
**International Journal of Production Research**
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713696255

# JIT mixed-model sequencing with batching and setup considerations via search heuristics

Patrick R. McMullen[a]
[a] Wake Forest University, Schools of Business, Winston-Salem, NC 27109, USA

First published on: 21 December 2009

**To cite this Article** McMullen, Patrick R.(2010) 'JIT mixed-model sequencing with batching and setup considerations via search heuristics', International Journal of Production Research, 48: 22, 6559 — 6582, First published on: 21 December 2009 (iFirst)
**To link to this Article: DOI:** 10.1080/00207540903321640
**URL:** http://dx.doi.org/10.1080/00207540903321640

*International Journal of Production Research*
Vol. 48, No. 22, 15 November 2010, 6559–6582

Taylor & Francis
Taylor & Francis Group

# JIT mixed-model sequencing with batching and setup considerations via search heuristics

Patrick R. McMullen*

*Wake Forest University, Schools of Business, Winston-Salem, NC 27109, USA*

(*Received 27 February 2009; final version received 6 September 2009*)

This research addresses the problem of sequencing items for production when it is desired that the production sequences result in minimal usage rates – surrogate measures for flexibility in a JIT environment. While seeking sequences with minimal usage rates, the number of required setups for the sequences is also considered, along with feasible batch-sizing combinations. The general intent is to find minimum usage-rate sequences for each associated number of setups and total batches. This multiple objective problem is addressed via a three-dimensional efficient frontier. Because the combinatorial nature of sequencing problems typically provides an intractable search space for problems of 'real world' size, the search heuristics of simulated annealing and genetic algorithms are presented and used to find solutions for several problem sets from the literature. Experimentation shows that the simulated annealing approach outperforms the genetic algorithm approach in both objective function and CPU performance.

**Keywords:** tabu search; simulation; genetic algorithms; simulated annealing; balancing; assembly line balancing



## 1. Introduction

Production sequencing has long been a source of concern for researchers engaged in JIT/Lean Manufacturing. Finding production sequences that provide minimal in-process inventories and flexibility is a position coveted by organisations. These production sequences consist of all of the unique items demanded by the customer, and it is desired that these unique items are distributed through the sequence as evenly as possible (Monden 1998). This even distribution of items throughout the sequence comes at the subsequent expense of having similar items in non-adjacent sequence positions, resulting in frequent changeovers. Frequent changeovers are not problematic if the effort required to change from one unique item to another is negligible. Unfortunately, this is not always the case. As such, there are often tradeoffs between JIT flexibility and setup cost (McMullen *et al.* 2000) – organisations who seek to minimise setups are often burdened with large in-process inventories and an inability to respond to customer demands, while organisations that succeed with JIT are often faced with the dilemma of finding ways to quickly change over from one unique item to another.

A fair amount of research has addressed finding sequences that address the tradeoff between JIT flexibility and the number of setups. The upshot of this research is essentially

*Email: patrick.mcmullen@mba.wfu.edu

ISSN 0020–7543 print/ISSN 1366–588X online

DOI: 10.1080/00207540903321640
http://www.informaworld.com

a set of strategies that attempt to find sequences that provide tolerable levels of both flexibility and setups, and/or find the most flexible sequences for some fixed number of setups.

For the research described above, there is an assumption that is essentially tacit throughout. Although not explicitly mentioned, the above-described research typically assumes that each item to be sequenced has a batch size of exactly one. In other words, it is assumed that each entity in the sequence is completely divisible, or separable from all other items in the sequence. In reality this divisibility assumption may not always be the case. The divisibility issue can be illustrated via the example scenario where four units of item A are demanded, two units of item B are demanded, and a single unit of item C is demanded. In total, seven units are demanded. One possible sequence is as follows: ABACABA. This particular sequence is particularly attractive if the intent is to optimise flexibility via the material usage rate (this point is explained in the next section). This example sequence assumes that the batch size for each unique item is one unit. Assume for one moment, however, that the required batch size for items A and B are two units each, while the required batch size for item C is one unit. If this were the case, the ABACABA sequence would be infeasible, as the 'batch size of two' constraint for items A and B would be violated. A sequence that would be feasible with respect to the 'batch size of two' constraint for items A and B would be as follows: $A_2B_2C_1A_2$, where the subscripts quantify batch sizes.

The author believes that considering batch size issues while simultaneously considering setups and JIT flexibility is important (McMullen 1998). In the face of a JIT supply chain, it is quite possible, even likely, that both suppliers and customers of the organistion of interest have discrete quantities in which they insist on *selling to* and/or *buying from* the specific organisation. In such situations, it may be appropriate to embrace the same batching scenarios as their suppliers and/or customers. As such, batch-sizing considerations, along with considering JIT flexibility and setups, make for an important discussion. It is also important to note that batch-sizing decisions may not necessarily be driven by the suppliers and/or clients of the organisation, but may simply be in the organisation's best interest to work with some non-trivial batch size that best enhances their competitive position.

As stated above, research efforts concerned with JIT sequencing typically assume the batch size to be fixed at one unit. The research presented here departs from this assumption, and considers all feasible batch sizes for various demand patterns. Previous research efforts have in fact considered JIT sequencing where batch sizes other than one are considered. Yavuz *et al.* (2006) and Yavuz and Tüfekçi (2007) considered JIT sequencing with an objective function concerned with optimisation of cycle time. Kubiak and Yavuz (2008) considered JIT sequencing with non-trivial batch constraints with multiple objectives including production rate variability and efficiency.

The research efforts cited above, however, do not simultaneously give explicit consideration to optimisation of JIT flexibility in the face of both setups and batch-sizing decisions. Such consideration is the motivation for this paper, and it is believed that operations managers would benefit from methodology addressing JIT sequencing in the presence of setups and batch-sizing considerations. To add support to this motivation, an example scenario is provided. Consider a situation where an assembly operation (such as appliances or electronic devices) receives their raw materials according to some fixed batch size. This fixed batch size, or some multiple thereof, can serve as a constraint for subsequent decisions. Given this scenario, production planners can then seek to find sequences that yield the best combinations of setups and material usage rate (JIT flexibility metric), via an

efficient frontier, for the fixed batch size. Conversely, consider a situation where production planners are required to find sequences such that the number of setups they incur must be minimised due to expensive changeovers. When this occurs, production planners can then exploit the efficient frontier between batch size and material usage rate (JIT flexibility metric) to find the best sequence given the fixed number of setups. The upshot of these two scenarios is that when an operation is committed to some fixed value of batch size or setups, they can then make the best possible decision by exploiting the relationship between the other, non-fixed entity and the material usage rate (JIT flexibility metric).

This relationship between JIT flexibility, batch sizing and setups is another motivation for this paper. The presented three-dimensional efficient frontier that shows the JIT flexibility measure as a function of both batch sizing and setups decisions is considered unique, and the author considers it of value to the body of literature.

The following sections discuss the problem in technical detail, describe the barriers to obtaining optimal solutions, present simulated annealing and genetic algorithm approaches to finding desirable solutions, describe the experimentation used to validate the presented methodology, present results, and offer general conclusions.



## 2. Optimal JIT sequencing

Miltenburg (1989) presented a metric to quantify a sequence's ability to evenly distribute the demand of unique items throughout. This metric is frequently referred to as the 'material usage rate', and is widely accepted as a measure of the JIT flexibility provided by the sequence. In layman's terms, the measure aggregates the difference between *what is* sequenced and *what should be* sequenced, through all positions in the sequence.

### 2.1 *Objective function in batch notation*

The Miltenburg metric can be thought of as the objective function when attempting to find sequences that optimise flexibility. From a batch-oriented standpoint, the following terms are defined (Yavuz and Tüfekçi 2004, 2007).

