2009

Order Acceptance Using Genetic Algorithms

Walter O. Rom
Cleveland State University, w.rom@csuohio.edu

Susan A. Slotnick
Cleveland State University, s.slotnick@csuohio.edu

Publisher's Statement

NOTICE: this is the author's version of a work that was accepted for publication in Computers & Operations Research. Changes resulting from the publishing process, such as peer review, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document. Changes may have been made to this work since it was submitted for publication. A definitive version was subsequently published in Computers & Operations Research, 36 (2009), 10.1016/j.cor.2008.04.010

Original Published Citation

Rom, W. O., Slotnick, S. A. (2009). "Order Acceptance Using Genetic Algorithms". Computers & Operations Research, 36, pp. 1758-1767.

Order acceptance using genetic algorithms

Walter O. Rom, Susan A. Slotnick*

Department of Operations Management and Business Statistics, Nance College of Business Administration, Cleveland State University, 2121 Euclid Avenue, Cleveland, OH 44115, USA

1. Introduction

The order-acceptance decision has gained increasing attention over the past decade. From the perspective of professional practice, a firm may choose to reject potential orders for a variety of reasons: market focus, competitive advantage, capacity limitations, or a combination of these. While an overabundance of orders might be welcomed by a manufacturing or service facility, demand that exceeds capacity brings with it some hard choices. There is an important trade-off between the profit-enhancing revenue associated with an order, and the costs of capacity that it may divert from other jobs. In addition, late delivery of some orders may result in penalties, reduced revenue and long-term loss of good-will and market share. In a competitive market, the importance of on-time delivery may make it cost- and profit-effective to reject some orders [1].

Consider, for example, a steel producer that faces cyclical demand and limited flexibility of capacity. If the firm turns away orders in times of high demand, when it is running close to capacity, it risks losing those customers who must go elsewhere for their product. Alternatively, it can continue to accept all prospective orders, and risk significant delays and a degradation of its on-time delivery performance. Both of these policies may result in loss of business that carries over into the low-demand periods, since customers may turn to other firms who have accepted their orders, or seek companies with more reliable delivery performance. How should the firm handle orders during the high-demand period? Should it reject some proportion of potential orders, and if so, which ones?

In this paper we consider a firm that makes its order-acceptance decisions when it has a pool of potential orders, for which it knows processing time, delivery date and price (revenue). In addition, the urgency of an individual order may be enhanced by the importance of the customer (for example, an important customer may sometimes submit orders that are not as lucrative as others, but have strategic importance for future business). An order delivered past the agreed-upon delivery date incurs a penalty that is proportional to the amount of time that it is late; however, there is no reward (or penalty) from completing an order before the promise date. The profit for each order is its revenue minus tardiness penalties. Capacity is fixed during the time that the decision is made. The firm must choose the subset of potential orders, together with the processing schedule that results in the highest profit.

This paper extends previous work that considered the order-acceptance decision with lateness [2,3] and tardiness penalties [4].

* Corresponding author. Fax: +1 216 687 9343.
E-mail address: s.slotnick@csuohio.edu (S.A. Slotnick).

In particular, we compare a previously tested myopic heuristic [4] with a genetic algorithm that also combines sequencing and job-acceptance decisions. We first fine-tune the genetic algorithm with a pilot study that compares the settings of various operators, including clone removal, mutation, immigration and population size, and different types of local search. We find that using a probabilistic local search provides results that are almost as good as exhaustive local search, with much shorter processing times. We then run the genetic algorithm, using the best combination of settings and probabilistic local search, against the myopic heuristic and an upper bound. Our computational study demonstrates that the genetic algorithm always dominates the myopic heuristic in terms of objective function, at the cost of increased processing time.

The contribution of this research is twofold. First, it demonstrates a new approach to the order-acceptance problem, that competes successfully with previous algorithms for large problems. Second, we explore the relative efficacy of different settings for local search, in combination with the diversity operators (clone removal, mutation, immigration, and population size). While the settings of these operators have some effect on performance, the use and type of local search makes the most difference. We expect that our results will be useful for the future application of genetic algorithms to scheduling problems.

2. Related work

The burgeoning research literature on order acceptance addresses the problem of which jobs the firm should process in order to maximize its profit. Previous approaches to this problem include integer and linear programming, dynamic programming, and various heuristics. Recent surveys of this literature are included in [4–6]. We discuss here those order-acceptance papers that are most closely related to our present work, and also relevant scheduling papers using genetic algorithms.

The present paper adds to a stream of research [2–4,7,8] that studies the job-acceptance problem with static arrivals, deterministic processing times, a customer weight and revenue associated with each job, and lateness or tardiness as the time-related penalty. It is an extension of [4], which presents optimal and heuristic methods for the order-acceptance problem that includes sequencing decisions. Job sequencing for the tardiness objective is known to be NP-hard [9], as is the order-acceptance problem with lateness penalty [7]. While small-scale problems can be solved to optimality, medium- and large-scale problems are more challenging, and so optimal algorithms are usually accompanied by simplifying assumptions [7,8,10,11]. Thus the order-acceptance problem with tardiness penalty is a natural candidate for heuristics.

Math programming approaches to this problem (integer, linear and mixed integer linear programs—MILP) solve for cost minimization [12–14] and profit objectives [2–4]. Combinations of MILP and heuristics [6,15–17] include computational studies to assess the performance of the heuristics. Versions of the order-acceptance problem include insertion of incoming orders into a current schedule while minimizing holding costs [6,18], and decisions that combine order acceptance, scheduling and due-date setting [16] as well as pricing [17]. Dynamic programming approaches consider order acceptance over time [3] and analyze greedy solutions [8].

Other approaches to the order-acceptance problem include decision theory [19–21], simulation [22–24], costing [10,25,26], neural networks [27], workload control [28–32], due-date assignment [33], and throughput maximization [34]. Most of these papers consider sequencing as well as order-acceptance decisions. The contribution of the present paper to the study of order acceptance is the development of a genetic algorithm that performs favorably on large problems, compared to a previously tested heuristic based on assignment relaxation.

Genetic algorithms have been used in a wide variety of applications, as diverse as supply-chain management [35], vehicle routing [36], cover printing [37], waste management [38], facility and network design [39–42], design of product lines [43,44], staff scheduling [45], thermal engineering [46], portfolio planning [47], and bankruptcy analysis [48]. There have been scores of papers in the last two decades that use genetic algorithms to solve scheduling problems. This approach is attractive for the set of scheduling problems that are difficult because of their combinatorial complexity, including those that are classified as NP-hard.

Some applications of genetic algorithms to scheduling include tardiness objectives [49,50], weighted number of late jobs [51], flow-shop scheduling [52–55], parallel machines [56–58], sequence-dependent setups [54,55,59,60], the early–tardy objective [61,62] and various versions of the job-shop scheduling problem [63–67]. For a discussion of the representation of scheduling problems in genetic algorithms see [68]. For recent reviews of applications of genetic algorithms to scheduling see [58,65,69–71].

