PERGAMON

Computers & Industrial Engineering 41 (2001) 335–353

computers & industrial engineering

www.elsevier.com/locate/dsw

An efficient frontier approach to addressing JIT sequencing problems with setups via search heuristics

Patrick R. McMullen*

Department of Management, College of Business, Auburn University, Lower Business Building, Auburn, AL 36849, USA

Accepted 30 July 2001

Abstract

This research presents a technique to obtain production sequences where scheduling flexibility and number of setups are considered. These two objectives of flexibility and setups are typically inversely correlated with each other, so simultaneous optimization of both is challenging. Another challenging issue is the combinatorial nature of such sequencing problems. An efficient frontier approach is exploited where simultaneous maximization of flexibility and minimization of setups is desired. The efficient frontier is constructed via the popular search heuristics of Genetic Algorithms, Simulated Annealing and Tabu Search. For several test problems from the literature, it is discovered that the efficient frontiers obtained via the three search heuristics provide near optimal results for smaller problems, and for larger problems, the Simulated Annealing and Tabu Search approaches outperform the Genetic Algorithm approach.

Keywords: Simulated Annealing; Genetic Algorithms; Mutation

1. Introduction

Successful implementation of JIT production systems can place manufacturing firms in an improved competitive position. Decreased inventories and waste, isolation of quality problems, shorter lead-times and increased manufacturing flexibility are perhaps the most coveted attributes of successful JIT implementation. These desired attributes, however, do not come easily. For JIT implementation to be successful, several prerequisites must be met. This research effort is concerned with the relationship between the flexibility obtained via successful JIT implementation and the setup requirement.

A good deal of prior research has been dedicated to finding the best mixed-model sequence in terms of

* Tel.: +1-334-844-6511.
E-mail address: pmcmullen@business.auburn.edu (P.R. McMullen).

0360-8352/01/$ - see front matter
PII: S0360-8352(01)00059-6

keeping the usage rate of materials as stable as possible. This popular objective makes the implicit assumption that setup times between differing products are negligible. The research here makes the assumption that setup times between differing products are *not* negligible. While it is acknowledged that small setup times between differing products are necessary for JIT success, it is assumed here that they are small, but not negligible, as opposed to other research. After all, firms successfully using JIT would likely prefer fewer setups for a production run as opposed to more.

Past research has determined that these two entities are frequently in opposition to one another (McMullen, 1998; McMullen & Frazier, 2000). This research is concerned with finding production sequences that permit flexibility with respect to JIT systems, with a minimum of setups. Therefore, there are two objectives of interest here: flexibility and setups.

The type of problem addressed here is concerned with finding solutions to a combinatorial sequencing problem having two objectives: minimization of setups and maximization of flexibility. The research presented in this paper presents a technique to address such problems. An efficient frontier approach is used to find solutions that perform well with regard to both setups and flexibility — a graphical representation of setups and flexibility. The frontier is constructed using three popular search heuristics: a Genetic Algorithm, Simulated Annealing and Tabu Search. This methodology is employed to solve several test problems from the literature using each of these search heuristics. The performance of these heuristics is compared against the optimal solution obtained via complete enumeration for the smaller problems. For the larger problems, complete enumeration is not possible, so the three search heuristics have their performances compared against each other.

It should also be noted that the search heuristics mentioned above have been used in the past to address multiple objective problems (Baykasoglu, Owen & Gindy, 1999; Cheng & Li, 1998; Suppapitnarm, Seffen, Parks & Clarkson, 2000). The research presented here only extends this work to the more specific problem of finding production sequences that provide stability of materials usage while simultaneously providing minimal setups.

The following sections of this paper detail the aforementioned issues, describe the experiment used to gauge performance of the heuristics, present experimental results and concluding comments.

2. JIT sequencing

2.1. Setups and flexibility

Monden (1983) discusses the need for JIT sequences to have the ability to provide stability in the material usage rate. This is what he refers to as 'production smoothing', or the 'means for adapting production to variable demand'. Miltenburg (1989) provides a metric to quantify this stability of material usage rate proposed by Monden. This usage rate is essentially used to evaluate the amount of product inter-mixing for a production sequence, and is thought of here as a simplifying surrogate for flexibility. Gaither and Frazier (1999) discuss the generality of this level-scheduling, while Wantuck (1989) presents more detailed discussions. Miltenburg's usage rate is as follows:

$$U = \sum_{k=1}^{D_T} \sum_{i=1}^{a} \left(x_{i,k} - k \frac{d_i}{D_T} \right)^2 \tag{1}$$

where U is the Usage rate of a production sequence, a the number of unique products to be produced, D_T

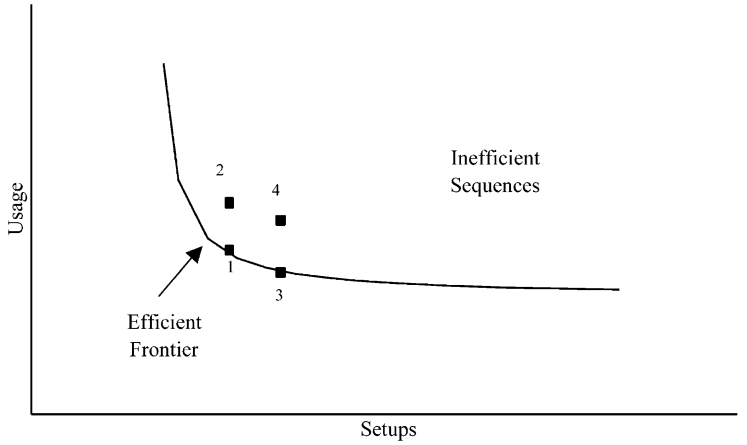

Fig. 1. Efficient frontier for JIT sequencing problem.

the total number of units for all products or total demand, d_i the demand for product i, $i = 1, 2, \ldots a$, and $x_{i,k}$ is the total number of units of product i produced over stages 1–k, where $k = 1, 2, \ldots, D_T$. Low values of this usage rate are desired.

