ELSEVIER

Int. J. Production Economics 72 (2001) 59–71

international journal of
production
economics

www.elsevier.com/locate/dsw

A Kohonen self-organizing map approach to addressing a multiple objective, mixed-model JIT sequencing problem

Patrick R. McMullen*

Department of Management, College of Business, Auburn University, Auburn, AL 36830, USA

Received 21 March 2000; accepted 21 June 2000

Abstract

A technique is presented which addresses a JIT production-scheduling problem where two objectives are present – minimization of setups between differing products and optimization of schedule flexibility. These two objectives are inversely related to each other, and, as a result, simultaneously obtaining desirable results for both is problematic. An efficient frontier approach is employed to address this situation, where the most desirable sequences in terms of both objectives are found. Finding the efficient frontier requires addressing the combinatorial complexity of sequencing problems. The artificial neural network approach of a Kohonen self-organizing map (SOM) is used to find sequences which are desirable in terms of both the number of setups and flexibility. The Kohonen SOM was used to find sequences for several problems from the literature. Experimental results suggest that the SOM approach provides near-optimal solutions in terms of the two objectives, in addition to comparing formidably with other search heuristics. Results also show, however, that the SOM approach performs poorly with regard to CPU time.

Keywords: JIT; Sequencing; Optimization; Heuristics; Artificial neural networks

1. Introduction

One of the many benefits of successful JIT implementation is increased flexibility. Making only the required products eliminates clutter on the manufacturing floor, which subsequently provides more opportunities for the manufacturing firm to be responsive to the customer's needs – whether it be expediting high-priority orders, isolating potential quality problems, etc. In short, manufacturing flexibility is a competitive weapon. To maintain flexibility, however, the manufacturing firm must not only process the demanded items only, but they must process them in such a way that product intermixing is optimized. In other words, differing items should be sequenced such that they are produced in direct proportion to their associated level of demand – this objective is referred to as level scheduling. This fact has been well-documented in the literature, commencing with the work of Monden [1]. One assumption that is implicit in this desire for level scheduling, however, is that setup times between differing products is negligible. In

* Corresponding author. Tel.: + 334-844-6511.
E-mail address: pmcmullen@business.auburn.edu (P.R. Mc Mullen).

0925-5273/01/$ - see front matter
PII: S0925-5273(00)00091-8

reality, this is not always the case, and this fact is the impetus for this research. In short, then, this research is concerned with obtaining sequences having product intermixing while simultaneously having a tolerable amount of setups.

Obtaining production sequences having a minimum number of setups can be done via inspection and is considered trivial. Obtaining production sequences having optimal levels of product intermixing, however, is not a trivial matter, and has subsequently been fertile ground for research efforts. Miltenburg [2] extended the work of Monden and provided a metric to gauge the level of product intermixing. This metric is referred to as a usage rate and measures the stability of materials usage. Sumichrast and Russell [3], Inman and Bulfin [4], and Ding and Cheng [5] provided other heuristic techniques in an attempt to minimize this usage rate. Bard et al. [6], Xiaobo and Ohno [7], and Tamura et al. [8] further extended this work by incorporating "real-world" variants into this problem. Bolat and Yano [9,10] and Bolat [11] address sequencing problems where addressing setup and utility costs is a subject of concern. Ghosh and Gagnon [12] and Yano and Bolatt [13] provide extensive literature reviews of this relevant body of literature.

None of the above literature explicitly addresses the dual objectives of simultaneous optimization of stability of material usage and setups. Simultaneous optimization of both setups and stability of material usage rate have been addressed by McMullen [14] and McMullen et al. [15,16] via a variety of search heuristics. These search heuristics deal with the multiple objectives via a composite objective function – one component dedicated to setups and the other dedicated to usage rate. The difficulty with use of these composite objective functions is twofold: first, the decision-maker is forced to deal with the different scaling associated with each of the two objectives; and second, the decision-maker must find weighting schemes which accurately reflect their objectives once the first challenge has been met.

Use of an efficient frontier can be employed to address simultaneous optimization of both of the above objectives, without the difficulties associated with composite functions. This research constructs an efficient frontier to find sequences having desirable levels of both required setups and usage rates.

Finding these desirable sequences, however, forces the decision-maker to address the fact that problems of this type are combinatorial in nature. Combinatorial optimization problems are difficult to solve to optimality because of their immense number of feasible solutions and subsequent CPU requirements. As a result, search heuristics can be applied, which are intended to find desirable, although not necessarily optimal solutions to combinatorial optimization problems. Simulated annealing, tabu search and genetic algorithms are perhaps the most popular search heuristic to address combinatorial optimization problems. This research effort, however, will use a different approach to search for near-optimal conditions – the artificial neural network (ANN) approach of the Kohonen self-organizing map (SOM). The reasons for pursuing the ANN approach to this problem are twofold. First, it is desired to approach the subjective, or "fuzzy" aspects of such a problem and second, most applications of ANNs to combinatorial optimization problem only address the traveling salesman problem and its variants. Therefore, this problem is considered as an opportunity for a new application of ANNs.

