ELSEVIER

Artificial Intelligence in Engineering 15 (2001) 309–317

ARTIFICIAL
INTELLIGENCE
in
ENGINEERING

www.elsevier.com/locate/aieng

An ant colony optimization approach to addressing a JIT sequencing problem with multiple objectives

Patrick R. McMullen*

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

Received 30 November 2000; revised 25 January 2001; accepted 28 March 2001

Abstract

This research presents an application of the relatively new approach of ant colony optimization (ACO) to address a production-sequencing problem when two objectives are present — simulating the artificial intelligence agents of virtual ants to obtain desirable solutions to a manufacturing logistics problem. The two objectives are minimization of setups and optimization of stability of material usage rates. This type of problem is NP-hard, and therefore, attainment of IP/LP solutions, or solutions via complete enumeration is not a practical option. Because of such challenges, an approach is used here to obtain desirable solutions to this problem with a minimal computational effort. The solutions obtained via the ACO approach are compared against solutions obtained via other search heuristics, such as simulated annealing, tabu search, genetic algorithms and neural network approaches. Experimental results show that the ACO approach is competitive with these other approaches in terms of performance and CPU requirements. © 2001 Elsevier Science Ltd. All rights reserved.

Keywords: Ant colony optimization; Simulated annealing; Tabu search; Genetic algorithms; Artificial neural networks; Heuristics; Optimization

1. Introduction

Successful implementation of just-in-time (JIT) production systems is very important to modern manufacturing firms. If firms can consistently maintain minimal inventories, they can maximize their flexibility and better manage product quality, and subsequently, enhance their competitive positions. Successful JIT implementation, however, requires discipline on the part of management, and prerequisites for this JIT success have been well documented — demand management and scheduling are two of the more common prerequisites. This research is concerned with the scheduling of JIT production. Of specific interest here is the sequencing of items to be made when there are two scheduling objectives of interest: minimization of setups between differing products and optimization of the usage rates of raw materials. It is also noted here that these objectives of finding minimal setups and optimal levels of usage rates are inversely related to each other. As a result, finding sequences having desirable levels of both objectives is difficult. This fact is demonstrated later.

Finding production sequences with desirable levels of both number of setups and material usage rates forces one to address the combinatorial aspects of the problem — these problems are NP-hard. As the size of the problem increases, the number of feasible solutions increase in an exponential fashion, making attainment of optimal solutions impractical.

There are then, two complicating features to the problem on hand: two objectives which are inversely related to each other and the problem is NP-hard. As a result, effort is required to find solutions are found providing desirable combinations of both required setups and usage rates in a reasonable amount of CPU time. Search heuristics can be exploited to address such a problem. Ant colony optimization (ACO) is used here to address this problem. ACO is an artificial intelligence procedure that simulates the behavior of social insects in an attempt to perform some complicated task — this issue is discussed in detail.

This paper presents an ACO approach to address the multiple-objective JIT sequencing problem as described above. The relevance and general contribution of this effort is that the artificial intelligence agents of virtual ants are used to address an important issue in manufacturing. The experimental results presented in a later section also suggest that the performance of the methodology is, in general, desirable.

---

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

PII: S0954-1810(01)00004-8

2. The mixed-model sequencing problem and its attributes

2.1. Objective functions

Prior to discussion of the models associated with this multiple objective problem, the following variables are defined: S is the number of setups required for sequence, U the usage rate associated with sequence, x_{ik} the number of item of type i through position k in sequence, s_k the setup indicator — if item in position $k \neq$ item in position $k-1$, $s_k = 1$; otherwise $s_k = 0$, d_i the demand for item of type i, n the number of types of items in sequence and D_T is the total demand for all items

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

$$D_\mathrm{T} = \sum_{i=1}^{n} d_i \tag{1}$$

Determination of the number of setups in a sequence is as follows:

$$S = 1 + \sum_{k=1}^{D_\mathrm{T}} s_k \tag{2}$$

Of course, lower values of S are desired. The usage rate, a metric measuring the stability of the parts usage as presented by Miltenburg [1], is as follows:

$$U = \sum_{k=1}^{D_\mathrm{T}} \sum_{i=1}^{n} \left(x_{ik} - k\frac{d_i}{D_\mathrm{T}} \right)^2 \tag{3}$$

The usage rate of Miltenburg in Eq. (3) is a formalized metric embellishing the earlier argument made by Monden [2] regarding the stability of material usage rates. Lower usage rates are desired — they are indicative of more stability in material usage (or production 'smoothing').

2.2. Efficient frontier

As there are two objectives of interest here (setups and usage rates), which happen to be measured using differing metrics, a single, ubiquitous measure of overall performance is not available. One possible remedy for this would be to use weights for each of the two factors and obtain a composite measure, but this approach is not taken due to the associated difficulties with weighting and scaling issues. Instead, an efficient frontier approach [3] is used 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 a simple example of how this efficient frontier works, consider the example where three unique items need to be made ($n = 3$), where demand for the items is as follows: $d_1 = 6$, $d_2 = 4$ and $d_3 = 3$. One possible sequence is: BBBBAAAAAACCC (where 'A' represents item 1, 'B' represents item 2 and 'C' represents item 'C'). From Eq. (2), the number of required setups for this sequence would be 3 ($S = 3$). From Eq. (3) the usage rate associated with this sequence is 69.7 ($U = 69.7$). Another possible sequence would be: ABCABACABACBA. This sequence [4] requires 13 setups ($S = 13$), and has an associated usage rate of 3.4 ($U = 3.4$).

