Int J Adv Manuf Technol (2008) 37:589–598
DOI 10.1007/s00170-007-0998-8

ORIGINAL ARTICLE

Customer order scheduling problem: a comparative metaheuristics study

Öncü Hazır · Yavuz Günalay · Erdal Erel

Received: 21 October 2006 / Accepted: 6 March 2007 / Published online: 3 April 2007

Abstract The customer order scheduling problem (COSP) is defined as to determine the sequence of tasks to satisfy the demand of customers who order several types of products produced on a single machine. A setup is required whenever a product type is launched. The objective of the scheduling problem is to minimize the average customer order flow time. Since the customer order scheduling problem is known to be strongly NP-hard, we solve it using four major metaheuristics and compare the performance of these heuristics, namely, simulated annealing, genetic algorithms, tabu search, and ant colony optimization. These are selected to represent various characteristics of metaheuristics: nature-inspired vs. artificially created, population-based vs. local search, etc. A set of problems is generated to compare the solution quality and computational efforts of these heuristics. Results of the experimentation show that tabu search and ant colony perform better for large problems whereas simulated annealing performs best in small-size problems. Some conclusions are also drawn on the interactions between various problem parameters and the performance of the heuristics.

Ö. Hazır · Y. Günalay (✉) · E. Erel
Faculty of Business Administration, Bilkent University,
06800 Ankara, Turkey
e-mail: gunalay@bilkent.edu.tr

Ö. Hazır
e-mail: oncuh@bilkent.edu.tr

E. Erel
e-mail: erel@bilkent.edu.tr

Ö. Hazır
Department of Industrial Engineering, Cankaya University,
06530 Ankara, Turkey

Keywords Metaheuristics · Customer order scheduling · Simulated annealing · Genetic algorithms · Tabu search · Ant colony optimization

1 Introduction

The customer order scheduling problem (COSP) is defined as sequencing multi-family customer orders on a single production facility with setups. A customer order involves more than one product that is processed on the same machine. The facility needs a setup whenever a product type is switched. The aim of the problem is to minimize the average customer order flow time, defined as the duration between the release time of the customer order and the completion time of all the jobs in that order.

The COSP is frequently encountered in make-to-order and make-to-assembly production environments in which a single facility produces different product types. In this scenario, a customer can request one or more of these products and his order is shipped only after all the products in the order have been produced and packed together. A setup is required when the facility switches among the products; hence, the management wishes to group customer orders within the same family to avoid non-value-adding setup activities. However, this policy may lead to undesirably long waiting times for the customers. This trade-off is well discussed in comprehensive review papers by Potts and Kovalyov [1] and Allahverdi et al. [2] on scheduling models with batch service and setup times.

The concept of customer order scheduling was first introduced by Julien and Magazine [3] who provided a dynamic programming formulation of the problem with two product families and a given order processing sequence. They also discussed the necessary properties of the optimal

solution for the general case. The COSP on single and multiple machines has attracted several other researchers over the last two decades such as Baker [4], Coffman et al. [5] and Vickson et al. [6] worked on single-machine cases, and Daganzo [7], and Yang and Posner [8], who studied the multi-machine case. Yang [9] in a different article studied the COSP with two machines and various objective functions, and showed that all variations have non-polynomial complexity.

Gerodimos et al. [10] addressed an equivalent multi-operation scheduling problem of which their job description matches with our customer definition. In their problem, each job has at most one operation from each family and a job is considered completed only after all of its operations have been processed. They provide properties of an optimal schedule for this problem. In a recent study, Erel and Ghosh [11] propose a dynamic programming algorithm to solve the general COSP, which is built on the dominance properties of Julien and Magazine [3]. They also show that the problem with the objective of minimum total customer order flow time is NP-hard in the strong sense.