The subsequent sections of the paper: provide a detailed description of the two objectives described above; discuss the efficient frontier approach to simultaneously addressing both objectives; detail the Kohonen SOM approach to the solution; and present an experiment to test the proposed methodology. General remarks are then offered regarding this ANN approach and its performance.

2. The mixed-model sequencing problem and its attributes

2.1. Objective functions

Before describing the formulae used to determine the objective functions associated with required setups and usage rate, the following variables are defined:

S setups required for sequence
U usage rate associated with sequence
x_{ik} number of item i through position k in sequence
s_k setup indicator – if item in position (k) ≠ item in position $(k-1)$, $s_k = 1$; otherwise $s_k = 0$
d_i demand for item i
n unique items in sequence
D_T total demand for all items

The total demand for all items in a sequence is determined as follows:

$$D_T = \sum_{i=1}^{n} d_i. \tag{1}$$

Determination of the number of setups in a sequence is straightforward:

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

Of course, lower values of S are required. The usage rate, a metric measuring the stability of the parts usage as presented by Miltenburg [2] – is as follows:

$$\sum_{k=1}^{D_T} \sum_{i=1}^{n} \left(x_{ik} - k\frac{d_i}{D_T} \right)^2. \tag{3}$$

It should be noted that lower usage rates are desired – these reflect more stability in material usage rates.

2.2. Efficient frontier

As stated earlier, because of the scaling and weighting difficulties associated with using composite functions, an efficient frontier approach is used here to find sequences with desirable levels of both required setups and usage rates. In the present context, the efficient frontier here is similar to the one used in basic economic principles – a collection of points that collectively dominate others. Here these "points" are combinations of required setups for sequences and their associated usage rates. A point (or sequence) is classified as efficient if its usage rate is minimal for the associated required number of setups. Otherwise, the point (or sequence) is classified as inefficient, or dominated by the efficient sequence for that particular required number of setups.

As an example of how this efficient frontier works, consider the example where five unique items need to be made ($n = 5$), where demand for the items is as follows: $d_1 = 4, d_2 = 3, d_3 = 2, d_4 = 2, d_5 = 1$. One possible sequence is *AAAABBBCCDDE* (where "*A*" represents item 1, "*B*" represents item 2, etc.). From Eq. (2), the number of required setups for this sequence would be 5 ($S = 5$). From Eq. (3) the usage rate associated with this sequence is 50.81 ($U = 50.81$). Another possible sequence would be *ABCDABEACDBA*. This sequence requires 12 setups ($S = 10$), and has an associated usage rate of 5.81 ($U = 5.81$).

From inspection of the example above, it should be noted that there is an apparent tradeoff between usage rate and required setups. The first sequence results in fewer setups with a higher usage rate, while the second provides a lower usage rate with the associate expense of more setups. This tradeoff is typical of problems having these two objective functions. As a result of this tradeoff, the efficient frontier approach provides the decision-maker an opportunity to find the sequences having minimal values of the usage rate for each associated required number of setups.

The efficient frontier for the example problem above is constructed by enumerating all possible sequences and finding the minimum usage rate associated with each required number of setups. A plot of the efficient frontier for the example problem is shown in Fig. 1.

From inspection of Fig. 1, it should be noted that it is not possible for sequences to be to the "southwest" of the efficient frontier – this is a collection of sequences that can only be equaled in terms of usage rate at the associated number of required setups, never surpassed. Sequences to the "northeast" of the frontier are inefficient, or dominated by the ones existing on the frontier. It should be noted that the first example sequence does not reside on the efficient frontier ($S = 5$, $U = 50.81$), while the second one does ($S = 12$, $U = 5.81$).

2.3. Combinatorial complexity

Determination of the efficient frontier above is not a trivial matter. The number of possible sequences for the example problem above is as

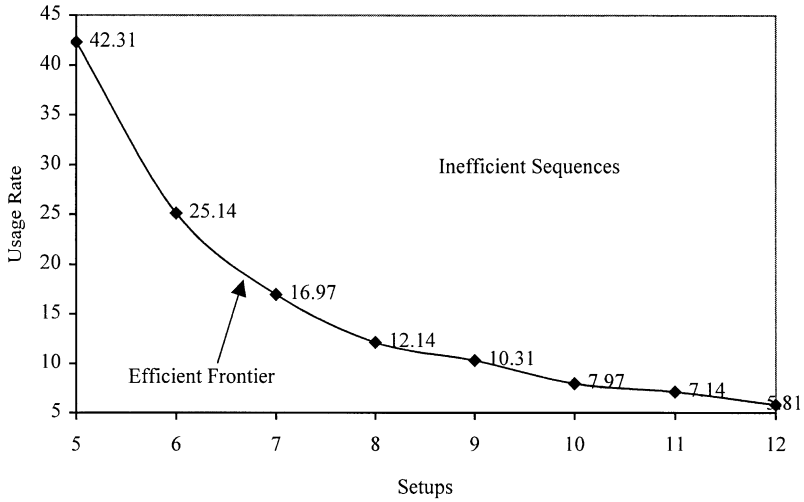

Fig. 1. Efficient frontier for example problem.

follows:

$$\text{Possible sequences} = \frac{(\sum_{i=1}^{n} d_i)!}{\prod_{i=1}^{n} (d_i!)}. \quad (4)$$

This results in 831,600 possible sequences:

$$\frac{(4+3+2+2+1)!}{(4!)(3!)(2!)(2!)(1!)} = 831{,}600.$$

If the demand for each of the five unique items were to be doubled ($d_1 = 8, d_2 = 6, d_3 = 4, d_4 = 4, d_5 = 2$), the total number of possible sequences would be 3.8552×10^{13} – an obvious demonstration of a combinatorial explosion. It is therefore impractical, if not outright impossible, to obtain optimal efficient frontiers to problems of real-world dimensions. As a result, decision-makers must strive to find desirable or near-optimal solutions to this type of problem while simultaneously keeping the required CPU resources to a reasonable amount.