Essafi et al. [72] apply a genetic algorithm with a search procedure to job-shop scheduling. They calibrate the settings of the genetic algorithm by fixing population size, encoding scheme and crossover (based on previous studies), and varying the schedule generator, local search and probability of mutation. The results demonstrate the power of combining a genetic algorithm with local search. We also design a genetic algorithm by tuning the settings with a design of experiments approach, and use local search, to develop a method that dominates previous approaches for our problem.

An application of a genetic algorithm to the problem of order acceptance is found in Roundy et al. [6]. Their formulation of the order-acceptance problem has multiple machines, where the decisions are whether to accept an order, what due date to quote and how to size batches. New orders are inserted into a current feasible schedule, but due-date constraints cannot be violated. The objective is to minimize setup and holding costs, while meeting as much of the customer's demand as possible. An MILP is developed for small problems, and various heuristics, including a genetic algorithm, are proposed for larger problems. A computational study shows that the genetic algorithm performs well, along with simulated annealing and other heuristics.

Malve and Uzsoy [73] compare the performance of a genetic algorithm with previous heuristics, for a scheduling problem: the minimization of maximum lateness on parallel batch processors, with incompatible job families and dynamic arrivals. The problem is strongly NP-hard, and so all solution procedures presented are heuristics. The authors employ genetic algorithms to improve upon the performance of the heuristics, which may fall into local optima. They use the heuristics, in turn, to enhance the performance of the genetic algorithms. A computational study shows that performance varies with the degree of congestion, and the genetic algorithms excel when inserted idle time results in better solutions.

Like these two papers, we compare the performance of heuristic procedures with that of genetic algorithms, for a scheduling problem that cannot be solved optimally in reasonable time for large problems. Our computational study reveals when the genetic algorithm performs well relative to the myopic heuristic, and which settings are likely to result in the best performance. So our contribution is both in adding a solution method to the order-acceptance literature, and providing insight into how to develop a genetic algorithm for this type of problem.

3. The model

The current paper models the order-acceptance decision with static arrivals, deterministic processing times, and a customer weight and revenue associated with each job. There are two decisions: which

jobs to accept, and in what order to process the selected subset. The objective is profit maximization, that is, the sum of per-job revenues minus total weighted tardiness. The objective function is

$$\max \sum_{i=1}^{n} x_i[Q_i - w_i(C_i - d_i)^+] \quad (1)$$

where i is the job index; $i<j$ implies that job i precedes job j in the processing order $i, j = 1, \ldots, n$, n being the total number of jobs in the set; x_i is 0 or 1 (job accepted or not); Q_i is the revenue of job i; w_i is the customer weight (proportional lateness discount) of job i; C_i is the completion time of job i, i.e. $C_i = \sum_{j=1}^{i} x_j p_j$, where p_j is the processing time of job j; d_i is the due date of job i.

4. Solution procedures

In this paper we compare two solution procedures, in order to gain insights about the order-acceptance problem itself and about the comparative advantages of each approach. We develop procedures that can tackle large problems (50, 75 and 100 jobs) and compare their efficacy with regard to solution value and computation time. Because of the difficulty of the problem (even the lateness version of the problem is NP-hard [7]), we cannot compute an optimal benchmark for problems this large, so we use an upper bound which is simply the assignment algorithm applied to a unit processing time relaxation of the problem.

4.1. Description of the myopic heuristic

The first solution procedure we employ is a fast and accurate myopic heuristic first described in [4]. Intuitively, this algorithm solves a relaxation of the original problem, and reassembles the accepted jobs, which are sequenced heuristically to minimize weighted tardiness.

Myopic heuristic:

1. Calculate the profit of the original problem with all jobs, in their original sequence.
2. Decompose the entire problem into joblets with unit processing time, apportion the weights and revenues accordingly, and use the assignment algorithm to find the optimal sequence of this relaxation by maximizing the return of accepting or rejecting each joblet.
3. Accept all jobs that have at least 75% of their joblets accepted in the relaxed solution.
4. Sequence these (reassembled) jobs in ascending order of completion time (the completion time of the latest component joblet scheduled in the assignment solution) minus processing time, and calculate the profit of this set.
5. Order the remaining jobs using the Rachamadugu and Morton heuristic for weighted tardiness [74,75], place them after the previously accepted jobs in 4 above, and calculate the profit of this set.
6. The solution is the best of 1, 4 and 5 above.

4.2. Description of the genetic algorithm

Genetic algorithms are general-purpose stochastic search procedures that mimic the process of natural selection. A *population* is a set of current problem solutions, or *chromosomes*, which are evaluated for *fitness* with regard to the objective function, and then undergo *reproduction* (selection and modification) to form the next generation of solutions (*offspring*). This process continues until a stopping criterion (number of generations or solution value) has been reached. For a general description of genetic algorithms see [76]; for a discussion of genetic algorithms for scheduling problems see [68–70,72,77].

For our problem, the chromosomes are sample solutions, that is, sequenced subsets of jobs from the candidate pool. We encode the job acceptance and scheduling problem as a sequence of random numbers (Section 4.2.1). The fitness function is our objective, that is, the maximization of profit (sum of revenues minus the sum of tardiness penalties). We employ crossover to generate offspring (Section 4.2.3), and to maintain population diversity we make use of clone removal, mutation and immigration between two populations (Section 5.1). To improve generated solutions we use local search (Section 5.2). The procedure is terminated when a maximum number of generations has been reached, or there has been no further improvement in the solution value for a certain number of generations.

The following pseudo code describes the general procedure of our genetic algorithm.

```
Generate an initial population
Do until Maximum number of generations G is reached
{
Evaluate solutions
Update best solution found so far
Choose solutions for crossover
Perform crossover
Generate new population
}
```

4.2.1. Initial solution

For a problem with N jobs, we begin by generating N uniform random numbers, which are then ordered, while the integers from one to N (corresponding to the jobs) are reordered in parallel. For example, when the random numbers (0.12, 0.56, 0.22, 0.91, 0.44) are reordered to (0.12, 0.22, 0.44, 0.56, 0.91), this random sequence will then represent the job sequence (1, 3, 5, 2, 4) (see [68] for a discussion of this representation method). This procedure is repeated until the desired population of solutions has been generated. The value of each solution is then evaluated as follows.

1. Until the end of the sequence is reached, choose the next job to evaluate.
2. Will this job increase profit if accepted?
   (a) If yes, accept it (add it to the sequence to be saved) and return to step 1.
   (b) If no, do not accept it and return to step 1.
3. When the end of the sequence has been reached, record the resulting set of accepted jobs and its profit.

The value of the best solution generated is then recorded.