As stated earlier, the intent of the Miltenburg metric is to minimise the aggregated gap between what is placed into sequence with what should be placed into sequence. In mathematical programming form, this can be stated via the following batch notation defined in Table 1 (McMullen *et al.* 2000):

$$\text{Min: } \sum_{k=1}^{Q}\sum_{i=1}^{n}\left(b_i\left(Y_{ik}-k\frac{q_i}{Q}\right)\right)^2, \quad \forall S, Q \tag{1}$$

subject to

$$b_i q_i \geq d_i, \quad \forall_i \tag{2}$$

$$Y_{ik} = \sum_{j=1}^{k} X_{ij}, \quad \forall_i \tag{3}$$

$$\sum_{k=1}^{Q} X_{ik} = 1, \quad \forall_i \tag{4}$$

Table 1. Definitions for formulation.

| Term | Description |
|---|---|
| *Endogenous variables* | |
| $n$ | Unique items to be sequenced |
| $d_i$ | Demand (in units) for item $i$ |
| $m_i$ | Unique batch size quantities for item $i$ |
| $p_j$ | Unique sequences for batch combination $j$ |
| $S$ | Total setups associated with sequence |
| $Q$ | Total number of batches associated with sequence |
| $M$ | Total number of batch combinations |
| $P$ | Total number of feasible sequences |
| $Y_{ik}$ | Number of batches of item $i$ through the $k$th batch sequence position |
| *Decision variables* | |
| $b_i$ | Batch size of item $i$ |
| $q_i$ | Number of batches of item $i$ |
| $X_{ik}$ | =1 if item $i$ is placed in the $k$th batch sequence position; 0 otherwise |
| *Indices* | |
| $i$ | Item index ($i=1,\ldots,n$) |
| $k$ | Batch-position index ($k=1,\ldots,Q$) |
| $j$ | Miscellaneous index |



where

$$Q = \sum_{i=1}^{n} q_i \tag{5}$$

$$S = 1 + \sum_{k=2}^{Q} \left(1 - \sum_{i=1}^{n} X_{i,k} X_{i,k-1}\right) \tag{6}$$

The model seeks to minimise the material usage rate. Equation (2) mandates that demand be met for each item. Equation (3) relates the decision variable $X_{ik}$ to the endogenous variable $Y_{ik}$. Equation (4) ensures that each position in the batch sequence be occupied with a single batch. Equation (5) details how the total batches quantity ($Q$) is determined, while Equation (6) details how the number of setups for the sequence ($S$) is determined. It is important to note that the usage rate and number of setups are sequence dependent, but the number of batches associated with the sequence is not sequence dependent.

### 2.2 *Batch-sizing issues*

As stated above, where variables are defined, it is noted that the batch size for each item and the number of batch sizes for each item are considered decision variables – the decision maker controls the values, with the following assumptions in mind.

- The minimum batch size for any item is 1, rendering the associated number of batches equal to demand. That is, $b_i$=1, $q_i = d_i$.

Table 2. Combinations for each item, example problem.

| Item A ($i=1$) | Item B ($i=2$) | Item C ($i=3$) |
|---|---|---|
| ($b_1=4$, $q_1=1$) | ($b_2=2$, $q_2=1$) | ($b_3=1$, $q_3=1$) |
| ($b_1=2$, $q_1=2$) | ($b_2=1$, $q_2=2$) | |
| ($b_1=1$, $q_1=4$) | | |
| $m_1=3$ | $m_2=2$ | $m_3=1$ |

- The maximum batch size for any item is assumed to be equal to the item's demand, rendering the number of batches equal to (1). That is, $b_i=d_i$, $q_i=1$.
- Setups are only required when transitioning from one batch to another when the batches are comprised of differing items.
- For batch sizes between 1 and $d_i$, the following algorithm is employed to develop a relationship between $b_i$ and $q_i$. For this algorithm, vector notation is used so that $b_i$ and $q_i$ relationships can be shown in list form. The $i$ subscript remains unchanged to represent item $i$, but a *counter* subscript is introduced to ordinal show position on a list.

```
for i = 1 to n
  counter = 1;
  q_{i,1} = 1;
  b_{i,1} = d_i;
  for j = 2 to d_i
    if ⌈d_i/j⌉ < ⌈d_i/(j − 1)⌉
      counter++;
      q_{i,counter} = j;
      b_{i,counter} = ⌈d_i/j⌉;
    end if
  next j
  m_i = counter;
next i
```

The above algorithm guarantees that any $b_iq_i$ mathematical product in excess of demand ($d_i$) is minimised. For example, if $d_i=6$ for some item, ($b_i$, $q_i$) combinations are (6, 1), (3, 2), (2, 3) and (1, 6). A ($b_i$, $q_i$), combination (2, 4) does not help because it results in satisfying demand of (8) units instead of the required (6), which is provided by (2, 3). As the algorithm proceeds, the ceiling function resulting in the first 'new' value of $\lceil d_i/j \rceil$ is used to determine the ($b_i$, $q_i$) combination.

In the context of the example problem above, where $d_1=4$, $d_2=2$ and $d_3=1$, Table 2 shows the breakdown of ($b_i$, $q_i$) combinations for each item.

The number of unique batch combination ($M$) for all $n$ items is determined as follows:

$$M = \prod_{i=1}^{n} m_i \tag{7}$$

For the example problem, then, there are $M=(3)(2)(1)=6$ unique batch combinations. Details of the ($b_i$, $q_i$) relationships for these combinations are shown in Table 3.

Table 3. Batch combinations for example problem.

| Comb. # ($j$) | Item A ($i=1$) | Item B ($i=2$) | Item C ($i=3$) | Base sequence | Feasible sequences ($p_j$) | Batches ($Q$) |
|---|---|---|---|---|---|---|
| 1 | $b_1=4,\ q_1=1$ | $b_2=2,\ q_2=1$ | $b_3=1,\ q_3=1$ | ABC | 6 | 3 |
| 2 | $b_1=4,\ q_1=1$ | $b_2=1,\ q_2=2$ | $b_3=1,\ q_3=1$ | ABBC | 12 | 4 |
| 3 | $b_1=2,\ q_1=2$ | $b_2=2,\ q_2=1$ | $b_3=1,\ q_3=1$ | AABC | 12 | 4 |
| 4 | $b_1=2,\ q_1=2$ | $b_2=1,\ q_2=2$ | $b_3=1,\ q_3=1$ | AABBC | 30 | 5 |
| 5 | $b_1=1,\ q_1=4$ | $b_2=2,\ q_2=1$ | $b_3=1,\ q_3=1$ | AAAABC | 30 | 6 |
| 6 | $b_1=1,\ q_1=4$ | $b_2=1,\ q_2=2$ | $b_3=1,\ q_3=1$ | AAAABBC | 105 | 7 |

Each unique batch combination has a 'base sequence'. This base sequence is listed in lexicographic order, but can be permuted in $p_j$ ways. The value of $p_j$ is determined as follows (Tucker 2007):

$$p_j = \left(\sum_{j=1}^{Q} q_i\right)! \Big/ \prod_{j=1}^{Q} (q_i!) \tag{8}$$

The number of all feasible sequences when all base sequences are considered is as follows:

$$p = \sum_{j=1}^{M} p_j \tag{9}$$

For example, the base sequence 'AAAABBCC' has $(4+2+1)!/(4!)(2!)(1!)=105$ unique permutations, and when all $M=6$ base sequences are permuted, there are 195 total sequences that are feasible when batch sizing and permutations are considered.

### 2.3 *Finding optimal sequences*

The objective function articulated in Equation (1) essentially states that it is desired to find sequences that minimise the parts usage rate for each combination of setups and number of total batches. There are, in principle, three different ways to obtain such an objective: (1) via enumeration of all possible solutions; (2) via mathematical programming; and (3) via a search algorithm (Hillier and Lieberman 2005).

A solution via enumeration would require one to first enumerate all unique batch combinations and then for each unique batch combination, permute all possible orderings to determine minimal usage rates. For our example problem, the solution is shown in Table 4.

This collection of data can be thought of as the *efficient frontier* – the minimal usage rates for each unique combination of total batches and setups. This is shown graphically in Figure 1.

This example problem is pedagogical in nature, and has little value from an application standpoint. Finding optimal solutions to larger problems becomes more difficult, even with continuous reductions in computational expense.

