ may be scheduled do
10         Determine $\boldsymbol{Q_i}$ (the set of possible starting times for operation $i$ on resource $d_i$)
11         while $i \in \boldsymbol{\Omega}$ do
12             Set $t_{id} = \min(\boldsymbol{Q_i})$
13             Calculate $f_{id}$ by applying Algorithms 1 and 2
14             if $t_{id} = \min(\boldsymbol{Q_i})$ results in a feasible schedule then
15                 Delete $i$ from $\boldsymbol{\Omega}$
16             else
17                 Delete $\min(\boldsymbol{Q_i})$ from $\boldsymbol{Q_i}$
18             end
19         end
20         Insert $i$ into $\boldsymbol{\Omega}$
21     end
22     $d_{min,i} \leftarrow \min_{d \in \boldsymbol{D_i}}(f_{id})$
23     Set $t_{id} = t_{id_{min,i}}$
24     Delete $i$ from $\boldsymbol{\Omega}$
25     Insert $i$ into $\boldsymbol{S}$
26     for All successors $j$ of $i$ do
27         if All other predecessors of operation $j \in \boldsymbol{S}$ then
28             Insert $j$ into $\boldsymbol{\Omega}$
29         end
30     end
31 end
```

test problems corresponding to actual problem size and complexity were derived from actual customer data and were adapted, as described in the rest of this section, to incorporate the changing customer requirement of *Optimatix*. All performance analyses were subsequently conducted on these data sets.

The effectiveness of a scheduling solution is highly dependent on the realism of the solution. In other words, it is important that the actual solution obtained corresponds to the requirements of the production environment which is to be scheduled. By effectively modeling the actual production environment the number of times rescheduling is required and the associated disruptions associated with frequent rescheduling can be drastically reduced. Furthermore, the time required to customize scheduling algorithms for each client's unique production environment can be reduced when the level of generality of a consulting firm's scheduling algorithms can be increased. To achieve these objectives, the exist-

Table 1 There are a number of parameters which have a significant effect on the performance of metaheuristics

Parameter	Value used
$\wp$	100
a	500
b	1500
n_s	27
D	3
I_{max}	200
c_1	$2.0 \longrightarrow 1.0$
c_2	$2.8 - c_1$
w	$0.8 \longrightarrow 0.4$

ing customer data sets were extended to include resource-dependent processing times and sequence-dependent set-up times.

The variation, σ_B^2, of all operation process times processed on different resources was calculated from the Kacem et al. (2002a) benchmark data set for flexible job shop scheduling problems, which address both the allocation of operations to resources and the sequencing of these operations on their associated resources. Subsequently the processing times and set-up times of the data sets used in this paper was randomly generated within the interval $[\mu_B - \sigma_B, \mu_B + \sigma_B]$, where μ_B denotes the operation-dependent data point as obtained from the original customer data set and half of the sequence-dependent set-up time data was initialized to zero. The data sets range in size from 56 to 256 operations which are to be scheduled on 216 resources and are available for comparison purposes from the corresponding author.

The results of the performance evaluation recorded in Tables 2 and 3 for the four PSO-based heuristics' results were recorded over 30 independent simulation runs. However, only 27 runs of the 256-operation problem (when solved by means of the Pen-PSO) resulted in feasible answers and subsequently only these feasible solutions were recorded.

Both accuracy and computational complexity were considered to be important performance measurements. In Tables 2 through 6, F_1 denotes makespan, F_2 the earliness/tardiness criteria, and F_3 the queue time. Goal programming, which minimizes the weighted deviation between each fitness function value and a target value set for it, was used to address the multiple objectives. The aggregated fitness function, F_4, is given as

$$F_4 = \sum_{i=1}^{3} |(F_i - G_i)| + \gamma F_i \tag{17}$$

where G_i denotes the target value of the i^{th} fitness function (F_i) and γ is selected as sufficiently small. The results in Tables 2 through 6 were obtained by selecting G_1 and G_2 as 250, and G_3 as 0. Throughout the rest of this section, μ and σ respectively denote the mean and standard deviation associated with the corresponding performance measurement.

