Hindawi Publishing Corporation
Discrete Dynamics in Nature and Society
Volume 2015, Article ID 350308, 12 pages
http://dx.doi.org/10.1155/2015/350308

Research Article

A Study on the Enhanced Best Performance Algorithm for the Just-in-Time Scheduling Problem

Sivashan Chetty and Aderemi O. Adewumi

School of Mathematics, Statistics and Computer Science, University of KwaZulu-Natal, University Road, Westville, Private Bag X 54001, Durban 4000, South Africa

Correspondence should be addressed to Aderemi O. Adewumi; laremtj@gmail.com

Received 4 November 2014; Accepted 12 January 2015

Academic Editor: Peng Liu

Copyright © 2015 S. Chetty and A. O. Adewumi. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The Just-In-Time (JIT) scheduling problem is an important subject of study. It essentially constitutes the problem of scheduling critical business resources in an attempt to optimize given business objectives. This problem is NP-Hard in nature, hence requiring efficient solution techniques. To solve the JIT scheduling problem presented in this study, a new local search metaheuristic algorithm, namely, the enhanced Best Performance Algorithm (eBPA), is introduced. This is part of the initial study of the algorithm for scheduling problems. The current problem setting is the allocation of a large number of jobs required to be scheduled on multiple and identical machines which run in parallel. The due date of a job is characterized by a window frame of time, rather than a specific point in time. The performance of the eBPA is compared against Tabu Search (TS) and Simulated Annealing (SA). SA and TS are well-known local search metaheuristic algorithms. The results show the potential of the eBPA as a metaheuristic algorithm.

1. Introduction

Scheduling problems altogether constitute a large and important field of study. It involves the allocation of production (or operational) resources with the intent of optimizing business objectives. Business objectives may include reduced operational costs, reduced production time, increased customer satisfaction, and increased profits in optimizing production processes or service delivery. Several categorizations of scheduling problems are found in the literature [1, 2]. However, of particular interest in this study is the problem of Just-in-Time (JIT) scheduling [3, 4].

JIT scheduling, as described by Taiichi Ohno (commonly referred to as the father of JIT) is, "in a flow process, the right parts needed in assembly reach the assembly line at the time they are needed and only in the amount needed" [5]. Ohno perfected JIT principles at Toyota manufacturing plants in Japan while being vice president of manufacturing. At the time, Toyota created high quality vehicles at relatively low costs, compared to its competitors. This was in spite of the disadvantage of having a lack of natural resources in the country. The success of implementing JIT techniques in manufacturing gave Toyota a prominent position within the automobile sector.

Observing Toyota's success, many organizations on a global scale have adopted and implemented JIT techniques with relative successes. Proper implementations of JIT techniques have resulted in documentation emphasizing improved product qualities, improved service deliveries, improved customer satisfaction, improved employer and employee relations, decreased production costs, reduced levels of inventory, and increased profit turnover [6]. Organizations have further benefited in remaining competitive within industry, in offering products and/or services without negotiating quality at competitive costs. These factors constitute important business objectives, as organizations remain competitive on the basis of cost, quality, and service delivery [7].

The JIT scheduling problem is largely studied in the sectors of engineering, manufacturing, and service delivery [1]. The objective is the optimized delivery of business resources that meet demand, rather than manufacturing or supplying less or in surplus. JIT scheduling objectives are summarized as follows [8].

(1) Competitiveness: companies remain competitive in offering high quality products and services at relatively low costs in meeting demands.

(2) Efficient production processes: the objective is greater productivity in maintaining quality and minimizing costs.

(3) Improved quality of products: production of smaller quantities allow for better assessment checks. This results in improved product quality.

(4) Minimizing wastage: this will reduce costs and save time and effort in business.

(5) Reduced inventory: this will minimize investments as capital will not be held (or lost) in holding excess inventory.

(6) Efficient space utilization: fewer inventories mean more space available.

(7) Improved customer satisfaction: the on-time delivery of quality products and services at competitive rates earn customer satisfaction.

(8) Improved supplier relations: supplier relations get strengthened in having organized delivery of goods and services as required.

The JIT scheduling problems is NP-Hard [3, 9, 10]. This study investigates the JIT scheduling problem and determines solutions using metaheuristic techniques. The study proposes an enhanced Best Performance Algorithm (eBPA) which is an enhancement of our recently introduced Best Performance Algorithms (BPA) in literature [11–13]. eBPA, though similar to BPA, improves on the initial implementation of the latter to enhance its efficiency and execution time. Further details on the initial BPA algorithm can be found in Chetty and Adewumi [11, 12]. This study compares eBPA with two other standard metaheuristics to ascertain its efficiency in handling this JIT scheduling problem. The objective of this study is therefore to test the potential of eBPA algorithm in comparing its solutions against well-known local search metaheuristic algorithms for an NP-Hard problem. This study constitutes initial research on the eBPA algorithm for scheduling problems.

Previous studies on JIT scheduling problems have investigated both the single and multiple machine scenarios. Many optimization techniques have been investigated in determining solutions. They include both the exact and heuristic algorithms. Ronconi and Kawamura [14] investigated a single machine JIT scheduling problem with restrictive common due dates. The objective was the minimization of the earliness and tardiness penalties. The study proposed a Branch and Bound algorithm which used lower bounds and pruning rules in exploiting properties of the problem in determining solutions. The algorithm was investigated using 280 jobs. These jobs were characterized by different due dates. The proposed algorithm showed to be effective in outperforming the CPLEX optimization software. Monette et al. [15] studied a JIT job-shop scheduling problem. Jobs were characterized by earliness and tardiness penalties with respect to their due dates. The objective was the minimization of the earliness and tardiness penalties. The study presented a constrained programming algorithm. This was a filtering algorithm based on machine relaxation. The study investigated a large range of benchmark test instances. 72 problems were studied in total. The algorithm showed to be very effective in determining 29 of the best-known solutions from the problems studied. Dereniowski and Kubiak [16] studied a JIT multislot scheduling problem. In this problem, processing time was divided into time slots rather than a single due date for the jobs. The intent of the study was to determine a minimization of the schedule makespan. The study presented algorithms for both the single and parallel machine problem instances.

Süer et al. [17] studied a single machine scheduling problem with nonzero ready times. Jobs were assumed to have arrived at different times, with the arrival times being known in advance. The objective was determining the job sequences in minimizing tardiness. For the problem setting, preemption was not allowed. The study investigated the Genetic Algorithm (GA) and compared its solutions to known optimal solutions for small to large size problems. Results showed that GA determined optimal solutions for smaller instances and near-optimal solutions for larger instances. van Laarhoven et al. [18] investigated the Simulated Annealing (SA) algorithm in finding the minimum makespan in large instances of job-shop scheduling problems. The results showed that SA found shorter makespan than tailored deterministic algorithms, at the expense of greater execution times. The conclusion was that the disadvantage of expensive computation times was compensated by the simplicity of the algorithm and the higher quality solutions determined. Sidhoum et al. [19] studied a JIT scheduling problem in a parallel machine environment. Jobs were characterized by distinct due dates and earliness and tardiness penalties. The research was motivated due to the difficulty of determining lower bounds for JIT scheduling problems in single and parallel machine environments. A simple heuristic algorithm was presented. Results showed the differences between the lower and upper bound values for the single and parallel machine environments being around 1% for the problem instances investigated. McMullen [20] investigated Tabu Search to a mix-model production scheduling problem at an assembly line. The objective of the algorithm was to best determine an assembly schedule based on the part-usage rates and the number of setups involved in the process. The problem objective was to determine an assembly sequence that optimized the assembly process. Results showed that the multiple-objective problem of minimizing part-usage and setup time could be valuable from a managerial perspective. Naso et al. [21] investigated a hybridized algorithm constructed using GA and a constructive heuristic for a JIT delivery problem in supply chain management. The problem setting is that of a ready-mixed concrete delivery service, in trying to best coordinate the supply of concrete from producers to customer's on-time. Apart from problem complexity, strict time constraints had forbid early or tardy delivery of ready-mixed concrete. The problems objective was scheduled delivery that maximized profit, in minimizing risk. The case study presented used actual industrial data. The hybridized algorithm was compared to that of four other constructive heuristics. Results

showed that the hybridized algorithm determined superior solutions to that of the constructive heuristics.