From inspection of this example, it is obvious that there is an apparent trade-off between usage rate and required setups [5]. 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 trade-off is typical of problems having these two objective functions. As a result of this trade-off, 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 as follows.

From inspection of Fig. 1, it is appropriate to note that it is not possible for sequences to exist 'southwest' of the efficient frontier — this is a collection of sequences that can only be equal 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 is also noted that the example problem above is a very simple one, and is intended to serve as an illustration for a real-world, industrial-scale problem (an industrial-scale problem would likely have more than 13 members requiring sequencing). While the industrial-scale problem is much larger than this simple example problem, the example

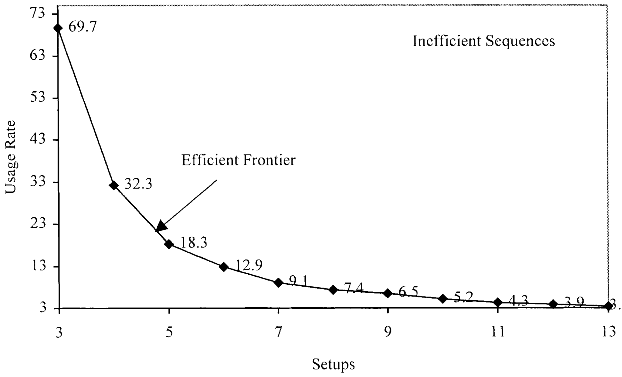

Fig. 1. Efficient frontier for example problem — numbers are usage rates for each associated number of setups.

problem does illustrate the trade-off between the number of setups and usage rate, which is also present in the larger, industrial-scale problems.

2.3. Combinatorial complexity

Capturing the efficient frontier above is not a trivial matter. The number of possible sequences for this type of problem is as follows:

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

For the example above, there are 60,060 possible sequences:

$$\frac{(6+4+3)!}{(6!)(4!)(3!)} = 60,060$$

If the demand for each of the five unique items were to be increased by one item each ($d_1 = 7$, $d_2 = 5$, $d_3 = 4$), the total number of possible sequences would be 1,441,440 — a demonstration of a combinatorial explosion. It is therefore difficult 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 provide such an opportunity. Techniques like simulated annealing, tabu search, genetic algorithms, and, most recently, ACO, have been demonstrated to provide near-optimal solutions to NP-hard problems with a relatively little computational effort.

3. Ant-colony optimization

ACO is a branch of a larger field referred to as Swarm Intelligence. Swarm Intelligence is the behavioral simulation of social insects such as bees, ants, wasps and termites. This behavioral simulation came about for many reasons — optimization of systems and learning about self-organization are two of many reasons why scientists are interested in simulating these insects [6,7]. More specifically, ACO simulates the collective foraging habits of ants — ants venturing out for food, and bringing their discovered food back to the nest. Ants have poor vision and poor communication skills, and a single ant faces a poor probability of longevity. However, a large group, or swarm, of ants can collectively perform complex tasks with proven effectiveness, such as gathering food, sorting corpses or performing division of labor.

The key to such group effectiveness is pheromone — a chemical substance deposited by ants as they travel. Pheromone provides ants with the ability to communicate with each other. Ants essentially move randomly, but when they encounter a pheromone trail, they decide whether or not to follow it. If they do so, they deposit their own pheromone on the trail, which reinforces the path. The probability that an ant chooses one path over another is governed by the amount of pheromone on the potential path of interest. Because of the pheromone, trails that are more frequently traveled by ants become more attractive alternative for other ants. Subsequently, less traveled paths become less likely paths for other ants.

With time, the amount of pheromone on a path evaporates. Prior to the establishment of the most desirable pheromone trails, individual ants will use all potential paths in equal numbers, depositing pheromone as they travel. But the ants taking the shorter path will return to the nest first with food. The shorter pathway will have the most pheromone because the path has ‘fresh’ pheromone and has not yet evaporated, and will be more attractive to those ants that return to the food source. The dynamics of a pheromone trail is illustrated by Fig. 2, which shows a distribution of ants over a set of pathways between a nest and a food source over time. Early on, the ants are equally distributed, but eventually they favor the shorter route.

There is, however, always a probability that an ant will not follow a well-marked pheromone trail. This probability (although perhaps small) allows for exploration of other trails, which is beneficial in that it allows for discovery of shorter or alternate pathways, or new sources of food. Given that the pheromone trail evaporates over time, the trail will become less detectable on longer trails, since these trails take more time to traverse. The longer trails will hence be less attractive, another benefit to the colony as a whole.

The above description of ants foraging for food has a strong association with the traveling salesman problem (TSP) — finding the most efficient route through a network [8]. This association is exploited to find production sequences concerned with optimization of the objective functions described above.

