American Journal of Operations Research, **2011, 1, 220-228**
doi:10.4236/ajor.2011.14025 Published Online December 2011 (http://www.SciRP.org/journal/ajor)

Limited Resequencing for Mixed Models with Multiple Objectives

Patrick R. McMullen
Schools of Business, Wake Forest University, Winston-Salem, USA
E-mail: *mcmullpr@wfu.edu*

Received August 29, 2011; *revised September* 30, 2011; *accepted October* 12, 2011

Abstract

This research presents a problem relevant to production scheduling for mixed models—production schedules that contain several unique items, but each unique item may have multiple units that require processing. The presented research details a variant of this problem where, over multiple processes, re-sequencing is permitted to a small degree so as to exploit efficiencies with the intent of optimizing the objectives of required setups and parts usage rate via an efficient frontier. The problem is combinatorial in nature. Enumeration is used on a variety of test problems from the literature, and a search heuristic is used to compare optimal solutions with heuristic based solutions. Experimentation shows that the heuristic solutions approach optimality, but with opportunities for improvement.

Keywords: Sequencing, Heuristics, Simulated Annealing

1. Introduction

Sequencing has been a popular topic in production research for as long as production research has been in existence. Typically, sequencing is done such that some objective associated with operational performance is optimized. Typically, these objectives involve entities such as labor use, in-process inventories, and flexibility.

1.1. Setups and Usage Rate

Of interest here are two objectives: one objective is concerned with minimization of setups, the other is concerned with stability of materials usage. The first objecttive is easily understood, while the second is less understood. The stability of materials usage is a matter of concern to those involved with lean manufacturing systems involving multiple quantities of each unique item requiring processing. Scenarios where items require processing when there are multiple units of each unique item are frequently referred to as *mixed models*. Manufacturing systems such as this desire some progression of each item's processing in proportion to the demand of the item of interest. A metric associated with the stability of materials usage was contributed by the seminal work of Miltenburg [1]. Miltenburg also contributed algorithms concerned with obtaining production sequences to minimize the material usage rate. To illustrate the principle of mixed model sequencing, consider a simple example where it is required to process four units of item A, two units of item B and one unit of item C. One possible sequence is as follows:

$$\text{AAAABBC} \qquad (S_1)$$

The sequence above does not accomplish the objective of stabilizing material usage. All of the *A*s are process, followed by the *B*s, finishing with the *C*s. The Miltenburg metric is optimized when the items are scrambled such that the presence of each item's sequence positions is proportional to its demand. The following sequence is more desirable with the material usage rate is considered:

$$\text{ABACABA} \qquad (S_2)$$

In layman's terms, one may better understand the need for stabilizing the material usage rate with a variant to this example. Imagine a scenario where demand is amplified by a factor of 1000 and that units A, B and C are needed by a customer for assembly (1000 assemblies required). Each single assembly requires four of A, two of B and one of C. One option is to employ a form of S_1 above—producing 4000 of A, then 2000 of B and finally 1000 of C—a production sequence of 7000 members. Now imagine a disaster (a power outage, for example) halfway through the sequence—only 3500 units of

A were produced. This disaster leaves the customer unable to assemble any of the 1000 demanded units—a death blow to the intentions of JIT/lean systems. On the other hand, consider employing S_2 1000 times and the same disaster occurs halfway through the sequence. When S_2 is employed, the customer is still able to complete 3500 assemblies, which are 3500 assemblies more than when the S_1 approach is employed.

While S_2 has its advantages over S_1 in terms of stability of material usage, it also has a major drawback—it results in many setups, frequently changing over from one unique item to another. S_1 is superior to S_2 in terms of setups. In fact, the two objectives of concern here are in conflict with each other—they are inversely correlated [2,3]. This is a common affliction for those interested in multiple-objective optimization [4]. Given the tradeoff between the number of setups and the stability of material usage, there is motivation to find sequences that provide desirability in terms of both objectives. In fact, multi-objective optimization problems are rife with conflicting objectives [5].

1.2. Sequencing and Limited Re-Sequencing

When multiple processes are required for a production problem, one can conceivably consider the situation an opportunity for improving the schedule from process to process. This "improving" the schedule can come in the form of re-sequencing. In this context, re-sequencing suggests making adjustments to the sequence prior to each process. In reality, these adjustments would typically be minor in nature, as physical limitations would prevent any re-sequencing to a large degree. As such, the re-sequencing associated with this research effort is considered *limited re-sequencing*. The work of Lahmar and Benjaafar [6] describes how physical buffers dictate the degree of re-sequencing that is feasible. Their work presents techniques to find desirable sequences, and is applied to automotive industry applications.