Note that this decision procedure is itself myopic. So the genetic algorithm complements the myopic heuristic described in Section 4.1 above in two important ways. First, instead of jointly optimizing the sequencing and acceptance decisions, it first sequences and then accepts jobs. Second, it uses the machinery of genetic search to construct a search space that is diverse, while keeping track of the best solutions found.

4.2.2. Successive generations

We perform the following steps to generate the successive generation:

1. Evaluate the profit of all solutions, and sort them in descending order of profit.
2. Keep the solutions whose profit is within 10% of the profit of the best solution found so far.
3. For the remaining solutions, we create a new array, ranking the solutions by an index that is calculated by modifying the profit

of each according to the following formula:

$$\text{ModProfit} = \text{Profit} - \left(\text{Rand} \times 0.05 \times \frac{\text{BestV}}{\text{GenNum}}\right) \quad (2)$$

where ModProfit is the modified profit for this solution, Profit is the original profit for this solution, Rand is the random number (uniform distribution $(0, 1)$), BestV is the best profit found so far, GenNum is the number of this generation.

So the profit of each solution is decreased by a random number, which decreases as the number of generations increases. We multiply by BestV to scale the decrease to the magnitude of the current profit values. The value 0.05 was determined empirically in a preliminary study.

4. Sort these remaining solutions in decreasing order of the index, and append them to the best solutions that were saved in (2) above.
5. Use the better (top) half of all solutions in crossovers (described below) to generate offspring that replace the bottom half. For example, when the population is of size 200, the first and the second solutions are used to generate two offspring which replace those ranked 101 and 102, and so on.

Notice that the modification effect of the index decreases with the number of generations, so that toward the end of the run, solutions are ranked by (unmodified) objective function value. This selection method is our implementation for the order-acceptance problem of a standard method of providing diversity in genetic algorithms: we first select the best solutions, and then randomize the selection of the rest, with a bias toward the better ones [76].

4.2.3. Crossover

We use a two-point crossover to generate the offspring. We randomly generate two distinct integers, n_1 and n_2, between 1 and N (the number of jobs), where n_1 and n_2 correspond to positions in the sequences. If $n_1 < n_2$ then the offspring are generated by switching the jobs between n_1 and n_2, that is, the jobs in positions $n_1, \ldots, n_2$ in the sequences. If $n_1 > n_2$ then the outer parts of the sequences (that is, the jobs before n_1 and after n_2) are switched to generate the offspring. Note that when parts of sequences are interchanged like this, the resulting sequence will not in general be feasible, since some jobs will be duplicated while others will be missing. The new sequence must thus be checked and the duplicated jobs replaced by those that are missing. The following example illustrates this process.

Example of crossover:

$n_1 = 3, n_2 = 5$
Sequence 1 {5,2,**3**,**8**,**4**,6,7,1}
Sequence 2 {3,5,**2**,**4**,**6**,7,1,8}
Offspring 1 {5,2,**2**,**4**,**6**,6,7,1} uncorrected (job 2 repeated and job 3 missing)
Offspring 2 {3,5,**3**,**8**,**4**,7,1,8} uncorrected (job 3 repeated and job 2 missing)
Offspring 1 {5,3,**2**,**4**,**6**,8,7,1} corrected
Offspring 2 {2,5,**3**,**8**,**4**,7,1,6} corrected

It is also possible to do the crossover (as well as mutation and clone removal) on the original random number sequence, and then reorder it to get a new job sequence [68]. This will always result in a feasible sequence. We did try this in a pilot study, but it did not work as well as performing the crossover on the job sequences: operating on the random number sequence took about 50% longer on average, with objective function values much worse (8.99% vs. 0.64% deviation from the benchmark on average).

5. Pilot study

In order to design a genetic algorithm for testing against the myopic heuristic, we conducted a pilot study in which we varied the settings of diversity operators, including clone removal, mutation, immigration, and population size, and also varied the type of local search. We designed the pilot study as follows. For the four diversity operators, we used a fractional factorial design, with two possible settings for each operator. This enabled us to run eight tests, and consider main effects as well as interaction effects among the diversity operators. We used probabilistic local search (see Section 5.2) in these eight tests. The fractional factorial design is shown in Table 1.[1] We used the resulting best settings of these four operators (Test 6), to test exhaustive local search and no search. In Sections 5.1 and 5.2 we discuss the details of these settings.

Table 1
Fractional factorial design.

Test number	Population size	Mutation type	Clone removal	Population switch
1	120	RISJ	No	No
2	200	RISJ	No	Yes
3	120	RR2J	No	Yes
4	200	RR2J	No	No
5	120	RISJ	Yes	Yes
6	200	RISJ	Yes	No
7	120	RR2J	Yes	No
8	200	RR2J	Yes	Yes

5.1. Population diversity

We maintain population diversity in four ways: checking for duplicate solutions, using mutation, maintaining two separate populations with immigration between them, and varying population size. Since comparing entire sequences to check for duplicate solutions is time consuming, we compare the values of solutions. If two solutions have the same value, we have the option of performing a mutation on one of them, to eliminate the duplication ("clone removal"). In the pilot study, we used two settings of this operator: clone removal or no clone removal.

In the pilot study, we compared two methods of performing mutations: (1) randomly interchange successive jobs (RISJ), and (2) randomly pick two locations in a sequence and randomly sequence the jobs between them (RR2J). Each time a new generation is created, there is a probability that each solution has a mutation applied. After preliminary experimentation we used 0.1 for the probability of a mutation.

A third method of maintaining population diversity is to maintain two separate populations. We performed immigration between the two populations by periodically switching portions of the populations between them. In preliminary studies, we employed three levels of population switch (every 20, 40 or 60 generations), and also considered the effect of no population switch. In the pilot study, we compare no population switch with population switch every 20 generations. Finally, we varied the population size, with the settings of 120 and 200.

5.2. Local search

Many implementations of genetic algorithms use local search to improve the generated solutions [70,72]. We employ local search when a solution has been created using a crossover, or modified by a mutation. That is, we search the neighborhood of the newly generated or modified solution for a better solution. With this approach, each sequence represents the best solution in a neighborhood.

[1] The two types of mutation are random interchange of successive jobs (RISJ) and random sequencing of jobs between two locations (RR2J), as defined in Section 5.1.

Table 2
Experimental design.

Problems	τ	Correlation
1–20	0.3	No
21–40		Yes
41–60	2.0	No
61–80		Yes
81–100	3.0	No
101–120		Yes

Using local search can improve the solutions generated, but it becomes computationally expensive for large problems.

We considered two types of local search. For the first type, we perform one pass through the job sequence and successively interchange the positions of all pairs of jobs. If an interchange improves the solution, we keep it. This exhaustive search requires $N*(N-1)/2$ comparisons and substantially increases the computation time. The second type of local search limits the number of pairwise interchanges to five, and only searches with a probability of 0.10. The number of pairwise interchanges and probability of search was set empirically, after preliminary tests.