Search heuristics have been demonstrated to provide near-optimal solutions to several types of combinatorial optimization problems with a reasonable computational effort. Among some of these, simulated annealing, genetic algorithms and tabu search are quite popular, and have seen much attention in relevant journals in the last several years.

3. The Kohonen self-organizing map

The Kohonen SOM [17] is a type of artificial neural network. ANNs simulate the activity of the human brain. Part of the attractiveness of using ANNs is to perform subjective analyses, as opposed to the typically more objective analyses offered by general analytical procedures. Detailed discussion of ANNs is beyond the scope of this paper, but the curious reader is referred to two informative and readable articles which explain ANNs in an understandable form [18,19].

The Kohonen SOM, like other ANNs, is used to perform tasks such as grouping, classification, forecasting and optimization. The SOM takes inputs in the form of vectors and classifies these inputs into groups – it is intended for each group to have similarity of some sort with respect to their input values. What sets the SOM approach apart from most other ANN approaches is that it is an unsupervised learning process, which means that there is no prior known classification, as opposed to the supervised learning process of feedforward, backpropagation ANNs.

An input vector is compared to all weight vectors – one weight vector for each classification group. The weight vector representing the group most "similar" to the input vector of interest is referred to as the "winner", and this particular weight vector has its weights adjusted to become even more similar to the input vector of interest. Weight vectors adjacent to the winner (these vectors are referred to as "neighbors") also have their weights adjusted to be more similar to the input vector of interest, but to a lesser degree than the actual winner. This procedure is repeated for each of the D_T input vectors.

The process described above is referred to as a single training run, or epoch, and is repeated until weights converge to a certain level. At that point, a sequence can be extracted, because each input vector will have been associated with a unique weight vector, or group. This classification process is repeated a user-specified number of times, so that a variety of different sequences can be obtained to construct an efficient frontier.

Prior to actual training of the ANN, initialization must occur. Initialization parameters include setting values for the learning rate, η, the neighbor learning rate, μ, and the adjustment factor for the learning rate, δ. All of these parameters are in the interval between 0 and 1.

3.1. Vector construction

For this particular effort, each input vector is two-dimensional, and the total number of inputs will be the total demand of all items needing sequencing, D_T. Input vector k, y_k, has the coordinates:

$$y_k = (y_{1k}, y_{2k}), \tag{5}$$

where

$$y_{1k} = 0.5 + 0.5\cos(360\,k/D_T), \quad \text{for } k = 1, \ldots, D_T, \tag{6}$$

$$y_{2k} = 0.5 + 0.5\sin(360\,k/D_T), \quad \text{for } k = 1, \ldots, D_T. \tag{7}$$

Each of these D_T vectors is constructed to be on the circle having an origin at [0.5, 0.5] and a radius of 0.5. Furthermore, each input vector has with it, an associated item as part of the desired product-mix. A geometric representation of the input vectors for the above example problem (along with associated items) is presented in Fig. 2.

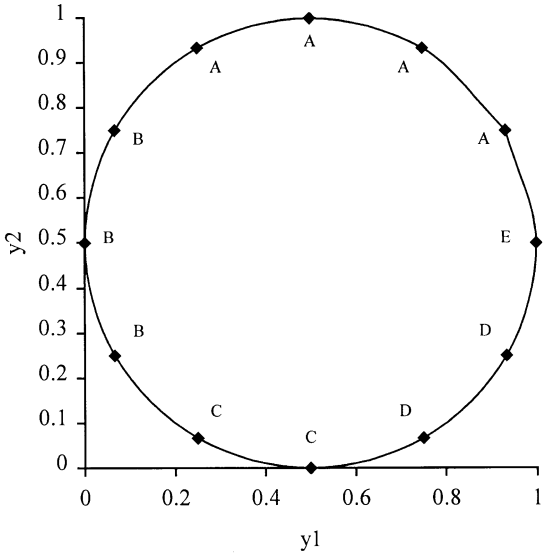

Fig. 2. Input vector construction, Part 1.

Notice how individual items at each input vector correspond with the product-mix in the example problem. The association between items in the product-mix and the input vectors favors sequences with minimal setups. This is due to the fact that the weight vectors will tend to organize themselves toward the circular patterns having minimal setups. While this does provide sequences with minimal setups, the expense of having high usage rates persists. To combat this, the input vector scheme above will be used for the first 50% of the solutions, while the second 50% of the solutions will exploit an input vector scheme which favors minimal usage – at the subsequent expense of more required setups. A geographic representation of this input vector scheme is shown in Fig. 3.