The rest of this paper is structured as follows. Section 2 describes and presents the JIT scheduling problem studied. Section 3 describes previous research work in this field. Section 4 describes and presents the local search metaheuristic algorithms. Section 4 presents and discusses the experimental results obtained. Finally, Section 5 draws conclusions and outlines possible future work.

2. Problem Description and Mathematical Formulation

The allocations of company resources to meet business demands are critical to the success of an organization. Therefore, in JIT problem formulation, the untimely scheduling of business resources that miss expected due dates is accompanied by penalties factors called earliness and tardiness penalties. An earliness penalty is incurred when a job (which implies a service rendered or an item being produced) is scheduled in business before its expected time. The implication of an earliness penalty relates to the cost of holding inventory before its expected time, as an example. Also, a tardiness penalty is incurred when a job is expected to complete after its expected due date. This could imply, for example, customer dissatisfaction.

The due date of a job refers to either a specific point in time or an interval specified by a window frame of time. The jobs due date is important. It relates to the demand of products or services at predetermined times. The inability of organizations to provide on-time delivery of products and/or services sets the stage for competitiveness in industry.

In a perfect scheduling environment resources will be made available as required. Realistically, however, the limited availability of resources and the differences in demands result in resources becoming available before or after expected due dates. Hence, the problem with JIT scheduling relates to either minimizing the earliness penalty, minimizing the tardiness penalty, or both in scheduling resources [1]. Optimizing a JIT schedule is difficult due to the conflicting objectives.

Most JIT investigations have studied the scheduling of n jobs on a single machine where the due dates are specific points in time. This research studies a JIT problem of scheduling n jobs on m parallel machines where the due dates are window frames of time. The single machine scenario is easier to model and solve. Although in industry the possibility of bottlenecking exists. Surprisingly, far fewer papers have surfaced on JIT problems of scheduling jobs on multiple and parallel machines.

The mathematical model presented in this study is that given in Adamu and Abass [22]. This study takes the opportunity of correcting the original mathematical formulation by removing irrelevant constraints and reformulating the objective function in terms of the schedule. Also, although the formulation is a maximization model, the original study presented solutions for a minimization model. These inconsistencies present the opportunity for this problem to be restudied.

In the mathematical formulation given below, the left and right hand sides of a window interval of time represent the earliest start time a_j (where the job becomes available for processing) and the latest due date d_j (where the job must be completed). The jobs are scheduled starting from time zero. The problems objective is the maximization of the total weight of all on-time jobs. w_j is the weight of a job. This relates to the importance of job x_{ij} being delivered on-time. This problem assumes equivalent earliness and tardiness penalties. These penalty factors are not considered in the objective function. The mathematical formulation is as follows.

Indices:

(i) i: indicative of each machine, that is, $i = 1, \ldots, m$.

(ii) j: indicative of each job, that is, $j = 1, \ldots, n$.

Parameters:

(i) a_j: representing the left hand side of the due window of job j. This is the earliest start time of job j.

(ii) d_j: representing the right hand side of the due window of job j. This is the expected completion time of job j.

(iii) p_j: representing the processing time of each job j.

(iv) t_{ij}: representing the actual start time of job j on machine i.

(v) $C_{ij}(S)$: given a schedule S, $C_{ij}(S)$ represents the completion time of job j on machine i, that is, $C_{ij}(S) = t_{ij} + p_j$. Hence, job j is said to be early if $C_{ij}(S) < a_j$, tardy if $C_{ij}(S) > d_j$ else on-time if $a_j \leq C_{ij}(S) \leq d_j$.

(vi) w_j: weight of job j.

Variables:

(i) $x_{ij}(S)$: representative if job j is allocated on machine i, in schedule S.

Objective function:

Maximize

$$f = \sum_{i=1}^{m} \sum_{j=1}^{n} w_j x_{ij}(S). \tag{1}$$

Subject to constraints:

$$a_j \leq \left\{ \max_{k=1}^{j} \left\{ C_{i(k-1)}(S), a_j - p_j \right\} + p_j \right\} x_{ij} \leq d_j, \quad \forall i = 1, \ldots, m; \quad \forall j = 1, \ldots, n \tag{2}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1, \quad \forall j = 1, \ldots, n \tag{3}$$

$$x_{ij} = \begin{cases} 1, & \text{iff } a_j \leq C_{ij}(S) \leq d_j \\ 0, & \text{otherwise,} \end{cases} \quad \forall i = 1, \ldots, m; \quad \forall j = 1, \ldots, n. \tag{4}$$

Equation (1) represents the total weight of all on-time jobs. Equation (2) ensures that if job j is scheduled on machine i it will start and complete processing between its earliest start time a_j and latest finishing time d_j. Equation (3) ensures that job j will be assigned to at most one machine i. Equation (4) represents a job being either on-time, early or tardy, with 1 representing on-time and 0 otherwise.

The problem assumptions are as follows.

(1) Setup time is included in processing time. Hence, preemption is not allowed. When job $j-1$ is completed, there is no delay in starting job j on machine i.

(2) There is no delay in machine processing. When job j starts, it is expected to be completed as represented by processing time p_j.

(3) Only one job can be processed at any given time on machine i.

3. Methodology

This study investigates three local search metaheuristic algorithms to the problem of JIT scheduling presented in Section 2. Meta (loosely speaking) refers to an algorithm beyond or higher level to that of a simple heuristic. Heuristic means "to find" or to "discover by trial and error" [23]. There are no formal definitions for what a heuristic and metaheuristic algorithm is in computational studies. However, the trend is to classify stochastic algorithms based on randomization into these categories. The metaheuristic algorithms investigated include eBPA, Tabu Search (TS), and Simulated Annealing (SA).

TS and SA are well-known local search metaheuristic algorithms. eBPA, on the other hand, is an enhancement of the Best Performance Algorithm (BPA) previously introduced in literature by Chetty and Adewumi [11, 12]. The improvement lies essentially in the implementation aspect of the algorithm. This study aims to present the potentials of eBPA in solving an NP-Hard problem. eBPA improves on both the efficiency and execution time performances of BPA, in having a simpler and completely new design. For this reason, standard implementations of all algorithms will be compared in determining solutions for the JIT scheduling problem studied. The presentation of the eBPA and the documentation of its potential are the primary objective of this study. The algorithms are presented and explained in the subsections below.