The number of setups in a production sequence is quite straightforward — it is the number of times adjacent members of the production sequence are different from one another. The calculation for number of setups is as follows:

$$S = 1 + \sum_{k=2}^{D_T} s_k \tag{2}$$

where S is the number of setups in a production sequence and $s_k = 1$ if setup required; 0 if not.

Consider the production sequence where there are three units of product A demanded, two of B, one each for C and D. One possible sequence would be: AAABBCD. This production sequence results in four setups and a usage rate of 11.71. Another possible sequence would be: ABCADBA. This particular sequence would result in seven setups and a usage rate of 2.86 (Ding & Cheng, 1993). This simple example suggests the inverse relationship between setups and stability of usage rate.

2.2. Combinatorial nature of sequencing

There are a total of 420 possible sequences for the example above when the following formula is used (Abranovic, 1997):

$$\text{Possible Sequences} = \frac{\left(\sum_{i=1}^{a} d_i\right)!}{\prod_{i=1}^{a} (d_i!)} \tag{3}$$

This increases to 1,261,260 when the demand for each product is doubled.

3. Efficient frontier

Because there are two objectives of interest here (usage rate and setups), where simultaneous minimization is desired, this type of problem can be thought of as a multiple-objective optimization problem. An efficient frontier provides an opportunity to address both objectives of interest here. An efficient frontier is a collection of points on a curve that possess a set of attributes collectively dominating all other points not on the frontier. Here, the attributes are the number of required setups and the usage rate for a sequence of interest. There is an axis for each of these attributes — number of setups is on the x-axis, and the usage rate is on the y-axis. A hypothetical efficient frontier is provided in Fig. 1.

The actual curve, also referred to as the efficient frontier, is a graphical representation of the combination of setups and usage rates that cannot be surpassed, or improved upon, by other sequences. Point 1 represents a production sequence which is on the efficient frontier, while Point 2 is not on the frontier (because of a higher usage rate), despite it having the same number of setups as Point 1. The sequences represented by Points 3 and 4 have the same relationship to one another as do the sequences represented by Points 1 and 2, respectively. The only difference here is that the sequences associated with Points 3 and 4 require more setups than the sequences associated with Points 1 and 2. Any points 'northeast' of the efficient frontier have their associated sequences classified as inefficient, or dominated.

One specific advantage to using an efficient frontier for a problem of this type is that the setups required objective is discrete, as opposed to continuous. This simplifies the effort needed to construct an efficient frontier, because for each possible number of setups, the user can work to find the minimum usage rate, because for each sequencing problem, there will be only $D_T - a + 1$ values for setups on the efficient frontier. If the number of setups were continuous, finding the number of values for setups on the efficient frontier would be much more tedious.

4. Search heuristics

The search heuristics of Genetic Algorithms, Simulated Annealing and Tabu Search are presented with examples in the following subsections. In the interest of this paper, they can simply be thought of as differing strategies for attainment of near-optimal efficient frontiers for the problem at hand. What these three different heuristics do have in common is that they essentially strive to obtain the global optimal condition, as opposed to the local optimal condition. The consequence of this goal is that in striving for global optima, relatively inferior solutions occasionally replace relatively superior solutions in an attempt to avoid being 'trapped' at such local optima.

4.1. Genetic Algorithms

A general description of Genetic Algorithms is beyond the scope of this paper, but Goldberg (1989) provides an informative introduction. In the context of the problem at hand, the 'best' production sequences are selected for 'crossing' with the other selected sequences in the creation of the next generation of production sequences. 'Best' sequences here are the sequences with the minimum usage value for each possible number of setups — these are the sequences selected for crossover. Specifically, selection for each generation works by selecting the production sequence having the lowest usage rate for each required number of setups. In each population, there will be as few as a required setups, and no

more than D_T required setups. Crossover is where two selected sequences are used to construct two new sequences — each of the two child sequences has characteristics of both parent sequences. Because the 'selected' sequences for crossover include sequences having as few as a setups and as many as D_T setups, the resultant number of child sequences obtained via the crossovers is as follows:

$$\text{Child Sequences} = (D_T - a + 1)(D_T - a) \tag{4}$$

The number of child sequences for each generation can be thought of as the population size, from which $(D_T - a + 1)$ sequences will be extracted for crossover and mutation for construction of the next generation. Again, these $(D_T - a + 1)$ sequences extracted from the population have the lowest usage for each unique number of setups.

After crossover, there is a small probability that a newly created production sequence will undergo a minor enhancement, independent of its parents. This enhancement is referred to as mutation, and occurs with the small probability of P_m. Mutation is done so that newly created entities will have some attributes somewhat unique from their parents.

This crossover, mutation and selection process continues until the specified number of generations is reached (*Gen*). At the end of the search process, the efficient frontier is reported by listing the lowest usage rate (found during the search) for each number of required setups.

4.1.1. Example of crossover and mutation

An example of crossover and mutation is important to detail because of its complexity. Crossover in several applications of Genetic Algorithms involves converting the data structure to a binary representation and subsequently 'swapping', or crossing-over the binary digits between the parents involved. This type of crossover is not possible for a problem such as the one at hand here due to feasibility considerations of the child sequences (Michalewicz, 1996). As a result, a genetic recombination approach must be used. Davis (1985) and Oliver, Smith and Holland (1987) offer details of the genetic recombination approach of interest here. Consider the following two sequences, selected for crossover, which will be referred to as parents:

Parent 1 : A A|A B B C|D Parent 2 : D B|B A A A|C

The vertical bars are randomly placed markers dictating the extent of parent's attributes passed along to the next generation — the child sequences. The sequence elements between the bars are replicated in the child sequences. The next step is to reorder these two sequences from the right of the rightmost bar and ending to the left of the rightmost bar

Parent $1'$: D **A A A B** B C Parent $2'$: **C** D **B B A** A A

The boldfaced characters in Parent $1'$ are jobs from Parent 2 that are directly copied into Child 2, and the boldfaced characters in Parent $2'$ are jobs from Parent 1 that are directly copied into Child 1. These particular products will not be involved in crossover and must be removed. After removal of the boldfaced jobs, the filtered parents are as follows:

Filtered Parent $1'$: D B C Filtered Parent $2'$: D A A

These filtered jobs are copied into the new child sequences. Jobs from filtered Parent $1'$ are used to construct Child 2, while the jobs from filtered Parent $2'$ are used for construction of Child 1. This 'genetic recombination' works by having filtered Parent $1'$ complete the Child 2

sequence starting at the right of the rightmost bracket, and finishing left of the leftmost bracket. Similarly, filtered Parent $2'$ completes the Child 1 sequence. The child sequences are as follows:

Child 1 : A A|A B B C|D Child 2 : B C|B A A A|D

Notice that the jobs between the bars for the sequences are identical to their corresponding parents.