Note the product intermixing from Fig. 3, especially compared to that from Fig. 2. The intermixing as shown in Fig. 3 was performed using the heuristic of Ding and Cheng [5], which is intended to provide minimal usage rates.

As stated earlier, the input vector scheme in Fig. 2 is used to generate the first 50% of the

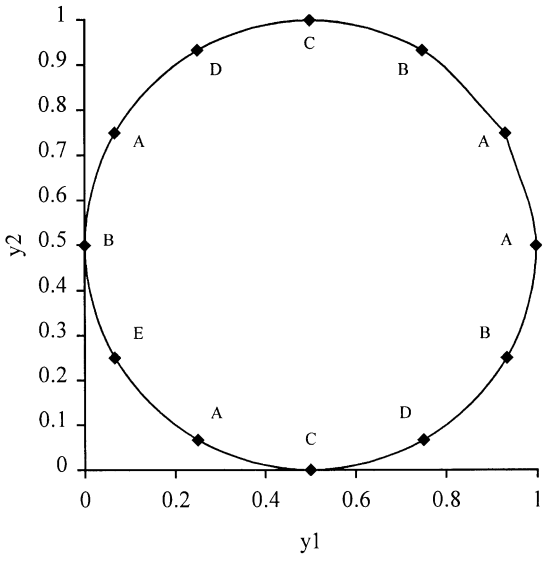

Fig. 3. Input vector construction, Part 2.

sequences, while the input vector scheme in Fig. 3 is used to generate the second 50% of the sequences. This is done so that sequences can be obtained that perform well with regard to required setups *and* usage rates.

The number of desired classification groups is also equal to D_T – each input vector will be in its own unique group. A representation of each group and their interrelationships is as follows:

Each of these groups can be thought of as a potential location for each input vector to reside in the sequence. Again, each unique vector will reside in exactly one group (Fig. 4).

A set of weights is generated. There are D_T weight vectors, where each vector is two-dimensional. Weight vector h, w_h, is constructed as follows:

$$w_h = (w_{1h}, w_{2h}) \tag{8}$$

where

$$w_{1h} = \text{random number on } [0, 1], \tag{9}$$

$$w_{2h} = \text{random number on } [0, 1]. \tag{10}$$

3.2. Training

After vector construction, the training process commences. This is a simple process where the similarity between each input vector and weight vector is determined, and the input vector is "assigned" to the weight vector where the most similarity occurs. The measure of similarity between input vector k (y_k) and weight vector h (w_h) is referred to as D_{kh}, (distance) and is determined as follows [20]:

$$D_{kh} = (y_{1k} - w_{1h})^2 + (y_{2k} - w_{2h})^2,$$

$$\text{for } h = 1, \ldots, D_T. \tag{11}$$

Group h having the minimum distance with input vector k (min D_{kh}) and not previously assigned to a group is declared the "winner." Input vector k is then assigned to group h, and weight vector h has its weights adjusted according to the following:

$$w_{\text{winner}} = w_{\text{winner}} + \eta^*(y_k - w_{\text{winner}}). \tag{12}$$

Groups adjacent to the "winning" weight vector (neighbors) also have their weights adjusted according to the following:

$$w_{\text{winner}+1} = w_{\text{winner}+1} + \mu^*\eta^*(y_k - w_{\text{winner}+1}), \tag{13}$$

$$w_{\text{winner}-1} = w_{\text{winner}-1} + \mu^*\eta^*(y_k - w_{\text{winner}-1}). \tag{14}$$

If the winning weight vector is weight vector 1, then the adjacent vectors (neighbors) are 2 and D_T. If the winning weight vector is weight vector D_T, then the adjacent vectors are 1 and D_{T-1}.

This process of computing distances, finding winning weight vectors (and their neighbors) and adjusting weights continues for each of the D_T input vectors. After all D_T input vectors have been associated with weight vectors (placed into groups), the learning rate is adjusted according to the following relationship:

$$\eta = \eta^*\delta. \tag{15}$$

This training procedure as described is referred to as an epoch, or iteration, and continues to some

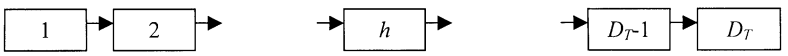

Fig. 4. Representation of groups.

user-specified point, or until the weights converge to some level.

3.3. Capture sequence

At the end of the training, a tally is taken to see which group each input vector has been associated with most often. The group that has been the "winner" for an input vector the most often becomes the position in the sequence for that input vector. For example, if input vector 12, with coordinates of $y_1 = 1$ and $y_2 = 0.5$, and associated with item E has been associated with weight vector 5 most often, then this input vector will be placed into the fifth position in the sequence – therefore, item E will be in the fifth position of the sequence.

3.4. Construction of efficient frontier

Each time the above steps of vector construction, training and capturing of the sequence occur, a feasible solution is obtained. This sequence then has its objective function values for number of setups and usage rate determined. If the usage rate at the associated number of setups is the minimum found for that particular number of setups, then this usage rate replaces the minimum usage rate at that particular number of setups. Therefore, this sequence resides on the efficient frontier.