As stated by Lahmar and Benjaafar [6], physical space availability dictates buffering ability, and subsequently the degree of re-sequencing that is feasible. For this research effort, it will be assumed that there is buffer space available to permit re-sequencing of only one item in the sequence. This means that one item is permitted to be taken out of the sequence and reinserted into a later part of the sequence. **Figure 1** provides a numerical example of a feasible re-sequencing:

Here, item A, the first item in the sequence, is taken out of sequence, and later re-inserted. It is pushed back in the sequence. Meanwhile, the intervening items in the sequence move forward one position because of item A being pushed back. **Figure 2** shows another type of feasible sequence:

Here, item A is pushed back, as is item B. Despite multiple pushbacks, this scenario is still feasible, as the buffer space of one unit is never exceeded. Furthermore, notice that none of the items have moved up more than one position in the sequence. **Figure 3** shows an illustration of an infeasible re-sequencing:

This particular sequence is not permitted. The reason for this is because the buffer capacity of one unit would be exceeded. The first two items in the sequence would be held in buffer at the same time, which is not permitted. Another indication as to the infeasibility of this sequence is that the first item C would have move forward two positions in the sequence, which is not permitted, given it exceeds the buffer size of one.

1.3. Multiple Processes and Combinatorial Difficulties

For some number of processes, limited re-sequencing can be performed. For three processes, for example, there are three opportunities for re-sequencing. For each process, there is some number of re-sequencing possibilities. This number of re-sequencing possibilities is equal to the number of feasible re-sequencing possibilities associated with the original sequence. In this context, the original sequence is referred to as the *parent sequence*, and the re-sequenced sequence is referred to as the *child sequence*. For example, consider a scenario where two units of A are demanded, one unit of both B and C are demanded. From this demand pattern, the trivial sequence of AABC is constructed. AABC is considered the *parent* sequence for this example. This parent sequence has ${}_4P_{2,1,1}$ = twelve feasible permutations, or sequences. Of these twelve permutations, there are only four which are considered feasible in terms of the limited re-sequencing with a buffer size of one unit. These sequences are AABC, AACB, ABAC, and ABCA. In the context of the description above, these can be considered *child* sequences. Note that the AABC sequence is essentially a direct mapping of its *parent* sequence. For the next

$\boxed{A}$ABB|CC (Initial Sequence)
ABBACC (New Sequence)

Figure 1. A feasible re-sequencing.

$\boxed{A}$AB|$\boxed{B}$C|C (Initial Sequence)
ABACBC (New Sequence)

Figure 2. A feasible re-sequencing with two pushbacks.

$\boxed{AB}$C|^{A}AB|^{B}C (Initial Sequence)
CAABBC (New Sequence)

Figure 3. An infeasible re-sequencing.

process, the four child sequences of AABC, AACB, ABAC and ABCA can be thought of a *parent* sequences, and their *child* sequences could be generated. **Figure 4** shows a tree diagram of limited re-sequencing through two processes for the initial sequence of AABC.
(www. joydivisionman.com/AJOR1/Figure4.html).

One will notice from **Figure 4** that even for a small sequence, there are many feasible re-sequences as the number of processes increase. For this particular example, there is one sequence at the 0^{th} level, four at the first level, and twenty-one at the second level. These twenty-one sequences at the second level suggest that there are twenty-one unique re-sequencing opportunities from the original sequence through two processes. Also included in **Figure 4** is the number of cumulative setups (S) and usage rate (U) through the appropriate number of processes. Calculation of the setups (S) and usage rates (U) is detailed in the next section.

Problems of this type, in the language of data structures, can be thought of as tree traversal problems. Each feasible re-sequence can be thought of as a *node*, and each of the individual required processes can be thought of as *levels*. Also similar to tree traversal problems in data structures, the problem presented here can be thought of as intractable from a combinatorial perspective.