The second option is to use mathematical programming to find the efficient frontier. This is not easy for problems of this type due to the mathematical properties of the formulation. Decision variables take on discrete values, the setups constraint is nonlinear,

Table 4. Efficient frontier in tabular form for example problem.

| | Batch | | | | |
|---|---|---|---|---|---|
| Setup | 3 | 4 | 5 | 6 | 7 |
| 3 | 1.33 | | | | |
| 4 | 2.25 | 1.25 | | | |
| 5 | 4.80 | 2.40 | 1.60 | | |
| 6 | 4.17 | 2.17 | 2.17 | | |
| 7 | 9.43 | 4.29 | 2.86 | 2.29 | 1.71 |

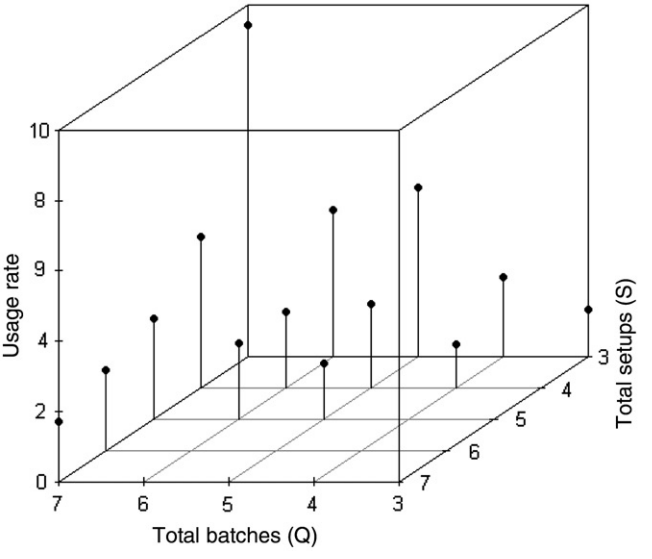


Figure 1. Efficient frontier in graphical form for example problem.



and the objective function is in the form of a non-convex surface. Such properties make finding optimal solutions highly unlikely.

The third option is to use search heuristics to find the efficient frontier. This is the chosen option given the inherent challenges of the two approaches described above. Search heuristics do not guarantee an optimal solution, but if well designed and executed, they have been successful in finding near-optimal solutions with a reasonable computational effort. Simulated annealing and genetic algorithms are both used for this research (Hillier and Lieberman 2005).

## 3. Simulated annealing and genetic algorithm methodology

It is desired that both of these search heuristics are used to find near-optimal solutions to problems. The task at hand here is somewhat unique compared with other sequencing problems in that, here, the researcher intends to find optimal sequences to a single problem

that takes on many forms. These 'forms' will have different combinations of total batches ($Q$) and total setups ($S$). Table 3 illustrates this point – each base sequence will have some specific number of total batches ($Q$) which is not sequence dependent (fixed), but each base sequence will have a total number of setups ($S$) and usage rates that are in fact sequence dependent. Because of this, it is appropriate to solve the problem several times via the chosen search heuristic, each time using a randomly selected base sequence as a starting point.

### 3.1 *Find a base sequence as a starting point for search*

Finding a base sequence as a starting point for the search is done via Monte-Carlo selection from a probability table. To construct the probability table, the notation given in Table 5 is used.

The values of $Q_j$ are easily understandable, and have been previously defined. The value of $F_j$ is simply the number of times that base sequence $j$'s value of $Q_j$ was present in other base sequences. The uniqueness ($U_j$) of base sequence $j$ is essentially the reversed value of $F_j$ – a measure of how different base sequence $j$ is from other base sequences, in terms of its value of $Q_j$. Mathematically, this value is determined as follows:

$$U_j = 1 + (\max(F) - F_j), \forall_j \tag{10}$$

A '1' is added to each value so that all base sequences have some non-zero probability of being selected. The weighted uniqueness is determined by applying the total number of batches ($Q_j$) to the uniqueness for each base sequence. Mathematically, this is as follows:

$$W_j = Q_j U_j, \forall_j \tag{11}$$

The probability of base sequence $j$ being selected is as follows:

$$prob_j = W_j \Big/ \sum_{j=1}^{M} W_j \tag{12}$$

The probability of selecting base sequence $j$ is a function of both its uniqueness and the number of batches. Higher values of each increase its probability of being selected. Such properties are incorporated to find as varied a group of solutions as possible. Table 6 shows how this selection computation works for the base sequences of our example problem.

Table 5. Definitions for selection probability.

| Term | Description |
|---|---|
| $Q_j$ | Total batches for base sequence $j$ |
| $F_j$ | Total number of times the quantity of $Q_j$ was observed in other base sequences |
| $U_j$ | Uniqueness of base sequence $j$ |
| $W_j$ | Weighted uniqueness of sequence $j$ |
| $Prob_j$ | Probability of selecting base sequence $j$ |

This base sequence is selected each time that the search heuristic is used. The number of times a base sequence is selected is referred to as *Simulations*.

### 3.2 *Simulated annealing approach to finding sequences*

Prior to detailing the steps for simulated annealing to find desirable sequences, the parameters are presented in Table 7 (Kirkpatrick *et al*. 1983, Eglese 1990).

The steps to finding sequences via the simulated annealing approach are straightforward. After a base sequence is selected (via Section 3.1), an initialisation process occurs, where the values of $T_1$, $T_F$, $CR$, *Iter* are chosen. The current solution and its properties are also initialised to their appropriate values.

During the search, the sequence undergoes a pairwise swap of its members – the members are always unique from each other and are randomly chosen. These pairwise swaps are intended to accomplish two things: the first is to find desirable solutions; the second is to find solutions that represent all parts of the continuum of feasible solutions. As an example of a pairwise swap, assuming that from our example problem, the base sequence AABBC is selected for modification. A possible modification is as follows:

$$\underline{A}ABB\underline{C}\ (\text{before modification}), \quad A\underline{C}BB\underline{A}\ (\text{after modification})$$

Underlined batches denote the batches that are selected for the swap. After modification, the usage is determined for the modified sequence, along with the number of setups. Total batches will remain unchanged for the base sequence and will therefore not require being

Table 6. Selection probabilities for example problem.

| Base sequence | $Q_j$ | $F_j$ | $U_j$ | $W_j$ | $Prob_j$ |
|---|---|---|---|---|---|
| ABC | 3 | 1 | 2 | 6 | 6/50 |
| ABBC | 4 | 2 | 1 | 4 | 4/50 |
| AABC | 4 | 2 | 1 | 4 | 4/50 |
| AABBC | 5 | 1 | 2 | 10 | 10/50 |
| AAAABC | 6 | 1 | 2 | 12 | 12/50 |
| AAAABBC | 7 | 1 | 2 | 14 | 14/50 |

Table 7. Simulated annealing parameters.

| Parameter | Description |
|---|---|
| $T$ | Current temperature |
| $T_1$ | Initial temperature |
| $T_F$ | Final temperature |
| *Iter* | Iterations for each temperature level |
| $CR$ | Cooling rate |
| $U_M$ | Usage rate of modified solution |
| $U_C$ | Usage rate of current solution |
| $\delta$ | Relative inferiority of test solution |
| $k_B$ | Boltzman constant |

determined for each modification. If the usage rate for the modified solution is less than the usage rate for the current solution (for the relevant values of $S$ and $Q$), then the modified solution replaces the current solution for the relevant values of $S$ and $Q$. If the modified usage rate is not less than the current solution's usage rate, then the Metropolis *et al.* (1953) criterion is explored via the degree of the modified solution's relative inferiority to the current solution's usage rage ($\delta$), which is determined by

$$\delta = (U_M - U_C)/U_C \tag{13}$$

This level of inferiority is then used to determine the probability of the inferior modified solution replacing the current solution via the following:

$$P(A) = \exp(-\delta/K_B T) \tag{14}$$

The value of $k_B$ is referred to as the Boltzman constant, and provides the researcher the opportunity to control for the probability of accepting a relatively inferior solution. Monte-Carlo simulation is then used to generate a number on the $[0, 1)$ interval to determine whether or not the inferior solution replaces the current solution.