5.3. Test problems and experimental design

For the pilot study, we ran 120 randomly generated problems, with 75 jobs each. The problems were generated as follows. Weights and processing times were drawn from a uniform distribution $(0, 1)$, then adjusted by a constant (multiplied by 10, with 1 added to the product), in order to adjust the difficulty of the problem in terms of profit margin and job size. Revenue was drawn from a lognormal distribution with an underlying normal distribution with mean 0 and standard deviation one.[2] Due dates were drawn from a uniform distribution, adjusted to the magnitude of processing times (p_i) in each problem. Each due date equals the generated uniform number plus an adjustment factor: $2 \times [\quad p_i/(1+\tau)] - 1$. The problems thus generated typify a scenario where job characteristics are similar, but revenue may vary more widely [4]. We used three different settings of τ $(0.3, 2.0, 3.0)$, resulting in successively tighter due dates. For each value of τ, we generated revenues for half of the problems that were not correlated with due dates (average correlation across all jobs 0.0002), and for the other half, we generated revenues that were inversely correlated with each due date for each job (average correlation -0.2062). So there were six types of problems in terms of due-date configuration (see Table 2). Test programs were coded in FORTRAN 90 and run on a Gateway computer with an Intel Pentium M 1.6 GHz processor.

5.3.1. Termination criterion

In a preliminary study, the algorithm was programmed to terminate after a predetermined number of generations. The numbers we tried were in the range of 100–700. We found, however, that for some problems there was no change in the best solution after the first few generations, while for others the best solution was found in a later generation. The termination criterion we used for the pilot study was to specify the maximum number of generations along with a limit on the number of generations with no improvement. We used 2500 as the maximum number of generations and 300 as the maximum number of generations with no improvement. This enabled the algorithm to terminate quickly when a very good solution was found early, while otherwise permitting it to continue to run if improvements were still being found.

5.4. Performance measures

The performance measure that we use for objective function value is deviation from an upper bound, which is found by using an assignment algorithm on a unit-time relaxation of the initial problem, which allows for joblets to be omitted. The average deviation is the average, over all problems in the set, of the difference between the upper bound and the objective function value, divided by the upper bound value:

$$\frac{1}{\text{number of problems}} \sum \frac{\text{upper bound} - \text{genetic or myopic solution}}{\text{upper bound}}$$

We also report the minimum and maximum deviation, the number of solutions which were equal to the upper bound (a lower bound on the number of times the algorithm achieved the optimal solution), and the average processing time (in CPU seconds).

5.5. Results of the pilot study

Table 3 and Fig. 1[3] show the results of the pilot study. In this set of problems, the genetic algorithm achieved the upper bound for a number of problems, and so the minimum deviation for all tests is zero. Looking at the average percentage deviation for tests 1–8, the best settings are found in test 6 (population size of 200, RISJ mutation type, clone removal, no immigration). The average running time of this test was also the longest, by about 16 s, compared to the next best performance (test 8). Comparing the results of test 6 with the exhaustive local search and the same settings of the other operators (penultimate row of Table 3), we find that the average objective function value is about the same (0.15% difference in average deviation), but the exhaustive local search took about 28 times as long to run. Using no local search at all (last row of Table 3) resulted in the worse performance of all (average deviation of 2%, more than twice as much as any setting with search, and maximum deviation almost twice as much as the worst for previous tests), with no saving in running time (it took twice as many generations, on average, as when probabilistic search was used). Test 6 had the lowest average percentage deviation, with test 5 running much faster with a small difference in objective function performance. Since all running times were manageable, we decided to use the settings that gave the best objective function value. So we chose test 6 for the settings for the main computational study: RISJ, clone removal, population of 200, no immigration and probabilistic local search.

6. Main computational study

After determining the best settings with the pilot study, we performed a computational study which compares the genetic algorithm and myopic heuristic under a variety of scenarios. As in the pilot study, we vary the due-date tightness (and so the difficulty) of the scheduling problem, and for each level of due-date tightness we consider whether or not the due date is inversely correlated with revenue (which illustrates a situation in which a customer pays more to have an order completed earlier).

[2] Note that these distributions guarantee that all numbers generated are positive. We used standard routines from the IMSL libraries: RNUN for the uniform distribution, and RNLNL for the lognormal distribution. For the latter, the probability density function is defined as

$$f(x) = \frac{1}{\sigma x\sqrt{2\pi}} \exp\left[-\frac{1}{2\sigma^2}(\ln x - \mu)^2\right] \quad \text{for } x > 0$$

For more details, see [78].

[3] Fig. 1 includes tests 1–8 only.

Table 3
Pilot study.

Test	Pop.	Mut.	Clone	Immig.	Avg. dev. (%)	Min dev. (%)	Max dev. (%)	Equal UB	Avg. time
1	120	RISJ	No	No	0.7762	0.0000	4.2824	16	23.60
2	200	RISJ	No	Yes	0.7356	0.0000	4.2080	16	29.42
3	120	RR2J	No	Yes	0.7645	0.0000	4.4504	18	17.38
4	200	RR2J	No	No	0.6117	0.0000	5.0926	18	39.29
5	120	RISJ	Yes	Yes	0.6591	0.0000	4.1253	18	21.64
6	200	RISJ	Yes	No	0.5957	0.0000	4.1253	18	46.07
7	120	RR2J	Yes	No	0.5980	0.0000	4.1419	18	30.53
8	200	RR2J	Yes	Yes	0.5995	0.0000	4.1419	18	37.61
Ex. LS	200	RISJ	Yes	No	0.4430	0.0000	4.0923	18	1319.90
No srch.	200	RISJ	Yes	No	2.0097	0.0000	9.3200	12	54.56

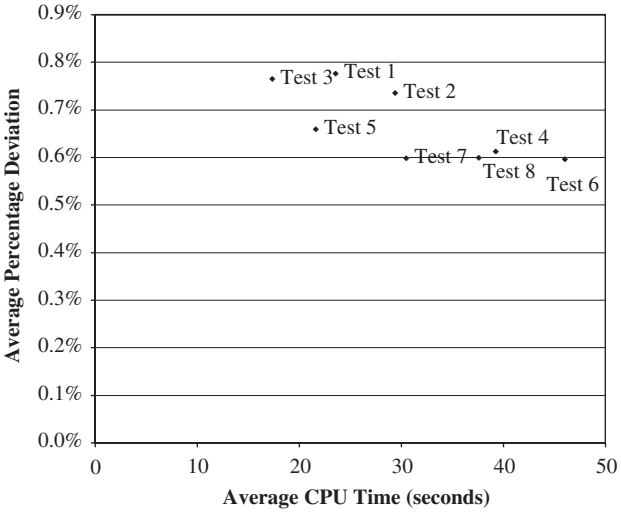

Fig. 1. Performance vs. CPU time (pilot study).