1.4. Multiple Objectives and Efficient Frontier

As mentioned in Section 1.1, there are two objectives of interest here, minimization of setups and minimization of the material usage rate. It was also mentioned that these two objectives are inversely related to each other. As such, to obtain sequences which are desirable in terms of both objectives, some sacrifices must be made. One could employ composite objective functions where each individual objective is weighted according at the researcher's discretion. This weighting scheme, however, is not desired for this research. Here, it is desired to avoid weighting schema and exploit an efficient frontier. Exploitation via an efficient frontier is well-suited for this type of problem because one of the objectives (setups) displays a variation of a discrete nature. The other objective (usage rate) displays a variation of a continuous nature. Given these circumstances, one can be motivated to find sequences that minimize the usage rate for each feasible number of associated setups [7]. **Figure 5** shows an efficient frontier through nine processes for the example problem. The line represents the minimum usage rate for each associated number of setups. For this example problem, there were 6,651,536 nodes traversed through the nine levels. 5,542,861 of these nodes were on the ninth level. Of these 5,542,861 nodes, there are ten unique numbers of setups. For each of these numbers of setups, the minimum usage rate is plotted.

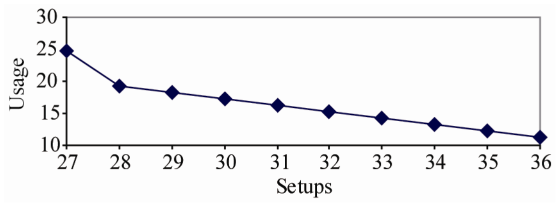

Figure 5. Efficient frontier through nine processes for example problem.

The usage rates associated with the numbers of setups that are not minima are not shown here—the sequences with these usage rates are considered inferior to those that are on the efficient frontier. It is, of course, desired to capture the sequences that comprise the efficient frontier.

1.5. Presented Research and Its Contributions

The research presented here is an extension of Lahmar and Benjaafar's earlier work [6]. While their work concentrates on minimization of changeover cost, the research presented here treats re-sequencing in a more generalized format, and recognizes the issues which are critical to the success of JIT/lean implementations, notably the number of setups required for each sequence and its material usage rate [1]. Obtaining the efficient frontier is the chosen approach to this multiple objective problem. Unfortunately, obtaining an efficient frontier for problems of any realistic size is difficult—**Figure 4** illustrates this unpleasant reality for a very small problem. Even with the high-speed computing resources that are available today, obtaining these efficient frontiers is prohibitive from both a CPU and memory standpoint. As such, obtaining optimal solutions for large-scale problems is not practical, and therefore desirable solutions must be obtained via search heuristics.

It is also important to emphasize the value of re-sequencing in general. One can make the argument that any permutation of demand could be used for all processes. This is true, but there are disadvantages to this. Consider the example above. The associated permutetions permit either three or four setups. If either of these options is employed for the nine processes in the example, there would be either (27) or (36) setups in total—no intervening number of setups would be possible. With re-sequencing, however, all numbers of sequences between (27) and (36) are possible, greatly adding flexibility to those concerned with finding schedules satisfactory with respect to multiple objectives. This point is considered a major contribution of this effort.

For this research, problem sets from the literature, along with some new problem sets, are used to perform limited re-sequencing with respect to the objectives of setups and usage rate through multiple processes. Optimal solutions via complete enumeration are obtained. These solutions are used as benchmarks for comparison to solutions obtained via the search heuristic of simulated annealing. General conclusions and observations are provided as well.

2. Methodology

Prior to presentation of the methodological details, assumptions of the problem in general are detailed:

- Changeover times between differing items are considered non-negligible. This assumption is common for past research efforts considering both setups and usage rates [2].
- The effort required to place a unit in buffer for later insertion into the sequence is considered negligible.

For a detailed presentation of the mathematical programming model the following variables, parameters and definitions are provided:

Item	Description
	Decision Variable
x_{ijq}	1 if item j is present in the i^{th} position of the sequence, for the q^{th} process; 0 otherwise
	Endogenous Variable
y_{ijq}	total units of item j placed in sequence through i sequence positions, for the q^{th} process
	Parameters
n	number of items to be sequenced
m	number of unique items to be sequenced
p	number of processes required
d_j	demand of item j
	Indices
i	Index for n ($1,2,\cdots i,\cdots n$)
j	Index for m ($1,2,\cdots j,\cdots m$)
q	Index for p ($1,2,\cdots q,\cdots p$)

