Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CI-Sched 2007)

A starting-time-based approach to production scheduling with Particle Swarm Optimization

Jacomine Grobler, Andries P. Engelbrecht, *Member, IEEE,* Johan W. Joubert, Schalk Kok

***Abstract*— This paper provides a generic formulation for the complex scheduling problems of *Optimatix*, a South African company specializing in supply chain optimization. To address the complex requirements of the proposed problem, various additional constraints were added to the classical job shop scheduling problem. These include production downtime, scheduled maintenance, machine breakdowns, sequence-dependent set-up times, release dates and multiple predecessors per job. Differentiation between primary resources (machines) and auxiliary resources (labour, tools and jigs) were also achieved. Furthermore, this paper applies Particle Swarm Optimization (PSO), a stochastic population based optimization technique originating from the study of social behavior of birds and fish, to the proposed problem. Apart from the significance of the paper in that the proposed problem has not been addressed before, the benefit of an improved production schedule can be generalized to include cost reduction, customer satisfaction, improved profitability and overall competitive advantage.**

I. Introduction

PRODUCTION scheduling plays an important role in the current business environment. Customers increasingly expect to receive the right product, at the right price, at the right time. In order to meet these requirements, manufacturing companies need to improve their production scheduling performance.

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 solution strategies for their production scheduling module.

However, research into improved production scheduling for complex job shop environments is not only critical to the competitive success of *Optimatix*, but also holds significant implications for production research in general. Hoitomt et al. [1] 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 has already received considerable attention in Operations Research literature. The deterministic job shop scheduling problem has developed a reputation of being notoriously difficult to solve. However, the business requirements of *Optimatix* requires that a much more complex variation of this problem should be addressed. 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 [2].

PSO 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 favourably with Genetic Algorithms and Tabu Search, which are often utilized to solve the job shop scheduling problem [3]. Secondly, PSO is one of the simplest optimization algorithms to implement. This inherent simplicity simplifies the design and enhances the user-friendliness of the algorithm [4]. Thirdly, even though PSO has been the focus of many research studies since its development, it does not have a rich literature with respect to scheduling problems [4].

The purpose of this paper is to discuss a starting-time-based formulation for the scheduling problem geared towards implementation in *Tactix Scheduling*, the production scheduling module of *Optimatix*. It should be noted that although distinct variations exist between the different production environments serviced by *Optimatix*, a number of standard requirements can be identified. The formulation should be capable of addressing 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. The operations are categorized into jobs for which release dates and due dates are defined. Each operation can be performed on any machine from a set of primary resources. Tools and labour may be required and can be selected from a set of auxiliary resources. The processing time of an operation includes sequence-dependent set-up times and is dependent on the resource on which it is produced. The actual production time for each operation may also be affected by scheduled maintenance, machine breakdowns or production calendars.

Manuscript received October 31, 2006.

Jacomine Grobler is a student in Industrial Engineering at the University of Pretoria, South Africa (corresponding author to provide e-mail: jacomine@tuks.co.za).

Andries P. Engelbrecht is with the Department of Computer Science at the University of Pretoria, South Africa.

Johan W. Joubert is with the Department of Industrial and Systems Engineering at the University of Pretoria, South Africa.

Schalk Kok is with the Department of Mechanical and Aeronautical Engineering at the University of Pretoria, South Africa.

In order to address these requirements the formulation was developed and implemented within a PSO-based algorithm and tested against test data obtained from *Optimatix*. Analysis of the algorithm behaviour indicated that the best performance was obtained with a Guaranteed Convergence Particle Swarm Optimization [14] algorithm used in conjunction with a compression algorithm implemented towards the end of the optimization process.

This paper is considered significant because no reference to this specific problem instance have been found in scheduling literature. Additionally, the complex nature of the search space and the specific formulation used, result in the existence of a number of unique optimization challenges, the most critical of which is handling the production-specific constraints. Furthermore, the inclusion of a penalty function requires the judicial handling of multiple objectives.