6.1. Test problems

For the main computational study, we ran three sets of 120 randomly generated problems, of size 50, 75 and 100. As for the pilot study (but using different random number seeds) weights and processing times were drawn from a uniform distribution $(0, 1)$, and due dates were drawn from a uniform distribution, adjusted to the magnitude of processing times in each problem. We used three different settings of τ $(0.3, 2.0, 3.0)$ and generated revenues for half of the problems without correlating due date and revenue (average correlation across all problems was 0.003), and for the other half, we generated revenues that were inversely correlated with each due date for each job (average correlation -0.187). This resulted in six sets of jobs with different job characteristics (see Table 2). To check the robustness of our results, we ran one set of 75-job problems with much tighter due dates ($\tau = 5.0$, 7.0, 10.0), and found that while (as expected) performance measures (deviation from upper bound) declined and processing times increased, the results that we describe in Section 6.3 were the same in terms of comparing the two procedures. Test programs were coded in FORTRAN 90 and run on a Gateway computer with an Intel Pentium M 1.6 GHz processor.

6.2. Performance measures

We compared the performance of the genetic algorithm and the myopic heuristic, in terms of objective function value and computation time. We compare the average, minimum and maximum deviation from the upper bound, and also report the number of solutions equal to the upper bound and the average processing time (in CPU seconds).

6.3. Results

The details of results for the sets of 50-, 75- and 100-job problems are given in Tables 4, 5 and 6. We separate the results into six problem types, characterized by due-date tightness (τ) and correlation (Crl.). Results aggregated by problem size are given in Table 7 and Fig. 2. In Tables 8 and 9, the results are aggregated by the problem characteristics of due-date tightness and correlation, respectively. The last two columns of these tables show the ratio of the genetic algorithm to the myopic heuristic, with regard to average percentage deviation from the upper bound, and average running time.

First we note that the upper bound is fairly tight; the maximum deviation for the genetic algorithm is 3.010% (Table 4), and for the myopic heuristic 9.628% (Table 5), with average deviations less than 3%. From all of the tables, and Fig. 2, we can immediately see that the genetic algorithm dominates the myopic heuristic in terms of objective function value. The average deviation from the upper bound of the genetic algorithm is always less than that of the myopic heuristic. See Figs. 3–5. The minimum and maximum deviation of the genetic algorithm is always less than that of the myopic heuristic (except when both minima equal zero). The genetic algorithm does worse than the myopic heuristic only 23 times in 360 problems. Aggregating by problem size, due-date tightness and correlation, the average deviation of the genetic algorithm is usually less than half of that of the myopic heuristic (penultimate column of Tables 7–9).

Looking at the breakdown by problem size (Tables 4–6, and Figs. 3–5), we see that as the value of τ rises (indicating more difficult problems in terms of tighter due dates), both procedures usually do worse, and the relative advantage of the genetic algorithm (the ratio in the penultimate column) is somewhat eroded. We also see this in the averages reported in Table 8.

The difference between no correlation and inverse correlation between due date and revenue shows a similar pattern; except when $\tau = 0.3$, both procedures do worse for correlated problems, with the performance of the genetic algorithm dropping more quickly (Tables 4–6). The relative advantage of the genetic algorithm is usually lower for correlated problems of size 50 and 75, but better for 100-job problems with correlation when $\tau = 2.0$ and 3.0. Across all problems (see Table 9), the relative advantage of the genetic algorithm is slightly lower for correlated than for uncorrelated problems.

However, the genetic algorithm takes longer to run (Fig. 2). The average times are always longer, and except for 20 times in 360 problems, the individual processing time for the genetic algorithm is greater than that of the myopic heuristic. On average the genetic

Table 4
50-job problems.

τ	Crl.	Genetic algorithm					Myopic heuristic					Genetic/myopic	
		Avg. dev. (%)	Min dev. (%)	Max dev. (%)	Eq. Bd.	Avg. time	Avg. dev. (%)	Min dev. (%)	Max dev. (%)	Eq. Bd.	Avg. time	Avg. dev.	Avg. time
0.3	0.025	0.059	0.000	0.685	11	10.88	0.333	0.000	1.104	2	2.96	0.176	3.671
0.3	−0.213	0.112	0.000	1.051	10	10.74	0.295	0.000	2.647	5	4.07	0.378	2.639
2.0	0.007	0.146	0.000	1.273	10	11.19	0.298	0.000	2.398	6	4.69	0.489	2.384
2.0	−0.161	0.313	0.001	1.137	0	18.50	0.588	0.011	2.558	0	3.24	0.532	5.702
3.0	0.009	0.527	0.000	3.010	1	20.18	0.893	0.004	4.076	0	4.19	0.590	4.821
3.0	−0.243	0.432	0.021	1.397	0	22.84	0.988	0.054	5.869	0	5.25	0.437	4.354

Table 5
75-job problems.

τ	Crl.	Genetic algorithm					Myopic heuristic					Genetic/myopic	
		Avg. dev. (%)	Min dev. (%)	Max dev. (%)	Eq. Bd.	Avg. time	Avg. dev. (%)	Min dev. (%)	Max dev. (%)	Eq. Bd.	Avg. time	Avg. dev.	Avg. time
0.3	−0.016	0.090	0.000	0.526	8	24.51	0.355	0.000	1.570	1	8.14	0.254	3.012
0.3	−0.168	0.175	0.000	1.151	6	20.67	0.338	0.000	1.934	2	16.58	0.516	1.246
2.0	0.034	0.612	0.037	1.632	0	67.68	1.247	0.070	6.715	0	9.81	0.490	6.901
2.0	−0.191	0.938	0.047	2.614	0	63.55	1.903	0.103	4.127	0	13.94	0.493	4.557
3.0	−0.005	0.535	0.032	1.406	0	52.32	1.405	0.071	6.408	0	9.30	0.381	5.623
3.0	−0.167	1.280	0.026	2.799	0	56.29	2.923	0.077	9.628	0	14.90	0.438	3.778

Table 6
100-job problems.

τ	Crl.	Genetic algorithm					Myopic heuristic					Genetic/myopic	
		Avg. dev. (%)	Min dev. (%)	Max dev. (%)	Eq. Bd.	Avg. time	Avg. dev. (%)	Min dev. (%)	Max dev. (%)	Eq. Bd.	Avg. time	Avg. dev.	Avg. time
0.3	−0.011	0.199	0.000	1.124	9	55.37	0.451	0.000	1.501	1	19.85	0.442	2.790
0.3	−0.219	0.194	0.000	1.411	7	46.57	0.366	0.000	1.684	2	38.74	0.532	1.202
2.0	0.021	0.651	0.074	1.503	0	119.21	1.009	0.000	3.064	1	22.22	0.645	5.364
2.0	−0.133	0.936	0.069	1.956	0	132.43	2.357	0.134	8.354	0	33.87	0.397	3.910
3.0	−0.034	0.853	0.023	2.259	0	137.16	1.675	0.106	4.100	0	23.90	0.509	5.740
3.0	−0.187	1.055	0.027	2.421	0	116.54	2.608	0.085	7.757	0	33.41	0.405	3.488

Table 7
By problem size.