Analysis of the behavior of the priority-based PSO on the 56-operation problem resulted in the parameter values listed in Table 1 being defined as suitable for comparison purposes. The number of particles in the swarm is denoted by n_s, $\wp$ is the weighting of the penalty function with respect to the other fitness functions, and a and b denote the interval sizes within which the decision variables are initialized. The size of the discretization intervals,

Table 2 Experimental results of alternative mapping strategies and particle representations with respect to makespan and earliness/tardiness

Problem	Algorithm	F_1		F_2	
		μ	σ	μ	σ
56-op	Pen-PSO	3972.26	234.86	11142.22	1242.85
	P-PSO	1582.37	7.76	3563.24	316.19
	RKPSO	2086.04	109.26	4060.96	353.53
	RBPSO	1567.71	0.00	3546.47	331.85
100-op	Pen-PSO	18671.19	2360.99	97363.16	19284.32
	P-PSO	1862.46	13.90	7045.34	378.46
	RKPSO	2271.42	128.58	8024.63	809.16
	RBPSO	1799.55	50.99	6431.98	369.90
256-op	Pen-PSO	24816.16	1537.58	465671.63	73818.10
	P-PSO	5059.85	153.17	39007.07	2789.33
	RKPSO	6191.17	731.57	38382.25	4102.06
	RBPSO	4922.67	62.13	30892.79	3508.45

Table 3 Experimental results of alternative mapping strategies and particle representations with respect to queue time and the aggregated objective function

Problem	Algorithm	F_3		F_4	
		μ	σ	μ	σ
56-op	Pen-PSO	3815.58	1459.38	18448.99	2548.62
	P-PSO	137.56	157.39	4788.46	390.82
	RKPSO	389.39	302.35	6042.93	609.69
	RBPSO	107.76	158.77	4727.16	429.11
100-op	Pen-PSO	52237.51	16511.37	167940.13	36632.48
	P-PSO	685.03	254.60	9102.42	574.98
	RKPSO	1376.47	452.24	11184.20	1248.05
	RBPSO	629.22	388.59	8369.60	775.02
256-op	Pen-PSO	303276.65	46100.14	794058.21	115527.70
	P-PSO	12760.31	2704.64	56384.06	5223.65
	RKPSO	10539.85	3118.69	54668.49	7143.55
	RBPSO	7504.28	2782.99	42863.06	5982.83

δ_i, of operation i is dependent on D, where

$$\delta_i = \frac{2a}{X_i D}. \tag{18}$$

X_i is the number of resources on which operation i may be scheduled, D is the number of discretization intervals which are allocated to a single operation-resource pair, and $m \longrightarrow n$ indicates that the associated parameter is decreased linearly from m to n over 95% of the total number of iterations, I_{max}.

The results of the investigation into alternative problem mapping mechanisms and particle representations are recorded in Tables 2 and 3. In all cases it is clear that the Pen-PSO algorithm performs very poorly. Penalty-based approaches have been successfully utilized to solve continuous, constrained problems, but the dual search spaces, and combinatorial and discrete nature of complex scheduling problems make this approach unsuitable for addressing the *Optimatix* scheduling problem. Although the function evaluations are relatively inexpensive in comparison to those of the P-PSO, the local search and initialization mechanisms did not aid the algorithm in finding those areas of the $\boldsymbol{S}$ space where good feasible solutions exist.

With reference to the other algorithms tested, the RBPSO outperforms the RKPSO and P-PSO in terms of accuracy for all three test instances. However, when the computational complexity and solution time of the algorithms are also considered, P-PSO is considered more suited to the *Optimatix* requirements. The solution time of the RBPSO is unfavorably large when compared to the other algorithms, whereas P-PSO is computationally much less complex while producing satisfactory results for most of the problems tested.

8 Comparison of the P-PSO algorithm against alternative solution strategies

To evaluate the significance of the PSO-based results, this section compares the best algorithm from the previous section to a number of selected benchmark algorithms. The P-PSO is, therefore, benchmarked against four other algorithms of which the first two algorithms are existing *Optimatix* algorithms currently in use. For the third algorithm Norman and Bean's (1999) RKGA was identified as the most promising algorithm identified from existing literature. Finally, the use of alternative metaheuristics within the priority-based algorithm framework was investigated through the implementation of a priority-based differential evolution (DE) algorithm (Grobler and Engelbrecht 2007). The rest of this section describes each of the benchmark algorithms in more detail before the results of the comparison are stated and discussed.

8.1 The existing *Optimatix* algorithms

Two of these currently used algorithms were selected as benchmark algorithms: the idea being to evaluate the improvements resulting from the use of the PSO-based heuristics instead of the existing algorithms. Both of these rule-based algorithms function on a very similar premise to the RBPSO discussed in the previous section, the only differences being the assignment of priorities to operations and the allocation of operations to the first available resource. While the operation priorities are evolved over time by a PSO-based algorithm in the RBPSO, the basic scheduling rule assigns priority values randomly to operations. The earliest due date (EDD) rule assigns operation priorities according to the earliest due date of the jobs corresponding to the operations under consideration. For the sake of completeness, the pseudocode for these algorithms is provided in Algorithm 6.