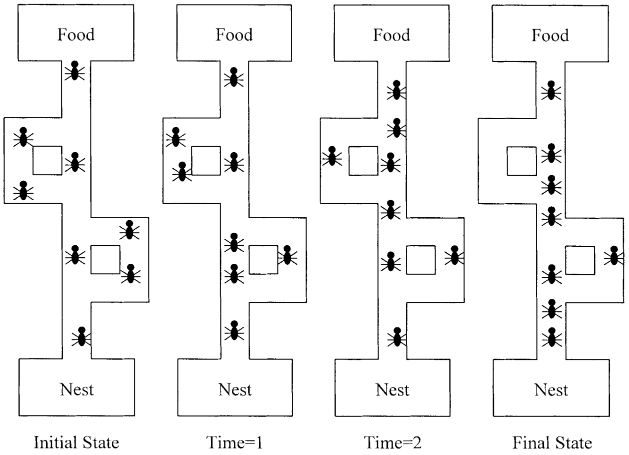

Fig. 2. Dynamics of pheromone trail — ants converge to the most ‘popular’ trails.

Table 1
Spatial strategies for sequencing problems (Note: i is an index commencing at 1 and ending at D_T)

Strategy	X Coord.	Y Coord.	Z Coord.	Objective
ACO-1	$0.5+0.5\cos(360(i/D_T))$	$0.5+0.5\sin(360(i/D_T))$	N/A	Minimal setups
ACO-2	$0.5+0.5\cos(360(i/D_T))$	$0.5+0.5\sin(360(i/D_T))$	Rand()	Setups/both
ACO-3	Rand()	Rand()	N/A	?
ACO-4	$0.5+0.5\cos(360(i/D_T))$	$0.5+0.5\sin(360(i/D_T))$	N/A	Minimal usage
ACO-5	$0.5+0.5\cos(360(i/D_T))$	$0.5+0.5\sin(360(i/D_T))$	N/A	Both
ACO-6	$0.5+0.5\cos(360(i/D_T))$	$0.5+0.5\sin(360(i/D_T))$	Rand()	Usage/both

4. ACO heuristic for JIT sequencing with multiple objectives

4.1. *'Spatialization' of problem*

The problem data is transformed into spatial data so that a TSP approach can be used to find production sequences offering desirable levels of both setups and usage rates. Each node in space is associated with an item to be placed in sequence. The route through the spatial system becomes the production sequence.

Specifically, six different spatial approaches are investigated, each offering a unique strategy in an attempt to find desirable production sequences. Each spatial strategy is either two-dimensional or three-dimensional and have minima at zero and maxima at one. Some of the strategies attempt to obtain sequences with minimal setup strategies, while others attempt to obtain sequences with minimal usage rates, while others attempt to excel at both. Table 1 details each of these strategies.

The ACO-1 strategy attempts to obtain sequences with few setups. ACO-2 does the same thing, with the addition of a third, randomly generated component intended to motivate discovery of sequences also favoring minimal usage rates. ACO-3 uses two randomly generated components explicitly preferring neither setups nor usage, but with ambitions of finding sequences excelling at both. ACO-4 is like ACO-1, except here minimal usage rates are explicitly preferred instead of minimal setups. ACO-5 is a strategy directed at obtaining desirable levels of both setups and usage rates. Here, minimal usage rates are preferred for the first 50% of the sequence, while minimal setups are preferred for the second 50% of the sequence. ACO-6 is similar ACO-2, but here, minimal usage rates are preferred along with the randomly generated third component. While Table 1 may provide ample information to assist in differentiating the strategies, Figs. 3–8 below are provided to show spatial details of these six strategies with regard to the example problem described above — note how the graph of each strategy reflects its intent.

4.2. *ACO heuristic*

As stated above, each 'item' in the sequence is mapped as a node in space. From this spatial mapping, a distance matrix is computed to obtain pairwise distances. The

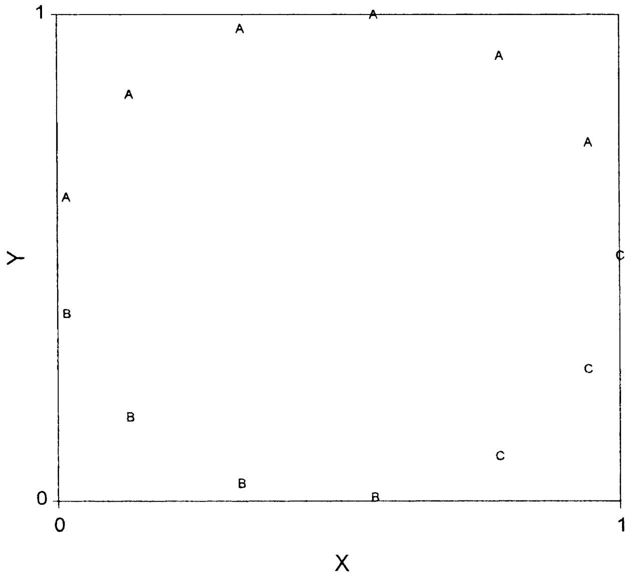

Fig. 3. ACO-1 strategy for example problem.