3.1. Enhanced Best Performance Algorithm. The eBPA is modeled on the analogy of professional athletes desiring to improve on their best registered performances within competitive environments. Numerous sporting disciplines exist yet the principles are the same in that professional athletes desire to perfect their levels of skills in order to beat their personal best performances and that of their competitors. Before entering professional, all athletes start off with a simple love for the sport and a desire to succeed. Thereafter, with constant practice and strategy their skill levels increase. They learn from trial and error techniques and improve on their strengths and weaknesses. Knowledge is also acquired in learning from mentors and/or other athletes. In becoming professional, the ultimate goal of an athlete is to develop a level of skill that would cause the athlete to give off a performance that would surpass their personal best registered performances.

Apart from coaching, an effective strategy could be to maintain an archive of a limited number of the athletes' best registered performances. This could be in the form of recordings. Recordings contain both the techniques used and the result determined in the delivery of performances. Athletes can use this information to identify strengths and weaknesses in the delivery of performances. With this knowledge, weaknesses can be improved upon or new techniques can be learned. In making appropriate changes and with sufficient practice, refined skills can be achieved. The objective is for the athlete to develop a level of skill that would allow for their best performance to be superseded.

Technique (or skill) in this context refers to a solution determined by an optimization algorithm. The result of executing a performance refers to the result of evaluating the objective function using this solution. Therefore, there are notable similarities between an athlete improving on skill level and an optimization algorithm determining improved solutions. Based on this analogy, the enhanced Best Performance Algorithm (eBPA) has been modeled. There are five guiding rules governing the eBPA. They include the following.

(1) An athlete maintains an archive of recordings for a limited number of their best performances delivered during competitive environments.

(2) An athlete reviews a performance from this archive and makes appropriate changes to the technique used with the hope of executing the new technique in trying to determine a performance that would at least meet the minimum criterion of being accepted into the archive.

(3) If an improved performance is determined, the archive is updated in replacing the recording of the worst performance with the recording of the new.

(4) The new recording is then chosen as the next recording to be reviewed by the athlete (given a certain probability).

(5) Only performances with unique techniques are allowed to be registered in the list.

To artificially simulate this analogy, the eBPA maintains a limited number of the best solutions found in a list called the Performance List (PL). Here, solutions refer to recordings stored in an archive. Solution attributes (i.e., the design variables of a solution) distinguish one solution from the next. Therefore, in allowing solutions to be inserted into the PL, only solutions that are unique must be considered. Disallowing duplicate solutions prevents the algorithm from working with solutions previously visited. In the PL, the best

(1) Initialize variables $bestIndex = 0$, $workingIndex = 0$, $worstIndex = 0$
(2) Set the size of the Performance List, that is, $listSize$
(3) Set probability p_a
(4) Set the first solution in the Performance List, that is, $\text{PL}_{workingIndex}$
(5) Calculate the fitness value of $\text{PL}_{workingIndex}$, that is, $\text{PL\_}Fitness_{workingIndex}$
(6) Set $working = \text{PL}_{workingIndex}$
(7) Set Boolean variable $toggle = true$
(8) **if** not Stopping_Criterion_Met() **then** (e.g., **for** $i = 1$ to $noOfIterations - 1$ **do**)
  (8.1) **if** resize() **then**
    (8.1.1) resize_PL()
  (8.2) **end if**
  (8.3) **if** $toggle$ **then**
    (8.3.1) $working$ = Determine_Solution($\text{PL}_{workingIndex}$)
  (8.4) **else**
    (8.4.1) $working$ = Determine_Solution($working$)
    (8.4.2) $toggle$ = true
  (8.5) **end if**
  (8.6) $f\_working$ = Determine_Fitness($working$)
  (8.7) **if** $f\_working$ better than $\text{PL\_}Fitness_{worstIndex}$ **then**
    (8.7.1) perform_Update()
  (8.8) **end if**
  (8.9) **if** random$[0, 1] \leq p_a$ **then**
    (8.9.1) $toggle = false$
  (8.10) **end if**
(9) **end if** (or e.g., **end for**)
(10) return $\text{PL}_{bestIndex}$

Algorithm 1: The enhanced Best Performance Algorithm.

and worst solutions (determined by their solution qualities or results) must be indexed. Another solution from the list that is considered to be worked with must also be indexed. This is called the working solution.

To try and determine improved solutions, over solutions already registered in the PL, local search changes are applied to the solution indexed as the working solution. The new solution determined in applying the change is referred to as an update of the working solution. If the result of this solution at least improves on the result of the worst solution or is equivalent in solution quality but unique in their design variables, then the PL is updated in replacing the worst solution with that of the new. The new solution is then indexed as the working solution. If this solution result improves on the best solution result (i.e., indexed by the best index) then it is also indexed as the new best solution. The worst solution would also need to be redetermined and reindexed upon an update of the PL being made. If an update of the PL is not achieved, then local search changes will continue to be applied to the solution indexed as the working solution. However, given a certain probability, the worked with solution for the next iteration could also be the solution that had been updated in the current iteration (i.e., the updated working solution). The possibility of the algorithm working with the updated working solution, for the next iteration, represents the willingness of an athlete to work with a new but disimproved technique. Working with disimproved solutions could possibly lead to improved solutions being found.

These strategies represent eBPA's ability to work with both improved and disimproved solutions. A solution is considered improved if the updated working solution result at least improves on the working solution result. A solution is considered disimproved in two ways. Firstly, if the result of the updated working solution causes the PL to be updated without the actual working solution result in the PL being improved upon. This means that a better solution is found, which although being a disimproved solution to the current working solution still meets the minimum requirements of being accepted into the PL. Secondly, a disimproved solution is accepted to be worked with if the updated working solution does not cause an update of the PL and if the probability factor is satisfied. This will cause the updated working solution to be the worked with solution in the next iteration. Accepting disimproved solutions is eBPA's strategy of escaping local entrapment and cycling.

To further constrain (or destrain) the acceptance into the PL, the eBPA allows for the PL to be dynamically resized. Large PL sizes have less restrictive acceptance criterion compared to smaller PL sizes. Reason being the quality of the worst solution is worse off. Therefore, strategically reducing the PL size is used to further intensify the search in promising neighborhood regions. Similarly, strategically increasing the PL size can be used to escape local entrapment.

After the termination criterion is satisfied, the solution indexed by the best index will be returned as the best solution found. This solution is representative of the best technique determined by an athlete. The eBPA is shown in Algorithm 1.

```
(1)  Initialize best to be the initial tour
(2)  Set current = best
(3)  Evaluate the fitness of best = f_best
(4)  Set f_current (the fitness of current) = f_best
(5)  Set the size of the Tabu List, that is, tabuListSize
(6)  Set the size of the Candidate List, that is, candidateListSize
(7)  Initiate the Tabu List (TL) and the Candidate List (i.e. CandidateList)
(8)  for i to noOfIterations do
     (8.1) CandidateList = Generate_New_Candidate_List(current)
     (8.2) current = Find_Best_Candidate(CandidateList)
     (8.3) f_current = Determine_Fitness(current)
     (8.4) if f_current better then f_best then
           (8.4.1) f_best = f_current
           (8.4.2) best = current
           (8.4.3) Update TL with current
     (8.5) else
           (8.5.1) if Intensification_Criterion_Met() then
                   (8.5.1.1) current = Reset_Current()
           (8.5.2) end if
     (8.6) end if
(9)  end for
(10) return best
```

Algorithm 2: Tabu Search.