Prob size	Genetic algorithm					Myopic heuristic					Genetic/myopic	
	Avg. dev. (%)	Min dev. (%)	Max dev. (%)	Eq. Bd.	Avg. time	Avg. dev. (%)	Min dev. (%)	Max dev. (%)	Eq. Bd.	Avg. time	Avg. dev.	Avg. time
50	0.265	0.000	3.010	32	15.72	0.566	0.000	5.869	13	4.07	0.468	3.865
75	0.605	0.000	2.799	14	47.50	1.362	0.000	9.628	3	12.11	0.444	3.922
100	0.648	0.000	2.421	16	101.21	1.411	0.000	8.354	4	28.66	0.459	3.531

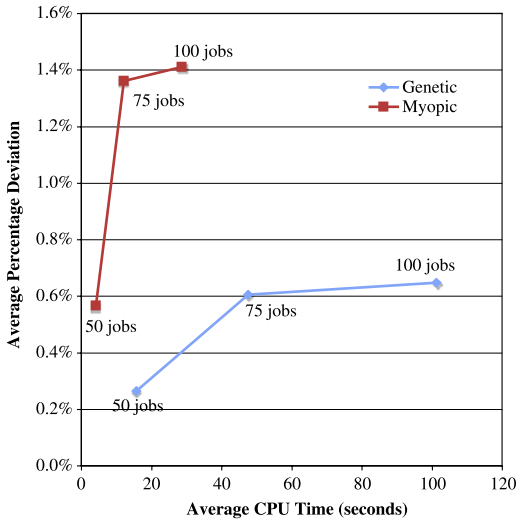

Fig. 2. Performance vs. CPU time (main computational study).

algorithm takes about four times as long to run (last column of Table 7). The myopic heuristic almost always takes longer for correlated problems of all sizes, but the genetic algorithm does not, and so we see that the time disadvantage of the genetic algorithm is lower for 75- and 100-job problems with correlation (see the last column of Tables 5 and 6).

7. Conclusion

We have developed a genetic algorithm that performs well for large instances of the order-acceptance problem. We used a fractional factorial experimental design to determine the best settings for the diversity operators, and then tried two different types of local search to improve the solutions. Our pilot study demonstrated that the most dramatic improvements in performance (objective-function value) are achieved by exhaustive local search, but at a high computational cost. A probabilistic search does nearly as well as exhaustive local search, at a fraction of the processing time, and so we use that procedure, as well as the best settings of population size, mutation, cloning and immigration for the main computational study.

Table 8
By due-date tightness.

τ	Genetic		Myopic		Genetic/myopic	
	Avg. dev. (%)	Avg. time	Avg. dev. (%)	Avg. time	Avg. dev. (%)	Avg. time
0.3	0.138	28.12	0.356	15.06	0.388	1.868
2.0	0.599	68.76	1.234	14.63	0.486	4.700
3.0	0.780	67.55	1.749	15.16	0.446	4.457

Table 9
By correlation (average correlation across all problem sizes and types).

Crl.	Genetic		Myopic		Genetic/myopic	
	Avg. dev. (%)	Avg. time	Avg. dev. (%)	Avg. time	Avg. dev. (%)	Avg. time
0.003	0.604	54.24	1.374	18.22	0.440	2.976
−0.187	0.408	55.39	0.852	11.67	0.479	4.745

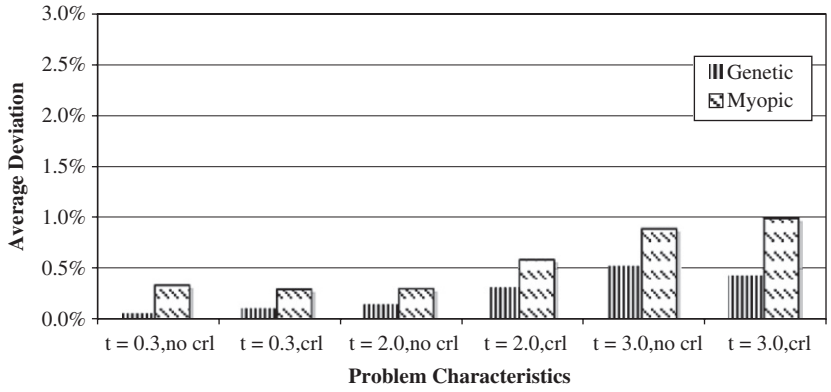

Fig. 3. Average deviation: 50-job problems.

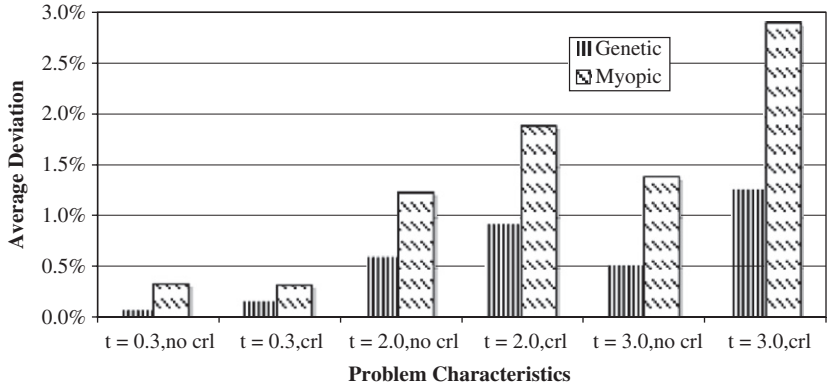

Fig. 4. Average deviation: 75-job problems.

The main results show that, while the upper bound is fairly tight for both procedures, the genetic algorithm dominates in terms of objective function value, but takes about four times as long to run. As problem difficulty increases, in terms of due-date tightness and correlation of due date and revenue, both procedures do worse, and the objective-function performance of the genetic algorithm degrades more than that of the myopic heuristic. Although the genetic algorithm takes longer to run than the myopic heuristic, computation times are still fast enough to enable its use for real-time job selection and due-date quotation. For example, when a firm has a set of orders with firm customer delivery dates, either algorithm can quickly identify which orders should be accepted for processing. If delivery schedules are negotiable, the procedures run fast enough to enable the decision-maker to perform sensitivity analysis with different due-date values. Our results, as well as our investigative methodology, have implications for the further study of scheduling problems in general, and the order-acceptance problem in particular, using genetic algorithms.