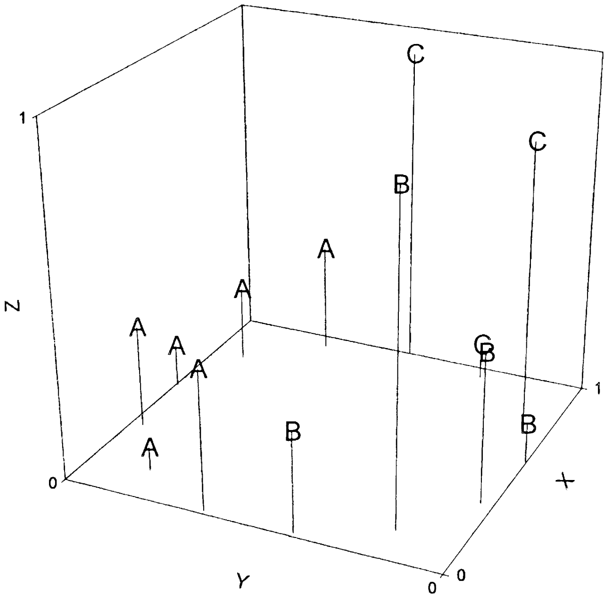

Fig. 4. ACO-2 strategy for example problem.

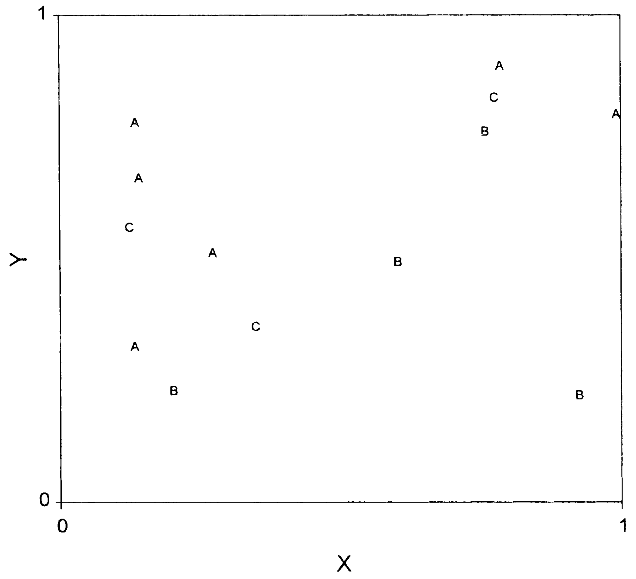

Fig. 5. ACO-3 strategy for example problem.

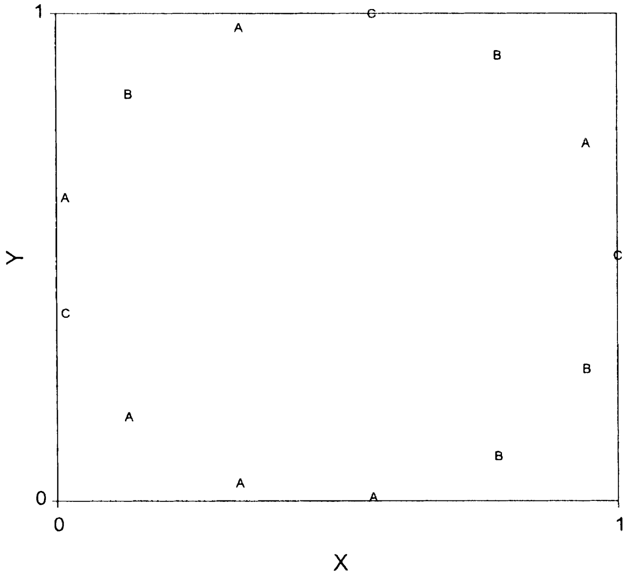

Fig. 7. ACO-5 strategy for example problem.

distance between nodes r and u is represented by $\delta(r,u)$ in the spatial system is as follows:

$$\delta(r,u) = \sqrt{\sum_{i=1}^{h} (y_{r,i} - y_{u,i})^2} \tag{5}$$

where $y_{r,i}$ and $y_{u,i}$ are the coordinates for nodes r and u, respectively, and where h is the number of dimensions in the spatial system — h will be either two or three for this research.

The following additional variables are also defined: $\tau(r,u)$ is the amount of pheromone between nodes r and u, M_k the set of all nodes assigned to the sequence, β the parameter weighing the relative importance of closeness between nodes, α the updating parameter, *ants* the number of ants employed in the search, and $p_k(r,s)$ is the probability that *ant* moves from nodes r to s.

Initialization of all parameters occurs. Values of β, α and *ants* are chosen, and pheromone values for all links ($\tau_0(r,u)$) are initialized as the reciprocal of the product of the number of nodes in the system (D_T) and the length of an initial tour (L_0) as determined by the nearest neighbor heuristic. Mathematically, this is as follows:

$$\tau_0(r,u) = (D_T L_0)^{-1} \tag{6}$$

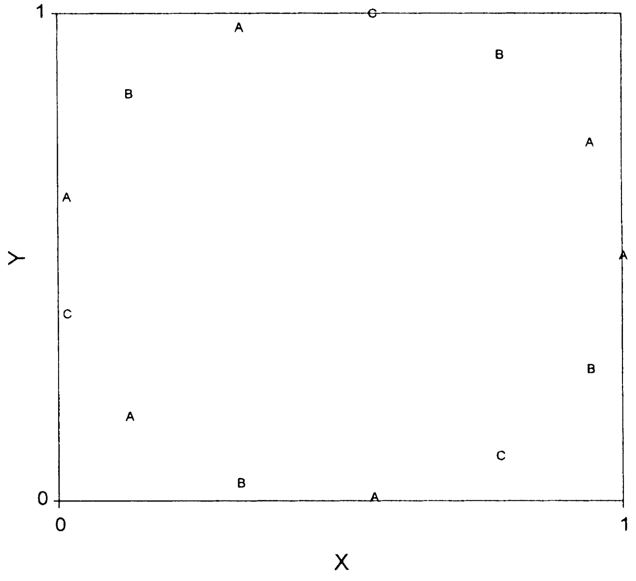

Fig. 6. ACO-4 strategy for example problem.