3.2. Tabu Search. TS is a neighborhood search algorithm based on the analogy of something that should not be touched or interfered with [24, 25]. This is implemented by maintaining a limited number of elite solutions (or specific solution attributes) in a list called the Tabu List (TL). The TL is commonly implemented in a first-in-first-out (FIFO) way, hence recording the most recent |TL| best solutions found. In searching neighborhood regions of solution x, that is, $N(x)$, the maximum number of neighbors considered is $N(x) - |\text{TL}|$, as any solution recorded in the TL has a tabu status and will not be interfered. This technique reduces the risk of cycling around local optima, as disimproved moves are accepted in implementing its metaheuristic technique to escape premature convergence.

TS also employs other strategies such as an aspiration condition, diversification, and intensification. An aspiration condition overrules a tabu status of a solution/attribute. For example, if a solution is found which improves on the best solution but uses a tabu attribute, the tabu status is overruled and the solution is accepted as the best solution found. Diversification is the analogy of a random restart. Intensification constructs other solutions, from some of the best attributes of the best solution/s found. Diversification and intensification additionally help prevent the trap of premature convergence. TS guides the search deterministically, with its strategies modeled around its main feature which is memory.

TS is implemented in this study by recording the *best* overall solution. Using a *current* solution x, a candidate list (CL) of solutions $C \subset \{N(x) - |\text{TL}|\}$ is determined neighboring x. The best candidate $x^* \in C$ is then selected as the new current solution x for the next iteration. If this solution improves on *best*, then *best* is updated to be x^*. x^* also gets inserted into TL in a FIFO manner. An intensification strategy is implemented such that if no improved solution over *best* is found, for *noOfIteration* objective function evaluations, then a random best solution is selected from the TL to be the next current solution x after a move being applied to it. The algorithm for TS is shown in Algorithm 2.

3.3. Simulated Annealing. SA [26, 27] is a Markov chain optimization technique modeled on the analogy of heated metal annealing to an equilibrium state. At higher temperatures the atomic composition of metal is more volatile, making the metallic structure unstable. However, when the metal starts to cool, the atomic structure becomes less volatile allowing it to stabilize. When completely cooled, an equilibrium state of stability is reached. For the annealing process to be successful, the decrease in the rate of temperature must be slow.

High temperatures allow SA to explore different neighborhood regions of solution space more easily. At these temperatures, local search changes will allow the search trajectory to wander from one neighborhood region to the next, in accepting both improved and disimproved solutions using elements of randomization. At higher temperatures the ability to accept disimproved solutions will be greater than when at lower temperatures. Accepting disimproved solutions is SA's technique of escaping local entrapment. The explorative ability of SA will identify the promising neighborhood regions, but as temperature T decreases by a constant rate of α, the explorative ability will decrease and exploitative ability will increase. At lower temperatures, exploitation searches neighborhood regions for local optimum points in trying to determine the best solution found by the algorithm. The best solution found by SA will be returned

```
(1) Initialize best to be the initial tour
(2) Set current = best
(3) Evaluate the fitness of best = f_best
(4) Set f_current (the fitness of current) = f_best
(5) Initiate starting temperature T and final temperature F
(6) while T ≥ F do
    (6.1) for i to stepsPerChange do
        (6.1.1) working = Determine_Solution(current)
        (6.1.2) f_working = Determine_Fitness(working)
        (6.1.3) if f_working better then f_current then
                (6.1.3.1) use_solution = true
        (6.1.4) else
                (6.1.4.1) Calculate acceptance probability P
                (6.1.4.2) if P > random[0, 1] then
                        (6.1.4.2.1) use_solution = true
                (6.1.4.3) end if
        (6.1.5) end else
        (6.1.6) if use_solution then
                (6.1.6.1) use_solution = false
                (6.1.6.2) f_current = f_working
                (6.1.6.3) current = working
                (6.1.6.4) if f_current better than f_best then
                        (6.1.6.4.1) best = current
                        (6.1.6.4.2) f_best = f_current
                (6.1.6.5) end if
        (6.1.7) end if
    (6.2) end for
    (6.3) Update T according to cooling schedule α
(7) end while
(8) return best
```

Algorithm 3: Simulated Annealing.

when its lowest temperature is reached, which is symbolic of the equilibrium state.

SA is implemented by starting off with equivalent *best* and *current* solutions. At each temperature T (reduced by a rate of $T \times \alpha$) a *stepsPerChange* number of local search moves are performed on the *current* solution to produce *working* solutions. If a *working* solution is found which improves on *current*, then *current* will be assigned to be *working*. However, given a certain metropolis probability, *current* can also be assigned to be a disimproved *working* solution. If *current* improves on *best*, then *best* will be assigned to be *current*. This process continues until T reaches its final temperature F. The algorithm for SA is shown in Algorithm 3.

4. Results and Discussion

Simulations were run using sets of jobs $n \in \{500, 1500, 2500\}$, tested on sets of machines $m \in \{2, 5, 10, 15, 20\}$. For each job $j = 1, \ldots, n$, its processing time p_j was randomly determined to fall within the interval $(1, 99)$. To set the starting and completion times a_j and d_j for job j two "Traffic Congestion Ratio" variables k_1 and k_2 were randomly selected from set $V \in \{1, 5, 10, 20\}$. Using k_1, a_j was randomly generated to fall within the interval $(0, n/mk_1)$. Using k_2, d_j was randomly generated to fall within the interval $(a_j + p_j, a_j + p_j + n/mk_2)$.

To test the algorithms fairly, a set of n jobs was initially generated and passed in as the input parameter to each of the algorithms. This was then used to test the algorithms on a particular machine m. Therefore, each algorithm used the same job set in testing on a particular machine. In this way the results were determined fairly for comparative purposes. To determine average performance results, each algorithm was run 10 times for each pair of job-machine combination. 10 runs were sufficient considering the expensive computational times of the metaheuristic algorithms. From the 10 runs, per job-machine combination, the best solution of each algorithm is compared. The best solution is referred to as the best fitness value (BFV). This is the highest total weight of all on-time jobs from the 10 runs, per job-machine combination per algorithm. Comparisons of average solution performances are also documented. This is for their average fitness value (AFV) solutions and their average execution time (AVG) performances.

To further test the algorithms fairly, their parameter settings were set such that each metaheuristic algorithm executed for exactly 1,000,000 objective function evaluations per run. The parameter settings were set as follows.

(i) eBPA: the *listSize* was set at 5. The *noOfIterations* was set at 1,000,000. p_a was set at 0.005.

Table 1: Statistics of the best fitness values (BFV) and average fitness values (AFV) for the class of 500 jobs.

Number of jobs	Methods	Fitness values	Number of machines				
			2	5	10	15	20
500	eBPA	BFV/AFV	**391.34**/**379.34**	**441.74**/427.02	**546.84**/**526.58**	**601.23**/581.25	**683.79**/665.76
	TS	BFV/AFV	325.91/317.44	382.02/371.01	462.70/453.36	519.78/511.09	584.56/575.07
	SA	BFV/AFV	388.70/370.75	441.10/**431.32**	532.90/519.58	593.44/**583.35**	680.78/**667.15**

Table 2: The average execution times in milliseconds, per machine set, for the class of 500 jobs.