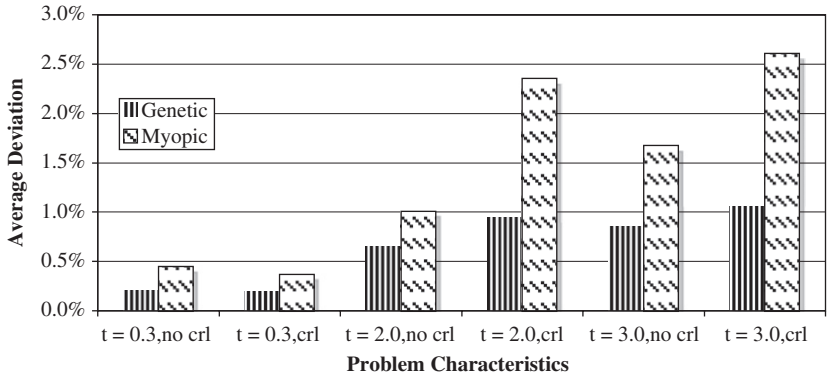

Fig. 5. Average deviation: 100-job problems.

Acknowledgments

We wish to thank two anonymous referees for their careful reading and helpful suggestions, particularly with regard to the computational study.

References

[1] Guerrero HH, Kern G. How to more effectively accept and refuse orders. Production and Inventory Management 1988;29(4):59–63.
[2] Slotnick SA, Morton TE. Selecting jobs for a heavily loaded shop with lateness penalties. Computers and Operations Research 1996;23(2):131–40.
[3] Lewis HF, Slotnick SA. Multi-period job selection: planning work loads to maximize profit. Computers and Operations Research 2002;28(1):1081–98.
[4] Slotnick SA, Morton TE. Order acceptance with weighted tardiness. Computers and Operations Research 2007;34:3029–42.
[5] Keskinocak P, Tayur S. Due date management policies. In: Handbook of quantitative supply chain analysis: modeling in the E-business era. Boston, MA: Kluwer; 2004. p. 485–556.
[6] Roundy R, Chen D, Chen P, Cakanyildirim M, Freimer MB, Melkonian V. Capacity-driven acceptance of customer orders for a multi-stage batch manufacturing system: models and algorithms. IIE Transactions 2005;37(12):1093–105.
[7] Ghosh J. Job selection in a heavily loaded shop. Computers and Operations Research 1997;24(2):141–5.
[8] Alidaee B, Kochenberger G, Amini M. Greedy solutions of selection and ordering problems. European Journal of Operational Research 2001;134:203–15.
[9] Du J, Leung JYT. Minimizing total tardiness on one machine is NP-hard. Mathematics of Operations Research 1990;15:483–95.
[10] Gietzmann M, Monahan G. Absorption versus direct costing: the relevance of opportunity costs in the management of congested stochastic production systems. Management Accounting Research 1996;7:409–29.
[11] Weng ZK. Strategies for integrating lead time and customer-order decisions. IIE Transactions 1999;31(2):161–71.
[12] Guerrero HH, Kern G. A conceptual model for demand management in the assemble-to-order environment. Journal of Operations Management 1990;9(1):65–83.
[13] Wu MC, Chen SY. A cost model for justifying the acceptance of rush orders. International Journal of Production Research 1996;34:1963–74.
[14] Wu MC, Chen SY. A multiple criteria decision-making model for justifying the acceptance of rush orders. Production Planning and Control 1997;8:753–61.
[15] Kolisch R. Integrated production planning, order acceptance, and due date setting for make-to-order manufacturing. In: Operations research proceedings. New York, NY: Springer; 1998. p. 492–7.
[16] Charnsirisakskul K, Griffin P, Keskinocak P. Order selection and scheduling with leadtime flexibility. IIE Transactions 2004;36:697–707.
[17] Charnsirisakskul K, Griffin P, Keskinocak P. Pricing and scheduling decisions with leadtime flexibility. European Journal of Operational Research 2006;171(1): 153–69.
[18] Wester F, Wijngaard J, Zijm W. Order acceptance strategies in a production-to-order environment with setup times and due-dates. International Journal of Production Research 1992;30:1313–26.
[19] Balakrishnan N, Patterson J, Sridharan V. Rationing capacity between two product classes. Decision Sciences 1996;27(2):185–214.
[20] Balakrishnan N, Patterson J, Sridharan V. An experimental comparison of capacity rationing models. International Journal of Production Research 1997;35(6):1639–49.
[21] Balakrishnan N, Patterson J, Sridharan V. Robustness of capacity rationing policies. European Journal of Operational Research 1999;115:328–38.
[22] Ono K, Jones C. An heuristic approach to acceptance rules in integrated scheduling systems. Journal of Operations Research Society of Japan 1973;16(1):37–58.
[23] ten Kate HA. Towards a better understanding of order acceptance. International Journal of Production Economics 1994;37:139–52.
[24] ten Kate HA. Order acceptance and production control. Ph.D. thesis, University of Groningen; 1995.
[25] Wouters M. Relevant cost information for order acceptance decisions. Production Planning and Control 1997;8(1):2–9.
[26] Verdaasdonk P, Wouters M. Defining an information structure to analyse resource spending changes of operations management decisions. Production Planning and Control 1999;10(2):162–74.
[27] Wang J, Yang JQ, Lee H. Multicriteria order acceptance decision support in over-demanded job shops: a neural network approach. Mathematical Computer Modelling 1994;19(5):1–19.
[28] Kingsman B. Modelling input–output workload control for dynamic capacity planning in production planning systems. International Journal of Production Economics 2000;68:73–93.
[29] Raaymakers WHM, Bertrand JM, Fransoo JC. The performance of workload rules for order acceptance in batch chemical manufacturing. Journal of Intelligent Manufacturing 2000;11:217–28.
[30] Raaymakers WHM, Bertrand JM, Fransoo JC. Using aggregate estimation models for order acceptance in a decentralized production control structure for batch chemical engineering. IIE Transactions 2000;32:989–98.
[31] Ivanescu CV, Fransoo JC, Bertrand JM. Makespan estimation and order acceptance in batch process industries when processing times are uncertain. OR Spectrum 2002;24:467–95.
[32] Ebben M, Hans E, Weghuis FO. Workload based order acceptance in job shop environments. OR Spectrum 2005;27:107–22.
[33] De P, Ghosh J, Wells C. Job selection and sequencing on a single machine in a random environment. European Journal of Operational Research 1993;70: 425–31.
[34] Yang B, Geunes J. Heuristic approaches for solving single resource scheduling problems with job-selection flexibility. Technical Report, Working Paper, Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL; 2004.
[35] Ko H, Evans G. A genetic-algorithm based heuristic for the dynamic integrated forward/reverse logistics network for 3PLs. Computers and Operations Research 2007;34:346–66.
[36] Alvarenga G, Mateus G, de Tomi G. A genetic and set partitioning two-phase approach for the vehicle routing problem with time windows. Computers and Operations Research 2007;34:1561–84.
[37] Elaoud S, Teghem J, Baouaziz B. Genetic algorithms to solve the cover printing problem. Computers and Operations Research 2007;34:3346–61.
[38] Bautista J, Pereira J. Modeling the problem of locating collection areas for urban waste management. An application to the metropolitan area of Barcelona. Omega 2006;34:617–29.
[39] Norman BA, Smith AE. Continuous approach to considering uncertainty in facility design. Computers and Operations Research 2006;33(6):1760–75.
[40] Tyni T, Ylinen J. Evolutionary bi-objective optimisation in the elevator car routing problem. European Journal of Operational Research 2006;169(3): 960–77.
[41] Vroblefski M, Brown EC. A grouping genetic algorithm for registration area planning. Omega 2006;34(3):220–30.
[42] Zhang GQ, Lai KK. Combining path relinking and genetic algorithms for the multiple-level warehouse layout problem. European Journal of Operational Research 2006;169(2):413–25.
[43] Alexouda G, Paparrizos K. A genetic algorithm approach to the product line design problem using the seller's return criterion: an extensive comparative computational study. European Journal of Operational Research 2001;134: 165–78.
[44] Alexouda G. An evolutionary algorithm approach to the share of choices problem in product line design. Computers and Operations Research 2004;31:2215–29.
[45] Moz M, Pato MV. A genetic algorithm approach to the nurse rerostering problem. Computers and Operations Research 2007;34:667–91.

