INT. J. PROD. RES., 2000, VOL. 38, NO. 12, 2653–2670

Using genetic algorithms to solve the multi-product JIT sequencing problem with set-ups

PATRICK R. McMULLEN†*, PETER TARASEWICH‡ and GREGORY V. FRAZIER§

This paper presents a methodology to solve the Just-in-Time (JIT) sequencing problem for multiple product scenarios when set-ups between products are required. Problems of this type are combinatorial, and complete enumeration of all possible solutions is computationally prohibitive. Therefore, Genetic Algorithms are often employed to find desirable, although not necessarily optimal, solutions. This research, through experimentation, shows that Genetic Algorithms provide formidable solutions to the multi-product JIT sequencing problem with set-ups. The results also compare favourably to those found using the search techniques of Tabu Search and Simulated Annealing.

1. Introduction

The benefits of successful Just-in-Time (JIT) production systems—such as reduced inventory levels, shorter lead times, and improved responsiveness—are well documented. One important part of successful JIT implementation, production scheduling, is the focus of this research. Specifically, this research is concerned with sequencing production or assembly lines where multiple products must be made simultaneously in an intermixed sequence. This is often referred to as *mixed-model scheduling*.

In mixed-model production systems, managers would ideally like to sequence the different products as evenly dispersed as possible, but without incurring an excessive number of set-ups due to switching between different products (assuming set-up times are non-negligible). A good sequence of products should not only have an acceptable level of product inter-mixing but also an acceptable number of required set-ups. In this environment, the sequencing decision becomes a multi-objective problem.

In this research, two specific sequencing objectives are examined: (1) the number of required set-ups, and (2) the usage rate. A set-up is required each time two consecutive items in the production sequence are different. The usage rate is a measure of the company's ability to keep the schedule level, or evenly inter-mixed, by keeping the raw materials for the different products arriving at the system at as constant a rate as possible. Because JIT systems are concerned with having the

Revision received January 2000.

†Auburn University, College of Business, Department of Management, Auburn University, Alabama 36849, USA.

‡University of Maine, Maine Business School, Orono, Maine 04469-5723, USA.

§The University of Texas at Arlington, College of Business Administration, Department of Information Systems and Management Sciences, Arlington, TX 76019-0377, USA.

*To whom correspondence should be addressed. e-mail: pmcmullen@business.auburn.edu

International Journal of Production Research ISSN 0020–7543 print/ISSN 1366–588X online
http://www.tandf.co.uk/journals

right parts at the right place at the right time, sequencing must be done so raw materials are introduced into the system at a fairly uniform rate. Miltenburg (1989) presented a metric to measure this usage rate, which is adopted for this research. This metric is based upon Monden's (1983) idea of level-scheduling. With both the number of set-ups and Miltenburg's usage rate metric, smaller values are more desirable.

Production schedules or sequences that provide a strong level of product intermixing clearly will require more set-ups. In addition to this trade-off, another issue that complicates the sequencing problem is its combinatorial nature. Typically, an enormous number of possible production sequences exist, even for relatively small problems, so that finding the optimal solution is usually impractical. For this type of optimization problem, search heuristics have been shown in the literature to be successful. In this research, a heuristic search technique known as a Genetic Algorithm is used to address this type of multiple objective sequencing problem. Genetic Algorithms are unique from most other types of heuristic search procedures in several ways. First, they sometimes replace a current solution with an inferior solution in an attempt to avoid getting trapped at local optima. Secondly, they use desirable solutions to construct new solutions.

After the presentation of the methodology, the Genetic Algorithm technique is implemented to address a set of sequencing problems from the literature. The resulting production sequences are then simulated so that production performance measures can be examined. These simulated results are then compared with the simulated results for the same set of problems from two other modern search techniques: Tabu Search and Simulated Annealing. These comparisons attempt to shed some light on the relative performance of the Genetic Algorithm. General conclusions are then offered.

2. JIT sequencing and combinatorial problems

The following variables are defined for the JIT sequencing problem with set-ups:

- U usage rate of a production sequence,
- S number of set-ups in a production sequence,
- Z objective function value of a production sequence,
- a number of unique products to be produced,
- D_T total number of units for all products or total demand—also represents number of positions in sequence,
- d_i demand for product i, $i = 1,2,\ldots,a$,
- s_k 1 if set-ups are required; 0 if not,
- $x_{i,k}$ total number of units of product i produced over stages 1 to k, where $k = 1,2,\ldots,D_T$.

The first objective addressed here is set-up minimization. The number of set-ups (S) in a production sequence is:

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

where k is the current position in the sequence. If the product in position k is different from the product in position $k-1$, then a set-up is required and $s_k = 1$. Otherwise, a set-up is not required and $s_k = 0$. It is assumed here that an initial set-

up is required regardless of sequence. It should be noted that the set-up times are assumed to be sequence-independent, so the set-up time for a product on a machine does not depend on which product preceded it on that machine. In this study, the number of set-ups required is used as a simplifying surrogate for the amount of set-up time required.