2.1. Minimal Sequences for Associated Number of Setups

Miltenburg's seminal effort quantified what is now referred to as the material usage rate. This metric is essentially an aggregated difference between has been sequenced and what should have been sequenced through all n sequence positions. For this research effort, the metric is modified to consider multiple processes and the number of setups associated with the sequence of interest:

Min

$$Usage = \sum_{q=1}^{p}\left(\sum_{i=1}^{n}\sum_{j=1}^{m}\left(y_{ijq} - i\left(d_i/n\right)\right)^2\right)_{Setups}, \tag{1}$$

where

$$setups = \sum_{q=1}^{p}\left(1+\sum_{i=2}^{n}\left(1-\left(\sum_{j=1}^{m} x_{ijq}x_{i-1,jq}\right)\right)\right), \tag{2}$$

and

$$y_{ijq} = \sum_{k=1}^{i}\left(x_{kjq} + x_{k-1,jq}\right), \quad \forall j,q \tag{3}$$

Subject to:

$$x_{0jq} = 0, \quad \forall j,q \tag{4}$$

$$\sum_{i=1}^{n} x_{ijq} = d_j, \quad \forall j,q \tag{5}$$

$$\sum_{j=1}^{m} x_{ijq} = 1, \quad \forall i,q \tag{6}$$

Equation (1) seeks to minimize the usage rate associated with the number of setups from the resultant sequence. The number of setups from the associated sequence is determined via (2). Both of these metrics are summed through all p processes. Equation (3) relates the endogenous variable y_{ijq} to the decision variable x_{ijq}. The first constraint in (4) initializes the decision variables to zero for the 0th position in the sequence. The constraint in (5) forces the decision variables to yield sequences consistent with the pre-determined item demand. The constraint in (6) guarantees that each sequence position contain exactly one item.

2.2. Problems with Objective Function

There are some features associated the above model that prevent one from obtaining optimal solutions via mathematical programming approaches. First, both Equations (1) and (2) are nonlinear and discontinuous in nature, which complicate the success of a mathematical programming approach [8]. Second, there are two objective functions (setups and usage) which further complicate finding an optimal solution.

2.3. Enumerative and Heuristic Approaches

With the complexities associated with the formulation above, other measures must be taken to capture the efficient frontier that is desired. Enumeration can be used, similar to that shown in **Figure 4**. Unfortunately, enumeration is not computationally feasible for large problems from both a CPU and memory standpoint—problems of this type are rigorous from a combinatorial standpoint to a very large degree. For small problems, enumeration is effective in capturing the optimal efficient frontier.

For larger problems, search heuristics can be used—algorithms used to obtain desirable, although not necessarily optimal solutions with a reasonable degree of computational effort. There are many search heuristics available to the research, but simulated annealing is used for this particular effort.

2.4. Simulated Annealing Heuristic

Simulated annealing [9,10] is the search heuristic used for this research. It is acknowledged that there are many search heuristics available for finding desirable solutions to combinatorial optimization problems with a reasonable computational effort. Genetic Algorithms [11], Tabu Search [12] and Ant Colony Optimization [13] are just a few. Choosing the "best" search heuristic for this type of problem is beyond the scope of the paper, and this point is emphasized later as an opportunity for subsequent research. The following subsections detail the steps associated with the search heuristic for the problem of interest.

2.4.1. Initialization (Step 0)

The first step in the heuristic is to initialize all of the parameters to appropriate values. The relevant parameters are shown below:

Parameter	Description
T_1	Initial temperature
T_F	Final temperature
T	Current temperature
Iter	Iterations for each temperature level
CR	Cooling rate
k_B	Boltzman constant

Relevant variable values which are endogenous (determined by the search heuristic) are defined as follows:

Variable	Description
$Usage_T[Setups]$	Usage for test solution with *Setups*
$Usage_C[Setups]$	Usage for current solution with *Setups*
$Usage_B[Setups]$	Usage for best solution with *Setups*
δ	Relative inferiority of test solution
P_A	Probability of accepting inferior solution

An initial sequence is chosen. For this research, the "trivial" sequence is always used for initialization. The "trivial" sequence assumes that all item A units will be placed at the beginning of the sequence, all item B units are next, etc. The value of the *q* index is set to zero. $Usage_C[Setups]$ and $Usage_B[Setups]$ are initialized to very high values for all possible values of *Setups*. T is initialized to T_1.