Given the NP-hard nature of the problem, there has been no successful attempt to solve large-size COSPs. In this paper, we propose four major metaheuristics to solve the COSP with a large number of customers and product families that may be encountered in real life. Since late 1970's, heuristic algorithms have been widely used to solve difficult combinatorial problems, e.g., the traveling salesman problem, the knapsack problem, the vehicle routing problem, as well as many scheduling problems. When it is known that the optimal solution of a problem is impractical to obtain, heuristic algorithms are the only salvage. The area of constructing heuristics has attracted the attention of numerous researchers, which has led to a vast number of articles being published; a recent survey by Blum and Roli [12] lists over 172 references. More recently a new paradigm in heuristic construction has emerged, leading to a class of heuristics called "modern" (Reeves [13]) or "metaheuristics" (Glover [14]). Blum and Roli [12] list the fundamental properties of these procedures as follows:

i. They are high-level strategies for efficiently exploring search spaces to find near-optimal solutions
ii. They are approximate, usually non-deterministic, and not problem-specific
iii. They all try to avoid getting trapped in local optima
iv. They range from simple local search procedures to complex learning processes that may utilize domain-specific knowledge

Having the above commonalities, metaheuristics also differ from each other with respect to their search mechanisms. We keep this in mind when choosing the four metaheuristics of our study: simulated annealing, genetic algorithms, tabu search, and ant colony optimization. The objectives of our study are to solve the COSP of realistic sizes with these metaheuristics, and also to review the most well-known metaheuristics. As a result of this review, the performance of these metaheuristics under various conditions is assessed, and conclusions on the relationships between various problem parameters and the performance of heuristics are drawn.

In the sequel, the problem considered in this study is explained in detail in the next section. A brief description of the metaheuristics is given in Section 3 and experimental design and parameter settings of the metaheuristics are given in Section 4. The results of the experiments are presented and discussed in Section 5. Finally, the last section is reserved for concluding remarks.

2 The model formulation

Suppose a single facility producing P different products satisfies the demands of C customers. The demand of customer j from product type i requires p_{ij} time units of processing. A setup of s_i time units is required prior to processing any product of type i if the previous product is of type l where $l \neq i$. Neither the processing nor the setup times are sequence-dependent. Since all orders are released at time zero, the flow time for a customer is the point in time at which the last product requested by that customer is completed. The objective of the problem is to minimize the sum of customer flow times. A binary integer programming model of the problem is presented below:

$$\text{Min.} \sum_{i}\sum_{j}\sum_{k}\sum_{m} p_{ij}\, U_{ijkm} + \sum_{i}\sum_{k}\sum_{m} s_i\, V_{ikm}$$

s.t.

$$\sum_{i}\sum_{j} X_{ijk} = 1 \qquad k = 1,..,N \tag{1}$$

$$\sum_{k} X_{ijk} = 1 \qquad i = 1,..,P; j = 1,..,C \tag{2}$$

$$k * X_{ijk} \leq a_j \qquad i = 1,..,P; j = 1,..,C; k = 1,..,N \tag{3}$$

$$\sum_{k} Z_{km} = a_m \qquad m = 1,..,C \tag{4}$$

$$Z_{k+1,m} \leq Z_{k,m} \qquad m = 1,..,C; k = 1,..,N-1 \tag{5}$$

$$\sum_{j} X_{ij1} \leq Y_{i1} \qquad i = 1,..,P \tag{6}$$

$$\sum_{j} X_{ijk} - \sum_{j} X_{ijk-1} \leq Y_{ik} \quad i = 1,..,P; k = 2,..,N \tag{7}$$

$$X_{ijk} + Z_{km} - 1 \leq U_{ijkm} \quad i = 1,..,P; j = 1,..,C; k = 1,..,N; m = 1,..,C \tag{8}$$

$$Y_{ik} + Z_{km} - 1 \leq V_{ikm} \quad i = 1,..,P; k = 1,..,N; m = 1,..,C \tag{9}$$

The decision variables are defined as:

- X_{ijk}=1 if product i of customer j is in position k; 0 otherwise
- Y_{ik}=1 if there exists a setup for product i prior to position k; 0 otherwise
- a_j = position at which all products of customer j are completed
- Z_{km}=1 if customer m is in the system at position k; 0 otherwise
- U_{ijkm}=1 if both X_{ijk}=1 and Z_{km}=1; 0 otherwise
- V_{ikm}=1 if both Y_{ik}=1 and Z_{km}=1; 0 otherwise

U_{ijkm} and V_{ikm} are indicator variables used to calculate the flow time of customer m. U_{ijkm} is the number of customers whose flow times include p_{ij}. Similarly, V_{ikm} is the number of customers whose flow times include s_i. Note that N represents the number of positions in the sequence; i.e., N=$P.C$.