The modification and comparison steps described above are repeated *Iter* times. At this point, the value of $T$ is updated via the following:

$$T = T(CR) \tag{15}$$

This continues while $T$ is greater in value than $T_1$. When $T$ is no longer greater than $T_1$, the minimal usage rates found via the search, for the relevant values of $S$ and $Q$, are recorded. This entire procedure repeats itself *Simulations* times. After this search heuristic is reported Simulations times, the best usage rate for each combination of $Q$ and $S$ is reported as the efficient frontier found via simulated annealing.

In pseudocode, the simulated annealing procedure (Kirkpatrick *et al.* 1983, Eglese 1990) is as follows:

```
for g=1 to Simulations
  select baseSequence();
  initialise();
  while (T>T1)
    for i=1 to Iter
      modification();
      findUsage();
      if usage<currentUsage replaceCurrent();
      else metroplisTest();
    next i
  T=T*CR;
  end while
  recordBestSolution();
next g
report bestSolution();
```

It is important to note that, for each *Simulation*, the total number of batches ($Q$) will never change. As such, it is necessary to perform several *Simulations*, so that different numbers of batches ($Q$) can be used to find the 'best'' efficient frontier that considers all possible values of $Q$.

Table 8. Genetic algorithm parameters.

| Parameter | Description |
|---|---|
| *Pop* | Population size of generation (assumed to be an even number) |
| *Gen* | Number of generations |
| *Uj* | Usage rate for sequence *j* |
| $U_B$ | Best usage rate for relevant *S*, *Q* |
| $\delta_j$ | Relative inferiority of sequence *j* |
| *P*(*Sj*) | Probability of selecting sequence *j* |
| $P_M$ | Probability of mutation |

### 3.3 *Genetic algorithm approach to finding sequences*

The parameters given in Table 8 are used to find desirable sequences via the genetic algorithm (Goldberg 1989, Michalewicz 1996).

As is the case with simulated annealing, the first step in the genetic algorithm process is to select a base sequence, which is detailed in Section 3.1.

The base sequence undergoes *Pop* random permutations – these *Pop* random permutations are the 0th generation via the genetic algorithm and comprise the *population*.

The usage rate and number of setups for each sequence is determined – as with simulated annealing, the total batches (*Q*) only needs to be determined for the base sequence, as this value is not sequence dependent. For each sequence in the population, the usage rate is compared with the best-known usage rate for that number of setups (*S*) and total batches (*Q*). This comparison determines the relative inferiority of the sequence:

$$\delta_j = (U_j - U_B)/U_B, \forall S, Q \tag{16}$$

From this value, the selection probability for each sequence can be quantified via the following:

$$P(S_j) = (\max(\delta) - \delta_j) \Big/ \sum_{j=1}^{M} (\max(\delta) - \delta_j) \tag{17}$$

Monte-Carlo simulation is then used to select *Pop* sequences from the population. Thus describes the selection process.

After selection, crossover is performed by pairing the population off into a number of pairs equal to *Pop*/2. Both sequences in each pair have a randomly selected partition, on the $[1, (Q-1)]$ interval placed after the *j*th sequence position for both members of the pair. The sequence members to the left of the partition are referred to as $\text{Parent}_1$ and $\text{Parent}_2$, respectively. The sequence members to the right of the partition are referred to as $\text{Reserve}_1$ and $\text{Reserve}_2$, respectively. Each child sequence is composed of the verbatim portion of its corresponding parent, followed by, in list order, the feasible portion of its opposite reserve, and then by, in list order, the feasible portion of its opposite parent (to the left of the partition). Parsing the list of feasible items in the opposite Reserve and opposite Parent is done until the child sequence is completed. Consider as an example the base sequence AAAABBC, and the crossover pair:

AAB|BCAA($\text{Parent}_1$) BCAA($\text{Reserve}_1$)

ABC|AAAB($\text{Parent}_2$) AAAB($\text{Reserve}_2$)

The randomly placed partition occurs after the third position. To construct $\text{Child}_1$, the first two units of item A are used from $\text{Reserve}_2$, along with the unit of item B from $\text{Reserve}_2$, and finally the unit of item C from $\text{Parent}_2$. The intervening items are not placed in the sequence due to feasibility preservation. Similarly, $\text{Child}_2$ is constructed from one unit of item B and then two units of item A from $\text{Reserve}_1$, followed by one unit of item A from $\text{Parent}_1$. Again, items that are 'passed over' are done so only to preserve feasibility. The two child sequences derived from the above example sequences are as follows:

$$\text{AAB|AABC}\,(\text{Child}_1)$$
$$\text{ABC|BAAA}\,(\text{Child}_2)$$

These child sequences are intended to have characteristics of both parent sequences. Thus describes the crossover process. This process is done for each pair of the *Pop*/2 pairs used for crossover. It should be noted that this crossover process then subsequently applies to the entire generation. It should also be noted that the crossover process is independent of the number of items requiring sequencing – as long as there are two or more items in a sequence, crossover can be performed for any population organised by pairs.

After crossover, mutation is performed. All of the *Pop* sequences in the population are candidates for mutation. A random number on the [0, 1) interval is generated. If this number is less than $P_M$, mutation is performed to the candidate sequence of interest, which is simply a pairwise switch of two unique, randomly selected items in the sequence. It is important to note that mutation occurs infrequently, but the literature has time and again shown that mutation is a necessary device in propagating some diversity in a population (Goldberg 1989).

After selection, crossover and mutation are performed, the population is considered to have been modified, and the usage rate is computed for each sequence in the population; the heuristic remembers the lowest usage rate for each combination of total batches ($Q$) and setups ($S$). This pattern of selection, crossover and mutation continues for *Gen* generations. It is important to note that this process of selection, crossover and mutation is done for two reasons, as was the case with the simulated annealing approach: the first intent is to find desirable solutions; the second is to find solutions from all parts of the feasible search space.

After each generation is completed, a new base sequence is selected in accordance with the description in Section 3.1, and the steps described immediately above are repeated *Simulations* times. At the end of the search, the minimum usage rates for each combination of total batches ($Q$) and setups ($S$) are reported. In pseudocode, the genetic algorithm takes on the following form:

```
for g = 1 to Simulations
  selectBaseSequence();
  initialise();
  for h = 1 to Generations
    selection();
    crossover();
    mutation();
    recordBestSolution();
  next h
next g
reportBestSolution();
```

## 4. Experimentation

To assess the performance of the simulated annealing and genetic algorithm approaches, five sets of test problems from the literature (McMullen 1998, 2001a,b) are used to determine the performance of each approach against each other, and with respect to an optimal solution that was obtained via enumeration.

### 4.1 *Test problems*

Tables 9a–e contain five problems sets used for experimentation, for a total of 37 different problems. Five unique items are present for the first four problem sets, and 10 unique items for the fifth problem set. The problem sets were originally used for exploring JIT sequencing with consideration for setups, but not batch-sizing considerations. Tables 9a–e detail the demand for each item, along with the total number of base sequences ($M$) and the total number of permutations for all base sequences ($P$). The number of grid entries is also presented, which is the total number of unique total batches ($Q$) and setups ($S$) combinations for the problem of interest. The CPU time in minutes required to find the optimal solution via enumeration is also presented – one will notice that, for the smaller problems, CPU time is not a concern, but for some of the larger problems, CPU time is extraordinary. The problems shown in Tables 9a–d were solved to optimality. As such, this information details optimality issues. The problems detailed in Table 9e were not solved to



Table 9a. Details of Problem Set $A_1$.

| Prob. | Item A | Item B | Item C | Item D | Item E | $M$ | $P$ | Grid size | CPU min |
|---|---|---|---|---|---|---|---|---|---|
| B | 6 | 1 | 1 | 1 | 1 | 4 | 6330 | 11 | 0.0020 |
| C | 5 | 2 | 1 | 1 | 1 | 8 | 24,444 | 21 | 0.0035 |
| D | 4 | 2 | 2 | 1 | 1 | 12 | 64,620 | 21 | 0.0076 |
| E | 4 | 3 | 1 | 1 | 1 | 9 | 40,740 | 21 | 0.0055 |
| F | 3 | 3 | 2 | 1 | 1 | 18 | 115,860 | 21 | 0.0115 |
| G | 3 | 2 | 2 | 2 | 1 | 24 | 183,840 | 21 | 0.0194 |
| H | 2 | 2 | 2 | 2 | 2 | 32 | 291,720 | 21 | 0.0276 |