Number of jobs	Methods	Average execution time (ms) for each machine set				
		2	5	10	15	20
500	eBPA	**8,394**	**15,637**	**26,763**	38,338	**47,594**
	TS	8,568	15,667	27,622	**37,804**	47,773
	SA	8,829	15,710	27,502	38,756	49,541

Table 3: Statistics of the best fitness values (BFV) and average fitness values (AFV) for the class of 1,500 jobs.

Number of jobs	Methods	Fitness values	Number of machines				
			2	5	10	15	20
1,500	eBPA	BFV/AFV	**696.21**/**653.63**	**841.96**/**778.05**	896.90/**877.80**	**952.07**/**905.63**	**1,066.63**/**1,022.07**
	TS	BFV/AFV	549.57/531.71	679.26/664.42	777.44/756.31	793.86/782.26	908.43/884.27
	SA	BFV/AFV	654.36/632.47	811.42/776.90	**909.04**/867.83	941.62/898.14	1,041.08/999.64

(ii) TS: the *tabuListSize* was set at 7. The *candidateListSize* was set at 100. The *noOfIterations* was set at 10,000.

(iii) SA: the *stepsPerChange* was set at 1,000. T was set at 115. F was set at 0.005. α was set at 0.99.

The program was written in Java. It was programmed using the Netbeans 7.0 Integrated Development Environment. All simulations were run on the same platform. The computer used had a Windows 7 Enterprise operating system, an Intel Core i5 CPU, 4 GB of RAM, and a 500 GB hard-drive. The findings of the simulations are documented in Table 1.

Table 1 gives the statistical values of the best (BFV) and average (AFV) fitness values of each algorithm, per machine set, for the class of 500 jobs. The overall best and average fitness value solutions, per machine set, are highlighted in bold font. This is for clarity. From Table 1 it is seen that eBPA determined the overall BFV solutions for all machine sets. On average, eBPA determined the overall AFV solutions for machine sets 2 and 10. SA determined the overall AFV solutions for machine sets 5, 15, and 20. However, it is seen that these solutions are only marginally superior to eBPA's solutions. TS has shown to be the weakest of the algorithms.

Graphical comparisons of the algorithms best and average fitness value solutions, as determined from Table 1, are seen in Figures 1 and 2.

Table 2 gives the statistical values of the average execution times in milliseconds (ms) for the algorithms, per machine set, for the class of 500 jobs. Although it is observed that the average execution times of the algorithms are fairly similar, eBPA executed the fastest for machine sets 2, 5, 10, and 20. TS executed the fastest for machine set 15. The relatively fast execution times of eBPA relate to its small Performance List (PL) size, which strategically decreased as the algorithm

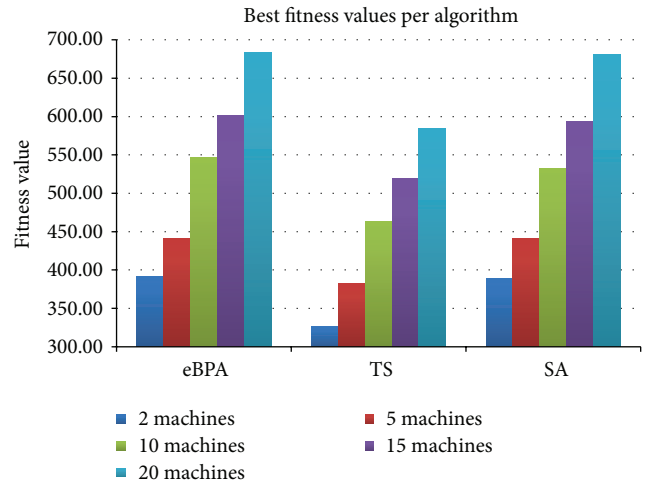


Figure 1: BFV comparisons for the class of 500 jobs.

iterated. This caused the acceptance criterion to become increasingly restrictive, allowing for greater exploitation in accepting fewer solutions to update the PL. This allowed eBPA to identify stronger solutions and explains its relatively fast execution times. A graphical comparison of the statistics given in Table 2 is seen in Figure 3.

For the class of 500 jobs it is concluded that the eBPA was the strongest algorithm.

Table 3 gives the statistical values for the overall BFV and AFV solutions, per machine set, for the class of 1,500 jobs. From Table 3 it is observed that eBPA determined the overall BFV solutions for all machine sets, except machine set 10. It also determined the overall AFV solutions for all machine

Table 4: The average execution times in milliseconds, per machine set, for the class of 1,500 jobs.

Number of jobs	Methods	Average execution time (ms) for each machine set				
		2	5	10	15	20
1,500	eBPA	**27,180**	49,508	**87,321**	117,160	**149,333**
	TS	27,964	**49,216**	88,229	117,477	150,678
	SA	28,184	49,281	88,037	**116,585**	150,116

Table 5: Statistics of the best fitness values (BFV) and average fitness values (AFV) for the class of 2,500 jobs.

Number of jobs	Methods	Fitness values	Number of machines				
			2	5	10	15	20
2,500	eBPA	BFV/AFV	993.52/960.94	1,100.46/**1,066.45**	**1,109.63/1,054.93**	**1,249.22/1,197.17**	1,307.02/1,267.87
	TS	BFV/AFV	799.60/789.21	948.02/919.61	951.80/930.43	1,071.70/1,057.45	1,150.72/1,130.19
	SA	BFV/AFV	**1,003.11/962.44**	**1,105.59**/1,039.86	1,085.21/1,039.86	1,234.31/1,182.19	**1,309.06/1,267.89**

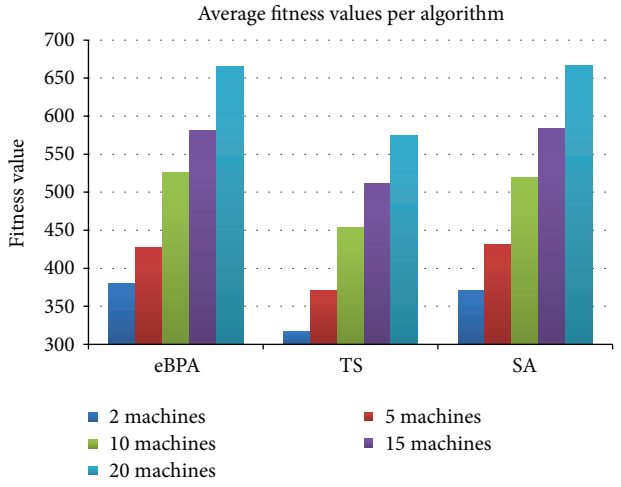


Figure 2: AFV comparisons for the class of 500 jobs.

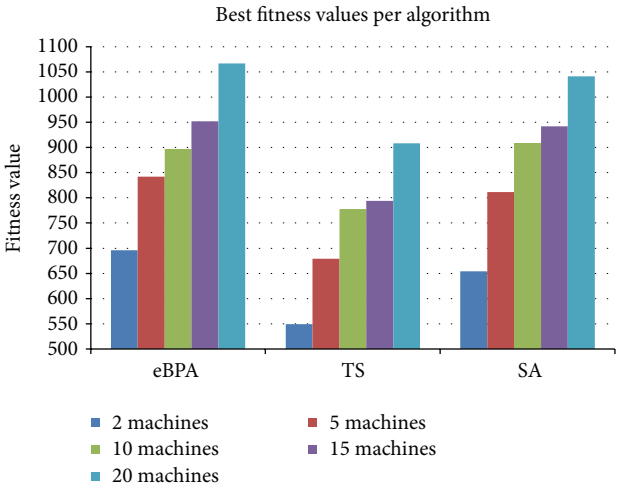


Figure 4: BFV comparisons for the class of 1,500 jobs.