Mutation for this problem is simpler to understand as compared to crossover — random selection and pairwise swapping of sequence members. An example is as follows:

Child 1 (Before) : A $\underline{A}$ A B B $\underline{C}$ D Child 1 (After) : A $\underline{C}$ A B B $\underline{A}$ D

It must be understood that mutation typically happens with a small probability ($\sim < 10\%$). Mutation is done so that diversity can occasionally be incorporated in the population, which can improve the solution beyond crossover alone.

It should be noted that many are many variations available for Genetic Algorithms. The genetic recombination approach proposed by Davis (1985) and Oliver et al. (1987), which is used here is one of several possible options.

4.1.2. Pseudocode for Genetic Algorithm

The following pseudocode provides a condensed version of the Genetic Algorithm used for this approach:

```
Initialize Search Parameters
Randomly Generate Initial Solutions
Construct Initial Frontier
For i = 1 to Gen
  For j = 2 to D_T
    For k = 1 to j − 1
      Perform Crossover(j,k)
      If Rand < P_m then Perform Mutation
    Next k
  Next j
  For j = a to D_T
    Find min U[j setups]
  Next j
  For j = a to D_T
    If min U[j setups]_i < overall_min U[j setups]_i then
      overall_min U[j setups]_i = min U[j setups]_i
  Next j
Next i
Report Resultant Efficient Frontier
```

4.2. Simulated Annealing

Similar to a Genetic Algorithm, Simulated Annealing can be used in an attempt to find global optima.

Eglese (1990) and Kirkpatrick, Gelatt and Vecchi (1983) provide informative introductory explanations to the Simulated Annealing process.

During the Simulated Annealing process, each modified sequence obtained has its objective function value determined. The required number of setups is noted and the subsequent usage rate for this test solution ($U[setups]_t$) is compared to the usage rate of the current solution for the same number of required setups ($U[setups]_c$). If this 'test solution' provides a more desirable objective function value than the 'current solution', the test solution replaces the current solution. Otherwise, a probabilistic test known as the Metropolis Criterion (Metropolis, Rosenbluth, Rosenbluth, Teller & Teller, 1953) is performed to determine the probability of accepting the relatively inferior sequence. The Metropolis Criterion states that the probability of accepting a relatively inferior solution ($P(A)$) is as follows:

$$P(A) = \exp\left(\frac{-\Delta E}{K_b T}\right) \tag{5}$$

where

$$\Delta E = \frac{U[setups]_t - U[setups]_c}{U[setups]_c} \tag{6}$$

$$K_b = \frac{-\Delta E_1}{T_1 \log(P(A)_1)} \tag{7}$$

The value K_b is referred to as the Boltzman constant, and reflects the probability of accepting a usage rate ($P(A)_1$) by the amount of E_1 above the usage rate of the current solution, at the initial temperature (T_1). This constant provides the user with control over the probability of relatively inferior solutions being accepted. If a randomly generated value on the [0,1] interval is less than the $P(A)$ value obtained via the Metropolis Criterion, then the relatively inferior production sequence replaces the current sequence.

This process continues a specified number of times (*Iter*) for each temperature level (T) until the stopping point (T_F) is reached. Each new temperature level is determined by the following relationship:

$$T = T * CR \tag{8}$$

At the end of the search process, the efficient frontier is reported by listing the lowest usage rate (found during the search) for each number of required setups.

4.2.1. Example of modification of sequence in Simulated Annealing

The following details an example of how a current solution is modified here for the Simulated Annealing search — random selection of sequence members for swapping. Unlike mutation with Genetic Algorithms, however, the swapping here occurs every iteration

Before Modification : $\underline{B}$ A A B A $\underline{C}$ D After Modification : $\underline{C}$ A A B A $\underline{B}$ D

Experimentation has determined that the number of items in a sequence involved in swaps is irrelevant (McMullen & Frazier, 2000). For example, three items involved in a swap does not provide better results as compared with two items in a swap (or vice versa).

4.2.2. Pseudocode for Simulated Annealing

The following details the algorithmic steps for the Simulated Annealing approach:

```
Initialize Search Parameters
Randomly Generate Initial Solutions
Select Initial Frontier
Randomly Select Starting Point from Initial Frontier
While T > T_F
  For i = 1 to Iter
    Generate Test Solution from Current Solution via Swap
    If U[setups]_t < U[setups]_c then
      Replace Current w/test solution
      If U[setups]_t < U[setups]_Best then
        Replace Best w/test solution
      Endif
    Else check Metropolis Criterion
      If Rand < P(A) then
      Replace Current w/test solution
    Endif
  Next i
  T = T * CR
End While
Report Resultant Efficient Frontier
```

4.3. Tabu search

Like its counterparts described above, Tabu search also strives for global optima. Bland and Dawson (1991) and Glover (1990,1993) offer clear descriptions of Tabu Search supported with examples.