In terms of the rest of the paper, Section II discusses the previous work on which this paper builds. Section III describes the PSO-based algorithm and Section IV discusses the experimental results. Section V compares the PSO-based algorithm with an insertion heuristic developed to facilitate algorithm evaluation. Finally, Section VI concludes the paper.

II. A BRIEF REVIEW OF APPLICABLE LITERATURE

A. The context of the problem within the scheduling literature

Production scheduling has fascinated researchers since the 1950s and a large number of scheduling problems exist to address almost every scheduling need. Zandieh et al's 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. The job shop with duplicate

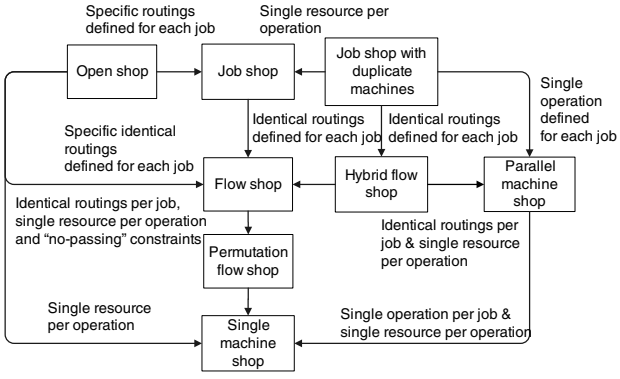

Fig. 1. Zandieh et al's classification of scheduling models based on their associated resource environments.

machines problem, the parallel machine scheduling problem and the single machine scheduling problem were identified as suitable candidates to use as points of departure for addressing the business requirements of *Optimatix*. The job shop with duplicate machines becomes an appropriate choice as all three of the identified problem instances can easily be addressed through judicial input parameter selection. However, it should be noted that Zandieh et al's classification only addresses problems on a relatively general level, and more specialized scheduling requirements necessitate a more elaborate classification.

An analysis of the variations on the most well-known production scheduling problem, the classical job shop scheduling problem (JSSP) result in three variations identified as applicable to the identified scheduling problem: the expanded job shop scheduling problem (EJSSP), the JSSP with sequence-dependent set-up times and the JSSP with precedence constraints.

The EJSSP is more general than the classical JSSP and incorporates both release dates and due dates. Job starting times are also restricted by technological planning constraints or operation enabling conditions [6]. Simply put, all cutting tools, machines and other resources required for processing an operation has to be available before the processing of the operation can start. This has positive implications for the scheduling of labour and auxiliary resources and implies that alternative resources can be specified for each operation.

Additional to the precedence constraints occurring between the various operations of each of the jobs, the JSSP with precedence constraints incorporates precedence relations between jobs — a very useful attribute in modeling assembly processes [7].

Set-up times are defined in literature as the time intervals between the completion of one operation and the start of the next operation. Set-up times, one of the most frequent additional complications in scheduling, is useful in situations where cleaning operations and tool changes play an important role in production [8].

With reference to the above discussion, it can be concluded that the scheduling problem is completely described by Zandieh's classification and the three identified JSSP variations. These problem instances provide a solid basis for the design of an algorithm capable of addressing the particular scheduling problem.

B. Popular solution strategies for complex job shop scheduling problems

Due to its extreme intractability, the basic job shop scheduling problem 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 done by [9] and [2] in this regard.

At this stage a number of more complex scheduling problems deserve special mention. Hoitomt et al. [1] solved a JSSP with a number of additional constraints by means of an augmented Lagrangian formulation. Bertel and Billaut [17] developed both a Greedy Algorithm and a Genetic Algorithm for a hybrid flow shop scheduling problem with reentrance

and release dates. Hwang and Sun [18] used a dynamic programming formulation for a reentrant 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.

C. Particle Swarm Optimization

PSO can be classified as a stochastic population-based optimization technique. Developed from the flocking behaviour of birds, 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 step $t+1$ is given by:

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

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, $y_{ij}(t)$ 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 $r_{1j}(t), r_{2j}(t) \sim U(0,1)$.