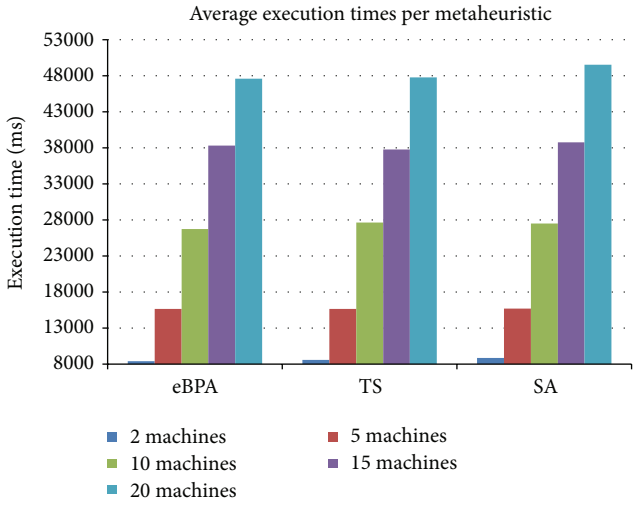


Figure 3: Average execution times per metaheuristic, per machine set, for the class of 500 jobs.

sets. SA determined the overall BFV solutions for machine set 10. SA again determined superior solutions over TS.

Graphical comparisons of the algorithms best and average fitness value solutions, as determined from Table 3, are seen in Figures 4 and 5.

Table 4 gives the statistics of the average execution times of the metaheuristic algorithms, per machine set, for the class of 1,500 jobs. It is observed that the average execution times were much more competitive for this class of jobs. eBPA performed faster on average for machine sets 2, 10, and 20. TS performed the fastest for machine set 5, and SA performed the fastest for machine set 15. Graphical comparisons of the execution time performances are seen in Figure 6.

For the class of 1,500 jobs it is also concluded that the eBPA was the strongest algorithm.

Table 5 gives the statistical values of the BFV and AFV solutions for each algorithm, per machine set, for the class of 2,500 jobs. From Table 5 it is seen that eBPA determine better BFV and AFV solutions for machine sets 10 and 15, while SA

Table 6: The average execution times in milliseconds, per machine set, for the class of 2,500 jobs.

Number of jobs	Methods	Average execution time (ms) for each machine set				
		2	5	10	15	20
2,500	eBPA	44,534.00	80,756.00	**139,053.00**	**195,479.00**	**260,926.00**
	TS	**44,461.00**	81,128.00	141,250.00	209,198.00	285,553.00
	SA	45,055.00	**80,646.00**	139,213.00	196,093.00	272,481.00

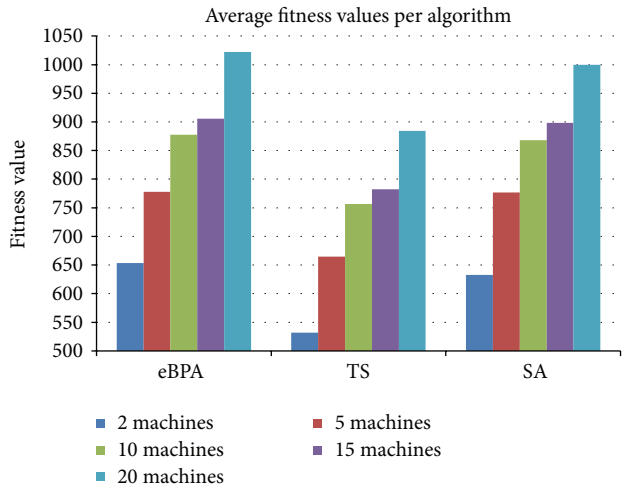


Figure 5: AFV comparisons for the class of 1,500 jobs.

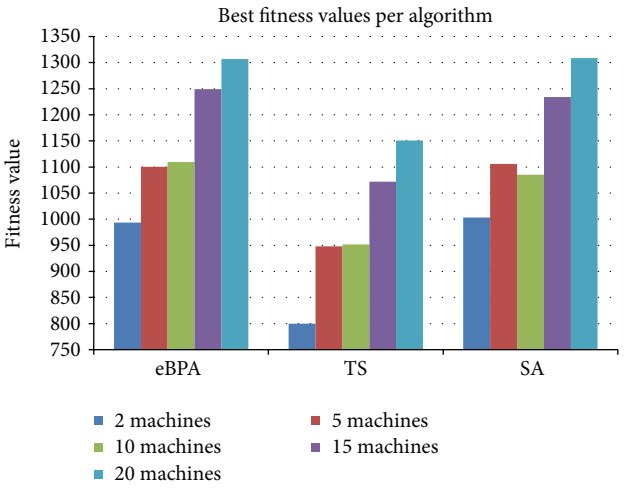


Figure 7: BFV comparisons for the class of 2,500 jobs.

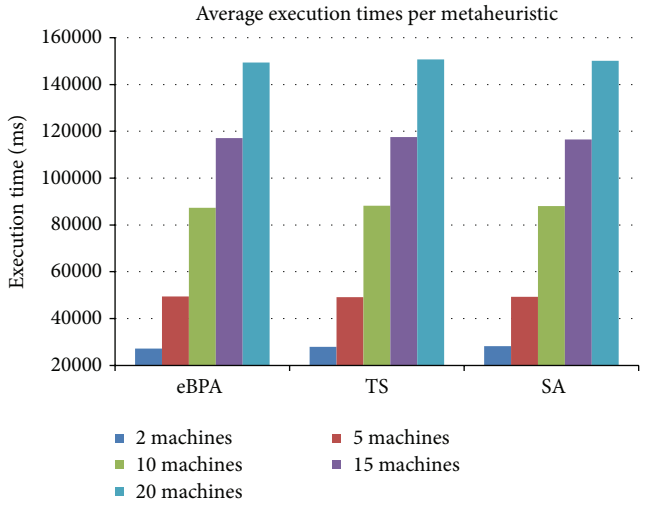


Figure 6: Average execution times per metaheuristic, per machine set, for the class of 1,500 jobs.

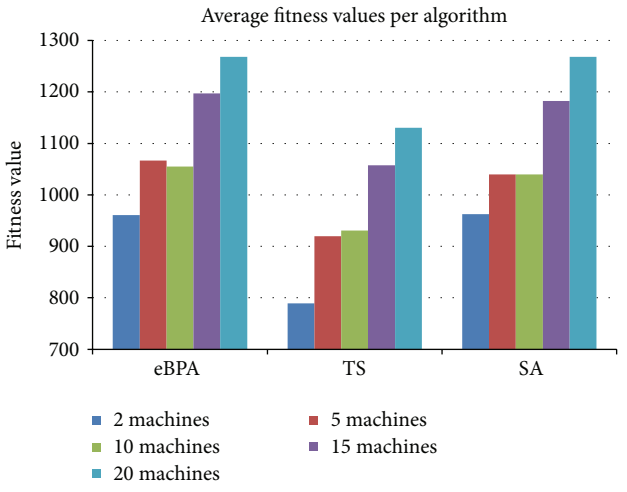


Figure 8: AFV comparisons for the class of 2,500 jobs.

determined better BFV and AFV solutions for machine sets 2 and 20. For machine set 5, eBPA determined a better AFV solution and SA determined a better BFV solution.