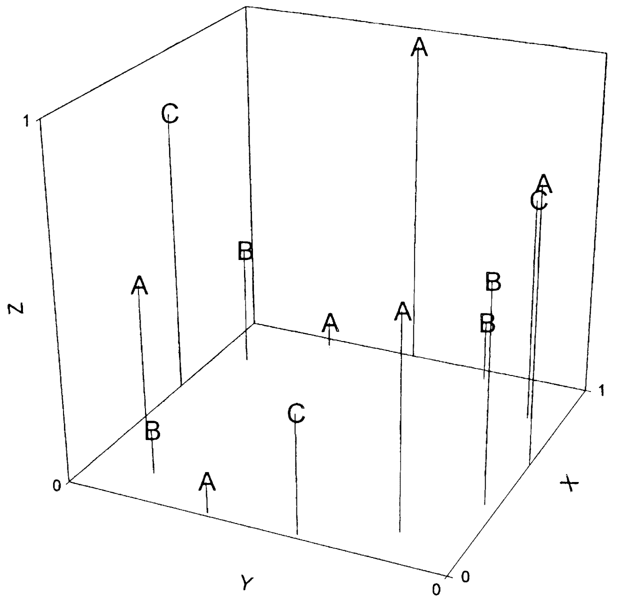

Fig. 8. ACO-6 strategy for example problem.

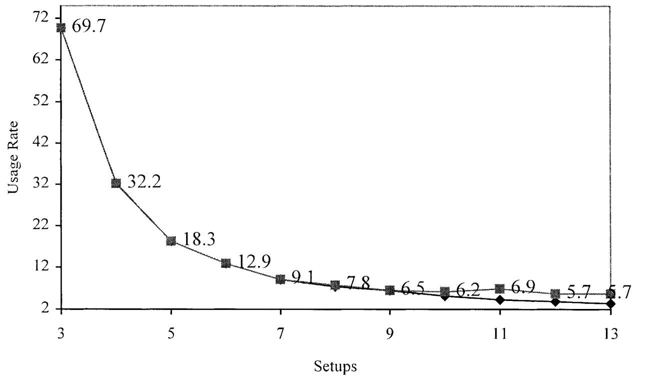

Fig. 9. Efficient frontier obtained via ACO-1 for example problem — numbers are usage rates for each associated number of setups for sequence obtained via ACO search.

When each *ant* commences its (singular) tour, the set of nodes included in the tour is empty. The probability that the current *ant* moves from node r to s is as follows:

$$p_k(r,s) = \frac{\tau(r,s)\delta(r,s)^{-\beta}}{\sum\limits_{u \notin M_k} \tau(r,u)\delta(r,u)^{-\beta}}, \qquad \text{if } s \notin M_k; \ \ 0 \text{ otherwise} \tag{7}$$

Monte-Carlo simulation is used to select the node to visit next. The selected node then joins the set of visited nodes (M_k). Local updating of the pheromone trail is performed to: (1) prevent various trail links from becoming dominant and (2) to emulate the evaporation in the pheromone trail. If node s is selected to follow node r, then local updating follows this relation:

$$\tau(r,s) = (1-\alpha)\tau(r,s) + \alpha\tau_0(r,s) \tag{8}$$

Each ant that completes a tour through the system results in a singular tour, or production sequence. The Euclidean distance of this sequence is determined (L). If this value of L is less than the smallest value of L found thus far, then global updating is performed to all links in this recently found tour according to the following relation:

$$\tau(r,s) = (1-\alpha)\tau(r,s) + \alpha L^{-1} \tag{9}$$

Global updating is done to enhance, or strengthen, the probability of these 'shorter' tours occurring. In short, local pheromone updating encourages 'exploration' of alternative solutions (in an attempt to find near-optimal solutions), while global pheromone updating encourages 'exploitation' of the most promising solutions.

As stated, each ant that completes its tour through the system results in a sequence. With each sequence, the number of required setups (S) and usage rate (U) are determined from Eqs. (2) and (3), respectively. If the usage rate for the associated number of setups is less than the lowest usage rate found thus far for the associated number of setups, than the newly discovered usage rate for that number of setups claims its place on the efficient frontier. Mathematically, this is as follows:

$$\text{If } U(S) < UBest(S), \text{ then } UBest(S) = U(S) \tag{10}$$

where $U(S)$ is the usage rate for the current sequence with S setups, and $UBest(S)$ is the lowest usage rate found thus far having S setups. After *ants* number of ants are simulated, the search process is complete. The efficient frontier is then reported.