Table 9b. Details of Problem Set $A_2$.

| Prob. | Item A | Item B | Item C | Item D | Item E | $M$ | $P$ | Grid size | CPU min |
|---|---|---|---|---|---|---|---|---|---|
| B | 8 | 1 | 1 | 1 | 1 | 5 | 14,880 | 15 | 0.0033 |
| C | 7 | 2 | 1 | 1 | 1 | 10 | 70,980 | 27 | 0.0103 |
| D | 6 | 3 | 1 | 1 | 1 | 12 | 164,220 | 35 | 0.0192 |
| E | 6 | 2 | 2 | 1 | 1 | 16 | 259,500 | 35 | 0.0301 |
| F | 5 | 3 | 2 | 1 | 1 | 24 | 620,364 | 36 | 0.0657 |
| G | 5 | 2 | 2 | 2 | 1 | 32 | 980,664 | 36 | 0.1036 |
| H | 4 | 3 | 2 | 2 | 1 | 36 | 1,663,500 | 36 | 0.1705 |
| I | 4 | 4 | 2 | 1 | 1 | 18 | 640,320 | 36 | 0.0655 |
| J | 3 | 3 | 2 | 2 | 2 | 72 | 4,594,200 | 36 | 0.4598 |

Table 9c. Details of Problem Set $A_3$.

| Prob. | Item A | Item B | Item C | Item D | Item E | $M$ | $P$ | Grid size | CPU min |
|---|---|---|---|---|---|---|---|---|---|
| B | 11 | 1 | 1 | 1 | 1 | 6 | 40,800 | 20 | 0.0073 |
| C | 10 | 2 | 1 | 1 | 1 | 12 | 237,888 | 33 | 0.0359 |
| D | 9 | 3 | 1 | 1 | 1 | 15 | 832,044 | 49 | 0.1131 |
| E | 7 | 5 | 1 | 1 | 1 | 20 | 2,718,444 | 56 | 0.3640 |
| F | 7 | 3 | 2 | 2 | 1 | 60 | 19,322,340 | 66 | 2.4318 |
| G | 6 | 3 | 3 | 2 | 1 | 72 | 46,356,960 | 66 | 5.7730 |
| H | 5 | 3 | 3 | 3 | 1 | 108 | 107,377,584 | 66 | 13.3674 |
| I | 4 | 3 | 3 | 3 | 2 | 162 | 289,603,680 | 66 | 35.5735 |
| J | 3 | 3 | 3 | 3 | 3 | 243 | 487,424,520 | 66 | 59.3068 |

Table 9d. Details of Problem Set $A_4$.

| Prob. | Item A | Item B | Item C | Item D | Item E | $M$ | $P$ | Grid size | CPU min |
|---|---|---|---|---|---|---|---|---|---|
| B | 16 | 1 | 1 | 1 | 1 | 7 | 136,200 | 25 | 0.0247 |
| C | 15 | 2 | 1 | 1 | 1 | 14 | 1,146,048 | 45 | 0.1992 |
| D | 13 | 4 | 1 | 1 | 1 | 21 | 17,930,304 | 69 | 3.1509 |
| E | 10 | 5 | 2 | 2 | 1 | 96 | 1,847,389,32 | 130 | 316.4182 |
| F | 8 | 7 | 2 | 2 | 1 | 100 | 3,850,514,040 | 136 | 655.5461 |
| G | 6 | 6 | 5 | 2 | 1 | 128 | 23,896,107,024 | 136 | 4,161.634 |
| H | 5 | 5 | 5 | 3 | 2 | 384 | 182,206,343,832 | 136 | 31,368.92 |

Table 9e. Details of Problem Set $A_5$.

| Prob. | Item A | Item B | Item C | Item D | Item E | Item F | Item G | Item H | Item I | Item J |
|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

optimality, given the sheer size of the problems. Because of this, Table 9e only shows information pertaining to product mix.

To put the simple example detailed in Table 4 and in Figure 1 in the context of the tables, there are six base sequences ($M = 6$), 195 total permutations ($P = 195$), and 14 grid entries. CPU time for this simple example is negligible.

### 4.2 *Performance measurements and research questions*

For the presented research, there are essentially three performance measures of concern: efficient frontier performance, frontier void performance and CPU performance. These entities are captured for both simulated annealing and genetic algorithm approaches for each test problem.

Efficient frontier performance is essentially the level of relative inferiority a solution obtained via the search heuristic is to the optimal solution obtained via enumeration. If $Usage_H$ represents the usage rate obtained via the search heuristic, and $Usage_O$ is the usage rate of the optimal solution, than the degree of inferiority, for each unique $Q$, $S$ combination, is as follows:

$$Inferiority = (Usage_H - Usage_O)/Usage_O, \forall Q, S \quad (18)$$

An average of relative inferiority is determined for each problem – this is simply the sum of relative inferiority divided by the number of *Inferiority* calculations made. It is of course desired that the relative inferiority associated with the search heuristics be minimised.

The 'frontier voids' performance measure is a measure of the heuristic's ability to find solutions in all parts of the feasible solution space. If a certain heuristic approach provides zero frontier voids, the heuristic has succeeded in finding solutions on all parts of the solution continuum. Conversely, if a heuristic yields multiple frontier voids, the heuristic has failed in finding solutions at multiple parts of the solution continuum. It is important to note that, for this research, it is desired to find solutions across the entire search continuum (Tan *et al*. 2005). An additional point to make regarding the 'frontier void' concept relates to the intent of the search heuristics used here – both search heuristics used in this research have sequences modified via randomisation with the intent being, in part, to find sequences that span all parts of the feasibility continuum, thereby minimising frontier voids.

As an illustration, one will note from Tables 9a–d that the grid size is provided. This is the number of unique $Q$ and $S$ combinations that total enumeration for each problem provides. The difference between this grid size and the number of unique $Q$ and $S$ combinations provided by the search heuristic is the number of frontier voids. In standardised form, the voids divided by the grid size provide the frontier void percentage. As an example of this, consider Problem Set $A_2$, problem F. The efficient frontiers for the optimal solution, the simulated annealing solution and genetic algorithm solution are shown in Figure 2.

If one were to count the number of $Q$, $S$ combinations for the optimal solution from the figure above, one would find 36 – this is the grid size. One would also find 36 combinations for the simulated annealing solution, and 35 for the genetic algorithm solution. This means that the simulated annealing approach resulted in 0 frontier voids (36–36), while the genetic algorithm approach resulted in one frontier void (36–35) – the genetic algorithm failed to find a solution when 12 total batches ($Q = 12$) and five total setups ($S = 5$) were encountered. In standardised form, the simulated annealing approach results in a frontier void percentage of 0%, while the genetic algorithm approach results in a frontier void percentage of (1/36)*100%. It is desired that the search heuristics result in minimal frontier voids.

An important follow-up to the concept of frontier voids relates to the concept of frontier performance/relative inferiority. When frontier voids are encountered, the voids are not incorporated into the average inferiority calculation. The reason for this is because there is no reasonable value to assume for inferiority. This point is elaborated upon in the paper's concluding section. The example of problem set $A_2$, problem F results in an average inferiority of 0.78% for the simulated annealing heuristic, and an average inferiority of 6.85% for the genetic algorithm heuristic.