Constraints (1) and (2) ensure that each task is scheduled at a certain position and each position is filled by exactly one task, respectively. Constraints (3), (4), and (5) are used to mark the position at which each customer order is fully completed. Constraints (6) and (7) check if there exists a product type change that leads to a setup prior to position k. Constraints (8) and (9) define the indicator variables to properly determine the flow times.

The number of binary integer variables and the number of constraints in the above model are both in the order of P^2C^3. The fast growth in the number of binary variables leads to excessive memory and computational requirements for the available optimization software for problems of realistic sizes.

3 Description of the metaheuristics

Four well-known metaheuristics, simulated annealing (SA), genetic algorithm (GA), ant colony optimization (ACO), and tabu search (TS) were chosen for comparison. Our selection criterion was to include metaheuristics with different characteristics, such as nature-inspired (GA and ACO) versus non-nature inspired (TS), population-based (GA and ACO) versus single-point search (SA and TS), memory usage (GA, TS, and ACO) versus memoryless (SA) (Blum and Roli [12]). Hence, we can assess and compare their performances with respect to these different characteristics. Similar comparative studies have been conducted by researchers for different problem domains, e.g., Pradhan and Lam [15] on job shop scheduling problem, and Baskar et al. [16] on machine tooling optimization. Brief descriptions of each metaheuristic are given below; the parameter fine tuning and implementation details are given in Section 4.2.

3.1 Simulated annealing

Simulated annealing (SA) is one of the oldest and most frequently used metaheuristics to find approximate solutions to combinatorial optimization problems. SA is a local search, improvement heuristic with a clever mechanism to prevent getting trapped at local optima. Starting from a randomly generated initial solution, the heuristic generates neighbor solutions by simple moves (i.e., simple perturbations in the current solution). Although improving moves are desired, a special mechanism occasionally allows moves to worse solutions to enrich the search domain and prevent getting trapped at local optima. SA was first proposed by Kirkpatrick et al. [17]. Subsequently, Johnson et al. [18, 19] reported a comprehensive study on the adaptation of simulated annealing to graph partitioning and the graph coloring problems and examined the effects of various parameters on the solution quality and computational requirements. Detailed description and discussion of the algorithm can be found in Johnson et al. [18, 19] and Reeves [13].

3.2 Genetic algorithm

The genetic algorithm (GA) is a population-based improvement heuristic based on natural selection and evolutional theory. A solution is represented by knowledge structures (also called chromosomes) that are composed of genes. A set of solutions comprises a population that evolves over time through competition and controlled variation. Each member of the population (individual) is evaluated and assigned a fitness value and then the next population is formed in two steps. First, individuals with high fitness values are selected for reproduction and a crossover operator generates two offspring from two parents. The role of the crossover operator is to form new fit individuals from fit parents (natural selection). Second, a mutation operator alters one or more components of a selected individual to provide new information into the knowledge base.

The mutation operator serves as a secondary search that ensures that all points in the search domain are reachable. The incumbent solution is expected to improve as populations are generated until a stopping criterion is reached. GA was first introduced by Holland [20] and then extensively used by numerous researchers to solve problems from various fields. Interested readers are referred to the several references written on GA (e.g., Holland [20] and Reeves [13]).

3.3 Ant colony optimization

The ant colony optimization (ACO) algorithm is another population-based metaheuristic inspired by the collective behavior of real ants for the survival of their colonies. Ants deposit pheromone on their path to the food sources and the ability of other ants to smell this chemical enables them to find the shortest path between their nest and the food. When more ants collectively follow a trail, the trail becomes more attractive for being followed in the future. The capability of a single ant to locate food is limited, but the collected/shared knowledge helps the colony to find efficient paths to a food source; the information about the good paths is passed on to the members of the colony via the amount of pheromone deposited on the paths. Dorigo and Gambardella [21] utilize this feature in solving combinatorial optimization problems.