4.3. Example of ACO heuristic on example problem

Fig. 9 shows the efficient frontier obtained when the ACO-1 strategy is employed for the example problem detailed earlier. Parameter values used were $\alpha = 0.1$, $\beta = 2$ and $ants = 1000$.

Close inspection of Fig. 9 shows two frontiers — the 'lower' frontier is the same as the one from Fig. 1. The 'upper' frontier is the one obtained from the ACO-1 heuristic described above, and its usage values are presented in the chart. From three setups to seven setups, and for nine setups, the ACO-1 frontier is identical to the optimal frontier. For eight setups, and for 10–13 setups, the ACO-1 frontier is inferior to the optimal frontier obtained via enumeration. On average, inferiority is 0.741%, which is obtained via computing the average percentage inferiority for each position on the frontier. This measure of performance is referred to as average inferiority hereafter. Of the 60,060 solutions obtained via enumeration, 37 of them were superior to the solutions obtained via the ACO-1 approach at the associated number of setups. This, then, places the ACO-1 strategy in the 99.9384th percentile as compared to the optimal frontier. Another way to express this percentile performance is in the number of inferiors per 1,000,000 solutions (via enumeration), which would be 616.05 for this example problem. This second measure of percentile performance is used hereafter because it better emphasizes differences in percentile performance as compared to the first measure.

5. Experimentation

To gauge the performance of the presented ACO methodology, the six strategies are used on some problems from the literature [9] (where several problem sets were presented in an earlier attempt to find sequences with minimal usage rates). Additional problem sets used here were taken from the literature [16] (in an earlier attempt to find sequences performing competitively with regard to both minimal setups and usage rates). The performance of these ACO-based search approaches is compared to results from other search heuristic techniques.

5.1. Design of experiment

Appendix A details the example problems used for this analysis. The six ACO strategies are compared to solutions

Table 2
Performance statistics for problem set 1 — ACO-1 and ACO-2 provide the most desirable performance of the ACO-based approaches

Approach	Inferiority rate (Std.)	Percentile (Std.)	CPU ratio (Std.)
ACO-1	0.68 (0.87)	49.13 (74.32)	1.20 (0.00)
ACO-2	0.85 (0.99)	42.52 (45.16)	1.20 (0.00)
ACO-3	1.60 (2.32)	137.63 (168.20)	1.20 (0.00)
ACO-4	3.15 (2.84)	142.04 (124.46)	1.20 (0.00)
ACO-5	2.10 (1.87)	90.39 (70.86)	1.20 (0.00)
ACO-6	2.72 (2.02)	111.49 (97.76)	1.20 (0.00)
GA	4.31 (2.52)	222.03 (164.23)	0.71 (0.00)
SA	0.69 (1.03)	102.04 (113.71)	1.20 (0.00)
ANN	0.73 (1.30)	24.57 (43.49)	0.28 (0.41)
TS	1.15 (1.39)	83.14 (104.35)	1.20 (0.00)

to the same problems obtained from simulated annealing (SA [10,11]), tabu search (TS [12,5]), genetic algorithm (GA [13,14]) and artificial neural network (ANN [15,16]) approaches. To make a fair comparison between the different heuristics, the same number of solutions were evaluated for each during the search process — this encourages a 'level playing field'. The number of solutions evaluated for the heuristics are listed in Appendix A. The three performance measures of interest here are: the average inferiority, percentile performance and the ratio of CPU time required to obtain the optimal, enumerated frontier to the CPU time required to obtain the efficient frontier based upon the heuristic search. For average inferiority and percentile performance (as described in Section 5) low values are desired, and for the CPU ratio, larger values are desired.

Generation of all efficient frontiers (both heuristic and enumerated) were done using C/C++ coding on dual Intel Pentium III 500 MHz processors. In all instances, efforts were made such that computations were done as efficiently as possible. For all ACO approaches, the following parameter values were used: $\alpha = 0.1$ and $\beta = 2$, because previous research and pilot testing have confirmed their robustness.

5.2. Experimental results and discussion

Tables 2–4 show performances of the search heuristics organized by the problem set. Table 5 shows mean performance measures of the search heuristics for all problem sets. From inspection of these tables, it is immediately clear that of the ACO heuristics, ACO-1 and ACO-2 outperform the others. ACO-3 through ACO-6 provide less promising results in terms of average inferiority. Because of this, discussion of ACO-3 through ACO-6 will be made sparingly.

Table 3
Performance statistics for problem set 2 — ACO-1 and ACO-2 provide most desirable inferiority rates of ACO-based approaches

Approach	Inferiority rate (Std.)	Percentile (Std.)	CPU ratio (Std.)
ACO-1	6.46 (6.31)	105.40 (73.65)	11.07 (12.69)
ACO-2	6.26 (6.59)	107.46 (101.20)	11.07 (12.69)
ACO-3	7.16 (5.13)	47.50 (36.20)	11.07 (12.69)
ACO-4	12.67 (9.85)	83.42 (51.93)	11.07 (12.69)
ACO-5	7.82 (6.22)	65.79 (39.44)	11.07 (12.69)
ACO-6	7.01 (5.01)	74.41 (42.47)	11.07 (12.69)
GA	1.13 (1.14)	53.98 (58.59)	5.23 (4.44)
SA	0.65 (0.64)	26.99 (28.14)	8.8 (7.46)
ANN	1.80 (1.55)	20.20 (18.59)	0.32 (0.34)
TS	0.79 (0.94)	32.33 (35.86)	8.80 (7.46)