The steps of vector construction, training and sequence capturing are performed a user-specified number of times (*Solutions*). The first 50% of these *Solutions* will construct input vectors in accordance with minimal setups (Fig. 2), while the last 50% of these *Solutions* will construct input vectors in accordance with minimal usage rates (Fig. 3). Both strategies are employed here in an attempt to find desirable, or near optimal, efficient frontiers.

3.5. Example of self-organization

Fig. 5 provides an example of how weight vectors organize themselves to be associated with input vectors. The Kohonen SOM approach was used

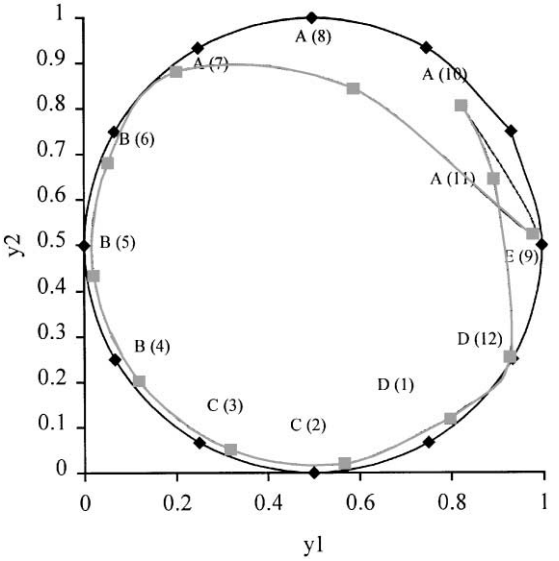

Fig. 5. Self-organization of weight vectors.

for the above example problem where 50 epochs were chosen for weight convergence to generate one feasible sequence, along with the following parameter values: $\eta = 0.9$, $\mu = 0.3$, and $\delta = 0.95$.

The input vectors were generated according to the strategy described in Fig. 2 – a strategy favoring minimal setups. The items requiring sequencing are present as they were in Fig. 2, but here, converged weight vectors are also shown. The weight vectors (in parentheses) show the sequence position of each of the individual items. This solution resulted in the following sequence: *DCCBBBAAEAAD*, with seven setups and a usage rate of 33.64. Each time vector construction occurs, weight vectors take on different initial values (randomly assigned), subsequently resulting in possibly different sequences.

3.6. Pseudocode for Kohonen SOM approach

The steps of the approach described above are presented in pseudocode. The loops involving a and e are indexed with "Next a" and "Next e," while more minor loops are bracketed with "{" and "}". Functions (or subroutines) are used to condense sections where full disclosure of code

is considered non-vital. The pseudocode is as follows:

Initialize η, δ, μ
For $a = 1$ to S*olutions*
For $e = 1$ to *Epochs*
/*Construct Input Vectors*/
For $k = 1$ to D_T $\{y_{1k} = 0.5 + 0.5*\cos((k/12)*360)$, $y_{2k} = 0.5 + 0.5*\sin((k/12)*360)\}$
/*Construct Weight Vectors*/
For $h = 1$ to D_T $\{w_{1h} = \text{rand}, w_{2h} = \text{rand}\}$
/*Train ANN*/
For $h = 1$ to D_T $\{D_{kh} = (y_{1k} - w_{1h})^2 + (y_{2k} - w_{2h})^2\}$
Find_min_D_{kh}( ) /*associated h is winner*/
$w_{\text{winner}+1} = w_{\text{winner}+1} + \mu*\eta*(y_k - w_{\text{winner}+1})$
$w_{\text{winner}-1} = w_{\text{winner}-1} + (\mu*\eta*y_k - w_{\text{winner}-1})$
/*Necessary adjustments must be made when 1 and D_T are winners*/
Next e
$\eta = \eta*\delta$
Capture_Sequence_and_Objective_Function_Values( )
If ($\text{Usage}_{\text{Setups}} < \text{Min\_Usage}_{\text{Setups}}$)
$\text{Min\_Usage}_{\text{Setups}} = \text{Usage}_{\text{Setups}}$
Sequence_Placed_on_Frontier( )
Endif
Next a
Present_Frontier( )

4. Experimentation

4.1. Experimental design

Experimentation is used to evaluate the performance of the Kohonen SOM approach to solving this sequencing problem. The presented methodology is used on three sets of problems from the literature [3,15]. These problems appear in the Appendix. Three performance measures are used here to gauge performance of the Kohonen SOM approach: percent inferior to optimal in terms of usage rate for each setup level; percentile performance of SOM approach compared to optimal; and ratio of CPU time requirement for optimal solution to CPU time requirement for heuristic solution. Low values of the percent inferiority performance measure are desired, while percentile performance values are desired to be as close to unity as possible, while the CPU ratio is desired to be as high as possible. Fig. 6 illustrates how the example problem developed throughout this paper has its performance measured against optimal.