Each ant constructs a solution by moving from one state to another adjacent state by applying a stochastic local search policy; a tour ends when all the ants of the colony generate solutions of different quality. The information gathered at the end of each tour is updated through a global pheromone updating rule. The ants are expected to generate better solutions by using this information in the next tour. The algorithm terminates when a stopping criterion is satisfied. Readers who are interested in ACO applications are referred to a recently published survey article by Fox et al. [22].

3.4 Tabu search

Tabu search (TS) is a local-search, improvement heuristic similar to SA with a punishment mechanism to avoid getting trapped at local optima by forbidding or penalizing moves that cause cycling among solution points previously visited. These forbidden moves are called "tabu". TS algorithm keeps a list of such moves for a specific number of iterations, called tabu tenure. There are two commonly used strategies to obtain good solutions: diversification is used to direct the search into less visited regions of the search space, whereas intensification is used to fully explore a certain region. However, tabu status of a move can be overridden if it leads to a solution better than the incumbent solution (aspiration criterion).

Glover's studies on TS have attracted numerous researchers to use the metaheuristics to solve problems from various fields due to its potential to solve difficult combinatorial optimization problems, TS is still a contemporary solution procedure vastly utilized. Interested readers are suggested to consult to books by Revees [13] and Glover and Laguna [23].

4 Experimental design and parameter setting

In order to compare the quality of solutions of each metaheuristic, SA, GA, ACO, and TS, experimentation is designed and 192 randomly generated problems are solved. In this section, the important problem characteristics as well as the critical parameter settings of each heuristic are presented.

4.1 Problem characteristics

The characteristic parameters of family scheduling problems are the number of customers, C, number of product types, P, variability and mean of setup times, and the density of the customer-product request matrix; if all customers request from each product type, this is a full customer-product matrix with density 1, and total number of tasks to be produced is N=P.C. Note that the system performance measure does not depend on processing times, but rather it depends on setup times. Note that the problem is trivial to solve as the setup times approach zero. Julien and Magazine [3] use *individual order scheduling* definition to explain this phenomenon. As the setup times get relatively larger, the problem gets more challenging to solve. Table 1 summarizes the levels of each problem characteristic.

The product processing times are drawn from the uniform distribution U(1, 15). Four different levels of setup times are considered: low mean-low variance (U(10, 20)), low mean-high variance (U(1, 30)), high mean-low variance (U(25, 35)), and high mean-high variance (U(15, 45)). For each problem set, setup time parameters are randomly generated from the above uniform distributions. Note that the setup means are relatively larger than the processing times. The customer-product mix specifies the percentage of products requested by a typical customer, namely the density of the problem, D. For example, if D is 0.5, then a customer's order contains half of the product types on

Table 1 Experimental design parameters

Problem parameters	Levels
Customer classes, C	5, 10, 15, 20
Product types, P	5, 10, 15, 20
Customer-product matrix density, D	1, 0.75, 0.5

average (i.e., N=0.5 P.C). Note that when D=1, each customer orders all product types of varying quantities. These problems are solved on an Ultra Enterprise 4000 248 MHz Sun Workstation. Each problem set is solved five times with different starting seeds by all metaheuristics and comparisons are done with respect to both the best of five and the average of five statistics. The problems of size, N, up to 15, can be optimally solved using CPLEX 7.5 solver in GAMS software. Hence, the performance of the metaheuristics is evaluated using optimal solutions for $N \leq$ 15, whereas for larger problems, the heuristic solutions are compared with the best found solution. A discussion of the results will be given in the next section.