2.4.2. Modification (Step 1)

Modification is performed in a fashion identical to that shown in **Figure 1**. Single "pushbacks" are used for the presented heuristic. The targeted item and its new "pushback" location are both chosen randomly according to a uniform probability distribution. The resultant sequence in process *q* is the parent sequence for process *q* + 1. In the spirit of the example problem shown in **Figure 4**, here's an example of modification through four processes.

Process:	$q = 0$	$q = 1$	$q = 2$	$q = 3$	$q = 4$
Seq.	AABC	ABAC	ABCA	BACA	BAAC

Note how that for each subsequent process, an item never moves forward more than one position in the sequence—this is the enforcement of the constraint restricting the buffer size to one.

2.4.3. Objective Function (Step 2)

After modification is performed for all *p* processes, the number of *Setups* and *Usage* rates are determined according to Equations (1) and (2). Continuing with the example above, *Setups* and *Usage* values through each of the four processes are as follows:

Proc.(q):	0	1	2	3	4
Seq:	AABC	ABAC	ABCA	BACA	BAAC
S/U:	N/A	4/1.75	4/1.25	4/1.75	3/2.25
ΣS/ΣU	N/A	4/1.75	8/3.00	12/4.75	**15/7.00**

The most important values associated with these *p* modifications are the cumulative Setups and Usage values for the p^{th} process. This is considered the objective function value. For this example, the four modifications resulted in a *Usage* rate of 15.00 for the (15) associated *Setups*. Again, it is desired to minimize the *Usage* rate for each associated number of *Setups*. This usage rate is referred to as $Usage_T[Setups]$.

2.4.4. Comparison (Step 3)

The desirability of the objective function is compared to that of the current solution via their *Usage* rates associated with their *Setups*. If the *Usage* rate for the test solution is less than that of the current solution, the test solution replaces the current solution. If the *Usage* rate for the test solution is less than that of the best solution, the test solution replaces the best solution.

Otherwise, the Metropolis criterion for replacing a current solution with one having relative inferiority is explored [8,9,14]. The relative inferiority of the test solution is computed via the following:

$$\delta = \frac{Usage_T\left[Setups\right] - Usage_C\left[Setups\right]}{Usage_C\left[Setups\right]} \qquad (7)$$

The probability of accepting this relatively inferior solution is determined as follows:

$$P_A = \exp\left(-\delta k_B / T\right) \qquad (8)$$

A random number on the (0, 1) interval is generated according to a uniform distribution. If this value is less than P_A, the relatively inferior test solution replaces the current solution. Otherwise, the current solution remains current.

2.4.5. Incrementation (Step 4)

Steps 1 through 3 are repeated *Iter* times. After *Iter* times, the current temperature is adjusted according to the following relation:

$$T = T \cdot CR \qquad (9)$$

This pattern continues while T exceeds T_F.

2.4.6. Completion (Step 5)

When the value of T no longer exceeds T_F, the search heuristic stops, and the efficient frontier values ($Usage_B$ [*Setups*]) are reported.

2.4.7. Pseudocode

The steps below show the steps of this heuristic in pseudocode:

```
Initialize();
while($T > T_F$){
    for(h = 1; h <= $Iter$; h++){
    modification();
    determineObjectiveFunction();
    if($Usage_T[Setups] < Usage_C[Setups]$){
        replaceCurrent();
    if($Usage_T[Setups] < Usage_B[Setups]$)
        replaceBest();}
    else testMetropolisCriterion();}
$T = T*CR$;}
reportEfficientFrontier();
```

3. Experimentation

To measure the performance of the heuristic methodology presented, several test problems are used to compare optimal solutions via enumeration to those found via the heuristic. Some of these test problems are from the literature [15]; some are new with this research. These problem sets are listed in the Appendix.

3.1. Performance Measures

The overall measure of performance is essentially two-fold. The first measure of performance is classified as *relative inferiority*—the degree to which the efficient frontier associated with the heuristic search is above, or inferior to, the efficient frontier obtained via enumeration. The second performance measure is the number of *frontier voids*—the frequency of the efficient frontier (obtained via the heuristic) not providing a specific solution yielding a number of setups yielded by the optimal solution (obtained via enumeration). It is, of course, desired that the *relative inferiority* and the number of *frontier voids* both be minimized.