It should be noted that the inclusion of the more complex customer requirements defined and discussed in Sect. 7 required that the calculation of the actual processing time of operation i on resource d_i (before the inclusion of production down time) had to be updated from

$$g_i = t_i + p_i + s_i \tag{19}$$

as used within the existing *Optimatix* algorithms, to

$$g_i = t_i + p_{id_i} + s_i \tag{20}$$

Algorithm 6: The rule-based benchmarking algorithms.

```
1  Let $\boldsymbol{S}$ be the partial schedule which contains scheduled operations
2  Let $\boldsymbol{\Omega}$ be the set of schedulable operations
3  Let $t_{id}$ be the starting time of operation $i$ on resource $d$
4  Let $f_{id}$ be the finishing time of operation $i$ on resource $d$
5  Initialize $\boldsymbol{S} = \emptyset$
6  Initialize $\boldsymbol{\Omega}$ to contain all operations without any predecessors
7  while $\boldsymbol{\Omega} \neq \emptyset$ do
8      Select $\boldsymbol{I}$ from $\boldsymbol{\Omega}$ as set of operations with the highest priority
9      for All operations $i \in \boldsymbol{I}$ do
10         for All resources $d \in \boldsymbol{D_i}$ on which operation $i$ may be scheduled do
11             Determine $\boldsymbol{Q_i}$ (the set of possible starting times for operation $i$ on
               resource $d_i$)
12             while $i \in \boldsymbol{\Omega}$ do
13                 Set $t_{id} = \min(\boldsymbol{Q_i})$
14                 Calculate $f_{id}$ by applying Algorithms 1 and 2
15                 if $t_{id} = \min(\boldsymbol{Q_i})$ results in a feasible schedule then
16                     Delete $i$ from $\boldsymbol{\Omega}$
17                 else
18                     Delete $\min(\boldsymbol{Q_i})$ from $\boldsymbol{Q_i}$
19                 end
20             end
21             Insert $i$ into $\boldsymbol{\Omega}$
22         end
23         $d_{min,i} \leftarrow \min_{d \in \boldsymbol{D_i}}(t_{id})$
24     end
25     $i \leftarrow \min_{i \in \boldsymbol{I}}(d_{min,i})$
26     Set $t_{id} = t_{id_{min,i}}$
27     Delete $i$ from $\boldsymbol{\Omega}$
28     Insert $i$ into $\boldsymbol{S}$
29     for All successors $j$ of $i$ do
30         if All other predecessors of operation $j \in \boldsymbol{S}$ then
31             Insert $j$ into $\boldsymbol{\Omega}$
32         end
33     end
34 end
```

where

$$s_i = \begin{cases} u_{ji} & \text{if } u_{ji} > 0 \\ h_{id_i} & \text{otherwise,} \end{cases} \tag{21}$$

as can be seen in Algorithm 6. This results in the existing algorithms being suitable almost "as-is" for benchmarking purposes and that no additional structural changes had to be made to enable these *Optimatix* algorithms to be able to solve the proposed problem.

8.2 An existing algorithm selected from literature

When selecting a benchmark algorithm from literature it is important to select an algorithm that addresses both the sequencing of operations and their allocation to resources. By considering the classical FJSP literature a number of potential benchmark algorithms can be identified. These include the work of Kacem et al. (2002a, 2002b), Gao et al. (2006), and Xia and Wu (2005). However, these algorithms specialize in solving classical FJSPs and it is the authors' opinion that they cannot be extended easily to address more constrained problems without significantly changing the structure of the algorithms, even when sequence-dependent set-up times and resource-dependent processing times are excluded from the problem. Furthermore, most of these algorithms are hybridizations of two or more solution strategies (Kacem et al. 2002a) or employ complex local search mechanisms (Gao et al. 2006) specific to the problem being solved. This greatly increases the time to solution, while a fast solution time is considered to be an important requirement in the South African manufacturing industry. For these reasons, the random keys genetic algorithm (RKGA) of Norman and Bean (1999) was identified as a more suitable alternative for benchmarking purposes. This algorithm was developed for a more highly constrained problem and does not make use of a local search mechanism.