Table 4
Performance statistics for problem set 3 — the desirability of CPU ratio of the ACO-based approaches becomes clear

Approach	Inferiority rate (Std.)	Percentile (Std.)	CPU ratio (Std.)
ACO-1	10.63 (8.89)	31.68 (40.99)	663.60 (959.72)
ACO-2	10.84 (6.12)	35.43 (42.63)	663.60 (959.72)
ACO-3	14.23 (10.03)	15.47 (21.70)	663.60 (959.72)
ACO-4	16.36 (11.27)	21.61 (28.21)	663.60 (959.72)
ACO-5	17.51 (13.96)	17.57 (16.37)	663.60 (959.72)
ACO-6	18.43 (11.69)	25.02 (28.16)	663.60 (959.72)
GA	4.10 (4.05)	80.89 (55.29)	302.67 (323.43)
SA	1.23 (1.42)	5.22 (7.05)	509.20 (544.14)
ANN	10.10 (6.54)	12.71 (18.98)	11.94 (17.04)
TS	2.23 (3.09)	13.66 (14.76)	509.20 (544.14)

From inspection of the summary information in Table 5, it is clear that both the SA and TS approaches perform well regarding inferiority rate and percentile performance, along with a desirable showing in terms of CPU needs. The GA approach offers a desirable performance in terms of inferiority rate, but shows a less competitive performance in terms of percentile and CPU performance. The ANN approach is very strong in percentile performance, reasonable for inferiority rate, but performs quite poorly in terms of CPU requirements.

ACO-1 and ACO-2 are clearly outperformed by SA and TS in terms of inferiority rate and percentile performance. They are more competitive with the GA and ANN approaches in terms of these two performance measures. The vast majority of the 'inferiority' associate with ACO-1 and ACO-2 resides in the 'high-setup end' of the efficient frontier. This claim is supported by the example problem as well in Fig. 9. Typically, these two strategies closely

Table 5
Performance statistics for all problem sets — performance for ACO-1 and ACO-2 are evident, as well as the CPU performance for all ACO-based approaches

Approach	Inferiority rate (Std.)	Percentile (Std.)	CPU ratio (Std.)
ACO-1	6.34 (7.49)	63.11 (69.62)	243.22 (640.81)
ACO-2	6.39 (6.60)	63.35 (75.35)	243.22 (640.81)
ACO-3	8.14 (8.39)	61.21 (101.12)	243.22 (640.81)
ACO-4	11.33 (10.31)	77.58 (86.23)	243.22 (640.81)
ACO-5	9.71 (10.94)	55.32 (52.89)	243.22 (640.81)
ACO-6	9.92 (10.02)	67.01 (67.18)	243.22 (640.81)
GA	3.09 (3.12)	110.72 (118.58)	111.04 (237.48)
SA	0.87 (1.07)	40.17 (71.78)	186.82 (399.52)
ANN	4.49 (5.84)	18.73 (27.06)	4.49 (11.37)
TS	1.41 (2.09)	39.84 (63.63)	186.82 (399.52)

resemble the optimal frontier of the enumerated solution on the 'low-setup end' of the frontier. This should not be surprising given the intent of ACO-1 and ACO-2 — they favor sequences with minimal setups. Unfortunately, the other strategies that favor objectives or usage rate alone generally do not perform well — their lack of performance mainly resides on the 'low setups end' of the frontier. Finding ways to improve inferiority rate and percentile performance for these ACO approaches does provide an opportunity for future research.

One of the more competitive features of the ACO strategies is the CPU performance. Also desirable in this department are the SA and TS approaches. Other research efforts have also made this point. From a computational standpoint, SA and TS approaches typically involve random selection of solution components, and make swaps. The ACO approach presented here makes Monte-Carlo based selections, and performs pheromone updating. These three techniques are typically computationally efficient, as reflected here. GA and ANN approaches, however, are not as computationally efficient, which is also reflected here. For this type of problem, GA approaches require complicated mappings of solutions, and their subsequent manipulations via crossover and mutation. ANN approaches require many iterations, or epochs, so that the required 'learning' can occur. This is very computationally inefficient.

6. Concluding comments

The JIT sequencing problem with non-negligible setup times has been addressed via a variety of ACO heuristics. Use of a spatial mapping approach and simulating the behavior of the food foraging habits of these social insects have resulted in competitive results for two of the six presented strategies. These two strategies (ACO-1 and ACO-2) provide desirable results in terms of average efficient frontier performance, percentile performance and CPU requirements. One of the advantages of using this approach is the CPU efficiency. While simulated annealing and tabu search approaches provide the most desirable results in terms of average inferiority and percentile performance, the author still considers the results of the ACO-1 and ACO-2 strategies competitive. Furthermore, ACO is still a new field, and with time its general performance should improve with new developments in the field.