The second objective addressed here is minimization of the usage rate (U), as presented by Miltenburg (1989). The computation of the usage rate is:

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

To illustrate, consider a situation where four products need to be put into a production sequence. The products are A, B, C, and D. The demand for product A is three units, B is two units, C is one unit and D is one unit. Because seven units are demanded ($3+2+1+1=7$), there are seven positions in the production sequence. The sequence that would minimize the required number of set-ups would be: AAABBCD, resulting in four set-ups (including one for the first product in the sequence). According to formula (2) above, the usage rate would be 11.71 for this sequence. Using the technique of Ding and Cheng (1993), the sequence that would minimize the usage rate would be: ABCADBA, with seven set-ups required and a usage rate of 2.86. At this point, the reader should notice a trade-off between usage rate and set-ups.

If seven distinct items (i.e. ABCDEFG) were to be sequenced, there would be $7! = 5040$ unique sequences, or permutations, possible. However, in the example sequence above (AAABBCD), not all of the items are distinct, which substantially reduces the number of unique sequences possible. Because there are 3 As, 2 Bs, 1 C and 1 D, the following calculation determines the number of possible unique sequences:

$$\frac{7!}{3!2!1!1!} = 420.$$

The general formula to compute the number of permutations of a multiset (Walpole and Myers 1985) is:

$$\frac{n!}{n_1!n_2!\ldots n_n!}, \quad \text{where } n_1 + n_2 + \cdots + n_a = n. \tag{3}$$

If the demand for each of the four items was doubled: six units of item A, four units of item B, and two units each of items C and D, then the number of possible unique sequences would be:

$$\frac{14!}{6!4!2!2!} = 1\,261\,260.$$

This simple example shows that small increases in the size of the problem result in very large changes in the number of possible unique sequences, a trademark of combinatorial problems. Problems with a large number of possible solutions usually cannot be solved to optimality within a reasonable amount of time. Given this, a heuristic, based on Genetic Algorithm techniques, is developed to find production sequences that perform well with regard to both the number of set-ups and the usage rate.

3. Genetic Algorithms and combinatorial optimization

Genetic Algorithms, which simulate the natural behaviour of biological systems, can be adapted to solve combinatorial optimization problems. In nature, the 'fittest' members of a population typically survive at higher rates compared to the 'weakest' members. These fittest members then reproduce with one another, resulting in a new generation of the population having attributes similar to that of their parents. In each new generation, mutations can also occur on an infrequent basis. In this situation, offspring develop attributes of their own, independent of their parents. Mutations can add diversity and a certain amount of robustness to the population.

These natural phenomena can be exploited to find good solutions to combinatorial optimization problems. Current solutions with the most desirable objective function values (the 'fittest' members of the solution population) are 'crossed' with one another. This crossing is analogous to reproduction, resulting in a new generation of solutions having attributes similar to that of the previous generation (the parent solutions). As is also the case in nature, these solutions can also undergo mutation, producing solutions that have their own traits. This is done so that diversity is incorporated into the 'gene pool'. The goal here is eventually to find generations of solutions to a combinatorial optimization problem where the objective function value approaches the global optimum.

4. Genetic Algorithm methodology

The actual Genetic Algorithm is quite straightforward. First of all, the decision-maker specifies the following input criteria: (1) the number of generations (G), (2) the number of randomly-generated initial solutions (Sol), (3) the number of solutions from each generation to be used for crossover ($Cross$), and (4) the probability of mutation (P_m). After these criteria are specified, an initial set of solutions is randomly generated. For each solution generated, the objective function, Z, is determined. The 'fittest' of these initial solutions are selected and crossovers are performed, resulting in a new generation. These new 'offspring' will undergo mutation with the probability of P_m. The 'fittest' of this new generation will then be chosen for the next set of crossovers and mutations, and this process is repeated until the desired number of generations has been produced. For each generation, the solution with the best objective function value is noted. Also recorded for each new generation is the best overall solution found to that point. At the completion of the algorithm, the overall best solution in terms of the objective function is presented.

The following is a pseudocode presentation of the Genetic Algorithm:

```
Input Parameters (G, Sol, Cross, P_m)
Generate Initial Solutions
For I = 1 to G
    Choose Best Solutions
    Perform Crossovers
    For H = 1 to New Offspring
        Determine Z(H)
        If Random Number < P_m then Perform Mutation
    Next H
    Note Best of Generation
    If Z(Best of Generation) is better than Z(Best Found thus Far) Then
            Replace Best Found thus Far with Best of Generation
```

Endif
Next *I*
Present Best Found thus Far

It is worth noting that Goldberg (1989) has an informative, yet highly readable, text pertaining to the basics of Genetic Algorithms.

4.1. *Crossover example*

To illustrate how a crossover works, consider the following two parent sequences:

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

Brackets designate the portion of the sequences that will remain intact and become part of the offspring in the crossover process. The location of the brackets is randomly determined, but the left bracket must be to the right of the first character in a sequence, and the right bracket must be to the left of the last character in the sequence. A new parent is then created by moving all the characters appearing after the right bracket (of the original parent) to the beginning of the sequence. The results of this are shown below.

Parent 1′: **B** C C D D **A A** A A A A **B** B B (C D D A A A A B B)

Parent 2′: **B** C **A** D **A** D A **B** A **B** C B A A (C D D A A C B A A)

From this new sequence, characters that match the characters between the brackets of the *other* original parent are removed (the characters to be removed are shown in bold). For example, from Parent 1′, the first two As, the first two Bs, and the first C are removed because Parent 2 had the sequence BCBAA between its brackets. The sequence between the brackets of the original parent and this shortened list (shown in parentheses above) from the other parent is then used to construct an offspring. For example, the sequence AABBB is taken from Parent 1, and the shortened list from Parent 2′ is added, starting at the right bracket and wrapping around to the beginning of the sequence. Using this technique, the following offspring result.