Figure 6, an extension of **Figure 5**, shows an example of the original efficient frontier, alongside a hypothetical efficient frontier obtained via the simulated annealing heuristic.

From this illustration, one will note that the efficient frontier obtained via the heuristic can only equal, never surpass, the optimal heuristic obtained via enumeration. The relative inferiority of the efficient frontier obtained via the heuristic approach is the average degree to which it is *above* the optimal frontier. The value of this performance measure is straightforward when the following notation is used:

Notation	Definition
$Usage_{Optimal}[Setups]$	Usage rate for optimal solution with *Setups*
$Usage_{SA}[Setups]$	Usage rate for heursitc solution with *Setups*
MinS	Minimum number of setups
MaxS	Maximum number of setups

The relative inferiority of the heuristic solution can then be determined via the following:

$$Rel.Inf. = \frac{1}{1+\left(\max S - \min S\right)} \sum_{i=\min S}^{\max S} \frac{Usage_{SA}\left[i\right] - Usage_{Opt}\left[i\right]}{Usage_{Opt}\left[i\right]} \qquad (10)$$

For this example, the relative inferiority of the heuristic solution is 4.90%.

To illustrate the concept of frontier voids, another variant of **Figure 5** is presented via another hypothetical heuristic solution to the original problem shown in **Figure 7**.

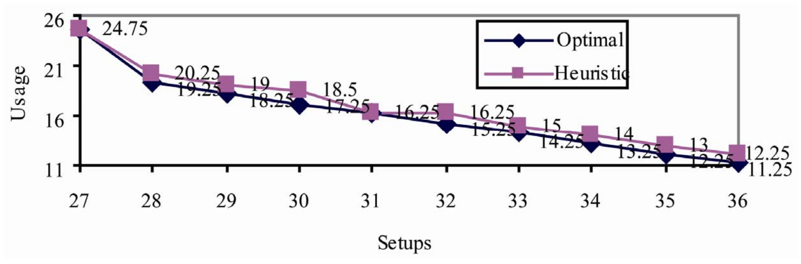

Figure 6. Efficient frontiers obtained via enumeration and search heuristic.

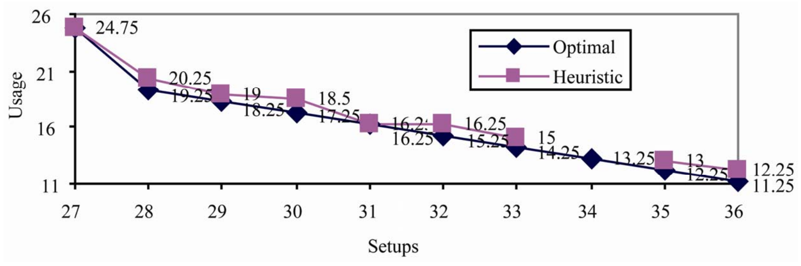

Figure 7. Example of one frontier void.

For this example, there is one frontier void—the heuristic did not find a solution associated with (34) setups. The relative inferiority associated with this solution is calculated the same way as described above, except that the frontier void statistics are omitted from the calculation, resulting in a denominator decrease equal to the number of frontier voids—in this case (34) Setups and the Usage rate of 13.25, resulting in a relative inferiority of 4.82%.

3.2. Problem Size

As stated earlier, the problem sets used are detailed in the Appendix. For each unique problem, multiple processes are explored, from as few as $p = 2$ to as large as $p = 10$. From **Figure 4**, there were only $p = 2$ processes used. The more processes used, the more rigorous the search. Across all problem sets and ranges of processes, there are (90) total problems studied. **Table 1** details the range of depth used for the various problem sets.

For each of the (90) problems studied, the optimal efficient frontier is captured via enumeration. The simulated annealing heuristic is then used to find the efficient frontier so that a comparison can be made to optimal. The parameters used for the simulated annealing search are adjusted according to problem size, so that the search is proportional to the number of nodes encountered via enumeration.

Table 1. Depth of re-sequencing problems.