The displacement of particle i at time t is defined as:

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

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.

A number of authors have managed to solve basic JSSPs with hybrid PSO algorithms [10]. Other scheduling applications of PSO include schedule optimization in a flexible manufacturing system [11], various permutation flow shop scheduling problems (PFSSPs) [4] and a resource-constrained project scheduling problem (RCPSP) [12]. One of the most complex production scheduling applications of PSO found to date, was performed in [13], where a Simulated Annealing-PSO (SA-PSO) based hybrid solution strategy is developed for the flexible job shop scheduling problem (FJSSP). In this specific implementation, 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. Additionally, multiple objectives are addressed by combining the relevant objectives into a single weighted sum objective. It is noteworthy that although the SA-PSO algorithm improved on current best known FJSSP benchmark values, the algorithm found a 56 operation problem challenging.

III. The PSO-based algorithm

The proposed algorithm, summarized in Figure 2, uses an initialization procedure to obtain semi-feasible solutions which is stored in the form of particle representations. These particle representations are subsequently converted to their associated decision variables in order to evaluate the fitness function. The fitness function is evaluated such that three standard scheduling objectives (makespan, lateness/earliness and queue time) is minimized simultaneously with the penalty function. This information is passed through to the optimization algorithm which intelligently updates the particle representations until a stopping criterion is satisfied. The rest of this section provides more detailed descriptions of each of the elements of the algorithm.

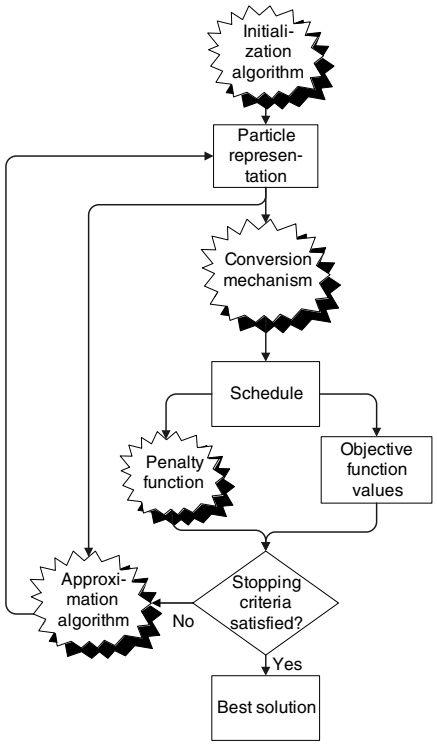

Fig. 2. The PSO-based algorithm.

A. Particle representation

The schedule representation can be loosely defined as the structure in which the optimization algorithm stores each of the scheduling solutions. With the emphasis on completeness it is imperative that both the sequencing and allocation decisions are addressed. For example, each particle is represented by $2 \times n$ dimensions, where n denotes the number of operations to be scheduled. Dimensions 1 to n addresses the allocation of operations to resources and is denoted by the decision variables x_{ij} which takes on a value of 1 if operation i is performed on resource j and is 0 otherwise. The sequencing decision is denoted by t_i

(the starting time of operation i), since the algorithm makes use of a starting-time-based formulation, and is stored in dimensions $n+1$ to $2n$.

B. Initialization

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

In contrast, allocation variables are initialized randomly and subsequently discretized. For each operation, unique intervals are defined for each resource on which the operation may be scheduled, such that a resource index (unique integer number corresponding to each primary resource) can be assigned to the operation depending on where the allocation variable is initialized.