4.2 Parameter settings of each algorithm

All the metaheuristics have some parameters that significantly affect computation time requirements and solution quality. These parameters have to be carefully set and fine-tuned for each problem domain; therefore, in order to find the best parameters for each metaheuristic, 30 test problems of varying problem sizes ($20 \leq N \leq 50$) are solved for a wide range of system parameters. Table 2 summarizes the tested system parameter ranges and the best values obtained for each metaheuristic. Detailed discussion of the parameter fine-tuning of the metaheuristics is presented in the section below. All metaheuristics employ a stopping (termination) criterion of 3-h CPU time limit. Besides this termination rule, the SA algorithm may also stop due to the temperature cooling process. Note that in most cases, the heuristics converge to their best values much faster (within minutes); we have taken these long runs to observe the convergence rates of these heuristics.

4.2.1 Simulated annealing

The algorithm starts with a randomly generated initial schedule and uses a two-opt neighborhood where a neighbor is obtained by reversing the processing sequence between any two items (themselves included) in the existing schedule. The initial temperature was set by using an acceptance probability of 0.4, representing the acceptance of probability of moving to a worse solution at the beginning of the procedure. Test runs indicate that the most significant parameters affecting the quality of the solution and the computation time are the minimum percentage factor and the cooling rate factor. Since wider solution space can be searched with a low minimum percentage factor or with a high cooling rate factor, the quality of the solution increases by decreasing minimum percentage factor and increasing the cooling rate factor. The change in cost per change in time (Δcost/Δtime) value is used to determine the best parameter levels that compensate the trade-off between computation time and solution quality. The parameters other than the minimum percentage and the cooling rate factor were set as recommended in Johnson et al. [18, 19].

4.2.2 Genetic algorithm

In our study, the pair (i,j) refers to the j^{th} job of customer i, i= 1,...,C, and j=1,...,P. We use permutation of N such pairs to construct the chromosomes and compute their fitness values using the setup and processing times of the schedule. Our code starts with a randomly generated 20 chromosomes (initial population), and stops after one of the termination criteria is reached. At each iteration, new offspring are created as a result of two operations: crossover and mutation. First, two fit individuals from the population are chosen with a probability, *pcross* and their chromosomes are crossovered. Note that, *pcross* is defined as the probability of a crossover in a chromosome, not in a gene. Then, the new individuals (offspring) are mutated with probability, *pmutat*. During the test runs, we tested various combinations of these parameters. For *pcross*, tested values are: 0.3, 0.4, 0.6, 0.8, and 0.9 and for *pmutat*, tested values are: 0.2, 0.3, 0.4, 0.5, and 0.6.

Table 2 Best parameter settings for each metaheuristic

	Parameters	Test values	Best value
SA	Minimum percentage	0.005, 0.01, 0.02, 0.03, 0.04, 0.05	0.02
	Cooling rate	0.99, 0.97, 0.95, 0.92, 0.90, 0.85	0.97
GA	Crossover probability	0.3, 0.4, 0.6, 0.8, 0.9	0.6
	Mutation probability	0.2, 0.3, 0.4, 0.5, 0.6	0.3
	Crossover operator	Position-based (Webster et al. [24]), Order crossover (Davids [25])	Position-based
ACO	q_0	0.7, 0.8, 0.9	0.8
	α	0.1, 0.2, 0.3	0.1
	β	1, 2	1
	p	0.1, 0.2, 0.3	0.2
TS	Initial solution	Random solution, Batched solution	Batched solution
	Tabu list size	5, 7,10	7

Furthermore, two different crossover methods are used: Position-based crossover (Webster et al. [24]), and order crossover (Davids [25]). Test runs explore the best parameters for the GA on our problem to be the position-based crossover operand with operation probabilities, *pcross* = 0.6, and *pmutat* = 0.3.