A current solution is used for modification (the current solution is modified n times — n being the sample size). This modification is done via pairwise swapping in the same way it was done for Simulated Annealing. The most promising of these n solutions in terms of usage rate $U[setups]_t$ is selected as the test solution. This solution is accepted as current unless its modification details appear on a list of recently accepted test solutions. If this is the case, the test solution is considered forbidden, or 'tabu', and is not accepted as current (more recent modifications, or swaps cannot be undone). This detail list of recently accepted solutions is referred to as the tabu list and has a user-specified length of *TL Length*. The tabu status of a tabu test solution can be overridden (and the test solution thereby accepted as current) if the usage rate of the tabu solution is less than the usage rate that the corresponding solution on the tabu list aspires to (at that given number of required setups: $U[setups]_t < Aspiration\ U[setups]_c$). In other words, the recently found tabu solution can be accepted as current (and the tabu status would subsequently be ignored) if and only if its usage rate is less than the usage rate of the corresponding solution on the tabu list (where $Aspiration\ U[setups]_c$ represents the usage rate that the corresponding/current solution on the

tabu list aspires to). The recently accepted test solution (regardless of acceptance type) has its modification details added to the top of the tabu list and other members of the tabu list have their positions demoted by one. The last entry on the tabu list prior to update, then, is subsequently removed. It should be noted that for this type of problem, there is a unique tabu list for each possible value of required setups. This process continues for a user-defined number of iterations (*Iter*). At the end of the search process, the efficient frontier is reported by listing the lowest usage rate (found during the search) for each number of required setups. Because the swapping here is identical to the swapping for the Simulated Annealing process, there is no need for such an example.

4.3.1. Pseudocode for Tabu search

The pseudocode for the Tabu Search approach for the problem at hand is as follows:

```
Initialize Search Parameters
Randomly Generate Initial Solutions
Select Initial Frontier
Randomly Select Starting Point from Initial Frontier
For i = 1 to Iter
  For j = 1 to n
    Generate Test Solution
  Next j
  Select Test Solution with minimum usage (U[setups]_t)
  For j = 1 to TL Length
    If Tabu Status = 1 then Check Aspiration of Tabu Solution
      If U[setups]_t < Aspiration(U[setups]_c) Then
        Replace Current w/Test Solution
        Update Tabu List
      Endif
    Else
      Replace Current w/Test Solution
      Update Tabu List
      If U[setups]_t < U[setups]_Best then Replace Best w/Test Solution
    Endif
  Next j
Next i
Report Resultant Efficient Frontier
```

5. Experimentation

The presented methodology is used to solve several test problems from the literature (Sumichrast & Russell, 1990), as well as some new problems developed for this research. Each of these test problems is solved via the presented heuristics of Genetic Algorithms, Simulated Annealing and Tabu Search. For the smaller test problems, the resultant efficient frontiers obtained via the search heuristics are compared to the optimal efficient frontiers obtained via complete enumeration. For the larger test problems, the resultant efficient frontiers obtained via the search heuristics are only compared to each other — the

Table 1
Search parameters for heuristics

Problem Set	Simulated Annealing		Tabu Search		Genetic Algorithm
	$P(A)_1$[a] (%)	E_1[a] (%)	*TL Length*	Sample size (n)	Probability of mutation (P_m) (%)
1	50	25	8	10	10
2	50	25	16	10	10
3	50	10	8	10	7.5
4	25	10	16	10	2.5
5	25	10	16	10	5

[a] Note: For Simulated Annealing, 25 iterations for each temperature level (*Iter*) are used for all problems. T_1 is 25 for all problems, and T_F is 1. For TS, the Aspiration criterion is a function of *TL Length*.

large number of feasible solutions prevents complete enumeration here. The Appendix A details the test problems and their problem-specific search parameters.

5.1. Search parameter values

Table 1 details the parameters used for the three different search heuristics.

The parameter values listed above provide the most desirable performance in terms of efficient frontier performance, and were determined via experimentation. In this experimentation to find the best parameter values, it should be mentioned that for each problem, several replications of solutions were found to ensure the fact that variation in performance measures was small. In other words, repeatability of results was obtained with success.

As there is an interest in finding the most desirable of these heuristics, it is imperative that a fair comparison between them be made. In attempt to do this, effort is made to measure their performance on an 'even playing field'. Finding the most favorable search parameters as described above helps to do this, but having each search heuristic process the same number of solutions during the search also contributes to fairness (Hooker, 1995). The number of solutions evaluated by each search heuristic is proportional to the total number of feasible solutions.

5.2. Example problem

To illustrate the concept of the efficient frontier beyond that of the elementary example in Fig. 1, an example is developed and explained here. Consider a sequencing problem with (5) units of product A required, (3) units of product B required, and (1) unit required for both products C and D. This is a total of (10) products required, which results in 5040 possible sequences. Fig. 2 shows the usage rates and setups for each of the 5040 sequences — the minimal usage rate for each setup comprises the efficient frontier.

An interesting feature of the scatter plot above is that the majority of possible solutions lie between the minimal number of required setups (4) and the maximum number of required setups (10). It is therefore likely that enumeration or heuristic search will largely involve itself in a search space between the extreme setups values — this is where the bulk of computational effort