As the name implies, the RKGA of Norman and Bean (1999) applies a genetic algorithm to the random keys representation used for the RKPSO in the previous section. Elitism is incorporated by automatically including the α best individuals from the parent population into the new population. The exploration ability of the algorithm is improved by the use of immigration i.e. β solutions of the new population are randomly re-initialized. After the first $\alpha + \beta$ individuals of the new population have been obtained by means of elitism and immigration, each of the remaining individuals are obtained by the application of crossover and selection operators. For each individual i, where $i \in \{1, \ldots, (n_s - \alpha - \beta)\}$ random selection is used to select two individuals from the current population, namely $\boldsymbol{x}_{r_1}(t)$ and $\boldsymbol{x}_{r_2}(t)$, where $x_{r_k j}(t)$ denotes the j^{th} dimension of the k^{th} vector of individual r_k of generation t. Then, Bernoulli crossover (Norman and Bean 1999) is applied such that for all dimensions, j, if $r \sim U(0, 1) \leq p_c$,

$$c_{i_1 j}(t) = x_{r_1 j}(t), \tag{22}$$

$$c_{i_2 j}(t) = x_{r_2 j}(t). \tag{23}$$

Otherwise $c_{i_1 j}(t) = x_{r_2 j}(t)$ and $c_{i_2 j}(t) = x_{r_1 j}(t)$, where p_c is the crossover probability. The better of the two candidate solutions, $c_{i_1 j}(t)$ and $c_{i_2 j}(t)$, is subsequently incorporated into the new population.

8.3 Using an alternative metaheuristic

The priority-based DE algorithm (P-DE) is a direct application of the P-PSO to the differential evolution (DE) paradigm (Storn and Price 1997) to evaluate the performance of the particle representation and decoding mechanism when applied within the context of another metaheuristic. Similar to PSO, DE is a continuous optimization algorithm. The standard DE operators can thus be applied directly to the priority-based representation of the *Optimatix* problem.

For each individual, i, in the population three different vectors are selected from the current population, namely $\boldsymbol{x}_{r_1}(t)$, $\boldsymbol{x}_{r_2}(t)$ and $\boldsymbol{x}_{r_3}(t)$, where $x_{r_k j}(t)$ denotes the j^{th} dimension of the k^{th} vector of individual i of generation t and $i \neq r_1 \neq r_2 \neq r_3$. Then, for all dimensions, j, if $r \sim U(0,1) \leq p_c$ or $j = i \sim U(1, \ldots, J)$

$$c_{ij}(t) = x_{r_1 j}(t) + F(x_{r_2 j}(t) - x_{r_3 j}(t)). \tag{24}$$

Otherwise $c_{ij}(t) = x_{ij}(t)$, where p_c is the probability of reproduction, J is the number of dimensions and F is the scaling factor. If the fitness of $\boldsymbol{c}_i(t)$ is better than the fitness of the i^{th} individual of the original population, this individual is replaced by $\boldsymbol{c}_i(t)$ (Storn and Price 1997 and Engelbrecht 2005).

The utilization of different strategies in the selection of $\boldsymbol{x}_{r_1}(t)$ results in different variations on the standard DE which differ with respect to the exploitation and exploration ability of the population (Storn and Price 1997). The three most common variations are DE/rand/bin, where $\boldsymbol{x}_{r_1}(t)$ is selected randomly from the population, DE/best/bin, where $\boldsymbol{x}_{r_1}(t)$ is taken as the best individual in the population and DE/rand-to-best/bin, which is a combination of the two. In all three cases, binary crossover (/bin) is applied as explained in (24). An empirical investigation into the effectiveness of these three strategies for solving the *Optimatix* problem have resulted in DE/rand-to-best/bin identified as the most suitable. Thus, in this paper,

$$x_{r_1 j}(t) = x_{rj}(t) + K(t)(x_{bj}(t) - x_{rj}(t)), \tag{25}$$

where $x_{bj}(t)$ is the best individual, $x_{rj}(t)$ is selected randomly from the population and $K(t)$ decreases linearly from 1 to 0 throughout the optimization run.

8.4 Comparative analysis

The selection of suitable parameters becomes even more important when algorithms with different characteristics and structures are compared. To address this issue, a great deal of emphasis was placed on the derivation of suitable algorithm parameters. For each of the algorithms, 144 uniformly distributed parameter combinations were selected and the means and standard deviations were recorded over 30 simulation runs for each unique parameter combination. The best parameter combination for each algorithm-data set pair could subsequently be selected, as indicated in Table 4.

In terms of the results recorded in Tables 5 and 6, the RKGA and P-DE results were recorded over 30 independent simulation runs. Due to the deterministic nature of the EDD-rule, only one simulation run was needed. Finally, in order to comply with current *Optimatix* scheduling practice, the basic scheduling rule was used to construct 100 schedules of which the best was selected as the result of the simulation. To ensure consistency between all stochastic algorithms this process was, again, repeated 30 times.

As can be seen from the results, the P-PSO performs significantly better than all the rule-based heuristics on every problem instance indicating that a definite performance improvement can be attributed to the PSO-based heuristics over the existing algorithms. Furthermore, the best performing algorithm for each problem is a priority-based algorithm. The P-PSO outperforms all other algorithms for the first two problems. Here the RKGA obtains a lower fitness value, which is improved upon by the P-DE.