C. Conversion from particle representation to decision variables

The process of deriving the decision variables and problem parameters from the particle representation provides an opportunity for the inclusion of production downtime. Comprising of scheduled maintenance, machine breakdowns and production calendars, this is the single most complicating factor in the proposed problem. 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 time ignores the time intervals where the required resources are not available. If g_i is defined as the proposed finishing time and s_i is the set-up time of operation i, then

$$g_i = t_i + b_{id_i} p_{id_i} + s_i \tag{3}$$

and

$$s_i = \begin{cases} c_{ji} u_{ji} & \text{if } u_{ji} > 0 \\ b_{id_i} v_{id_i} & \text{otherwise} \end{cases} \tag{4}$$

where

$$b_{id_i} = \begin{cases} 1 & \text{if operation } i \text{ is performed on resource } d_i \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$$c_{ji} = \begin{cases} 1 & \text{if operation } i \text{ is performed immediately after} \\ & \text{operation } j \text{ on resource } d_i \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where 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 (3) allows the algorithm to make use of default set-up time values for the first operation processed on each resource as well as when no sequence-dependent set-up times are defined in the problem data. Incorporating the downtime 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 downtime intervals associated with the primary and auxiliary resources on which it is to be scheduled. If the starting time (q_m) and the finishing time (u_m) is given for each of the m production downtime intervals, the actual finishing time of operation i, denoted by f_i, can be determined by the procedure described in Algorithm 1.

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

```
for All operations i do
    for All downtime intervals m do
        if f_i ≤ q_m or t_i ≥ u_m then
            Interval m is not an intersected downtime
            interval of operation i
        end
    end
    for Intersected downtime intervals k of operation i
    to K do
        if q_k ≤ f_i and t_i ≤ q_k then
            f_i = f_i + u_k − q_k
        end
        if q_k < t_i and t_i ≤ u_k then
            f_i = f_i + u_k − t_i
        end
    end
    for All downtime intervals m from k to M do
        if f_i > q_m then
            f_i = f_i + u_k − q_k
        else
            Break to operation i + 1
        end
    end
end
```

D. The penalty function

Successfully addressing the proposed problem requires the enforcement of a number of scheduling-specific problem constraints:

Precedence relationships between operations and jobs: These relationships enforce processing sequences associated with the product design. In case of violation of Equation (7), where the set A contains all precedence relationships and operation i must be completed before processing of operation j may start, a penalty corresponding to the number of time units the precedence constraint is violated can be calculated

for each operation according to Equation (8). Summing over all precedence relationships in Equation (9) gives the total penalty value associated with the violation of precedence relationships.

$$f_i \leq t_j \qquad \forall (i,j) \in A \qquad (7)$$

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

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

Release dates: Constraint (10) ensures that the first operation of job k, defined in set F, is only released on the production floor after the arrival of the job release date R_k and the associated penalties are calculated by Equation (11) and Equation (12).

$$t_k \geq R_k \qquad \forall k \in F \qquad (10)$$

$$p_{2k} = |\min(0, (t_k - R_k))| \qquad \forall k \in F \qquad (11)$$

$$p_{2a} = \sum_{k\in F} p_{2k} \qquad (12)$$

No intersecting operations: If operation i and operation j is performed on the same finite capacity primary resource, the relationship between f_i, f_j, t_i and t_j determines the value of the penalty assigned. The four mutually exclusive scenarios which can occur is incorporated into the calculation of $p_{3(i,j)}$, where J_{d_i} consists of the set of all operations performed on resource d_i and p_{3a} denotes the total penalty value corresponding to intersecting operations.

$$p_{3(i,j)} = \begin{cases} f_i - t_i & \text{if } t_i \geq t_j,\ f_i < f_j \text{ and } z_{ij} = 1 \\ & \forall (i,j) \in J_{d_i}. \\ f_j - t_i & \text{if } t_i \geq t_j,\ f_j \leq f_i \text{ and } z_{ij} = 1 \\ & \forall (i,j) \in J_{d_i}. \\ f_j - t_j & \text{if } t_j > t_i,\ f_j < f_i \text{ and } z_{ij} = 1 \\ & \forall (i,j) \in J_{d_i}. \\ f_i - t_j & \text{if } t_j > t_i,\ f_i \leq f_j \text{ and } z_{ij} = 1 \\ & \forall (i,j) \in J_{d_i}. \end{cases} \qquad (13)$$