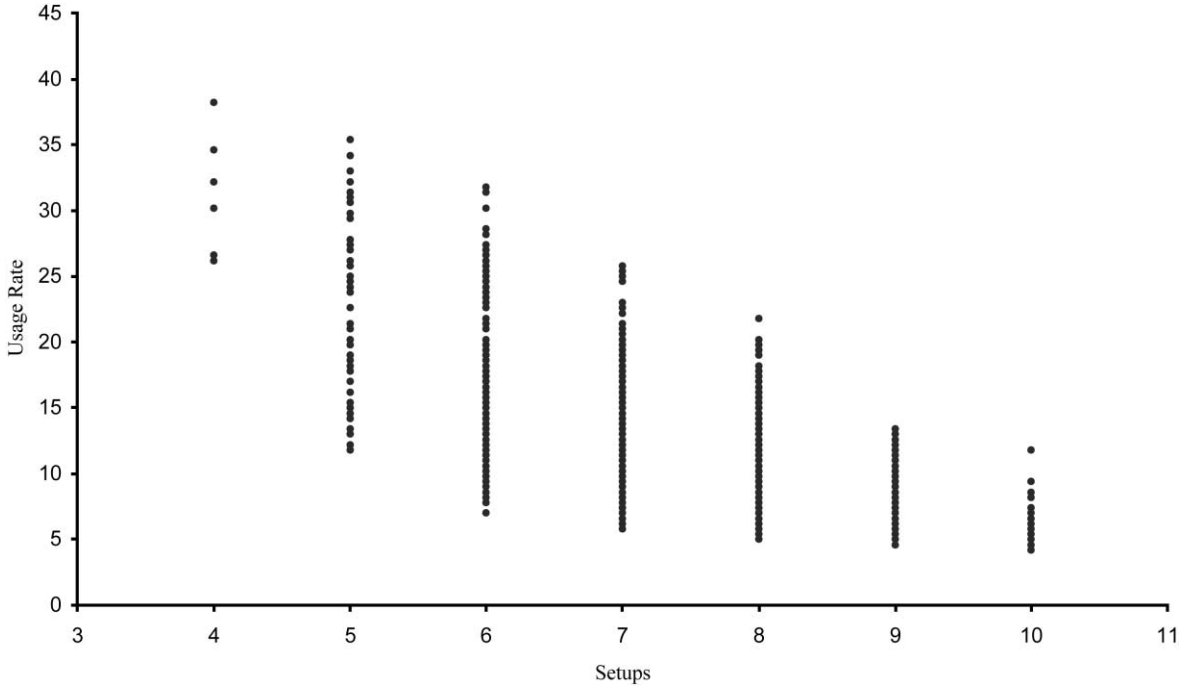

Fig. 2. All combinations for example problem.

lies. Another interesting feature of the scatter plot suggests that the required number of setups between (4) and (10) also show the greatest variation. It is also important, then, to appreciate the especially important need to strive for optimal usage rates at each of these intermediate setup values, because inferiority well above optimal usage for these intermediate setup values can be costly in terms of overall performance. These points are illustrated in Table 2, which gives usage information for each corresponding setup value (coefficient of variation is the ratio of standard deviation to mean) — intermediate setup values have the highest coefficient of variation. It should be mentioned that experimentation has shown that the type of behavior for this example problem as described above applies to all sequences evaluated in this research.

Fig. 3 shows the efficient frontier of the optimal solution obtain via complete enumeration compared to efficient frontiers obtained via the GA, SA and TS approaches.

Fig. 3 does not show the usage rates association with (4) setups so that more detail can be shown for other setups values. From inspection of Fig. 3, it appears that the SA approach comes closest to optimal for this example problem, followed by TS, and finally the GA approach. Table 3 shows efficient frontier information for these approaches.

The GA approach provides a frontier that is an average of 5.33% above the optimal frontier, while the SA and TS approaches provide frontiers that are 0.94 and 2.30% above the optimal frontier, respectively. These amounts above optimal are also referred to as ‘inferiority’. The GA approach provided usage rates equal to or superior to all but (22) of the 5040 possible solutions at the corresponding setup values, for a

Table 2
Breakdown of example problem's efficient frontier by setups

Setups	Optimal usage	Solution count	Percentage distribution	Mean usage	Standard deviation usage	Coefficient of variation
4	26.2	24	0.48	31.33	4.35	0.14
5	11.8	216	4.29	21.85	7.02	0.32
6	7	840	16.67	15.75	5.21	0.33
7	5.8	1560	30.95	12.11	4.03	0.33
8	5	1536	30.48	9.69	2.97	0.31
9	4.6	768	15.24	7.84	2.06	0.26
10	4.2	96	1.90	6.60	1.77	0.27
Total		5040	100.00			

percentile performance of 99.58. The SA and TS approaches provided percentiles of 99.90 and 99.84 respectively, with (5) inferiors for SA, and (8) for TS.

6. Experimental results

Prior to presenting actual results, it is appropriate to note that the search heuristics were written in C/C++ and executed on an Intel Pentium II, 300 MHz processor. For all three heuristics, CPU times for the largest problems (Problem Set 5, problem J, etc.) required less than two minutes of CPU time.

6.1. Results

Table 4 shows performance detail for the smaller problem sets (Problem Sets 1, 2 and 3, problems B–G). For each problem set and specific problem, the average inferiority is provided along with the percentile performance of the search heuristic of interest. The average inferiority is the average amount that the efficient frontier usage rate obtained via the search heuristic exceeds the efficient frontier usage rate obtained via enumeration for each setup on the frontier. The percentile performance is simply the percent of heuristic solutions found superior to the all solutions (at corresponding setup values) obtained via complete enumeration. Percentile performance is also presented here because it provides a compelling argument as to the near-optimality of the solutions offered by these heuristics.

Table 5 shows performance results for the larger problems (Problem Set 3, problems H–J, Problem Sets 4 and 5). Here, complete enumeration of all feasible solutions is not possible due to problem size, and subsequently, comparison to optimal efficient frontiers is simply not possible. Instead, the three search heuristics are compared to each other. Specifically, the efficient frontiers obtained via the Genetic Algorithm are used as a basis for comparison in the same way that the optimal efficient frontier was used as a basis for comparison above. For example, for Problem Set 4, Problem C, the Simulated Annealing approach provides an efficient frontier 3.57% superior to that of the Genetic Algorithm. Similarly, the Tabu Search approach provides an efficient frontier 4% superior to its Genetic Algorithm counterpart.