Some considerations regarding implementation of this methodology must be made for industrial-scale problems. Management must realize that they need to have a clear understanding of the relative importance between setups and usage rates. Some manufacturing firms may be better off to minimize setups at the subsequent expense of high usage rates, while other firms may be better off with the opposite scenario (minimizing usage rates at the subsequent expense of more setups). Furthermore, some firms may have the need to perform reasonably well with regard to both setups and usage rates. The point is, firms implementing this methodology must understand which of these opposing objectives is most important to them.

The use of social insects to assist with combinatorial optimization problems has tremendous potential. The methodology presented here could be used for other scheduling and sequencing problems. There are other potential applications for this use of social insects: ants and termites perform division of labor with regularity — this concept could be used to assist with manufacturing design problems such as assembly line balancing or assigning tasks to robots. This idea could be enhanced to assist to designing layouts for manufacturing firms. This division of labor concept could also be extended to model a layout problem on a much smaller physical scale — such as the design of circuit boards. On a much larger scale, this concept could be used for physically large-scale applications, such as placement of satellites for cellular communications systems.

Given the relatively efficient CPU needs for these techniques, and the virtually unlimited opportunities for application, ACO techniques, and more generally, Swarm Intelligence should be considered a serious optimization tool.

Appendix A

Tables A1–A3.

Table A1
Problem Set 1: number of each product type in product mix — total demand is 10

Problem	Item 1	Item 2	Item 3	Item 4	Item 5	Solutions	*ants*
B	6	1	1	1	1	5040	5554
C	5	2	1	1	1	15,120	6269
D	4	2	2	1	1	37,800	6866
E	4	3	1	1	1	25,200	6602
F	3	3	2	1	1	50,400	7054
G	3	2	2	2	1	75,600	7318
H	2	2	2	2	2	113,400	7582

Table A2
Problem Set 2: number of each product type in product mix — total demand is 12

Problem	Item 1	Item 2	Item 3	Item 4	Item 5	Solutions	*ants*
B	8	1	1	1	1	11,880	6112
C	7	2	1	1	1	47,520	7015
D	6	3	1	1	1	110,880	7567
E	6	2	2	1	1	166,320	7831
F	5	3	2	1	1	332,640	8283
G	5	2	2	2	1	498,960	8547
H	4	3	2	2	1	831,600	8880
I	4	4	2	1	1	415,800	8428
J	3	3	2	2	2	1,663,200	9331

Table A3
Problem Set 3: number of each product type in product mix — total demand is 15 (*Note*. For all problems, *ants* are the number of ants simulated, or the number of solutions obtained via the heuristic. This is also the number of solutions evaluated for the other search heuristics)

Problem	Item 1	Item 2	Item 3	Item 4	Item 5	Solutions	*ants*
B	11	1	1	1	1	32,760	6773
C	10	2	1	1	1	180,180	7884
D	9	3	1	1	1	600,600	8668
E	7	5	1	1	1	2,162,160	9502
F	7	3	2	2	1	10,810,800	10,551
G	6	3	3	2	1	25,225,200	11,103
H	5	3	3	3	1	50,450,400	11,554
I	4	3	3	3	2	126,126,000	12,151
J	3	3	3	3	3	168,168,000	12,339

References

[1] Miltenburg J. Level schedules for mixed-model assembly lines in just-in-time production systems. Management Sci 1989;35(2):192–207.
[2] Monden Y. Toyota production system. Norcross, GA: The Institute of Industrial Engineers, 1983.
[3] McMullen PR. Using search heuristics and an efficient frontier approach to address JIT sequencing problems with setups. Proceedings of the YOR 11 Conference, Cambridge, England. 2000.
[4] Ding F, Cheng L. An effective mixed-model assembly line sequencing heuristic for just-in-time production systems. J Oper Management 1993;11(1):45–50.
[5] McMullen PR. JIT sequencing for mixed-model assembly lines with setups using tabu search. Production Plann Control 1998;9(5):504–10.
[6] Bonabeau E, Dorigo M, Theraulaz G. Swarm intelligence: from natural to artificial systems. Oxford: Oxford University Press, 1999.
[7] Bonabeau E, Theraulaz G. Swarm smarts. Scient Am 2000; March:72–9.
[8] Dorigo M, Gambardella LM. Ant colonies for the travelling salesman problem. BioSystem 1997;43(1):73–81.
[9] Sumichrast RT, Russell RS. Evaluating mixed-model assembly line sequencing heuristics for just-in-time production systems. J Oper Management 1990;9:371–90.
[10] Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by simulated annealing. Science 1983;220:671–9.
[11] McMullen PR, Frazier GV. A simulated annealing approach to mixed-model sequencing with multiple objectives on a JIT line. IIE Trans 2000;32(8):679–86.
[12] Glover F. Tabu search: a tutorial. Interfaces 1990;20:74–94.
[13] Michalewicz Z. Genetic algorithm + data structures = evolution programs. Berlin: Springer, 1994.
[14] McMullen PR, Tarasewich P, Frazier GV. Using genetic algorithms to solve the multi-product JIT sequencing problem with setups. Int J Production Res 2000;38(12):2653–70.
[15] Kohonen T. The self-organizing map. Proc IEEE 1990;78(9):1464–80.
[16] McMullen PR. A Kohonen self-organizing map approach to addressing a multiple objective, mixed-model JIT sequencing problem. Int J Production Econ. Forthcoming.