Graphical comparisons of the algorithms best and average fitness value solutions, as determined from Table 5, are seen in Figures 7 and 8.

Table 6 gives the statistics of the average execution times for the metaheuristic algorithms, per machine set, for the class of 2,500 jobs. It is observed that for this class, TS executed the fastest for machine set 2, SA executed the fastest for machine set 5, and eBPA executed the fastest for machine sets 10, 15, and 20. Graphical comparisons of the execution time performances are seen in Figure 9.

For the class of 2,500 jobs, both eBPA and SA performed similarly in determining an equivalent number of best solutions. However, eBPA executed the fastest for most machine sets.

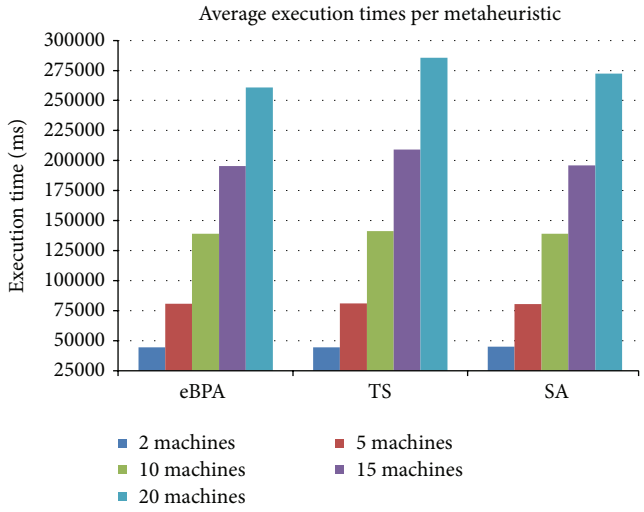


Figure 9: Average execution times per metaheuristic, per machine set, for the class of 2,500 jobs.

The strong performances of the eBPA for this JIT scheduling problem show its potential as a metaheuristic algorithm. Although standard implementations of the algorithms were compared, the results documented are significant in that the techniques employed by the eBPA show to be very competitive in being compared to the techniques of TS and SA. TS and SA are very competitive and well-known local search metaheuristic algorithms in the literature. The strength of eBPA lays in its memory structure, and the techniques used in allowing the population of solutions contained within to direct the search. Solutions registered in the PL would have identified the most attractive points within the neighborhood regions of the solution space. However, it uses the information of the worst solution in the list as a strategic point to move the search forward. The memory structure adapts dynamically in accepting solutions that satisfy the acceptance criterion. It uses each solution inserted into the PL as the next *working* solution. This strategy allows eBPA to use a population of solutions to direct the search rather than using the population as a network to exploit a neighborhood region.

As the search iterates and the worst solution in the PL is improved upon, the acceptance criteria become more restrictive allowing for greater levels of exploitation. Exploitation is further increased with the PL dynamically reducing in size by cutting away worst solutions in a strategic manner. This constrains the acceptance criteria further. This allows the algorithm to exploit quality solutions as the PL narrows in size. The solutions accepted into the PL does not need to be the best overall. However, along the way the best solution will be found. An added advantage of eBPA is its simplistic design and the few parameter settings that it requires.

5. Conclusion

The Just-In-Time (JIT) scheduling problem is of great significance to both academics and industries. The objective is to determine operational processes that would allocate limited business resources efficiently to optimize given business objectives. These objectives may include the optimization of operational costs, operational times, inventory storage, customer and supplier relations, and profits margins. In this study, the JIT problem of allocating a large number of jobs required to be processed on m parallel machines was investigated. A job represents a business resource required to be made available during a specific window interval of time. An example may be the delivery of vehicles to customers that require rented vehicles within a specific time frame. The objective was therefore to determine a schedule that would maximize the total weighed number of all on-time jobs that could be scheduled. This is an NP-Hard problem.

To determine solutions, we proposed the eBPA algorithms and investigated its results alongside well-known TS and SA techniques for comparison purposes. Results obtained show that eBPA performed competitively well with both TS and SA in terms of best and average fitness values obtained as well as the execution times thus presenting a good potential for other NP-Hard problems. Further study will therefore investigate the performance of eBPA for other types of discrete optimization problems and perhaps compare it with other population-based techniques such as Genetic Algorithms [28] and Particle Swarm Optimization [29]. We also aim to further improve on its performance through hybridization and parameter improvement.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The financial assistance of the National Research Foundation (DAAD-NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at are those of the author and are not necessarily to be attributed to the DAAD-NRF.

References

[1] P. Brucker, *Scheduling Algorithms*, Springer, 5th edition, 2007.

[2] A. O. Adewumi, B. A. Sawyerr, and M. Montaz Ali, "A heuristic solution to the university timetabling problem," *Engineering Computations*, vol. 26, no. 8, pp. 972–984, 2009.

[3] M. O. Adamu and A. O. Adewumi, "A comparative study of meta-heuristics for identical parallel machines," *Journal of Engineering and Technology Research*, vol. 5, no. 7, pp. 207–216, 2013.

[4] M. O. Adamu and A. O. Adewumi, "A survey of single machine scheduling to minimize weighted number of tardy jobs," *Journal of Industrial and Management Optimization*, vol. 10, no. 1, pp. 219–241, 2014.

[5] A. A. N. Fateha, M. Y. Nafrizuan, and A. Razlan, "Review on elements of JIT implementation," in *Proceedings of the International Conference on Automotive, Mechanical and Materials Engineering (ICAMME '12)*, Penang, Malaysia, 2012.

[6] A. J. Kootanaee, K. N. Babu, and H. F. Talari, "Just-in-time manufacturing system: from introduction to implement," *International Journal of Economics, Business and Finance*, vol. 1, no. 2, pp. 7–25, 2013.

[7] S. Kumar, V. Ranga, R. Chopra, and M. S. Sehrawat, "Scope of JIT management in Indian service industries," in *Proceedings of the International Conference on Intelligent Systems and Networks (ISN '08)*, Klawad, India, 2008.

[8] S. Singh and D. Garg, "JIT system: concepts, benefits and motivation in Indian industries," *International Journal of Management & Business Studies*, vol. 1, no. 1, pp. 26–30, 2011.

[9] M. O. Adamu and A. O. Adewumi, "Metaheuristics for scheduling on parallel machine to minimize weighted number of early and tardy jobs," *International Journal of Physical Sciences*, vol. 7, no. 10, pp. 1641–1652, 2012.

[10] M. O. Adamu and A. O. Adewumi, "Unweighted parallel machine scheduling: a meta-heuristic approach," in *Proceedings of the International Conference in Electrical and Electronics Engineering*, pp. 65–72, Istanbul, Turkey, 2013.

[11] S. Chetty and A. O. Adewumi, "Three new stochastic local search algorithms for continuous optimization problems," *Computational Optimization and Applications*, vol. 56, no. 3, pp. 675–721, 2013.

[12] S. Chetty and A. O. Adewumi, "Three new stochastic local search metaheuristics for the annual crop planning problem based on a new irrigation scheme," *Journal of Applied Mathematics*, vol. 2013, Article ID 158538, 14 pages, 2013.