Due to the poor performance of the P-PSO on the 256-operation problem, a closer investigation into the scalability of the RKGA, P-DE and P-PSO was performed. Algorithm performance was investigated on two additional problems, namely a 146 operation and a 200 operation problem. The results obtained in Figs. 9 through 12 indicate that P-PSO perform the best for problems containing approximately 125 operations or less. Since the average

Table 4 Algorithm parameters as selected for benchmarking purposes

PSO		w	c_1	c_2
P-PSO	56-op	$1.1 \longrightarrow 0.72$	$1.8 \longrightarrow 0.2$	$0.2 \longrightarrow 1.8$
	100-op	$0.9 \longrightarrow 0.3$	$2.6 \longrightarrow 0.2$	$0.2 \longrightarrow 2.6$
	256-op	$1.1 \longrightarrow 0.3$	$1.4 \longrightarrow 0.2$	$0.2 \longrightarrow 1.4$
GA		α	β	p_c
RKGA	56-op	14	1	$0.5 \longrightarrow 0.1$
	100-op	10	5	$0.5 \longrightarrow 0.3$
	256-op	10	1	$0.5 \longrightarrow 0.5$
DE		p_c	F	
P-DE	56-op	$0.5 \longrightarrow 0.3$	$0.7 \longrightarrow 0.1$	–
	100-op	$0.5 \longrightarrow 0.3$	$0.7 \longrightarrow 0.5$	–
	256-op	$0.7 \longrightarrow 0.1$	$0.5 \longrightarrow 0.5$	–

Table 5 Experimental comparison of alternative solution strategies with respect to makespan and earliness/tardiness

Problem	Algorithm	F_1		F_2	
		μ	σ	μ	σ
56-op	Basic rule	2124.62	43.48	3362.32	142.64
	EDD	2071.1	–	4242.8	–
	RKGA	1652.57	59.80	3578.04	331.49
	P-PSO	1653.71	44.20	3579.17	231.81
	P-DE	1594.19	24.71	3861.50	231.19
100-op	Basic rule	2266.31	51.20	6969.04	188.56
	EDD	2381.34	–	8168.33	–
	RKGA	2317.02	160.65	9770.28	372.64
	P-PSO	1858.57	11.29	7151.72	235.12
	P-DE	2276.37	66.72	9754.20	227.96
256-op	Basic rule	6446.67	304.80	49446	1329.76
	EDD	6360	–	62380	–
	RKGA	5514.86	189.35	31444.15	1240.96
	P-PSO	5010.93	45.57	38169.64	3065.49
	P-DE	5343.74	143.92	30931.67	1018.09

number of operations which need to be scheduled by *Optimatix* is 100, the P-PSO will be the most suitable alternative most of the time. For larger problems the P-DE is the best performing algorithm of the three algorithms tested.

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

Finally, although the performance of the RKGA is very similar, though worse, than the P-DE algorithm over all problem sizes, the fact that it does not ever outperform the priority-based algorithms highlight the contribution made by the development of the priority-based

Table 6 Experimental comparison of alternative solution strategies with respect to queue time and the aggregated objective function

Problem	Algorithm	F_3		F_4	
		μ	σ	μ	σ
56-op	Basic rule	328.47	136.77	5321.22	80.04
	EDD	753.8	–	6574.8	–
	RKGA	300.84	190.52	5036.98	495.28
	P-PSO	90.64	129.21	4828.84	202.77
	P-DE	497.38	122.14	5459.02	328.29
100-op	Basic rule	323.13	155.44	9522.51	378.58
	EDD	1330.92	–	11392.47	–
	RKGA	1379.68	321.07	12980.46	724.83
	P-PSO	675.70	214.73	9195.67	367.53
	P-DE	1408.47	214.59	12952.48	418.30
256-op	Basic rule	24589.83	1304.41	80062.98	1870.93
	EDD	33090	–	101430	–
	RKGA	5475.59	873.54	41977.04	1889.94
	P-PSO	12893.70	1788.01	55630.34	4088.77
	P-DE	5143.53	811.32	40960.35	1392.98

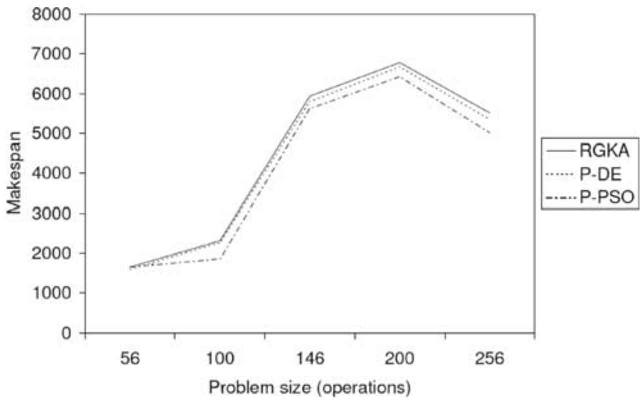


Fig. 9 Investigating algorithm scalability with respect to makespan

representation for the multi-objective FJSP with sequence-dependent set-up times, auxiliary resources, and machine downtime.