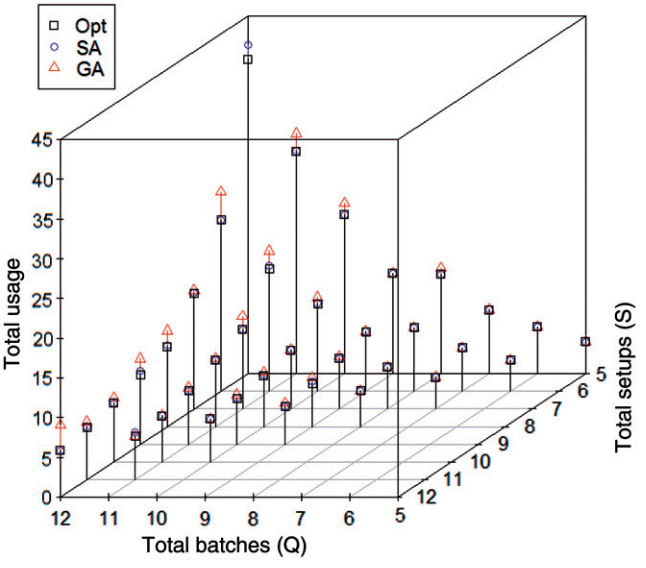


Figure 2. Efficient frontiers for Problem Set $A_2$, Problem F.

The third performance measure of interest is the CPU time – CPU time is captured for both search heuristics for all problems.

For the performance measures involving relative inferiority and frontier void percentage, a general research question is formulated where the null hypothesis states the performance measure of interest is zero, while the alternative hypothesis states the performance measure of interest is greater than zero. These questions are addressed via one-sample *t*-tests. This null hypothesis is a surrogate measure for 'statistical optimality' – performance measures of zero inferiority.

Given that the problems detailed in Table 9e are not solved to optimality, a comparison to optimal is not possible. Because of this, relative comparisons are made between the two approaches in terms of relative inferiority, frontier voids and CPU time.

### 4.3 *Heuristic parameters*

For a fair comparison to be made between the simulated annealing and genetic algorithm heuristics, it is important to select parameters that result in the search heuristics providing an equal number of solutions to evaluate for each problem. The number of *Simulations* is the same for each heuristic for each problem. The *CR* and *Iter* parameters for simulated annealing are calculated to result in the same number of solutions as the genetic algorithm solutions, which, for each problem, are determined by calculated values of *Gen* and *Pop* size. The probability of mutation ($P_M$) for the genetic algorithm approach is 0.02 for all problems. In general, and beyond the concern related to the magnitude of the search space, extensive pilot testing was performed on several randomly selected problems of different sizes to determine parameter values that would result in the most desirable solutions possible. Of course, in all instances, efforts were made to ensure that the search heuristics are as computationally efficient as possible.

Table 10a. Heuristic parameters for Problem Set $A_1$.

| Problem | *Simulations* | SA *CR* | SA *Iter* | GA *Gen* | GA *Pop* |
|---|---|---|---|---|---|
| B | 12 | 0.6976 | 6 | 12 | 6 |
| C | 20 | 0.7862 | 10 | 16 | 8 |
| D | 30 | 0.7862 | 15 | 20 | 10 |
| E | 24 | 0.7862 | 12 | 18 | 9 |
| F | 38 | 0.7862 | 19 | 24 | 12 |
| G | 48 | 0.7862 | 24 | 26 | 13 |
| H | 58 | 0.7862 | 29 | 30 | 15 |

Table 10b. Heuristic parameters for Problem Set $A_2$.

| Problem | *Simulations* | SA *CR* | SA *Iter* | GA *Gen* | GA *Pop* |
|---|---|---|---|---|---|
| B | 18 | 0.7415 | 9 | 14 | 7 |
| C | 28 | 0.8181 | 14 | 22 | 11 |
| D | 38 | 0.8497 | 19 | 28 | 14 |
| E | 46 | 0.8497 | 23 | 32 | 16 |
| F | 68 | 0.8531 | 34 | 38 | 19 |
| G | 84 | 0.8531 | 42 | 42 | 21 |
| H | 106 | 0.8531 | 53 | 48 | 24 |
| I | 68 | 0.8531 | 34 | 38 | 19 |
| J | 170 | 0.8531 | 85 | 60 | 30 |

Table 10c. Heuristic parameters for Problem Set $A_3$.

| Problem | *Simulations* | SA *CR* | SA *Iter* | GA *Gen* | GA *Pop* |
|---|---|---|---|---|---|
| B | 24 | 0.7799 | 12 | 18 | 9 |
| C | 46 | 0.8426 | 23 | 30 | 15 |
| D | 68 | 0.8891 | 34 | 44 | 22 |
| E | 108 | 0.9042 | 54 | 60 | 30 |
| F | 238 | 0.9225 | 119 | 98 | 49 |
| G | 352 | 0.9225 | 176 | 120 | 60 |
| H | 514 | 0.9225 | 257 | 144 | 72 |
| I | 798 | 0.9225 | 399 | 180 | 90 |
| J | 1002 | 0.9225 | 501 | 200 | 100 |

Tables 10a–d detail the search heuristic parameters for the $A_1$–$A_4$ problem sets compared with optimal. For Problem Set $A_5$, the simulated annealing parameters were (for all problems in this set) $Iter = 550$ and $CR = 0.995$. The genetic algorithm parameters for Problem Set $A_5$ were $Gen = 841$ and $Pop = 421$. For both approaches, 1000 simulations were performed.

### 4.4 *Computational experience*

Both search heuristics, along with the enumeration approach to find optimal solutions, are coded using the Java Development Kit, version 1.6, via the NetBeans IDE, version 6.5,

Table 10d. Heuristic parameters for Problem Set $A_4$.

| Problem | *Simulations* | SA *CR* | SA *Iter* | GA *Gen* | GA *Pop* |
|---|---|---|---|---|---|
| B | 38 | 0.8085 | 19 | 26 | 13 |
| C | 82 | 0.8793 | 41 | 46 | 23 |
| D | 222 | 0.9274 | 111 | 98 | 49 |
| E | 470 | 0.9944 | 235 | 518 | 259 |
| F | 270 | 0.9990 | 135 | 930 | 465 |
| G | 554 | 0.9990 | 277 | 1336 | 668 |
| H | 1066 | 0.9990 | 533 | 1850 | 925 |

Table 11a. Results for Problem Set $A_1$.

| Problem | SA inferiority (%) | GA inferiority (%) | SA void (%) | GA void (%) | SA CPU | GA CPU |
|---|---|---|---|---|---|---|
| B | 1.30 | 7.46 | 0.00 | 9.09 | 0.0009 | 0.0012 |
| C | 1.03 | 13.64 | 9.52 | 4.76 | 0.0013 | 0.0009 |
| D | 1.43 | 11.97 | 0.00 | 0.00 | 0.0013 | 0.0011 |
| E | 4.73 | 10.70 | 0.00 | 9.52 | 0.0013 | 0.0009 |
| F | 1.19 | 7.13 | 0.00 | 0.00 | 0.0020 | 0.0016 |
| G | 0.41 | 4.62 | 0.00 | 0.00 | 0.0023 | 0.0022 |
| H | 0.00 | 1.21 | 0.00 | 4.76 | 0.0026 | 0.0026 |
| Aggregate | 1.44 | 8.11 | 1.46 | 3.65 | 0.0117 | 0.0105 |

Table 11b. Results for Problem Set $A_2$.

| Problem | SA inferiority (%) | GA inferiority (%) | SA void (%) | GA void (%) | SA CPU | GA CPU |
|---|---|---|---|---|---|---|
| B | 1.15 | 5.96 | 13.33 | 0.00 | 0.0013 | 0.0008 |
| C | 6.56 | 8.63 | 0.00 | 0.00 | 0.0024 | 0.0014 |
| D | 2.60 | 8.72 | 0.00 | 5.71 | 0.0039 | 0.0021 |
| E | 2.79 | 6.37 | 2.86 | 2.86 | 0.0035 | 0.0029 |
| F | 0.78 | 6.85 | 0.00 | 2.78 | 0.0062 | 0.0051 |
| G | 1.58 | 4.19 | 0.00 | 0.00 | 0.0070 | 0.0073 |
| H | 2.42 | 7.69 | 0.00 | 2.78 | 0.0121 | 0.0119 |
| I | 1.55 | 6.29 | 0.00 | 0.00 | 0.0060 | 0.0050 |
| J | 1.21 | 2.50 | 0.00 | 0.00 | 0.0288 | 0.0295 |
| Aggregate | 2.29 | 6.36 | 1.03 | 1.71 | 0.0712 | 0.0660 |

under the Microsoft Vista Operating System. The CPU is an Intel Pentium Dual, Model 3180, with CPU speeds of 2.0 GHz. The system memory is 2 GB.