where

$$z_{ij} = \begin{cases} 0 & \text{if } f_i \leq t_j \text{ or } f_j \leq t_i\ \forall (i,j) \in J_{d_i} \\ 1 & \text{otherwise} \end{cases} \qquad (14)$$

$$p_{3a} = \sum_{(i,j)\in J_{d_i}} p_{3(i,j)} \qquad (15)$$

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 m_l and n_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 Equation (16) as the operation production times of infeasible operations

$$p_{4a} = \sum_i a_i (f_i - t_i) \qquad (16)$$

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

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 ≤ m_l or t_i ≥ n_l then
                        Operation i will not 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
```

The total penalty function value, P, for each schedule can be calculated according to Equation (17). It should be noted that due to the total per-schedule penalty function value being calculated as the sum of the penalty values associated with each set of constraints, the penalization of the objective function is directly proportional to the extent of infeasibility.

$$P = p_{1a} + p_{2a} + p_{3a} + p_{4a} \qquad (17)$$

E. Using PSO for schedule optimization

In order to determine how PSO performs on the proposed problem, the basic PSO algorithm as described by [3] had to be adapted. To avoid premature algorithm stagnation, the Guaranteed Convergence Particle Swarm Optimization (GCPSO) algorithm was used [14]. The GCPSO algorithm applies different velocity and displacement updates, respectively indicated by Equation (18) and Equation (19), to the global best particle.

$$v_{\tau j}(t+1) = -x_{\tau j}(t) + Y_j(t) + wv_{\tau j}(t) + \rho(t)(1-2r_j(t)) \quad (18)$$

$$x_{\tau j}(t+1) = Y_j(t) + wv_{\tau j}(t) + \rho(t)(1-2r_j(t)) \quad (19)$$

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.

IV. Improving the PSO-based algorithm

In order to investigate the behaviour of the GCPSO algorithm on the problem formulation, a number of experiments were performed. To be able to come to meaningful conclusions it was necessary to define suitable conditions under which the PSO-based algorithm should be executed. Repeated execution of the algorithm with different parameter values resulted in the values listed in Table I being defined as suitable for comparison purposes. The number of particles in the swarm is denoted by n_s, δ denotes the fraction of the domain of each dimension which in turn is used to calculate V_{max}, P is the weighting of the penalty function with respect to the other fitness functions and a and b denote the interval size within which the decision variables are initialized.

TABLE I
PARAMETER VALUES USED AFTER COMPLETION OF THE PARAMETER DERIVATION STUDY

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

All experiments were performed on a test problem derived from both customer data and the *T-FJSP* benchmark problems developed by [15]. This data is available from the authors upon request. However, it should be noted that the algorithm does not converge for any of the experimental results. Tests with regards to the convergence discovered that convergence is not even obtained at 10000 iterations. The algorithm simply progresses through different stages of exploration and exploitation depending upon whether any of the particles have found a better solution. Therefore, the results obtained from the experiments performed in this and the next section (indicated in Tables II, III and IV) is not an indication of how well PSO solves the proposed problem, but rather of the best solutions which can be obtained within the time constraints provided by the client.

During initial phases of algorithm development, the best results obtained, measured in hours, was recorded in Table II. Upon closer inspection, it became clear that the schedule incorporated a large amount of slack time, where slack time is defined as the amount of queue time, which can not be attributed to the unavailability of resources. Including slack time into the schedule was an effective means for the algorithm to obtain feasible schedules with minimal effort. Although queue time was used as an objective function, the simultaneous minimization of the four objective functions ensured that not enough emphasis could be placed on this objective to completely avoid slack time.

TABLE II
RESULTS OBTAINED DURING INITIAL PHASES OF ALGORITHM DEVELOPMENT

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

A. Removing slack time from the schedule