[13] S. Chetty and A. O. Adewumi, "On the performance of new local search heuristics for annual crop planning: case study of the Vaalharts irrigation scheme," *Journal of Experimental & Theoretical Artificial Intelligence*, 2014.

[14] D. P. Ronconi and M. S. Kawamura, "The single machine earliness and tardiness scheduling problem: lower bounds and a branch-and-bound algorithm," *Computational & Applied Mathematics*, vol. 29, no. 2, pp. 107–124, 2010.

[15] J.-N. Monette, Y. Deville, and P. van Hentenryck, "Just-in-time scheduling with constraint programming," in *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS '09)*, pp. 241–248, Thessaloniki, Greece, September 2009.

[16] D. Dereniowski and W. Kubiak, "Makespan minimization of multi-slot just-in-time scheduling on single and parallel machines," *Journal of Scheduling*, vol. 13, no. 5, pp. 479–492, 2010.

[17] G. A. Süer, X. Yang, O. I. Alhawari, J. Santos, and R. Vazquez, "A genetic algorithm approach for minimizing total tardiness in single machine scheduling," *International Journal of Industrial Engineering and Management*, vol. 3, no. 3, pp. 163–171, 2012.

[18] P. J. van Laarhoven, E. H. Aarts, and J. K. Lenstra, "Job shop scheduling by simulated annealing," *Operations Research*, vol. 40, no. 1, pp. 113–125, 1992.

[19] S. K. Sidhoum, Y. R. Solis, and F. Sourd, *Lower Bounds for the Earliness-Tardiness Scheduling Problem on Single and Parallel Machines*, Laboratoire d'Informatique de Paris 6 (LIP6), Paris, France, 2004.

[20] P. R. McMullen, "JIT sequencing for mixed-model assembly lines with setups using Tabu Search," *Production Planning & Control*, vol. 9, no. 5, pp. 504–510, 1998.

[21] D. Naso, M. Surico, B. Turchiano, and U. Kaymak, "Genetic algorithms for supply-chain scheduling: a case study in the distribution of ready-mixed concrete," *European Journal of Operational Research*, vol. 177, no. 3, pp. 2069–2099, 2007.

[22] M. O. Adamu and O. Abass, "Parallel machine scheduling to maximize the weighted number of just-in-time jobs," *Journal of Applied Science and Technology*, vol. 15, no. 1-2, pp. 27–34, 2010.

[23] X. S. Yang, *Nature-Inspired Metaheuristic Algorithms*, Luniver Press, 2nd edition, 2010.

[24] F. Glover, "Tabu search—part I," *ORSA Journal on Computing*, vol. 1, no. 2, pp. 190–206, 1989.

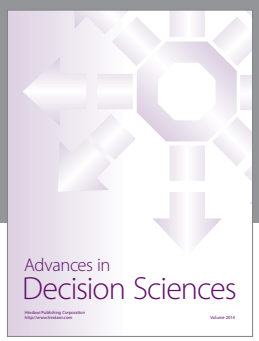
[25] F. Glover, "Tabu search—part 2," *ORSA Journal on Computing*, vol. 2, no. 1, pp. 4–32, 1990.

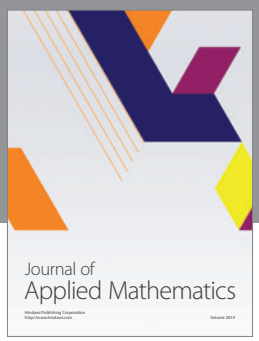
[26] S. Kirkpatrick, J. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

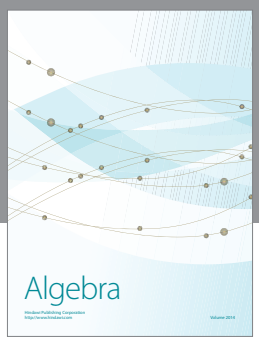
[27] C. M. Tan, *Simulated Annealing*, InTech Publisher, 2008.

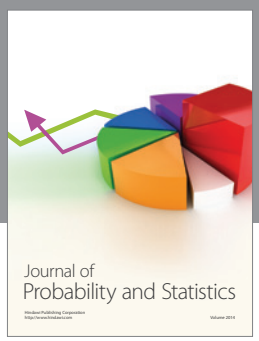
[28] A. O. Adewumi and M. M. Ali, "A multi-level genetic algorithm for a multi-stage space allocation problem," *Mathematical and Computer Modelling*, vol. 51, no. 1-2, pp. 109–126, 2010.

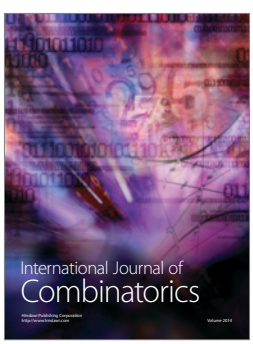
[29] M. A. Arasomwan and A. O. Adewumi, "An adaptive velocity particle swarm optimization for high-dimensional function optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '13)*, pp. 2352–2359, Cancún, Mexico, June 2013.

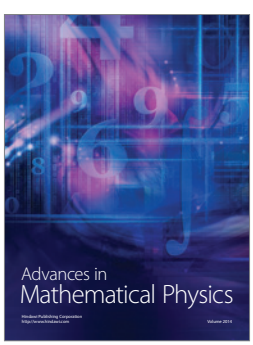


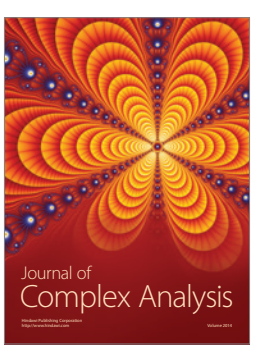


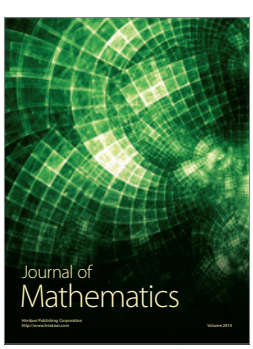


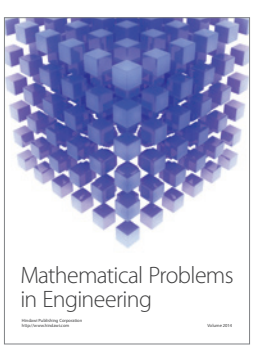


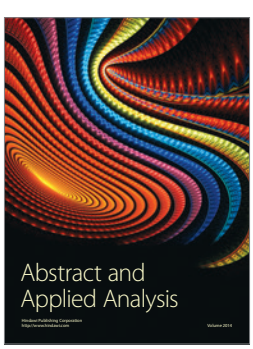


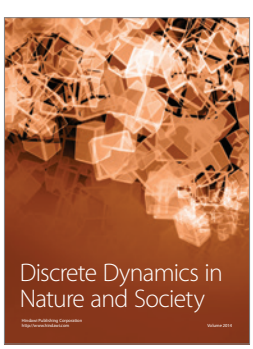


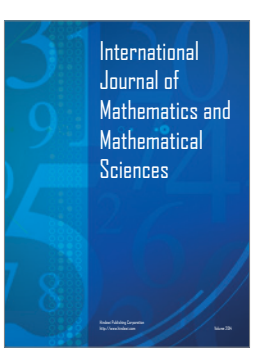












Hindawi

Submit your manuscripts at
http://www.hindawi.com