From Fig. 6, it is clear that the Kohonen SOM frontier never surpasses the optimal frontier – it can only "equal" the optimal frontier. Table 1 details the usage rates for each level of setups for both the

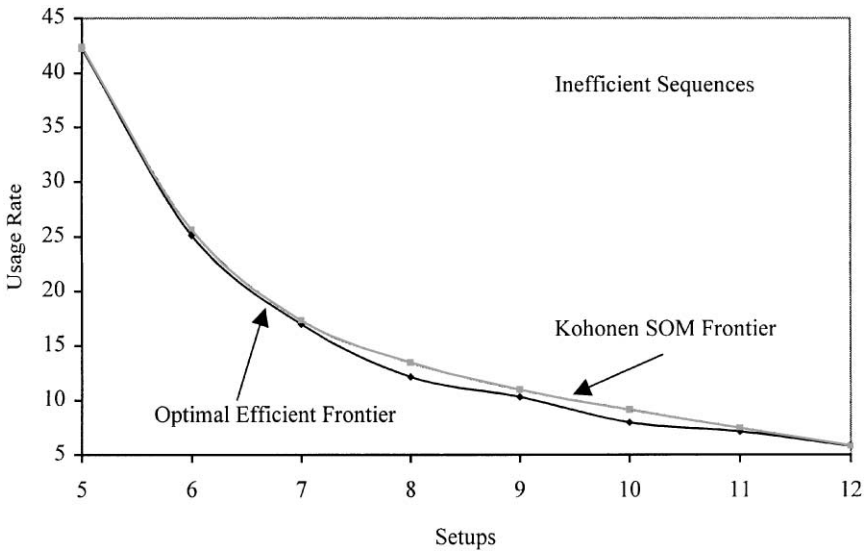

Fig. 6. Comparison of optimal to kohonen SOM frontiers.

Table 1
Comparison of optimal to Kohonen SOM frontiers for example problem

Setups	Optimal usage	Kohonen SOM usage	% inferior	No. of inferiors
5	42.3056	42.3056	0.00	0
6	25.1389	25.6389	1.99	1
7	16.9722	17.3056	1.96	1
8	12.1389	13.4722	10.98	6
9	10.3056	10.9722	6.47	3
10	7.9722	9.1389	14.63	5
11	7.1389	7.4722	4.67	4
12	5.8056	5.8056	0.00	0

optimal and Kohonen SOM approaches, along with comparison data.

The first three columns are self-explanatory. The "% inferior" column represents the percentage that the Kohonen SOM approach usage rate is in excess of the optimal solution usage rate at the required setups of interest. The average inferiority for this example is 5.09%. The "No of inferiors" column is a listing of the number of times a solution was found to have a lower usage rate than the Kohonen SOM approach for the required setups of interest when enumeration was performed to obtain the optimal solution. There were a total of 20 solutions found via the Kohonen SOM approach which were inferior to the optimal solutions obtained via complete enumeration. Given that there were 831,600 total solutions found via enumeration, the 20 inferior solutions found via the Kohonen SOM approach places the Kohonen SOM approach in the 99.997595th percentile. The CPU requirement to obtain the optimal solution via complete enumeration was 0.35 minutes, while the CPU requirement to obtain the Kohonen SOM solution was 0.60 minutes, for a CPU ratio of 0.5833. It should be noted that for this problem, the following parameter values were used: Solutions = 8,880; $\eta = 0.9$; $\mu = 0.3$; and $\delta = 0.95$.

These performance-related results are compared with results from the search heuristic approaches of simulated annealing, genetic algorithms and tabu search for the problems detailed in the appendix.

4.2. Parameter values

For all problems solved via the Kohonen SOM approach, the initial learning rate, η, was set to 0.9, the neighbor learning rate, μ, was set to 0.3, and the learning rate adjustment factor, δ, was set to 0.95. The number of solutions processed for the problems were set to values proportional to the number of feasible solutions the problem has. Problem Set 1 used 25 epochs per solution for training, while Problem Sets 2 and 3 used 50 and 75, respectively. It should be noted that regardless of the search heuristic used for comparison, the same number of solutions for each heuristic was generated. This was done so that fair comparisons could be made regarding heuristic performance, and was applied to all problem sets.

4.3. Computational experience

The Kohonen SOM approach and the three search heuristics used for this research (simulated annealing, genetic algorithm and tabu search) were all coded using C/C++ and executed on dual 500 MHz Intel Pentium III processors. The same environment was used obtain optimal solutions via complete enumeration. In all instances, every effort was made to ensure that the algorithms would run as efficiently as possible.

5. Experimental results and discussion

Table 2 shows means of the three performance measures of interest: average inferiority %, average percentile performance and CPU ratio organized by the type of search procedure used: simulated annealing, tabu search, genetic algorithm and the kohonen SOM.

The Kohonen SOM approach provides solutions which are generally competitive with the other search heuristics in terms of average inferiority% and percentile performance. The average inferiority of the Kohonen SOM approach is the least desirable – its Tukeys pairwise difference with the genetic algorithm approach is not significant at the $\alpha = 5\%$ level, while its pairwise differences with the simulated annealing and tabu search