Offspring 1:	C	B	A	A	A	A	B	B	B	C	D	D	A	A
Offspring 2:	A	A	B	B	B	C	B	A	A	C	D	D	A	A

Notice how the some of the characters of each offspring match that of their parents. This technique is referred to as order crossover (OX Michalewicz 1996, Davis 1985). Other Genetic Algorithm crossover techniques available for general sequencing problems are partially-mapped crossovers (PMX—Goldberg and Lingle 1985) and cycle crossovers (CX—Oliver *et al.* 1987). Michalewicz (1996) provides elaboration of these techniques.

For each match, there will be two offspring. Consider an example where four sequences (A, B, C and D) can be matched for crossover:
An 'x' denotes a situation where a crossover will be made. A '–' denotes a situation where a sequence, or solution could be crossed with itself, but will not be, despite its feasibility. Crossovers will not be performed here simply because new sequences will not result. For example, crossing sequence C with itself results in sequence C, which adds nothing new to the search. Since there are two offspring for each match, the

	A	B	C	D
A	–			
B	x	–		
C	x	x	–	
D	x	x	x	–

following formula shows the number of offspring from n solutions chosen for cross-over.

$$\text{New Offspring} = 2\sum_{i=1}^{n-1} i = n(n-1). \tag{4}$$

4.2. *Mutation example*

As stated earlier, mutation is sometimes performed so that a solution (sequence) has an occasional trait that is unique from its parents. This is essentially done so that diversity remains in the gene pool, where the diversity component contributes to robustness in the solution population. Mutation for this research is the simple swapping of two unique elements in the sequence of interest. Consider the following sequence:

Before Mutation: A A A A A A B B B B C C D D.

The two underlined elements are randomly selected unique elements that are targeted for swapping. After swapping, or mutation, the sequence is as follows:

After Mutation: A A A B A A B B B A C C D D.

4.3. *Objective functions*

For each sequence obtained, its objective function value is determined. For this research, one of five different objective functions is selected by the decision-maker. The objective functions are as follows.

The first objective finds the sequence that minimizes the number of required set-ups (as determined in equation (1)):

$$Z_1 = S = 1 + \sum_{k=2}^{D_T} s_k. \tag{5}$$

The second objective finds the sequence that minimizes the sequence usage rate. The methodology used for this objective is the result of the Ding and Cheng (1993) methodology.

$$Z_2 = U = \sum_{k=1}^{D_T}\sum_{i=1}^{a}\left(x_{i,k} - k \cdot \frac{d_i}{D_T}\right)^2. \tag{6}$$

The third objective attempts to find the sequence that minimizes a composite function of both set-ups and usage rate, and each of these attributes are equally weighted (details of weighting issues are forthcoming).

$$Z_3 = w_S S + w_U U. \tag{7}$$

The fourth objective attempts to find the sequence that minimizes a composite function of both set-ups and usage rate, where set-ups are weighted three times as much as usage.

$$Z_4 = 3w_S S + w_U U. \tag{8}$$

The fifth and final objective attempts to find the sequence that minimizes a composite function of both set-ups and usage, where usage is weighted three times as much as set-ups.

$$Z_5 = w_S S + 3w_U U. \tag{9}$$

It is important to note that the first two objectives do not require a Genetic Algorithm—the sequences obtained from these objectives come directly from application of a heuristic that respectively minimizes the number of required set-ups and the usage rate. The last three objectives, however, do require a Genetic Algorithm. The weights, w_S and w_U, used for these objective functions are intended to reflect the level of importance that the attributes of set-ups and usage rates have with respect to the objective of interest. The weights are determined as follows:

$$w_S = 1000/\text{number of set-ups from initial solution} \tag{10}$$

$$w_U = 1000/\text{usage rate from initial solution} \tag{11}$$

For objectives three and four, the 'initial solution' used for weight determination is the one where the number of set-ups is minimized (objective three places equal weight on both set-ups and usage, while objective four places more weight on set-ups). For objective five, the 'initial solution' used for weight determination is the one where the usage rate is minimized (objective five places more emphasis on usage).

4.4. *Example problem*

Consider again, the problem where there are six units of A demanded, four or B, and two each for C and D. One possible production sequence is: AAAAAABBBBCCDD—one that minimizes the number of set-ups. This particular production sequence results in four set-ups and a usage rate of 88.86 obtained from Miltenburg's formula in (2). Now consider the following multiple objective function, where both the number of set-ups and the usage rate are considered simultaneously:

$$\text{Min:} \quad Z = w_S S + w_U U.$$

The set-ups weight, w_S, and the usage weight, w_U, could be determined such that both set-ups and usage make equal contributions to the multi-criteria objective function for this solution (objective 3 from above). This could be done where $w_S = 1000/4$, and $w_U = 1000/88.86$, which means that the objective function value to this initial solution would be 2000. Hence:

$$Z = \frac{1000}{4} \cdot 4 + \frac{1000}{88.86} \cdot 88.86 = 2000.$$

This objective function and weighting scheme are used for the example problem. The Genetic Algorithm outlined above is used to solve the example problem with the following parameters: 10 randomly generated initial sequences (*Sol*), 6 extracted sequences for each generation (*Choose*), a mutation probability (P_m) of 6%, and 100 generations (G). The chart in figure 1 shows the objective function value for each of 100 generations of a test run.