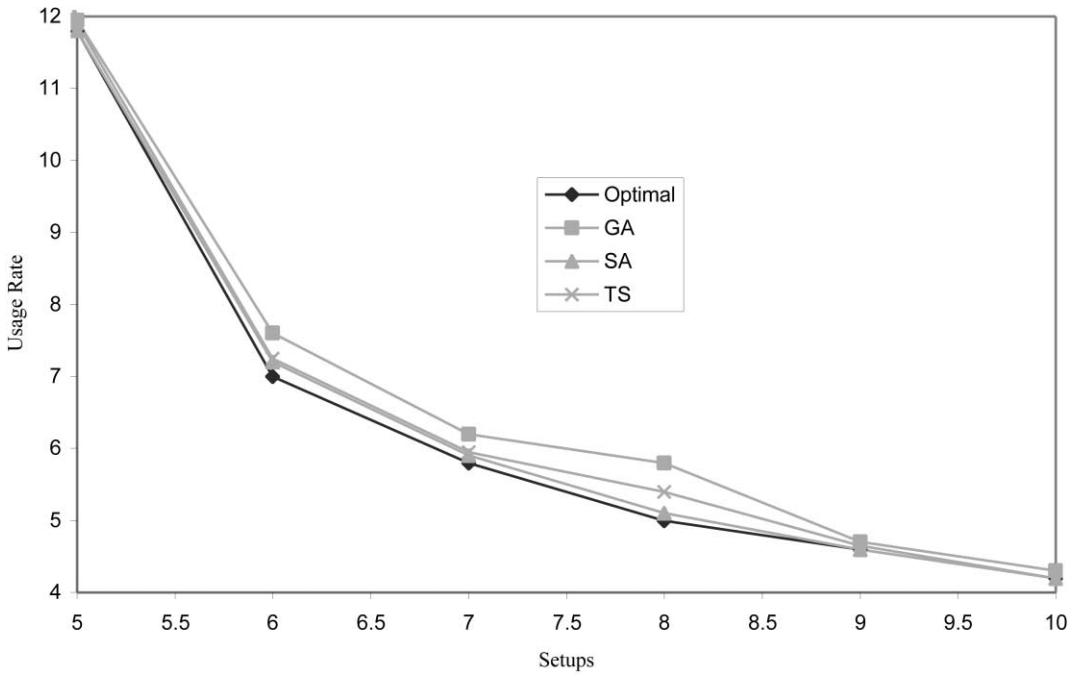

Fig. 3. Efficient frontier obtained via enumeration and search heuristics.

6.2. Discussion of results

For the smaller problems, Table 4 indicates that the three search heuristics provide near-optimal results in terms of percentile performance. In terms of mean inferiority, the Genetic Algorithm results in mean inferiority of 4.3%, while its Simulated Annealing and Tabu Search counterparts provide mean inferiority of 1.14 and 1.88%, respectively.

One thing noticeable is that the Genetic Algorithm performance is not as competitive as compared to its Simulated Annealing and Tabu Search counterparts in terms of either percentile or inferiority. This conjecture is supported by Tukey's pairwise comparisons, which classifies the Genetic Algorithm approach as statistically unique (inferior) from its Simulated Annealing and Tabu Search counterparts.

Table 3
Efficient frontier usage rates for search heuristics

Setups	Optimal	GA	SA	TS
4	26.2	26.2	26.2	26.2
5	11.8	11.95	11.8	11.9
6	7	7.6	7.2	7.25
7	5.8	6.2	5.9	5.95
8	5	5.8	5.1	5.4
9	4.6	4.7	4.6	4.65
10	4.2	4.3	4.2	4.2

Table 4
Performance of search heuristics compared to optimal for small problems

Prob.	Solutions	SA Inf.	SA %ile	TS Inf.	TS %ile	GA Inf.	GA %ile
Problem Set 1							
B	11,880	0.00	100	0.00	100	0.00	100
C	47,520	0.00	100	0.00	100	0.00	100
D	110,880	0.00	100.	0.00	100	0.00	100
E	166,320	0.00	100.	0.96	99.9928	1.65	99.9904
F	332,640	1.29	99.9964	0.29	99.9976	2.42	99.9868
G	498,960	1.32	99.9952	0.00	100	0.00	100
H	831,600	1.39	99.9962	2.27	99.9918	2.75	99.9865
I	415,800	0.69	99.9923	1.54	99.9923	1.50	99.9913
J	1,663,200	1.13	99.9957	2.01	99.9964	1.88	99.9964
Problem Set 2							
B	32,760	0.00	100	0.00	100	0.00	100
C	180,180	0.00	100	0.00	100	1.39	99.9967
D	600,600	0.00	100	0.23	99.9960	1.82	99.9860
E	2,162,160	0.35	99.9994	0.23	99.9994	1.85	99.9900
F	10,810,800	2.77	99.9979	4.56	99.9983	9.90	99.9869
G	25,225,200	3.09	99.9994	5.81	99.9989	11.24	99.9853
H	50,450,400	1.77	99.9997	1.19	99.9985	5.96	99.9894
I	126,126,000	3.03	99.9989	8.09	99.9966	3.61	99.9955
J	168,168,000	0.00	100	0.00	100	1.14	99.9974
Problem Set 3							
B	116,280	0.00	100	0.00	100	0.00	100
C	930,240	0.80	99.9884	0.00	100	2.57	99.9755
D	97,675,200	0.15	99.9999	0.66	99.9997	3.01	99.9979
E	1,396,755,360	4.06	99.9953	7.94	99.9995	20.12	99.9902
F	2,993,047,200	1.20	99.9999	3.08	99.9999	14.66	99.9981
G	19,554,575,040	4.21	$>$ 99.9999	6.29	99.9999	15.74	99.9960

The same Tukey's pairwise comparisons show that there is no significant difference between the Simulated Annealing and Tabu Search approaches for either performance measure of percentile or inferiority. All Tukey's pairwise comparisons were made at the $\alpha = 0.05$ level of significance.

Table 5 also supports the claim regarding the relative inferiority of the Genetic Algorithm approach when larger problems are investigated. Simulated Annealing provides an efficient frontier performance with a usage rate of 21.1% superior to that of the Genetic Algorithm approach across setups. Similarly, Tabu Search provides an efficient frontier with usage rates 19.7% superior to its Genetic Algorithm counterpart for each associated setup. These differences are significant (t-statistics of -4.94 and -5.09, respectively, which assumes that these distributions were normally distributed). It was also determined that there is no significant difference between the performances of the Simulated Annealing and Tabu Search approaches.