Table 2
Performance Measures of Solution Approaches

	Problem Set 1	Problem Set 2	Problem Set 3	Overall Mean
Simulated annealing				
Avg. inferiority	0.6394%	0.6467%	1.2233%	0.852
Percentile	99.9898%	99.9973%	99.9995%	99.9960
CPU ratio	2.3097	7.8381	391.9962	144.6
Tabu search				
Avg. inferiority	0.7813%	0.7856%	2.2344%	1.306
Percentile	99.9917%	99.9968%	99.9986%	99.9960
CPU ratio	2.3097	7.8381	340.2037	125.9
Genetic algorithm				
Avg. inferiority	4.4811%	1.1333%	4.1011%	3.139
Percentile	99.9778%	99.9946%	99.9919%	99.9890
CPU ratio	2.3097	7.8381	340.2037	125.9
Kohonen SOM				
Avg. inferiority	0.7271%	1.7978%	10.1000%	4.49
Percentile	99.9976%	99.9980%	99.9987%	99.9980
CPU ratio	0.4615	0.3196	11.8943	4.53

approaches are significant at the $\alpha = 5\%$ level. The inferiority% associated with the Kohonen SOM approach is formidable for two smaller problem sets, but its poor performance for the third problem set is due in part to a very poor performance for Problem G (20.63%) – the reason for this is unclear.

The percentile performance of the Kohonen SOM approach is superior to that of the other three heuristics, but the only significant pairwise difference at the $\alpha = 5\%$ level the Kohonen SOM approach has is with the genetic algorithm approach.

The CPU ratio associated with the Kohonen SOM approach is clearly inferior to that of the other heuristics – this fact will be addressed in the concluding section.

5.1. Isolation of inferiority in Kohonen SOM

Table 3 shows a breakdown by problem set of the number of inferior solutions found via the Kohonen SOM approach as compared to optimal.

Problem Set 1 has a maximum of 10 possible setups, while Problem Sets 2 and 3 have 12 and 15, respectively. Considering these varying number of

Table 3
Breakdown of inferior solutions by problem set

Setups	Problem Set 1	Problem Set 2	Problem Set 3
5			1
6		1	8
7	2	5	18
8	2	16	29
9	5	16	47
10		11	66
11		5	94
12			73
13			19
14			19
15			4

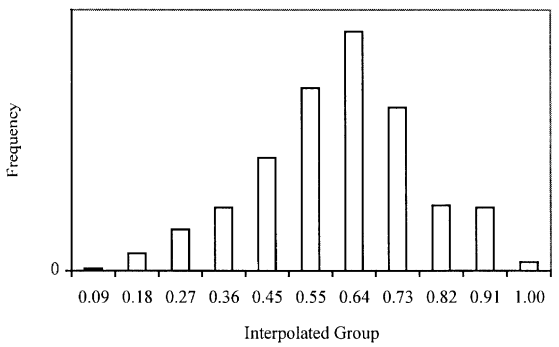

Fig. 7. Breakdown of inferiority along efficient frontier.

possible setups, the data above is "mapped" onto a new scale where the minimum setups value is zero, the maximum setups value is unity, and values in between are interpolated according to their position between the minimum and maximum number of setups. With this new scale, the total number of inferior solutions is summed for each interpolated group on the new scale. A histogram of this is presented in Fig. 7.

Both Table 3 and Fig. 7 suggest that for the Kohonen SOM approach, most of the inferiority comes from the middle portions of the efficient frontier. That is, the sequences comprising the frontier whose associated number of setups are greater than the minimum, but less than the maximum. This should not come as a surprise because the input vector construction for the first 50% of

solutions favors minimal setups, while the second 50% of solutions favors minimal usage – there is no overt consideration for setups or usage in this intermediate group. Addressing this problem is left as an opportunity for future research.

6. Concluding comments

A technique has been presented to address the JIT mixed-model sequencing problem with non-negligible setup times using the artificial neural network approach of the Kohonen self-organizing map. Several test problems from the literature were solved using this approach, and the results were compared to solutions from the popular search heuristics of simulated annealing, genetic algorithms, and tabu search. In terms of performance as compared to optimal and other search heuristics, the Kohonen SOM approach performed formidably – its inferiority % was least desirable, but not by a large degree, and its percentile performance was the most desirable, also not by a large degree. Its overall performance as compared to optimal is considered competitive with the other search heuristics.

The biggest setback of this Kohonen SOM approach is its CPU intensity. It performs much worse than the other search heuristics in terms of the CPU ratio. The Kohonen SOM approach only becomes attractive (when compared to complete enumeration to obtain optimal solutions) for larger problem sets – the CPU ratio for Problem Set 3 is 11.8943 (it is less than unity for the smaller problem sets). Given the combinatorial explosion of these types of problems, the Kohonen SOM approach would continue to gain in attractiveness (as compared to optimal) as the problem size grows – this point should be emphasized, given that "real-world" sequencing problems are frequently quite large.

The fact that this approach does not perform competitively in terms of CPU resources is not unique to artificial neural network approaches. The reason for this is because of the many replications needed to generate feasible solutions. Roughly speaking, an ANN approach will be inferior to other approaches by a factor of the number of epochs, or iterations, as chosen by the decision-maker. The large number of epochs is necessary to ensure that weight convergence is obtained, and subsequently, feasibility. An obvious opportunity for further research would be to enhance the computational efficiency of this approach by using hybrid techniques which could combine the advantages of the ANN approach with that of one or more of the other less CPU-intensive approaches.