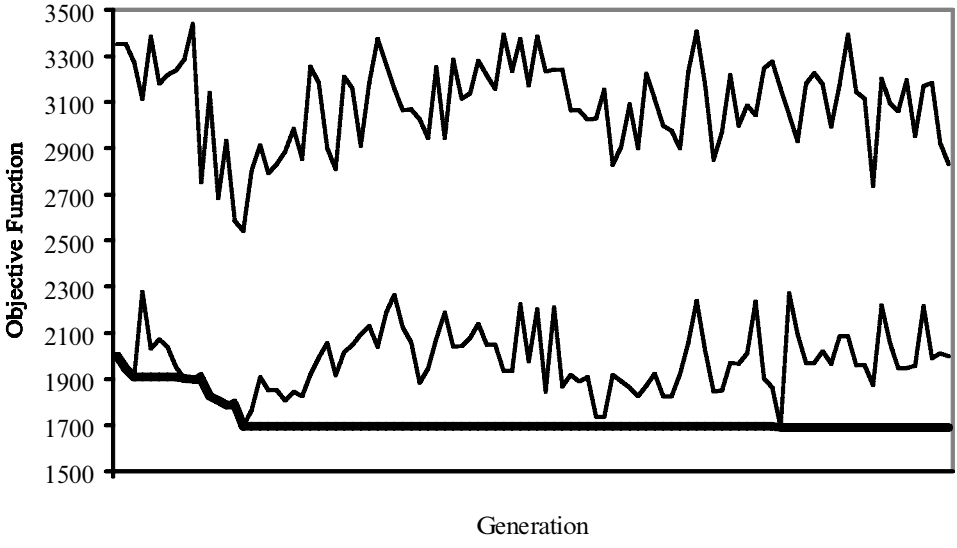

Figure 1. Generation history of example problem.

The bold line is the objective function value of the overall 'best' solution found so far through each generation—this value will only decrease from generation to generation. The two fine lines represent the objective function values of the 'best' and 'worst' solutions for each generation. Something interesting to note here is that the best of generation line 'meanders', trying to find the global optima, because it frequently explores inferior solutions caused by crossover and mutation. This is a characteristic that sets metaheuristics apart from more naïve local heuristics. The best solution that the Genetic Algorithm found for this problem was the sequence AAACCDDBBBBAAA, with an objective function value of 1690.51. When complete enumeration was used to evaluate all of the 1 261 260 sequences, the sequence resulting in the globally optimal condition was AAACCBBBBDDAAA, with an objective function value of 1664.78. Therefore, in terms of the objective function, the Genetic Algorithm solution was within 98.45% of optimal. Furthermore, when complete enumeration was performed, only 10 solutions (of the 1 261 260 possible) had objective function values that were superior to that obtained from the Genetic Algorithm. That means that the Genetic Algorithm objective function value is better than 99.9992% of all other solutions for this problem—an obviously strong argument in support of the performance of Genetic Algorithms.

5. Design of the experiment

To compare the results of sequencing according to a Genetic Algorithm with that of other search techniques, an experiment is presented. Specifically, the results of Genetic Algorithm sequencing for a sample problem set (Sumichrast and Russell 1990) are compared with sequencing the same problem set via Tabu Search (McMullen 1998) and Simulated Annealing (McMullen and Frazier 1998, 2000) from earlier published work. The relative 'desirability' of each approach is measured across several performance criteria as follows.

- Required set-ups: the number of set-ups required from a production sequence (from equation (1)).
- Usage rate: the measure of how uniformly raw materials are being introduced to the production flow (from equation (2)).
- Makespan: the length of a simulated production run as the result of the sequence.
- Average work-in-process inventory level: the average number of units in the simulated production system at any given time as the result of the production sequence.
- Average flow-time: the average amount of time a job spends in the simulated production system.

The first two performance criteria are associated with the sequence itself, while the last three are measures of how well the production sequences perform with respect to simulated production conditions.

Simulated Annealing borrows from the physical annealing of solids—heating a solid to a high temperature and slowly cooling it until desired properties of the solid are obtained. This technique of introducing 'chaos' into a solid can also be used to introduce 'chaos' into a combinatorial optimization problem such that it obtains desired properties—in this case, the objective function value. Tabu Search is a memory-based procedure where recent solutions are documented. If, during the search process, a solution is found that is similar to a recent, documented solution, the solution of interest is forbidden. This tabu status of such a solution can be overridden if it shows potential for reasonable improvement. Detailed discussion of Tabu Search and Simulated Annealing are beyond the scope of this paper, but the interested reader is referred to Glover (1990, 1993) for an introduction to Tabu Search and to Eglese (1990) and Kirkpatrick *et al.* (1983) for introductions to Simulated Annealing. Although they are different approaches, both will occasionally accept relatively inferior solutions in an attempt to avoid being trapped at local optima.

5.1. *Research questions*

To determine the formidability of the Genetic Algorithm presented here, the following research question is presented.

- Are the performance criteria presented above sensitive to the sequencing heuristics of Genetic Algorithms, Tabu Search and Simulated-Annealing? If so, which ones are most preferred?