The relatively inferior performance of the Genetic Algorithm seems to worsen as the size of the problem gets larger. For the smallest problem set, the inferiority is perhaps not even noticed, but as the problem sizes increase, it is more noticeable. For the largest problem set, the inferiority of the Genetic

Table 5
Comparison of search heuristics to each other for larger problems

Prob.	Problem Set 3		Problem Set 4		Problem Set 5	
	SA superior (%)	TS superior (%)	SA superior (%)	TS superior (%)	SA superior (%)	TS superior (%)
B			0.31	0.31	25.62	23.47
C			3.57	4.00	34.78	30.78
D			3.53	3.53	36.47	32.90
E			4.85	4.91	38.31	34.69
F			3.39	3.85	42.63	38.79
G			8.21	8.55	44.23	41.27
H	9.48	10.94	6.34	5.50	52.67	47.88
I	9.48	10.94	8.58	9.15	50.13	47.53
J	7.11	6.22	0	0	52.60	49.12

Algorithm as compared to the other two search approaches is substantial. One possible reason for this is when modifications are made to the sequences during the search (crossover), a very large part of the production sequence is subject to modification. Conversely, modifications made to the sequences subjected to the Simulated Annealing and Tabu Search approaches have only two members of the sequence modified (specifically, swapped). As the production sequence becomes larger, this 'aggressiveness' of Genetic Algorithm crossover could perhaps have an adverse effect on efficient frontier performance — further investigation of this issue is an opportunity for future research. Other crossover/mutation mechanisms in an effort to improve Genetic Algorithm performance are another potential avenue for investigation.

Another issue important to note is with regard to the fact that most of the relative inferiority for all three heuristic approaches lies between the extreme setup values (this also occurs for solutions obtained via complete enumeration). In other words, all three heuristic approaches perform reasonably well in terms of usage rate for minimum and maximum setup values, but their performance suffers at intermediate setup values. Also, given the fact that the vast majority of all sequences lie at intermediate setup values, one must understand the need to approach optimal usage rates at these intermediate setup values. Otherwise, one could find their solution 'far away' from the efficient frontier. This point was made in Section 5.2, but is emphasized here due to its gravity. Experimentation shows that all three of the search heuristics are equally sensitive to this phenomenon.

7. Concluding comments

Attaining flexible production schedules while simultaneously minimizing the required number of setups is a challenging issue facing production management. This issue is challenging due to a few reasons. One of these reasons is that flexibility of production sequences and their associated number of setups is inversely related. Another complicating factor is that these two objectives imply a multiple objective type of problem, which complicates construction of an objective function. Yet another reason is that the combinatorial 'explosion' of possible sequences of such problems prohibits the use of enumerating all possible solutions in hopes of finding the 'best' ones.

In an attempt to find desirable production sequences in terms of flexibility for the associated number of

Table A1
Product mix and search parameters

Problem	Pr 1	Pr 2	Pr 3	Pr 4	Pr 5	SA *CR*	TS *Iter*	GA *Gen*
Problem Set 1								
B	8	1	1	1	1	0.9869	611	109
C	7	2	1	1	1	0.9886	702	125
D	6	3	1	1	1	0.9894	757	135
E	6	2	2	1	1	0.9898	783	140
F	5	3	2	1	1	0.9904	828	148
G	5	2	2	2	1	0.9906	855	153
H	4	3	2	2	1	0.9910	888	159
I	4	4	2	1	1	0.9905	843	151
J	3	3	2	2	2	0.9914	933	167
Problem Set 2								
B	11	1	1	1	1	0.9882	677	62
C	10	2	1	1	1	0.9899	788	72
D	9	3	1	1	1	0.9908	867	79
E	7	5	1	1	1	0.9916	950	86
F	7	3	2	2	1	0.9924	1055	96
G	6	3	3	2	1	0.9928	1110	101
H	5	3	3	3	1	0.9931	1155	105
I	4	3	3	3	2	0.9934	1215	110
J	3	3	3	3	3	0.9935	1234	112
Problem Set 3								
B	16	1	1	1	1	0.9895	760	32
C	15	2	1	1	1	0.9911	895	37
D	13	4	1	1	1	0.9933	1198	50
E	10	5	2	2	1	0.9942	1372	57
F	8	7	2	2	1	0.9944	1421	59
G	6	6	5	2	1	0.9948	1544	64
H	5	5	5	3	2	0.9952	1660	69
I	5	4	4	4	3	0.9953	1708	71
J	4	4	4	4	4	0.9953	1723	72

required setups, an efficient frontier approach is used. With this approach, a collection of points is monitored and managed so that for each possible number of setups associated with a sequence, the lowest usage rate is found (again, it is emphasized here that this usage rate is a surrogate for flexibility) via search heuristics. The three search heuristics of Simulated Annealing, Tabu Search and Genetic Algorithm used here provide near-optimal efficient frontiers for smaller problems, with the Genetic Algorithm being outperformed by the other two approaches. Additionally, for the larger problems, the Genetic Algorithm approach is out-performed by both the Simulated Annealing and Tabu Search approaches.

For practical implementations of this approach, management could obtain production sequences by first specifying a tolerable number of setups, and subsequently selecting a sequence resulting in the minimum usage rate for that particular number of setups.