4.2.3 Ant colony optimization

The ACO implementation starts with W ants. An ant w constructs the schedule, assigns job v to position k, by applying the state transition rule given below (Dorigo and Gambardella [21]):

$$v = \begin{cases} \underset{u \in S_w(k)}{\operatorname{argmax}} \left\{ [\tau(k,u)][\eta(k,u)]^{\beta} \right\} & \text{if } q \le q_0 \\ J & \text{otherwise} \end{cases}$$

where $\tau(k,u)$ indicates the desirability of assigning job u to position k of the schedule. $S_w(k)$ is the set of jobs that can be assigned by ant w for position k. β is the parameter that determines the relative importance of pheromone level and heuristic information. The problem-dependent heuristic value, $\eta(k,u)$, is determined as follows:

$$\eta(k,u) = \frac{1}{p_u + y_{k-1,u} \cdot s_u}$$

where p_u represents processing time of job u, s_u represents setup time of job u; $y_{k\text{-}1,u}$ is a binary variable that is equal to 1 in cases where scheduled element at position k-1 belongs to the family of job u. Every feasible schedule starts with a setup, i.e., $y_{1,u}=1 \ \forall u$. Variable q is a uniformly distributed random number over [0, 1] and $0 \le q_0 \le 1$ determines the relative importance of exploitation versus exploration. J is a random variable that gives the probability with which ant w chooses job v to position k and it is selected according to the probability distribution, called a "random-proportional rule", and is given below:

$$p_w(k,v) = \begin{cases} \dfrac{[\tau(k,v)][\eta(k,v)]^{\beta}}{\sum_{u \in S_w(k)} [\tau(k,u)][\eta(k,u)]^{\beta}} & \text{if } v \in S_w(k) \\ 0 & \text{otherwise} \end{cases}$$

Every time an ant constructs a schedule, the local updating rule decreases pheromone on visited edges. Therefore, the jobs at a specific position in one ant's tour will be chosen with a lower probability at the same position in other ants' tours. Consequently, ants explore the edges not yet visited and prevent converging to a common path. Local updating rule is applied as shown below (Dorigo and Gambardella [12]) where τ_0 is the initial pheromone level and p is the pheromone evaporating parameter:

$$\tau(k,v) = (1-p)\tau(k,v) + p\tau_0$$

Test runs indicate the best values of the parameters of q_0, α, β, and p are 0.8, 0.1, 1, and 0.2, respectively.

4.2.4 Tabu search

The tabu search (TS) algorithm starts with an initial heuristic solution and keeps searching for efficient directions until the stopping criterion is satisfied. During parameter testing experiments, we consider the following alternatives for system initialization: random and batched solution in which all customer orders from the same family are sequenced contiguously. Within a batch, the final operations, which are the last operations of particular customer orders, are sequenced first; they follow the shortest processing time (SPT) order among themselves and non-final operations follow the final operations (Gerodimos et al. [10]).

Table 3 Results of the ANOVA test

Source	DF	SS	Adj SS	Adj MS	F	*p*
P	3	15.92161	1592161	5.30720	6840.38	0.000
C	3	5.46032	5.46032	1.82011	2345.91	0.000
E[setup]	1	0.77766	0.77766	0.77766	1002.31	0.000
VAR(setup)	1	0.00101	0.00101	0.00101	1.31	0.253
D	2	4.03332	4.03332	2.01666	2599.25	0.000
Method	3	0.01830	0.01830	0.00610	7.86	0.000
P.Method	9	0.01359	0.01359	0.00151	1.95	0.043
C.Method	9	0.01985	0.01985	0.00221	2.84	0.003
VAR(setup).Method	3	0.00005	0.00005	0.00002	0.02	0.996
D.Method	6	0.01512	0.01512	0.00252	3.25	0.004
E[setup].Method	3	0.00162	0.00162	0.00054	0.69	0.556
Error	724	0.56173	0.56173	0.00078		
Total	767	26.82417				

Table 4 The mean and variance of error terms (with respect to the best found solution) for each method

	D=0.5		D=0.75		D=1.0		All 192 Instances	
Method	Mean(Err) (%)	VAR(Err) (%)	Mean(Err) (%)	VAR(Err) (%)	Mean(Err) (%)	VAR(Err) (%)	Mean(Err) (%)	VAR(Err) (%)
ACO	2.70	0.04	2.18	0.02	3.47	0.22	2.78	0.27
GA	6.04	0.09	6.80	0.14	11.58	4.02	8.14	4.24
SA	1.64	0.01	5.60	0.79	12.05	8.02	6.43	8.82
TS	3.55	0.08	2.50	0.03	1.31	0.02	2.45	0.12