The simplest way to remove slack time from the schedule is through the use of a compression algorithm. The compression algorithm functions similar to a local search. This algorithm determines whether any of the operations can be scheduled at an earlier time without violating the problem constraints and reschedules these operations by removing the slack time. The experiments focused on the timing of the compression algorithm and best results (Table III) were obtained when particles were allowed more time to roam throughout the search space before being pulled towards the nearest optimum by the compression algorithm.

B. Improving the optimization of multiple objectives

In the previous experiments, the multiple objectives were handled by means of goal programming. This approach assigns target values to each of the fitness functions and attempts to minimize the sum over all the fitness functions of the deviation between the actual values obtained and the set targets. The second experiment focused on the development of a more involved approach to addressing the multiple

TABLE III

THE RESULTS OBTAINED FROM THE INCLUSION OF A COMPRESSION ALGORITHM TOWARDS THE END OF THE OPTIMIZATION PROCESS

Objective function	Makespan	Earliness/Lateness	Queue time	Total function
Mean	3160	10108	7354	20142
Standard deviation	197	1198	1490	2626
Sample size	30	30	30	30
Upper bound of 5% confidence interval	3230	10536	7887	21081
Lower bound of 5% confidence interval	3089	9679	6821	19202

objectives. In order to address issues of scalability between the various fitness function coefficients, these coefficients were normalized and the aggregated fitness function was changed to the following:

$$f(\boldsymbol{t}, \boldsymbol{b}) = \max_i\{k_i\} + k_p \tag{20}$$

where k_i denotes the normalized deviation between the target and actual value obtained of fitness function i and k_p denotes the deviation between the target and actual value of the penalty function.

As the results in Table IV indicate, these improvements did not have a significant influence on algorithm performance. Depending upon which fitness function performed the poorest, this method excludes the other fitness functions from the fitness calculation. Even though the penalty function was always included within the fitness function evaluation, elimination of queue time had a definite impact on the results obtained. In a formulation where excessive queue time remains a problem, queue time should never be excluded from the optimization process. Furthermore, the percentage of feasible schedules generated decreased from close to 100% to 93.3% over 30 algorithm runs. The results for the remaining 28 algorithm executions which did produce feasible schedules are indicated in Table IV.

V. BENCHMARKING AGAINST AN ALTERNATIVE SOLUTION STRATEGY

Due to the non-existence of benchmark problems and alternative optimization algorithms for the proposed problem, an insertion heuristic was developed and used to obtain an objective comparison of algorithm performance. In the past, *Optimatix* has had significant success with rules-based heuristics applied to less complex problems.

The insertion heuristic aims to exploit the structure of the problem to a greater extent than can be done by rules-based heuristics. Starting off with a priority-based sequence of operations, the operations are scheduled by means of an iterative process which attempts to insert each operation into its best possible position in the schedule.

The results obtained (Table V), are significantly better than the PSO-based algorithm results. This suggests that there exists significant room for improvement and another experiment was subsequently conducted to investigate how the underlying concepts of the insertion heuristic could be used to obtain improved algorithm results.

A. Initializing the PSO-based algorithm with the insertion heuristic

The rationale behind this experiment was that if the PSO algorithm could start off with higher quality information at the start of the optimization process, the algorithm would be able to find better solutions more quickly. The results obtained indicated that the PSO algorithm was unable to improve on the initial solution since the *gbest* particle was never updated. This can be directly attributed to the fact that the insertion heuristic makes use of the problem's structure to generate solutions which tend to be local optima. Since the search space consists of a large number of local optima surrounded by infeasible space, it is very difficult for the PSO algorithm to improve upon this initial solution.