Problem Set	Range of Processes (p)
A0	$p = 2, p = 10$
A1	$p = 2, p = 4$
A2	$p = 2, p = 4$
A3	$p = 2, p = 4$
A4	$p = 2, p = 6$
A5	$p = 2, p = 9$

3.3. Computational Experience

The capturing of the optimal efficient frontier via both enumeration and simulated annealing was done via programs written with the Java Development Kit, version 1.6.02 and executed on the Microsoft Vista operating system with an AMD Turion 64 × 2 processor. CPU times for the enumeration effort are reported in the results section. CPU times for the simulated annealing heuristic effort were not of consequence and are not reported. The number of nodes associated with the enumerative search is reported in the results section, as this measure discloses how much memory is required for the traversal of the search tree.

It is also noted here that the simulated annealing heuristic parameters of iterations (*Iter*), cooling rate (*CR*) and the Boltzman constant (k_B) were determined for each test problem in a manner proportional to the total number of nodes. Specifically, these three parameters were chosen so that the simulated annealing heuristic search space was 10% of the size of the total number of nodes.

4. Experimental Results

Tables 2(a)-(f) (http://www.joydivisionman.com/AJOR1/Table2.html) show the performance results of the six problem sets used for this research. *Nodes* is the number of sequences encountered in the enumerative search tree. *CPU Time* (in minutes) the length of time the CPU spent finding the optimal solution via enumeration. *Inferiority* is the degree to which the heuristic was inferior to the optimal solution. *Voids* depicts the number of times that the heuristic search failed to find a solution corresponding with the number of setups for the optimal solution.

4.1. Results for Problem Sets

Tables 2(a)-(f) are available on the internet via www.joy-divisionman.com/AJOR1/Table2.html.

4.2. Interpretation of Results

For the (90) problems studied, the simulated annealing heuristic provided a solution that was, on average, 3.74% above (or inferior to) the optimal solution obtained via enumeration. The sum of the frontier voids across all (90) problems as compared to the total number of unique setups across all problems is 9.48%.

The CPU time required to solve the problems to optimality illustrates a few forces at work. One force is the number of processes involved in the enumeration (*p*). As *p* increases, the number of nodes increases to an exponential degree. Another force at work is the number of permutations associated with each individual problem. As the number of permutations associated with the problem increases, the number of potentially feasible child sequences also increases, and these must potentially feasible sequences must be examined for feasibility. The number of items in the sequence (*n*), and the number of unique items in the sequence (*m*) also dictate computational resource needs. Of course, as *n* and *m* increase, so does the number of permutations that must be checked for feasibility.

5. Research in Perspective

This research was mainly dedicated to presenting a problem and associated solution which can be used to enhance the competitive positions of organizations concerned with multiple-objective sequencing over several processes. While the author considers an average inferiority of 3.74% to optimal reasonable, frontier voids of 9.48% is considered a performance where improvement is desired. The following sections detail further research limitations, as well as suggestions for subsequent opportunities.

5.1. Limitations of Research

There are a few limitations to this research which should be itemized. First of all, as mentioned earlier in the paper, the buffer size is one unit throughout. This means that only one item can be withheld and later reinserted into the sequence, which ultimately means that when re-sequencing for a single process, a unit can only move up one position in the sequence. A buffer size of one does limit the flexibility of the research.

Another limitation of this research is an assumption that cannot be avoided in one form or other—the initial sequence used. For purposes of consistency, the initial sequence (the level 0 sequence) used for this effort was always one where the item A's were sequenced first, followed by the item Bs, and so on. Re-sequencing was then performed. The result of this assumption was that the efficient frontiers were always well-represented on the *low-setup end*, but the *high-setup end* of the efficient frontier was frequently the source of inferiority and/or frontier voids. Nevertheless, this initial sequence assumption was made in the interest of consistency. Other assumptions could be considered for initial sequences, but this is left as a future research opportunity.

Another limitation associated with this research is associated with the problem size. Despite some of the enumerated solutions showing more than 20 million nodes, the problems used for experimentation could be considered, from a combinatorial sense, small. Of course, finding solutions via the simulated annealing heuristic has no real size limitations, but the optimal solutions obtained via enumeration provide a basis for comparison. Given this, simulated annealing was only done for problems that could be solved via enumeration as well. The constraining factor for finding optimal solutions for this research had more to do with system memory than the CPU. This caveat is not uncommon for problems requiring tree-traversal [16].

5.2. Opportunities for Future Research

Fortunately, some of the limitations associated with this effort also provide opportunities for new research efforts. One obvious opportunity is to improve the performance of the heuristic approach, especially regarding frontier voids. This could be explored via other search heuristics, such as tabu search, genetic algorithms, or natural agent systems such as ant-colony optimization.