The large CPU requirement for this Kohonen SOM approach should not detract from the attractive features of the approach. The primary attractive feature here is that this approach presents a new optimization-related application of ANNs – most previous such optimization-related applications of ANNs only address the traveling salesman problem and its extensions [21]. It should again be emphasized here that finding a more CPU-efficient way to perform the desired sequencing should be considered as an opportunity for future research.

Table 4
Problem Set 1 (number of each product type in product mix – total demand is 10)

Problem	Product 1	Product 2	Product 3	Product 4	Product 5	Solutions
B	6	1	1	1	1	5,040
C	5	2	1	1	1	15,120
D	4	2	2	1	1	37,800
E	4	3	1	1	1	25,200
F	3	3	2	1	1	50,400
G	3	2	2	2	1	75,600
H	2	2	2	2	2	113,400

Table 5
Problem Set 2 (number of each product type in product mix – total demand is 12)

Problem	Product 1	Product 2	Product 3	Product 4	Product 5	Solutions
B	8	1	1	1	1	11,880
C	7	2	1	1	1	47,520
D	6	3	1	1	1	110,880
E	6	2	2	1	1	166,320
F	5	3	2	1	1	332,640
G	5	2	2	2	1	498,960
H	4	3	2	2	1	831,600
I	4	4	2	1	1	415,800
J	3	3	2	2	2	1663,200

Table 6
Problem Set 3 (number of each product type in product mix – total demand is 15)

Problem	Product 1	Product 2	Product 3	Product 4	Product 5	Solutions
B	11	1	1	1	1	32,760
C	10	2	1	1	1	180,180
D	9	3	1	1	1	600,600
E	7	5	1	1	1	2162,160
F	7	3	2	2	1	10,810,800
G	6	3	3	2	1	25,225,200
H	5	3	3	3	1	50,450,400
I	4	3	3	3	2	126,126,000
J	3	3	3	3	3	168,168,000

Appendix A. Details of problem sets

The performance-related results are compared with results from the search heristic approaches of simulated annealing genetic algorithims and tabu search and one given in Tables 4–6.

References

[1] Y. Monden, Toyota Production System, The Institute of Industrial Engineers, Norcross, GA, 1993.
[2] J. Miltenburg, Level schedules for mixed-model assembly lines in just-in-time production systems, Management Science 35 (1989) 192–207.
[3] R.T. Sumichrast, R.S. Russell, Evaluating mixed-model assembly line sequencing heuristics for just-in-time production systems, Journal of Operations Management 9 (1990) 371–390.
[4] R. Inman, R. Bulfin, Sequencing JIT mixed-model assembly lines, Management Science 37 (1991) 901–904.
[5] F. Ding, L. Cheng, An effective mixed-model assembly line sequencing heuristic for just-in-time production systems, Journal of Operations Management 11 (1993) 45–50.
[6] J. Bard, E. Dar-El, A. Shtub, An analytic framework for sequencing mixed-model assembly lines, International Journal of Production Research 30 (1992) 35–48.
[7] K. Xiaobo, A. Ohno, A sequencing problem for a mixed-model assembly line in a JIT production system, Computers & Industrial Engineering 27 (1994) 71–74.
[8] T. Tamura, H. Long, K. Ohno, A sequencing problem to level part usage rates and work loads for a mixed-model assembly line with a bypass subline, International Journal of Production Economics 61 (1999) 557–564.
[9] A. Bolat, C.A. Yano, Scheduling algorithms to minimize utility work at a single station on a paced assembly line, Production Planning and Control 3 (1988) 393–405.
[10] A. Bolat, C.A. Yano, A surrogate objective for utility work in paced assembly lines, Production Planning and Control 3 (1998) 406–412.

[11] A. Bolat, Sequencing jobs on an automobile assembly line: Objectives and procedures, International Journal of Production Research 32 (1994) 1219–1236.
[12] S. Ghosh, R. Gagnon, A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems, International Journal of Production Research 27 (1989) 637–670.
[13] C.A. Yano, A. Bolatt, Survey, development and applications of algorithms for sequencing paced assembly lines, Journal of Manufacturing and Operations Management 2 (1989) 172–198.
[14] P.R. McMullen, JIT sequencing for mixed-model assembly lines with setups using tabu search, Production Planning & Control 5 (1998) 504–510.
[15] P.R. McMullen, G.V. Frazier, A simulated annealing approach to mixed-model sequencing with multiple objectives on a JIT line, IIE Transactions 32 (2000) 679–686.
[16] P.R. McMullen, P. Tarasewich, G.V. Frazier, Using genetic algorithms to solve the multi-product JIT sequencing problem with setups, International Journal of Production Research 38 (2000) 2653–2670.
[17] T. Kohonen, The self-organizing map, Proceedings of the IEEE 78 (1990) 1464–1480.
[18] A.K. Jain, J. Mao, K.M. Mohiuddin, Artificial neural networks: A tutorial, Computer 30 (1996) 31–44.
[19] K. Knight, Connectionist ideas and algorithms, Communications of the ACM 33 (1990) 59–74.
[20] K. Mehrotra, C.K. Mohan, S. Ranka, Elements of artificial neural networks, MIT Press, Cambridge, MA, 1997.
[21] K.A. Smith, Neural networks for combinatorial optimization: A review of more than a decade of research, INFORMS Journal on Computing 11 (1999) 15–34.