9 Conclusions and future work

This paper investigated the application of PSO to a real-world complex scheduling problem. Four PSO-based heuristics, differing in terms of particle representation and problem mapping mechanism were developed and the priority-based PSO (P-PSO) algorithm was found to be the best performing PSO algorithm when quality of solution and computational

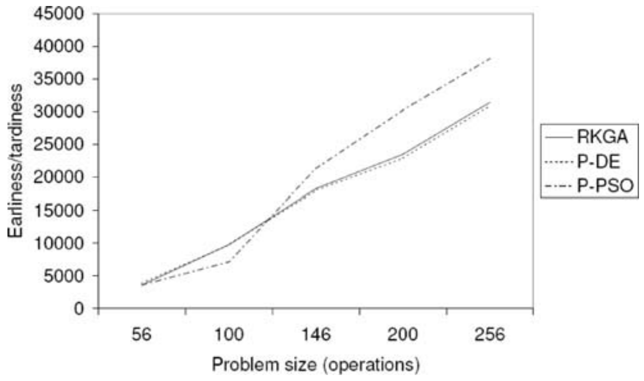


Fig. 10 Investigating algorithm scalability with respect to earliness/tardiness

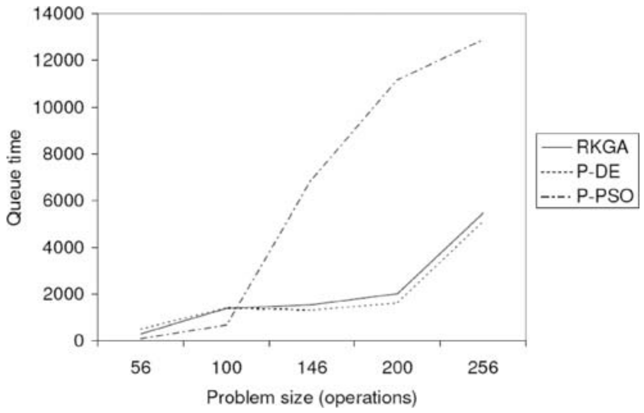


Fig. 11 Investigating algorithm scalability with respect to queue time

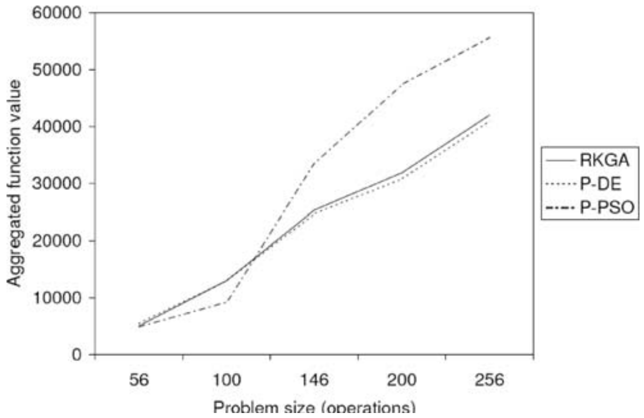


Fig. 12 Investigating algorithm scalability with respect to the aggregated objective function, F_4

complexity was considered. Further benchmarks against existing rule-based algorithms and Norman and Bean's (1999) RKGA indicated that the P-PSO algorithm outperforms all of the other benchmark algorithms when 125 operations or less are to be scheduled. The P-DE algorithm was the best performing algorithm for larger problems.

Significant future research opportunities exist in the evaluation of alternative multi-objective optimization methods and scheduling-specific updates for PSO and DE in order that the complex problem requirements can be addressed effectively. Additionally, research into improving the scalability of the P-PSO algorithm could also prove to be useful.

As technology is evolving and more advanced optimization techniques are becoming the norm, both the academic and business worlds are starting to realize the importance of addressing increasingly complex scheduling scenarios. As much as this paper attempts to make a contribution towards the application of PSO to complex scheduling scenarios, it also highlights the large number of opportunities for future research in this field.

Acknowledgements The financial assistance of the National Research Foundation towards this research is hereby acknowledged. Opinions expressed in this paper and conclusions arrived at, are those of the authors and not necessarily to be attributed to the National Research Foundation. Furthermore, the first author wishes to thank the anonymous reviewers whose comments helped improve the quality of this paper.