## 5. Experimental results

Tables 11a–e summarise the performance measure results by problem sets. The tables are self-explanatory in terms of how the performance measures were defined in Section 4.2.

Table 11c. Results for Problem Set $A_3$.

| Problem | SA inferiority (%) | GA inferiority (%) | SA void (%) | GA void (%) | SA CPU | GA CPU |
|---|---|---|---|---|---|---|
| B | 3.19 | 3.21 | 0.00 | 10.00 | 0.0023 | 0.0014 |
| C | 4.59 | 5.71 | 0.00 | 0.00 | 0.0112 | 0.0028 |
| D | 0.86 | 3.88 | 0.00 | 2.04 | 0.0091 | 0.0073 |
| E | 0.55 | 4.01 | 1.79 | 0.00 | 0.0202 | 0.0213 |
| F | 1.60 | 6.00 | 1.52 | 1.52 | 0.1175 | 0.1249 |
| G | 0.18 | 3.19 | 0.00 | 0.00 | 0.2601 | 0.2776 |
| H | 1.71 | 2.63 | 0.00 | 1.52 | 0.5372 | 0.5811 |
| I | 0.65 | 3.99 | 0.00 | 0.00 | 1.2679 | 1.4018 |
| J | 0.00 | 1.22 | 0.00 | 1.52 | 1.9672 | 2.1116 |
| Aggregate | 1.48 | 3.76 | 0.41 | 1.23 | 4.1927 | 4.5298 |



Table 11d. Results for Problem Set $A_4$.

| Problem | SA inferiority (%) | GA inferiority (%) | SA void (%) | GA void (%) | SA CPU | GA CPU |
|---|---|---|---|---|---|---|
| B | 0.00 | 0.00 | 0.00 | 0.00 | 0.0028 | 0.0026 |
| C | 1.22 | 3.95 | 0.00 | 2.22 | 0.0139 | 0.0121 |
| D | 2.07 | 1.75 | 0.00 | 0.00 | 0.1288 | 0.1392 |
| E | 0.31 | 5.30 | 1.52 | 3.03 | 7.9029 | 8.4736 |
| F | 0.77 | 6.28 | 0.00 | 0.74 | 14.1132 | 16.1451 |
| G | 0.26 | 8.54 | 0.00 | 1.47 | 58.0793 | 68.0232 |
| H | 2.34 | 8.83 | 0.00 | 0.74 | 211.2852 | 261.3644 |
| Aggregate | 0.99 | 4.95 | 0.29 | 1.33 | 291.5261 | 354.1602 |

Table 11e. Results for Problem Set $A_5$.

| Problem | GA inferiority over SA (usage) (%) | GA inferiority over SA (voids) | GA inferiority over SA (CPU) (%) |
|---|---|---|---|
| F | 0.37 | 0 | 9.65 |
| G | 0.91 | 0 | 8.36 |
| H | 0.73 | 0 | 15.26 |
| I | −0.87 | 0 | 9.46 |
| J | 2.31 | 4 | 9.86 |

CPU times are given in minutes. Aggregate measures are means for relative inferiority and void percentage, and sums for CPU minutes.

For Problem Set $A_5$, a comparison with optimal is not made due to the large size of the problems. Because of this, Table 11e details only a comparison between the two approaches. Results are presented in a form where the inferiority of the genetic algorithm approach to the simulated annealing approach is presented. In terms of the *Usage* measure and the CPU time measure, the relative inferiority (in %) of the genetic algorithm is stated.

Table 12. Summary of performance measures for Problem Sets $A_1$–$A_4$.

| Problem set | SA inferiority (%) | GA inferiority (%) | SA void (%) | GA void (%) | SA CPU | GA CPU |
|---|---|---|---|---|---|---|
| $A_1$ | 1.44 | 8.11 | 1.46 | 3.65 | 0.0117 | 0.0105 |
| $A_2$ | 2.29 | 6.36 | 1.03 | 1.71 | 0.0712 | 0.0660 |
| $A_3$ | 1.48 | 3.76 | 0.41 | 1.23 | 4.1927 | 4.5298 |
| $A_4$ | 0.99 | 4.95 | 0.29 | 1.33 | 291.5261 | 354.1602 |
| Aggregate | 1.59 | 5.70 | 0.95 | 2.10 | 295.80 | 358.77 |
| Std. Dev | 1.50 | 3.17 | 2.87 | 2.91 | | |
| *t*-Statistic | 6.03 | 10.18 | 1.88 | 4.07 | | |
| *p*-Value | <0.0001 | <0.0001 | 0.0346 | 0.0001 | | |

For the voids measure, the number of voids associated with the genetic algorithm approach compared with the number of voids associated with the simulated annealing approach is stated.

Table 12 shows a performance measure summary for problem sets $A_1 - A_4$. The aggregate measures of relative inferiority and void percentage are averages for these 32 problems. CPU times are summed for the problem sets. The three bottom lines of Table 12 detail the hypothesis tests that involve the relative inferiority measures and the frontier void percentage values being zero (optimal) vs. being greater than zero (suboptimal).

As one can see from Table 12, the simulated annealing heuristic clearly outperforms the genetic algorithm in all facets: relative inferiority, frontier void percentage and CPU minutes all show that the simulated annealing is a superior approach to the genetic algorithm. In terms of CPU time, there is no consequential difference until the larger problem sets are encountered – at that point, the difference in CPU requirements becomes profound. Neither of the performance measure types for either heuristic show insignificant differences with zero (statistical optimality) at the $\alpha = 0.05$ level, but the frontier void percentage for the simulated annealing approach does show an insignificant difference with zero (statistical optimality) at the $\alpha = 0.03$ level.

For the larger problems, whose performance is summarised in Table 11e, it is shown that, with the exception of one problem (Problem I) where the genetic algorithm provides a better inferiority performance compared with simulated annealing in terms of usage, simulated annealing is dominant over the genetic algorithm in terms of usage, voids and CPU requirements.

## 6. Concluding comments

This research effort has presented two heuristic techniques which strive to find production sequences that provide JIT flexibility while simultaneously considering both the number of setups and the total number of batches which result from the sequences. This combination of attributes is, in the author's opinion, something in which the literature can benefit. Another contribution of this research, in the opinion of the author, is the three-dimensional efficient frontier that is sought by the search process. Two-dimensional efficient frontiers have been used to minimise the usage rate for given setups, but the incorporation of the third dimension, via total batches, is also something in which the literature can reap benefit.

There are some issues associated with this research effort that require further discussion, and these are discussed in the subsections below.

### 6.1 *Discussion of heuristic performance*

The simulated annealing approach outperforms the genetic algorithm approach with respect to the three performance measures of interest: relative inferiority, frontier void percentage, and CPU requirement. In essence, the genetic algorithm approach is dominated by the simulated annealing approach. Given consideration, this is not terribly surprising. Previous research efforts with the intent of sequencing for JIT purposes have uncovered the relative inferiority of genetic algorithms to other sequences (McMullen *et al.* 2000). The general reason for the inferior genetic algorithm performance (when CPU time is not considered) pertains to the aggressiveness of the genetic algorithm compared with that of simulated annealing. When simulated annealing is performed during the search, two items within a single sequence are altered. When the genetic algorithm is performed during the search, several items within each sequence of the population are altered. This continual large modification of the sequences clearly leads to sub-optimal results in terms of relative inferiority. In terms of the objective of frontier void percentage, the aggressiveness of the genetic algorithm fails to reach all parts of the search space. In layman's terms, one could classify, at least in the context of the present application, the genetic algorithm approach is like 'throwing out the baby with the bathwater'. As far as CPU performance is considered, the genetic algorithm approach is manipulating more items in the sequence when compared with its simulated annealing counterpart, which requires more CPU resources, increasing its search time.