[46] Cheng YH, Shih C. Maximizing the cooling capacity and COP of two-stage thermoelectric coolers through genetic algorithm. Applied Thermal Engineering 2006;26(8/9):937–47.
[47] Jiao J, Zhang Y, Wang Y. A heuristic genetic algorithm for product portfolio planning. Computers and Operations Research 2007;34:1777–99.
[48] Lensberg T, Eilifsen A, McKee TE. Bankruptcy theory development and classification via genetic programming. European Journal of Operational Research 2006;169(2):677–97.
[49] Avci S, Akturk MS, Storer R. A problem space algorithm for single machine weighted tardiness problems. IIE Transactions 2003;35:479–86.
[50] Mattfeld D, Bierwirth C. An efficient genetic algorithm for job shop scheduling with tardiness objectives. European Journal of Operational Research 2004;155:616–30.
[51] Sevaux M, Dauzère-Pérès S. Genetic algorithms to minimize the weighted number of late jobs on a single machine. European Journal of Operational Research 2003;151:296–306.
[52] Bolat A, Al-Harkan I, Al Harbi B. Flow-shop scheduling for three serial stations with the last two duplicate. Computers and Operations Research 2005;32: 647–67.
[53] Etiler O, Toklu B, Atak M, Wilson J. A genetic algorithm for flow shop scheduling problems. Journal of the Operational Research Society 2004;55:830–35.
[54] Franca P, Gupta J, Mendes A, Moscato P, Veltink K. Evolutionary algorithms for scheduling a flowshop manufacturing cell with sequence dependent family setups. Computers and Industrial Engineering 2005;48:491–506.
[55] Ruiz R, Maroto C, Alcarez J. Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. European Journal of Operational Research 2005;165:34–54.
[56] Chiu NC, Fang SC, Lee YS. Sequencing parallel machining operations by genetic algorithms. Computers and Industrial Engineering 1999;36:259–80.
[57] Koh SG, Koo PH, Ha JW, Lee WS. Scheduling parallel batch processing machines with arbitrary job sizes and incompatible job families. International Journal of Production Research 2004;42(19):4091–107.
[58] Jou C. A genetic algorithm with sub-indexed partitioning genes and its application to production scheduling of parallel machines. Computers and Industrial Engineering 2005;48:39–54.
[59] Kurz M, Askin R. Scheduling flexible flow lines with sequence-dependent setup times. European Journal of Operational Research 2004;159:66–82.
[60] Lee SM, Asllani AA. Job scheduling with dual criteria and sequence-dependent setups: mathematical versus genetic programming. Omega 2004;45(32): 145–53.
[61] Hino CM, Ronconi DP, Mendes AB. Minimizing earliness and tardiness penalties in a single-machine problem with a common due date. European Journal of Operational Research 2005;160:190–201.
[62] M'Hallah R. Minimizing total earliness and tardiness on a single machine using a hybrid heuristic. Computers and Operations Research 2007;34:3126–42.
[63] Ghedjati F. Genetic algorithms for the job-shop scheduling problem with unrelated parallel constraints: heuristic mixing method machines and precedence. Computers and Industrial Engineering 1999;37:39–42.
[64] Prins C. Competitive genetic algorithms for the open-shop scheduling problem. Mathematical Methods of Operations Research 2000;52:389–411.
[65] Ponnambalam SG, Aravindan P, Rao PS. Comparative evaluation of genetic algorithms for job-shop scheduling. Production Planning and Control 2001;12(6):560–74.
[66] Park BJ, Choi HR, Kim HS. A hybrid genetic algorithm for the job shop scheduling problems. Computers and Industrial Engineering 2003;45:597–613.
[67] Cavory G, Dupas R, Goncalves G. A genetic approach to solving the problem of cyclic job shop scheduling with linear constraints. European Journal of Operations Research 2005;161:73–85.
[68] Bean JC. Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing 1994;6:154–60.
[69] Cheng R, Gen M, Tsujimura Y. A tutorial survey of job-shop scheduling problems using genetic algorithms—I. Representation. Computers and Industrial Engineering 1996;30:983–97.
[70] Cheng R, Gen M, Tsujimura Y. A tutorial survey of job-shop scheduling problems using genetic algorithms—II. Hybrid genetic search strategies. Computers and Industrial Engineering 1999;36:343–64.
[71] Ruiz R, Maroto C. A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research 2005;165: 479–94.
[72] Essafi I, Mati Y, Dauzère-Pérès S. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. Computers and Operations Research 2008;35:2599–616.
[73] Malve S, Uzsoy R. A genetic algorithm for minimizing maximum lateness on parallel identical batch processing machines with dynamic job arrivals and incompatible job families. Computers and Operations Research 2007;34: 3016–28.
[74] Morton TE, Rachamadugu RM. Myopic heuristics for the single machine weighted tardiness problem. Working Paper No. 30-82-83, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA; 1982.
[75] Morton TE, Pentico DW. Heuristic scheduling systems: with applications to production engineering and project management. New York, NY: Wiley; 1993.
[76] Goldberg DE. Genetic algorithms in search, optimization and machine learning. Reading, MA: Addison-Wesley; 1989.
[77] Cheng R, Gen M, Tsujimura Y. A tutorial survey of job-shop scheduling problems using genetic algorithms—part II. Hybrid genetic search strategies. Computers and Industrial Engineering 1999;37:51–5.
[78] IMSL. Random number generation. User's manual: FORTRAN subroutines for statistical analysis. 1991 [chapter 18].