References

Abdedda'im, Y., & Maler, O. (2002). Pre-emptive job shop scheduling problem using stopwatch automata. In *Proceedings of the 8th international conference on tools and algorithms for the construction and analysis of systems* (pp. 113–126).

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

Anghinolfi, D., & Paolucci, M. (2009). A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, *193*(1), 73–85.

Aydin, M. E., & Oztemel, E. (2000). Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, *33*, 169–178.

Bertel, S., & Billaut, J. C. (2004). A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation. *European Journal of Operational Research*, *159*, 651–662.

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

Brizuela, C.A., Zhao, Y., & Sannomiya, N. (2001). No-wait and blocking job-shops: challenging problems for GA's. In *Proceedings of the 2001 IEEE international conference on systems, man, and cybernetics* (pp. 2349–2354).

Brucker, P. (2004). *Scheduling algorithms* (4th ed.). Berlin: Springer.

Brucker, P., & Kampmeyer, T. (2005). Tabu search algorithms for cyclic machine scheduling problems. *Journal of Scheduling*, *8*, 303–322.

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

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

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

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

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

Engelbrecht, A. P. (2005). *Fundamentals of computational swarm intelligence*. New York: Wiley.

Ercan, M. F., & Fung, Y. (2007). Performance of particle swarm optimization in scheduling hybrid flow-shops with multiprocessor tasks. In *Lecture notes in computer science* (Vol. 4707, pp. 309–318). Berlin: Springer.

Essafi, I., Mati, Y., & Dauzère-Pérès, D. (2008). A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers and Operations Research*, *35*, 2599–2616.

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

Gao, J., Gen, M., & Sun, L. (2006). A hybrid of genetic algorithm and bottleneck shifting for flexible job shop scheduling problem. In *Proceedings of the 8th annual conference on genetic and evolutionary computation* (pp. 1157–1164).

Giffler, J., & Thompson, G. L. (1960). Algorithms for solving production scheduling problems. *Operations Research*, *8*, 487–503.

Gonzalez, M. A., Vela, C. R., Sierra, M., Gonzales, I., & Varela, R. (2006). Comparing schedule generation schemes in memetic algorithms for the job shop scheduling problem with sequence dependent setup times. In *Lecture notes in artificial intelligence* (Vol. 4293, pp. 472–482). Berlin: Springer.

Grobler, J., & Engelbrecht, A. P. (2007). A scheduling-specific modeling approach for real world scheduling. In *Proceedings of the 2007 IEEE international conference on industrial engineering and engineering management* (pp. 85–89).

Grobler, J., Engelbrecht, A. P., Joubert, J. W., & Kok, S. (2007). A starting-time based-approach to production scheduling with particle swarm optimization. In *Proceedings of the 2007 IEEE symposium on computational intelligence in scheduling* (pp. 121–128).

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

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

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

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

Jia, Z., Chen, H., & Tang, J. (2007a). An improved particle swarm optimization for multi-objective flexible job-shop scheduling problem. In *Proceedings of the 2007 IEEE international conference on grey systems and intelligent services* (pp. 1587–1592).

Jia, Z., Chen, H., & Tang, J. (2007b). A new multi-objective fully-informed particle swarm algorithm for flexible job-shop scheduling problems. In *Proceedings of the 2007 international conference on computational intelligence and security workshops* (pp. 191–194).

Kacem, I., Hammadi, S., & Borne, P. (2002a). Approach by localization and multi-objective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, *32*(1), 1–13.

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

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of the IEEE international conference on neural networks* (Vol. 4, pp. 1942–1948).

Kennedy, J., & Mendes, R. (2002). Population structure and particle performance. In *Proceedings of the IEEE congress on evolutionary computation* (Vol. 2, pp. 1671–1676).

Kennedy, J., Eberhart, R. C., & Shi, Y. (2001). *Swarm intelligence*. San Mateo: Morgan Kaufmann.

Le Pape, C., & Baptiste, P. (1999). Heuristic control of a constraint-based algorithm for the preemptive job-shop scheduling problem. *Journal of Heuristics*, *5*, 305–325.

Lee, C. (2004). Machine scheduling with availability constraints. In J. Leung (Ed.), *Handbook of scheduling: algorithms, models and performance analysis* (Vol. 22). London: Chapman and Hall/CRC.

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

Lei, D., & Xiong, H. (2007). An efficient evolutionary algorithm for multi-objective stochastic job shop scheduling. In *Proceedings of the 6th international conference on machine learning cybernetics* (pp. 867–872).