However, the apparent failure of this final experiment should not lead to the conclusion that PSO cannot be used to solve complex production scheduling problems. Improved integration of the insertion heuristic within the PSO-based algorithm might be all that is needed to produce significantly better results.

VI. CONCLUSION AND FUTURE WORK

This paper discussed the application of Particle Swarm Optimization to a starting-time-based formulation of a complex variation on the classical job shop scheduling problem. The best results were obtained by a GCPSO algorithm used in conjunction with a compression algorithm.

Although the PSO-based algorithm results do not compare as favourably to the insertion heuristic as the authors would have liked, the conclusion that PSO cannot be used to solve complex production scheduling problems should not be made. This paper merely addresses one formulation of the proposed problem. The use of alternative formulations will result in different challenges and opportunities than those faced with the starting-time-based formulation. Two of the most obvious alternatives to consider include a priority-based formulation and the application of a binary optimization algorithm to [16]'s adaptation of the disjunctive graph representation. Furthermore, due to the computational advantages associated with the starting-time-based formulation, future research should also investigate the application of alternative solution strategies to this formulation.

Eliciting both commercial and academic interest, the proposed problem is a prime example of a real-world optimiza-

TABLE IV

THE RESULTS OBTAINED FROM NORMALIZATION OF THE FITNESS FUNCTION COEFFICIENTS

Objective function	Makespan	Earliness/Lateness	Queue time	Total function
Mean	3136	10325	7774	20755
Standard deviation	199	1108	1693	2648
Sample size	28	28	28	28
Upper bound of 5% confidence interval	3209	10735	8401	21736
Lower bound of 5% confidence interval	3062	9914	7147	19774

TABLE V

INSERTION HEURISTIC RESULTS

Objective function	Makespan	Earliness/Lateness	Queue time
Mean	2164	4390	539
Standard deviation	63	401	310
Sample size	100	100	100
Upper bound of 5% confidence interval	2176	4469	599
Lower bound of 5% confidence interval	2152	4312	478

tion problem. Finding an effective solution strategy for this extremely intractable problem still remains a challenge.

REFERENCES

[1] D. J. Hoitomt, P. B. Luh, and K. R. Pattipati, "A practical approach to job-shop scheduling problems," *IEEE Transactions on Robotics and Automation*, vol. 9, no. 1, pp. 1–13, February 1993.

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

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

[4] Z. Lian, X. Gu, and B. Jiao, "A similar particle swarm optimization algorithm for permutation flowshop scheduling to minmize makespan," *Applied Mathematics and Computation*, vol. 175, no. 1, pp. 773–785, 2006.

[5] M. Zandieh, S. M. T. F. Ghomi, and S. M. M. Husseini, "An immune algorithm approach to hybrid flow shop scheduling with sequence-dependent setup times," *Applied Mathematics and Computation*, 2006, forthcoming.

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

[7] P. Brucker, *Scheduling Algorithms*, 4^{th} ed. Springer, 2004.

[8] A. G. Lockett and A. P. Muhlemann, "A scheduling problem involving sequence dependent changeover times," *Operations Research*, vol. 20, no. 4, pp. 895–902, 1972.

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

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

[11] J. Jerald, P. Asokan, G. Prabaharan, and R. Saravanan, "Scheduling optimisation of flexible manufacturing systems using particle swarm optimisation algorithm," *Springer-Verlag*, pp. 964–971, August 2004.

[12] H. Zhang, X. Li, H. Li, and F. Huang, "Particle swarm optimization-based schemes for resource-constrained project scheduling," *Automation in Construction*, vol. 14, no. 2005, pp. 393–404, 2004.

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

[14] A. P. Engelbrecht, *Fundamentals of computational swarm intelligence*. Wiley, 2005.

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

[16] P. Ivens and M. Lambrecht, "Extending the shifting bottleneck procedure to real-life applications," *European Journal of Operational Research*, vol. 90, pp. 252–268, February 1996.

[17] S. Bertel and J. C. Billaut, "A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation," *E*uropean Journal of Operational Research, vol. 159, pp. 651–662, 2004.

[18] H. Hwang and J. U. Sun, "Production sequencing problem with reentrant work flows and sequence-dependent set-up times," *C*omputers and Industrial Engineering, vol. 33, pp. 773–776, 1997.