As a secondary question, it is desired to see if the performance criteria are sensitive to the five differing objectives presented above. Specifically:

- Do the user-chosen objectives have an effect on the performance measures? If so, which perform best with regard to the various criteria?

These questions are addressed via multivariate analyses of variance (MANOVA), with appropriate univariate analyses of variance and Tukey's pairwise comparisons as follow-ups. Additionally, the first two objectives (the ones exclusively addressing number of set-ups and usage rate respectively) will be omitted from these analyses of variance. The reason for this is to isolate the three composite, search-based objectives so that group differences in their means are more easily interpreted.

Search parameters	Problem set 1	Problem set 2	Problem set 3
Randomly generated sequences (*Sol*)	25	25	25
Sequences used for crossover (*Cross*)	5	6	6
Probability for mutation (P_m)	6%	6%	6%
Generations (G)	30	85	85
Total products (Problem Size)	20	20	100
Product types (Problem Size)	5	10	15

Table 1. Search parameters for genetic algorithm.

5.2. *Genetic Algorithm search parameters*

Table 1 lists parameters used for the Genetic Algorithm search. Experimentation was used to find the probability of mutation (P_m) resulting in the most desired objective function values. P_m values of 2%, 4%, 6%, 8% and 10% were explored. Other Genetic Algorithm search parameters (*Sol*, *Cross*, G) were selected such that the total number of solutions evaluated for the three search heuristics (Genetic Algorithm, Simulated Annealing and Tabu Search) are the same for each problem evaluated. This contributes to a fair comparison of heuristics. For all three search heuristics, search parameters are selected such that each heuristic performs in its best possible light. Prior research (McMullen and Frazier 2000) details the efforts used in finding search parameters for the Simulated Annealing and Tabu Search parameters.

It is also appropriate to note that 12 additional test problems were used to assist with determination of the search parameters used above. These problems range from as few as 10 items in a sequence to as many as 180 items in a sequence. These experimental problems themselves were not found to have an effect on the objective function of interest ($F = 0.22$, $p = 0.9270$) and were subsequently not considered as an experimental factor here. Instead, the three test problems from the literature (Sumichrast and Russell 1990) are used for detailed analysis. Appendix 3 details these 12 additional problem sets.

5.3. *Simulated production runs*

As previously stated, the sequences obtained from the Genetic Algorithm methodology are used as production sequences for simulated production runs. The software used for these simulated production runs was SlamSystem along with user-written inserts in FORTRAN v 5.1. For each sequence obtained via the Genetic Algorithm, the sequence was repeated five times for the simulated production run. This meant that the two smaller problem sets resulted in the simulated production of 100 units (there are 20 products involved in each sequence), and the largest problem set resulted in the simulated production of 500 units (there are 100 products involved in each sequence).

Relevant assumptions associated with the simulated production runs are detailed as follows.

- A pull system is used.
- Parts scheduled for production are dispatched in a deterministic fashion.
- The production sequence is repeated five times when the problem sets are simulated.
- Each unit is processed through seven different resources.

- The process time for each resource is assumed to be a normally distributed random variable with a standard deviation being 15% of its expected value.
- The expected process time for each different product through each resource is independent of process times for other units.
- When set-ups are required between differing products, the expected duration of the set-up is a normally distributed random variable, where the mean is 20% of the resource requiring the set-up, with a standard deviation being 15% of its mean.

5.4. *Database and computational experience*

For the problem set showing the sample problems in Appendix 1, note that there are (3) differing problem sizes. For each size problem, there are (9) separate problems having differing product mixes. Each separate problem will be addressed via the (5) differing objectives. For each of these solutions, the simulated production run will be replicated (25) times, so reasonable estimates of the means of the performance measures can be obtained. This results in $(3) \times (9) \times (5) \times (25) = 3375$ observations in the database, but only rules 3, 4 and 5 are interpreted in the Analyses of Variance. The database also contains performance data for the production sequences obtained from both Tabu Search and Simulated Annealing, structured in the same fashion as above, giving a total of $(3) \times (3375) = 10\,125$ observations.

The Genetic Algorithm was developed using Microsoft's Visual Basic and was run on a Pentium II processor (300 MHz). Average CPU times for the three differing problem sizes are presented in table 2, along with comparisons of CPU times for the Simulated Annealing and Tabu Search heuristics.

From table 2, it is clear that the Genetic Algorithm sequencing approach is more CPU intensive than its Simulated Annealing and Tabu Search counterparts, despite the fact that the GA approach processes slightly fewer solutions than SA. The reason for the higher CPU requirement lies in the fact that the GA approach manipulates more members of the production sequence than the SA and TS approaches. Because the SA and TS approaches involve only pairwise 'swapping' of sequence members, process time is comparatively minimal. Despite the fact that the GA approach requires slightly more CPU time, the authors are not overly concerned due to the availability of high-speed computing.

6. Experimental results

Prior to addressing the individual research questions, it should be noted that due to the high correlation between performance measures, some are omitted and subsequently explained by others. Specifically, average WIP level, system flow-time and the number of set-ups were highly correlated with makespan (correlations of 0.98, 0.99, and 0.92 for each problem set, respectively). As a result, these three variables

	Problem set 1	Problem set 2	Problem set 3
Genetic Algorithm	0.12 (300)	0.84 (1,275)	6.10 (1,275)
Simulated annealing	0.09 (320)	0.55 (1,280)	2.38 (1,920)
Tabu search	0.02 (variable)	0.20 (variable)	2.38 (variable)

Table 2. Listing of CPU time in minutes (and number of solutions evaluated).

will be dropped from further analysis, and will be explained by makespan. It should also be noted, from this point forward, that all relevant statistics will be reported for each of the three problem sets, so that the large variation in size across problem sets does not bias interpretation.