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

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

Liu, B., Wang, L., & Jin, Y. (2005). Hybrid particle swarm optimization for flow shop scheduling with stochastic processing time. In *Lecture notes in artificial intelligence* (pp. 630–637). Berlin: Springer.

Liu, H., Abraham, A., Choi, O., & Moon, S. H. (2006). Variable neighborhood particle swarm optimization for multi-objective flexible job-shop scheduling problems. In *Lecture notes in computer science* (pp. 197–204). Berlin: Springer.

Liu, B., Wang, L., & Jin, Y. (2007a). An effective hybrid particle swarm optimization for no-wait flow shop scheduling. *International Journal of Advanced Manufacturing Technology*, *31*, 1001–1011.

Liu, H., Abraham, A., & Grosan, C. (2007b). A novel variable neighborhood particle swarm optimization for multi-objective flexible job-shop scheduling problems. In *Proceedings of the 2nd international conference on digital information management* (pp. 138–145).

Liu, B., Wang, L., & Jin, Y. (2008). An effective hybrid pso-based algorithm for flow shop scheduling with limited buffers. *Computers and Operations Research*, *35*, 2791–2806.

Mascis, A., & Pacciarelli, D. (2002). Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, *143*, 498–517.

Meloni, C., Pacciarelli, D., & Pranzo, M. (2004). A rollout metaheuristic for job shop scheduling problems. *Annals of Operations Research*, *131*, 215–235.

Nakamura, M., Tome, H., Hachiman, K., Ombuki, B. M., & Onaga, K. (2000). Cyclic job-shop-scheduling based on evolutionary petri nets. In *Proceedings of the 26th annual conference of the IEEE industrial electronics society* (pp. 2855–2860).

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

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

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

Pan, Q., Tasgetiren, M. F., & Liang, Y. (2006). Minimizing total earliness and tardiness penalties with a common due date on a single-machine using a discrete particle swarm optimization algorithm. In *Lecture notes in computer science* (pp. 460–467). Berlin: Springer.

Pan, Q., & Wang, L. (2008). No-idle permutation flow shop scheduling based on a hybrid discrete particle swarm optimization algorithm. *International Journal of Advanced Manufacturing Technology*, *39*(7–8), 796–807.

Pan, Q., Wang, L., Tasgetiren, M. F., & Zhao, B. H. (2008). A hybrid discrete particle swarm optimization algorithm for the no-wait flow shop scheduling problem with makespan criterion. *International Journal of Advanced Manufacturing Technology*, *38*(3–4), 337–347.

Petrovic, S., Carole, F., & Petrovic, D. (2005). Job shop scheduling with lot-sizing and batching in an uncertain real-world environment. In *Proceedings of the 2nd multidisciplinary international conference on scheduling* (pp. 363–379).

Potts, C. N., & Kovalyov, M. Y. (2000). Scheduling with batching: a review. *European Journal of Operational Research*, *120*, 228–249.

Potts, C. N., Strusevich, V. A., & Tautenhahn, T. (1998). *Scheduling batches with simultaneous job processing for two-machine shop problems* (Technical report). University of Southampton.

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

Schuster, C. J., & Framinan, J. M. (2003). Approximative procedures for no-wait job shop scheduling. *Operations Research Letters*, *31*, 308–318.

Sha, D. Y., & Hsu, C. (2006). A hybrid particle swarm optimization for job shop scheduling problem. *Computers and Industrial Engineering*, *51*, 791–808.

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

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

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

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

Van den Bergh, F., & Engelbrecht, A.P. (2002). A new locally convergent particle swarm optimiser. In *Proceedings of the IEEE international conference on systems, man and cybernetics* (Vol. 3, pp. 6–12).

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

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

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

Xia, W., & Wu, Z. (2006). A hybrid particle swarm optimization approach for the job-shop scheduling problem. *International Journal of Advanced Manufacturing Technology*, *29*, 360–366.

Xia, W., Wu, Z., & Yang, G. (2004). A new hybrid optimization algorithm for the job-shop scheduling problem. In *Proceedings of the 2004 American control conference* (pp. 5552–5557).

Yoshitomi, Y., & Yamaguchi, R. (2003). A genetic algorithm and the Monte Carlo method for stochastic job-shop scheduling. *International Transactions in Operations Research*, *10*, 577–596.

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

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

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

Zhao, F., Hong, Y., & Yu, D. (2005). A hybrid approach based on artificial neural network and genetic algorithm for job-shop scheduling problem. In *Proceedings of the 2005 international conference on neural networks and brain* (pp. 1687–1692).

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