Table A2
Product mix

Problem	Pr 1	Pr 2	Pr 3	Pr 4	Pr 5	Pr 6	Pr 7	Pr 8	Pr 9	Pr 10	Pr 11	Pr 12	Pr 13	Pr 14	Pr 15
Problem Set 4															
B	11	1	1	1	1	1	1	1	1	1					
C	10	2	1	1	1	1	1	1	1	1					
D	9	3	1	1	1	1	1	1	1	1					
E	8	4	1	1	1	1	1	1	1	1					
F	7	5	1	1	1	1	1	1	1	1					
G	6	5	2	1	1	1	1	1	1	1					
H	5	5	3	1	1	1	1	1	1	1					
I	4	4	4	2	1	1	1	1	1	1					
J	2	2	2	2	2	2	2	2	2	2					
Problem Set 5															
B	40	40	8	1	1	1	1	1	1	1	1	1	1	1	1
C	35	35	10	5	5	1	1	1	1	1	1	1	1	1	1
D	30	30	15	10	5	1	1	1	1	1	1	1	1	1	1
E	25	25	20	15	5	1	1	1	1	1	1	1	1	1	1
F	20	20	20	15	15	1	1	1	1	1	1	1	1	1	1
G	20	20	15	15	10	6	6	1	1	1	1	1	1	1	1
H	15	15	15	10	10	10	10	5	4	1	1	1	1	1	1
I	15	15	10	10	10	10	10	10	4	1	1	1	1	1	1
J	7	7	7	7	7	7	7	7	7	7	6	6	6	6	6

Table A3
Search parameters

Problem	SA *CR*	TS *Iter*	GA *Gen*
Problem Set 4			
B	0.9950	1618	67
C	0.9954	1729	72
D	0.9956	1807	75
E	0.9957	1860	78
F	0.9958	1891	79
G	0.9959	1972	82
H	0.9960	2017	84
I	0.9962	2092	87
J	0.9965	2306	96
Problem Set 5			
B	0.99907	8631	12
C	0.99920	10084	14
D	0.99926	10855	15
E	0.99929	11251	15
F	0.99932	11787	16
G	0.99937	12704	17
H	0.99942	13788	19
I	0.99942	13950	19
J	0.99950	15999	22

One common attribute shared by these three approaches is that they use a stochastic process to guide the search — 'parts' of the sequence at hand are randomly chosen for modification. This is not the only way that a search can be guided. Another way is to pursue the sequential structure of the problem via search trees — depth-first, breadth-first or beam-search approaches (which includes the possibility of using branch and bound approaches). With these tree-oriented search processes, the stochastic aspects of search are not necessary, only traversal of the search tree (Russell & Norvig, 1995). While depth-first and breadth-first searches are completely enumerative searches, beam-search is not and could be considered a candidate for future research efforts for the problem type presented here. Beam-search is promising here because it only branches on nodes in the tree-search that show the most potential for desirable objective functions (Winston & Horn, 1988), therefore it is not an exhaustive search, which offers computational efficiency.

Appendix A

Tables A1–A3.

References

Abranovic, W. A. (1997). *Statistical thinking and data analysis methods for managers*, Reading, MA: Addison-Wesley.

Baykasoglu, A., Owen, S., & Gindy, N. (1999). A taboo search based approach to find the Pareto Optimal set in multiple objective optimization. *Engineering Optimization*, *1* (1), 1–18.

Bland, J. A., & Dawson, G. P. (1991). Tabu Search and design optimization. *Computer-Aided Design*, *23*, 195–201.

Cheng, F. Y., & Li, D. (1998). Genetic algorithm development for multiobjective optimization of structures. *AIAA Journal*, *33* (1), 59–85.

Davis, L. (1985). Applying adaptive algorithms to epistatic domains. *Proceedings of the International Joint Conference in Artificial Intelligence*, 162–164.

Ding, F., & Cheng, L. (1993). An effective mixed-model assembly line sequencing heuristic for just-in-time production systems. *Journal of Operations Management*, *11* (1), 45–50.

Eglese, R. W. (1990). Simulated annealing: a tool for operational research. *European Journal of Operational Research*, *46*, 271–281.

Gaither, N., & Frazier, G. V. (1999). *Production and operations management*, . (8th ed)Cincinnati, OH: Southwestern Publishing.

Glover, F. (1990). Tabu Search: a tutorial. *Interfaces*, *20* (1), 74–94.

Glover, F. (1993). A user's guide to Tabu Search. *Annals of Operations Research*, *41* (1), 3–28.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*, Reading, MA: Addison-Wesley.

Hooker, J. N. (1995). Testing heuristics: We have it all wrong. *Journal of Heuristics*, *1* (1), 33–42.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220* (4598), 671–679.

McMullen, P. R. (1998). JIT sequencing for mixed-model assembly lines with setups using tabu search. *Production Planning and Controlling*, *9* (5), 504–510.

McMullen, P. R., & Frazier, G. V. (2000). A simulated annealing approach to mixed-model sequencing with multiple objectives on a JIT line. *IIE Transactions*, *32* (8), 679–686.

Metropolis, N., Rosenbluth, A., Rosenbluth, N., Teller, A., & Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, *21*, 1087–1092.

Michalewicz, Z. (1996). *Genetic algorithms + data structures = evolution programs*, New York: Springer.

Miltenburg, J. (1989). Level schedules for mixed-model assembly lines in just-in-time production systems. *Management Science*, *35* (2), 192–207.

Monden, Y. (1983). *Toyota production system*, Norcross, GA: The Institute of Industrial Engineers.

Oliver, I. M., Smith, D. J., & Holland, J. R. C. (1987). A study of permutation crossover operators on the traveling salesman problem. *Proceedings of the Second International Conference on Genetic Algorithms*, 224–230.

Russell, S., & Norvig, P. (1995). *Artificial intelligence: a modern approach*, Upper Saddle River, NJ: Prentice-Hall.

Sumichrast, R. T., & Russell, R. S. (1990). Evaluating mixed-model assembly line sequencing heuristics for just-in-time production systems. *Journal of Operations Management*, *9*, 371–390.

Suppapitnarm, A., Seffen, K. A., Parks, G. T., & Clarkson, P. J. (2000). A simulated annealing algorithm for multiobjective optimization. *Engineering Optimization*, *1* (8), 1–27.

Wantuck, K. A. (1989). *Just-in-time for America*, Southfield, MI: KWA Media.

Winston, P. H., & Horn, B. K. P. (1988). *LISP*, . (3rd ed.)Reading, MA: Addison-Wesley.