Another opportunity for further research relates to the number of buffers used. For this research, a buffer-size of one was used throughout. This restricts re-sequences to be limited to *moving up* no more than one unit. Larger buffer sizes would permit more flexibility in re-sequencing in this regard. On the down side of this, however, more computational recourses are needed for larger buffer sizes, in terms of both CPU and memory resources.

With computing resources in a state of perpetual growth, larger buffer-sizes are not unrealistic in the future.

The size of the problems studied is also something that could be further explored as computing resources become more plentiful. At present, some of the smaller problem sets of earlier research were used [15]. Some larger problem sets from this research could be used as computing resources permit. The problem sets can be found via
www.joydivisionman.com/AJOR1/Appendix. html.

6. References

[1] J. Miltenburg, "Level Schedules for Mixed-Model Assembly Lines in Just in Time Production Systems," *Management Science*, Vol. 35, No. 2, 1989, pp. 192-207. doi:10.1287/mnsc.35.2.192

[2] P. R. McMullen, "JIT Sequencing for Mixed-Model Assembly Lines Using Tabu Search," *Production Planning and Control*, Vol. 9, No. 5, 1998, pp. 504-510. doi:10.1080/095372898233984

[3] P. R. McMullen and G. V. Frazier, "A Simulated Annealing Approach to Mixed-Model Sequencing with Multiple Objectives on a Just in Time Line," *IIE Transactions*, Vol. 32, No. 8, 2000, pp. 679-686. doi:10.1080/07408170008967426

[4] A. Joly and Y. Frein, "Heuristics for an Industrial Car Sequencing Problem Considering Paint and Assembly Shope Objectives," *Computers & Industrial Engineering*, Vol. 55, No. 2, 2008, pp. 295-310. doi:10.1016/j.cie.2007.12.014

[5] M. Masin and Y. Bukchin, "Diversity Maximization Approach for Multiobjective Optimization," *Operations Research*, Vol. 56, No. 2, 2008, pp. 411-424. doi:10.1287/opre.1070.0413

[6] M. Lahmar and S. Banjaafar, "Sequencing with Limited Flexibility," *IIE Transactions*, Vol. 39, No. 10, 2007, pp. 937-955. doi:10.1080/07408170701416665

[7] P. R. McMullen, "A Kohonen Self-Organizing Map Approach to Addressing a Multiple Objective, Mixed Model JIT Sequencing Problem," *International Journal of Production Economics*, Vol. 72, No. 1, 2001, pp. 59-71. doi:10.1016/S0925-5273(00)00091-8

[8] LINGO, "The Modeling Language and Optimizer," LINDO Systems, Inc., Chicago, 1995.

[9] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol. 220, No. 4598, 1983, pp. 671-679. doi:10.1126/science.220.4598.671

[10] R. W. Eglese, "Simulated Annealing: A Tool for Operational Research," *European Journal of Operational Research*, Vol. 46, No. 3, 1990, pp. 271-281. doi:10.1016/0377-2217(90)90001-R

[11] J. H. Holland, "Adaptation in Natural and Artificial Systems," University of Michigan Press, Ann Arbor, 1975.

[12] F. Glover, "Tabu Search: A Tutorial," *Interfaces*, Vol. 20, No. 1, 1990, pp. 74-94. doi:10.1287/inte.20.4.74

[13] M. Dorigo and L. M. Gambardella, "Ant Colonies for the Traveling Salesman Problem," *Biosystem*, Vol. 43, No. 1, 1997, pp. 73-81. doi:10.1016/S0303-2647(97)01708-5

[14] N. Metropolis, A. Rosenbluth, N. Rosenbluth, A. Teller and E. Teller, "Equation of State Calculations by Fast Computing Machines," *Journal of Chemical Physics*, Vol. 51, No. 6, 1953, pp. 177-190.

[15] P. R. McMullen, "An Ant Colony Optimization Approach to Addressing a JIT Sequencing Problem with Multiple Objective," *Artificial Intelligence in Engineering*, Vol. 15, No. 3, 2001, pp. 309-317. doi:10.1016/S0954-1810(01)00004-8

[16] R. Sedgewick, "Algorithms in Java," 3rd Edition, Addison-Wesley, New York.