The first research question deals with the effect that the sequencing heuristic has on the outputs of makespan and usage. For all three problem sets, the search heuristic does have an overall multivariate effect on the both makespan and usage. Presentation of the multivariate and univariate statistics, as well as order of preference for each response is presented in table 3.

The univariate statistics in table 3 suggest that the performance measure of usage contributes more to the overall multivariate effect than does makespan. As a follow-up to this, table 4 presents means of makespan and usage for each problem set organized by a search heuristic.

Problem set	Wilks' lambda	Multi-F	Make (F)	Make Pref.	Usage (F)	Usage Pref.
1	0.7193	180.96	11.81	TS > GA > SA	105.21	TS > GA > SA
2	0.8184	106.49	21.58	TS > GA > SA	39.47	TS > SA > GA
3	0.6707	223.34	114.94	GS > SA > TS	438.19	TS > GA > SA

Table 3. Multivariate and univariate statistics for effect of search heuristic on performance measures.

Heuristic	Problem set 1	Problem set 2	Problem set 3
GA Makespan	18.92 (0.94)	19.33 (0.64)	88.24 (6.47)
GA Usage	98.30 (89.08)	60.10 (31.22)	9556.06 (8726)
SA Makespan	18.92 (0.91)	19.37 (0.66)	88.60 (6.64)
SA Usage	99.40 (90.20)	53.99 (23.96)	10 900 (8649)
TS Makespan	18.75 (0.32)	19.18 (0.29)	92.36 (2.47)
TS Usage	47.34 (30.99)	47.89 (19.01)	396.75 (165.37)

Table 4. Means (and standard deviations) of performance measuers by problem set and search heuristic

Problem Set	Wilks' lambda	Multi-F	Make (F)	Make Pref.	Usage (F)	Usage Pref.
1	0.7217	178.99	5.165	4 > 3 > 5	168.97	5 > 3 > 4
2	0.6795	215.40	6.301	4 > 3 > 5	171.20	5 > 3 > 4
3	0.6078	285.63	0.595*	4 > 3 > 5	114.49	5 > 3 > 4

Make refers to makespan. The associated p-values for all F-statistics were < 0.001 (* with the single exception of Problem Set 3, makespan, which is not significant at the $\alpha = 0.05$ level). Underlines for preferences show differences not significant at the $\alpha = 0.05$ level of significance when using Tukey's Pairwise comparisons.

Table 5. Multivariate and univariate statistics for effect of objective function on performance measures.

The second research question addresses the effect that the objective has on the performance measures of makespan and usage. Table 5 shows multivariate and univariate statistics, as well as the order of preference that the objectives have on these two performance measures.

Objective	Problem set 1	Problem set 2	Problem set 3
3 (Makespan)	18.53 (0.35)	19.12 (0.32)	86.96 (3.98)
3 (Usage)	86.02 (69.10)	54.49 (12.60)	7692 (6393)
4 (Makespan)	18.34 (0.30)	18.88 (0.30)	85.90 (3.13)
4 (Usage)	142.77 (76.99)	76.88 (27.04)	12 961 (9652)
5 (Makespan)	19.72 (0.72)	19.88 (0.47)	96.34 (3.27)
5 (Usage)	16.24 (7.48)	30.62 (4.78)	199.09 (38.58)

Table 6. Means (and standard deviations) of performance measures by problem set and objective.

Problem Set 1:			
Measure	Rule 3	Rule 4	Rule 5
GA Makespan	18.39 (0.35)	18.24 (0.27)	20.13 (0.51)
GA Usage	113.92 (78.26)	169.22 (70.20)	11.77 (1.87)
SA Makespan	18.42 (0.33)	18.25 (0.28)	20.08 (0.51)
SA Usage	102.66 (70.99)	183.83 (67.40)	11.71 (1.86)
TS Makespan	18.76 (0.25)	18.54 (0.23)	18.94 (0.34)
TS Usage	41.49 (12.26)	75.27 (37.31)	25.26 (6.25)
Problem Set 2:			
Measure	Rule 3	Rule 4	Rule 5
GA Makespan	19.11 (0.23)	18.75 (0.23)	20.12 (0.35)
GA Usage	53.96 (7.90)	98.66 (16.60)	27.67 (2.73)
SA Makespan	19.09 (0.44)	18.87 (0.27)	20.16 (0.30)
SA Usage	60.84 (10.07)	73.13 (22.94)	28.01 (2.58)
TS Makespan	19.16 (0.25)	19.04 (0.32)	19.35 (0.19)
TS Usage	48.67 (15.46)	58.84 (24.11)	36.17 (2.86)
Problem Set 3:			
Measure	Rule 3	Rule 4	Rule 5
GA Makespan	84.63 (2.17)	83.95 (1.66)	96.13 (4.95)
GA Usage	9802 (5689)	18 674.3 (5007)	191.6 (11.60)
SA Makespan	84.21 (1.84)	83.95 (1.66)	97.65 (1.72)
SA Usage	12 909.5 (2205)	19 610.9 (4951)	179.78 (39.24)
TS Makespan	92.04 (0.77)	89.81 (0.94)	95.25 (1.36)
TS Usage	365.87 (29.99)	597.97 (94.99)	226.42 (40.18)

Note: Means and standard deviations for Rules 1 and 2 are provided in Appendix 2.