The simulated annealing approach is a more deliberate approach than the genetic algorithm approach – modifications are made at a slower pace. This practice results in a more thorough exploration of the search space, resulting in better relative inferiority and frontier void percentage than its genetic algorithm counterpart. Given that less modification is made when compared with its genetic algorithm counterpart, less CPU requirements result in greater efficiency.

### 6.2 *Limitations of research*

Unfortunately, frontier voids bias findings. This is due to the fact that when frontier voids are discovered, they are removed from the relative inferiority calculation, for lack of a better alternative. This practice places a downward bias on the relative inferiority calculation. An example of this is when a search heuristic is used to find the efficient frontier for a problem. If a single frontier void exists, but all other usages for each $Q$, $S$ combination are found to be optimal, then the relative inferiority of this solution is zero, despite having one frontier void. Given the presence of this issue, it is important to interpret both relative inferiority and frontier void percentage for all problems studied.

Another issue with this research that requires further elaboration is the batch-size selection. It was stated in Section 2.2 that the minimum batch size is assumed to be one unit, while the maximum batch size is assumed to be the total demand for the item of interest ($d_i$). The search heuristic computes all feasible batch-size possibilities on the $[1, d_i]$ interval. In reality, this continuum of batch sizes may not be feasible. Stakeholders outside the control of the decision-makers may have input as to what batch sizes are actually used.

This point must be considered with regard to implementation of the presented methodology.

Another limiting aspect to this research effort pertains to the concept of setup time. Setup times are given general treatment in this research – setup times are not 'added', but only 'counted' as setups when changeovers between different items are needed.

Sequencing problems are combinatorial in nature, which is suggestive of computational intractability. This research is no exception to that. The largest problem solved to optimality for this effort (via complete enumerations) had a total of 182,206,343,832 feasible solutions. The CPU time required to obtain this optimal solution was 31,368.92 minutes. This translates into more than three weeks of CPU time to find the optimal solution. Of course, a fraction of this time was needed to find solutions to this problem via the two presented search heuristics. Nevertheless, it is not possible to find an optimal solution as a benchmark without this practice. All of this suggests that if the decision-maker is interested in comparing their work with optimal (for problem sizes of any consequence), a large investment in CPU time is required.



### 6.3 *Future research opportunities*

Limitations associated with a particular research effort translate directly into opportunities for new research. This research effort is no exception to this rule. One opportunity for further investigation relates to attempting to improve the performance of the genetic algorithm approach. Genetic algorithms have seen too much success in the literature to be 'written off' for JIT sequencing issues. Perhaps a modification of the genetic algorithm presented here could benefit the literature. At the same time, it is believed that any modification of a simulated annealing approach would only provide research of incremental value relative to what is presented here. Of course, other search meta-heuristics, such as tabu search, could also be explored.

Another opportunity for new research relative to what is presented here pertains to batch-sizing policies. Here, all feasible batch sizes on the $[1, d_i]$ interval are explored. Considering batch sizes as dictated by supplier and/or customers may also have value.

As opposed to counting the number of needed setups, new research could be performed to investigate the setup times, and the stochastic nature of these setup times, that are associated with various solutions. Considering storage/capacity constraints on the model would also provide more realistic value to subsequent research efforts.

Finally, as high-speed computing resources become more and more attainable, problems of a larger size may be something worth pursuing – this would provide the ability to study more realistic scenarios.

## Acknowledgement

The author would like to thank the two anonymous referees for their assistance in the improvement of this paper.

## References

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.

Hillier, F.S. and Lieberman, G.J., 2005. *Introduction to operations research*. 8th ed. Boston, MA: McGraw-Hill Higher Education.

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

Kubiak, W. and Yavuz, M., 2008. Just-in-time smoothing through batching. *Manufacturing & Service Operations Management*, 10, 506–518.

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

McMullen, P.R., 2001a. An efficient frontier approach to addressing JIT sequencing problems with setups via search heuristics. *Computers & Industrial Engineering*, 41, 335–353.

McMullen, P.R., 2001b. An ant colony optimization approach to addressing a JIT sequencing problem with multiple objectives. *Artificial Intelligence in Engineering*, 15, 309–317.

McMullen, P.R., Tarasewich, P., and Frazier, G.V., 2000. Using genetic algorithms to solve the JIT sequencing problem with setups. *International Journal of Production Research*, 38, 2653–2670.

Metropolis, N., *et al*., 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, 192–207.

Monden, Y., 1998. *Toyota production system: an integrated approach to just-in-time*. 3rd ed. GA: Engineering & Management Press.

Tan, K.C., Khor, E.F., and Lee, T.H., 2005. *Multiobjective evolutionary algorithms and applications*. London, UK: Springer.

Tucker, A., 2007. *Applied combinatorics*. New York: Wiley.

Yavuz, M. and Tüfekçi, S., 2004. Some lower bounds on the mixed-model level scheduling problems. *In*: *Proceedings of the 10th international conference on industry, engineering and management systems*, Cocoa Beach, FL, 385–395.

Yavuz, M., Akcali, E., and Tüfekçi, S., 2006. Optimizing production smoothing decisions via batch selection for mixed-model just-in-time manufacturing systems with arbitrary setup and processing times. *International Journal of Production Research*, 44, 3061–3081.

Yavuz, M. and Tüfekçi, S., 2007. Analysis and solution to the single-level batch production smoothing problem. *International Journal of Production Research*, 45, 3893–3916.



## Appendix A: Explanation of usage rate computation

The notation in Table 1 provides details on the objective of finding the minimal usage rate for each combination of total batches ($Q$) and setups ($S$). This notation provides a usage rate result that is not in consistent agreement with the usage rate computation from the original Miltenburg (1989) paper. The algorithm below converts a sequence in batch notation to a sequence in single-item notation:

```
counter = 0;
for k = 1 to Q
  for i = 1 to n
    if(X_{i,k} == 1) then
      marker = i;
      break;
  next i
  for j = 1 to b_{marker}
```

```
        counter = counter + 1;
        for i = 1 to n
          x_{i,counter} = X_{i,k};
      next i
      next j
    next k
```

where $x_{i,k} = 1$ if item $i$ is in the $k$th position of the single-item notation sequence; 0 otherwise. The usage rate via this notation is as follows:

$$\text{Min: } \sum_{k=1}^{D_T} \sum_{i=1}^{n} \left( y_{ik} - k \frac{b_i q_i}{D_T} \right)^2, \quad \forall S, Q \tag{A1}$$

where

$$D_T = \sum_{i=1}^{n} b_i q_i \tag{A2}$$

and

$$y_{i,k} = \sum_{j=1}^{k} x_{i,j} \tag{A3}$$

One will notice that this is similar to what is presented in batch notation, with the general exception being the batch notation indexes through all $Q$ batch positions, while the single-unit notation indexes through all $D_T$ sequence positions.

An example is the problem below, resulting in the following batch sequence: $A_2B_2C_1A_2$, where the subscripts represent batch sizes. Properties in batch notation are as follows:

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 2 & 2 & 1 \end{bmatrix}, \quad q = \begin{bmatrix} 2 & 1 & 1 \end{bmatrix}, \quad S = 4, \quad Q = 4$$

According to Equation (1), the usage rate is 3.875. Setups ($S$) are four, and there are four total batches ($Q = 4$).

When the above algorithm is used to convert the batch notation into single-item notation, the sequence is AABBCAA, with relevant properties

$$x = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 2 & 0 \\ 2 & 2 & 1 \\ 3 & 2 & 1 \\ 4 & 2 & 1 \end{bmatrix}, \quad D_T = 7, \quad S = 4, \quad Q = 4$$

This results in a usage rate of 4.2857, different from that above. Both usage rate computations provide informative values – their intent is to be compared with usage rates associated with other sequences. JIT-friendly sequences provide low usage rates regardless of which type of notation is used, and JIT-unfriendly sequences provide high usage rates regardless of which type of notation is used.

For this research, the usage rate is determined via the batch notation being converted into single-item notation, so as to be consistent with the original Miltenburg (1989) equation. Setups ($S$) and total batches ($Q$) are not affected by the conversion from batch notation to single-item notation.