Table 7. Means (and standard deviations) of performance measures by problem set, search heuristic and objective.

Table 4 shows that usage makes more meaningful contributions to the overall multivariate effect of objective compared with makespan. Means are presented on table 6.

Table 7 presents means for makespan and usage by problem set, search heuristic and objective.

7. Discussion of results

Initially, it should be emphasized that Objectives 1 and 2 do not employ searches at all. These are direct, one-time applications of heuristics, which explains why the three different searches yield the same results for these two objectives. As a result, they are essentially omitted from the prior analyses in order to isolate differences in the three composite, search-based approaches.

From inspection of the data in tables 3 and 4, the search heuristic does have an effect upon the performance measures of makespan and set-ups. Generally, Tabu Search performs well with regard to usage for all three problem sets, but poorly for makespan on the large problem set. The Simulated Annealing and Genetic Algorithm search heuristics are quite competitive with each other—Tukey's pairwise comparisons show non-significant differences for all cases with the exception of usage for the two largest problem sets, where they alternate better performance. For the two smaller problem sets, differences are not as 'large' as they are for the largest problem set.

Additional explanation needs to be provided with regard to the Tabu Search heuristic. When this methodology was constructed, the initial solution was always the one that minimized the usage rate (which is the second objective here—see equation (6)). In addition, weights for the Tabu Search heuristic were determined from sampling across many solutions. Recall from the methodology section that for the Genetic Algorithm Search, initial solutions were randomly generated. Furthermore, weights were determined by taking the number of set-ups and usage rate from the sequence that minimized set-ups for objectives (3) and (4). And for objective (5), by taking the number of set-ups and usage rate for the sequence that minimized usage rate determined weights. For the Simulated Annealing search heuristic, the initial sequence for objectives (3) and (4) were the ones that minimized the number of set-ups, while the initial sequence for objective function (5) was the one that minimized usage. Therefore, weighting for the Simulated Annealing was done the exact same way as it was for the Genetic Algorithm. This explanation essentially means the following: the Tabu Search objective functions were treated differently from those of the Genetic Algorithm and Simulated Annealing. Essentially, the Tabu Search 'favours' usage at the expense of makespan—and, in general, Tabu Search yields an undesirable performance for the largest problem set. Table 8 demonstrates the inverse relationship between makespan and usage. Notice that Tabu Search results in the most preferred makespan for the smaller two problem sets, which

Problem set	Correlation (makespan, usage)
1	−0.685
2	−0.718
3	−0.744

Table 8. Correlations between usage and makespan.

Problem set 1	Makespan preference	Usage preference
1	GA > SA	SA > GA
2	SA > GA	GA > SA
3	SA > GA	GA > SA

Table 9. Direct pairwise comparisons between Genetic Algorithm and simulated anealing for performance measures for objective 3.

contradicts the prior claim. Despite the Tabu Search providing superior levels of makespan for these two smaller problem sets, these significant differences are quite small, and (the significance of differences) can be attributed to the very large sample sizes. In short, Tabu Search does not address both objectives of set-ups and usage in an effective manner.

Given the fact that Tabu search favours usage at the expense of makespan, effort is directed toward a more detailed comparison of Genetic Algorithm and Simulated Annealing. Table 9 shows pairwise comparisons between the Genetic Algorithm and Simulated Annealing search heuristics for objective 3, where both the number of set-ups and usage are equally considered. Underlines show where Tukey's pairwise comparisons are not significant at the $\alpha = 0.05$ level.

Perhaps table 9 suggests a slight advantage for the Genetic Algorithm in terms of usage for the two larger problem sets, but there are no meaningful differences with respect to makespan.

There is nothing really surprising about the fact that the objective does have a multivariate effect on makespan and usage. For makespan, the order of objective preference is always: $4 > 3 > 5$, and for usage the preference order is reversed: $5 > 3 > 4$—further evidence of the trade-off between makespan and usage. This finding should present the importance of designing an objective that simultaneously addresses the conflicting objectives of makespan and usage—supporting the result of Objective 3, being the median performer for both makespan and usage. Objectives 4 and 5 also perform according to their intent, in that unequal weighting places varying emphases on these conflicting goals. It is appropriate to note that, in all instances, Rule 1 is the best performer with respect to makespan and worst with respect to usage. Similarly, Rule 2 is the worst performer with respect to makespan, and the best with respect to usage.

8. Conclusions

A technique has been presented to address a production sequencing problem where there are multiple objectives inversely related to each other. This type of sequencing problem can be thought of as combinatorial, which makes finding optimal solutions difficult. A Genetic Algorithm is the specific technique presented—an approach that can provide desirable solutions to challenging problems of a combinatorial nature, with a reasonable amount of CPU time.

The sequencing problem was treated as a biological organism, and was 'engineered' in such a way that the solutions showing the most potential for desirability were used as 'blueprints' for future solutions. This evolution process resulted in production sequences measured by the attributes of makespan time and a production usage rate. The desirability of these sequences was compared with that of two other approaches—Simulated Annealing and Tabu Search. Tabu Search did not perform

well in comparison to Simulated Annealing and the Genetic Algorithm across the attributes of makespan and usage rate. The direct comparison of Genetic Algorithms and Simulated Annealing essentially show that neither approach dominates the other—both are formidable in addressing the multiple objectives. The relative disadvantage of using the Genetic Algorithm over Simulated Annealing is that the Genetic Algorithm approach is computationally more expensive than Simulated Annealing. The reason for this is that, with the Genetic Algorithm, a larger part of the production sequence is manipulated as compared to the Simulated Annealing approach. However, this increase in CPU time is not a source of major concern because of the relatively easy availability of high-speed computing.

Acknowledgements

This research was supported in part by a competitive award from the Northeast Business and Economics Association.

Appendix 1: Sample problems

Problem Set 1: (number of each product type)

Problem	Product 1	Product 2	Product 3	Product 4	Product 5
A	20	0	0	0	0
B	16	1	1	1	1
C	15	2	1	1	1
D	13	4	1	1	1
E	10	5	2	2	1
F	8	7	2	2	1
G	6	6	5	2	1
H	5	5	5	3	2
I	5	4	4	4	3
J	4	4	4	4	4

Problem Set 2: (number of each product type)

Problem	Pr 1	Pr 2	Pr 3	Pr 4	Pr 5	Pr 6	Pr 7	Pr 8	Pr 9	Pr 10
A	20	0	0	0	0	0	0	0	0	0
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 3: (number of each product type)

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
A	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0
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	6	6	6	6	6	6

Note: Problem Set A is not used in these analyses because of the absence of product-mix.

Appendix 2: Means (and standard deviations) of makespan and usage for Rules 1 and 2.

Rule 1

	Problem Set 1	Problem Set 2	Problem Set 3
Makespan	18.23 (.28)	18.74 (.25)	83.94 (1.66)
Usage	193.72 (49.32)	179.54 (26.57)	19,612 (4,951)

Rule 2

	Problem Set 1	Problem Set 2	Problem Set 3
Makespan	20.15 (.51)	20.24 (.25)	97.65 (1.72)
Usage	11.71 (1.86)	27.56 (2.58)	179 78 (39.24)

Note: Regardless of whether GA, SA or TS is used, measures of makespan and usage will be the same for Rules 1 and 2, since Rules 1 and 2 do not involve search.

Appendix 3: Problems sets used for experimentation purposes

	Number of each item in product-mix														
Set	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	2	2	2	2	2	x	x	x	x	x	x	x	x	x	x
B	6	1	1	1	1	x	x	x	x	x	x	x	x	x	x
C	4	4	4	4	4	4	4	4	4	4	x	x	x	x	x
D	31	1	1	1	1	1	1	1	1	1	x	x	x	x	x
E	3	3	2	2	2	x	x	x	x	x	x	x	x	x	x
F	8	1	1	1	1	x	x	x	x	x	x	x	x	x	x
G	3	3	3	3	3	x	x	x	x	x	x	x	x	x	x
H	11	1	1	1	1	x	x	x	x	x	x	x	x	x	x
I	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
J	16	1	1	1	1	1	1	1	1	1	1	1	1	1	1
K	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
L	166	1	1	1	1	1	1	1	1	1	1	1	1	1	1

References

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

DING, F. and CHENG, L., 1993, An effective mixed-model assembly line sequencing heuristic for just-in-time production systems. *Journal of Operations Management*, **11**, 45–50.

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

GOLDBERG, D. E., 1989, *Genetic Algorithms in Search, Optimization & Machine Learning* (Reading, MA: Addison-Wesley).

GOLDBERG, D. E. and LINGLE, R., 1985, Alleles, loci, and the TSP. *Proceedings of the First International Conference on Genetic Algorithms*, pp. 154–159.

GLOVER, F., 1990, Tabu search: a tutorial. *Interfaces*, **20**, 74–94; 1993, A user's guide to tabu search. *Annals of Operations Research*, **41**, 3–28.

KIRKPATRICK, S., GELATT, C. D. and VECCHI, M. P., 1983, Optimization by simulated annealing. *Science*, **220**, 671–679.

MCMULLEN, P. R., 1998, JIT sequencing for mixed-model assembly lines with set-ups using Tabu Search. *Production Planning & Control*, **9**, 504–510.

MCMULLEN, P. R. and FRAZIER, G. V., 1998, Using simulated annealing to address JIT sequencing for mixed-model assembly lines with set-ups. *Proceedings of the Decision Sciences Institute Annual Conference*; 2000, A simulated annealing approach to mixed-model sequencing with multiple objectives on a JIT line. *IIE Transactions*, **32**, 679–686.

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**, 192–207.

MONDEN, Y., 1983, *Toyota Production System* (Norcross, GA: The Institute of Industrial Engineers).

OLIVER, I. M., SMITH, D. J. and 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*, pp. 224–230.

SUMICHRAST, R. T. and RUSSELL, R. S., 1990, Evaluating mixed-model assembly line sequencing heuristics for just-in-time production systems. *Journal of Operations Management*, **9**, 371–390.

WALPOLE, R. E. and MYERS, R. H., 1985, *Probability and Statistics for Engineers and Scientists*, 3rd edn (New York: Macmillan).