The stopping criterion is set to 1,000 iterations. In order to fully explore the neighborhood of generated solutions, the entire neighborhood is examined with both swap and inserts. In the test runs, a tabu list of size 5, 7, and 10 are tried. The best solutions for our model can be achieved under the following conditions:

i. Start with a batched solution.
ii. Keep a tabu list of size 7

5 Computational results

In the computational study, each problem instance is solved five times by four different metaheuristics and two different statistics from these experimentations are collected: (i) the best of five, and (ii) average of the five replications. Since both statistics show similar results, we only report the *average of five* results in this section. Note that this is a more conservative performance measure than the best of five experiments.

ANOVA test is applied to the results in order to see the variance effect of the experimental design factors. These factors and their corresponding factor levels are: number of product families (P=5, 10, 15, 20), number of customer classes (C=5, 10, 15, 20), mean of setup times (low, high), variance of setup times, (low, high), density (D=1, 0.75, 0.5), and the method (SA, GA, ACO, TS). Before the test is performed, the necessary assumptions for ANOVA are checked and Box-Cox transformation is utilized to satisfy constant variance assumption. The output of the test is depicted in Table 3.

The ANOVA test reveals that the method used has a significant effect on the variance of results and more importantly the method has a significant interaction with the problem size parameters (P, C, and D). In order to show the effect of the method distinctly for each problem instance, the relative performance of each method with

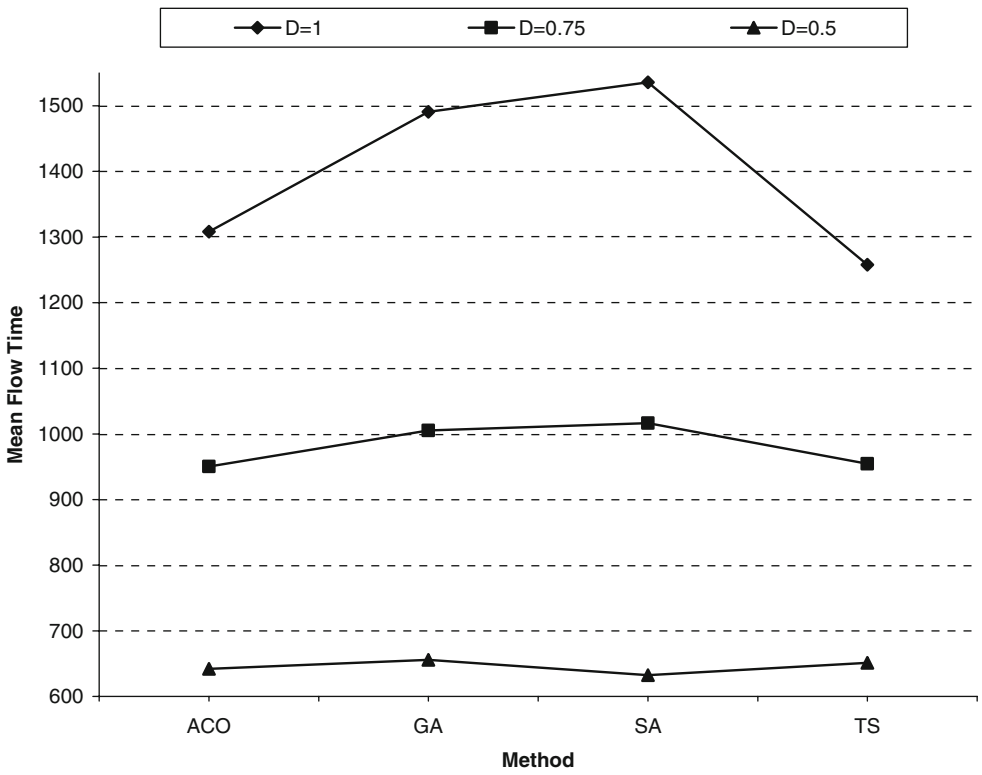

Fig. 1 The average performance of each metaheuristics as the product family density of a problem set changes from D=0.5 to D=1.0

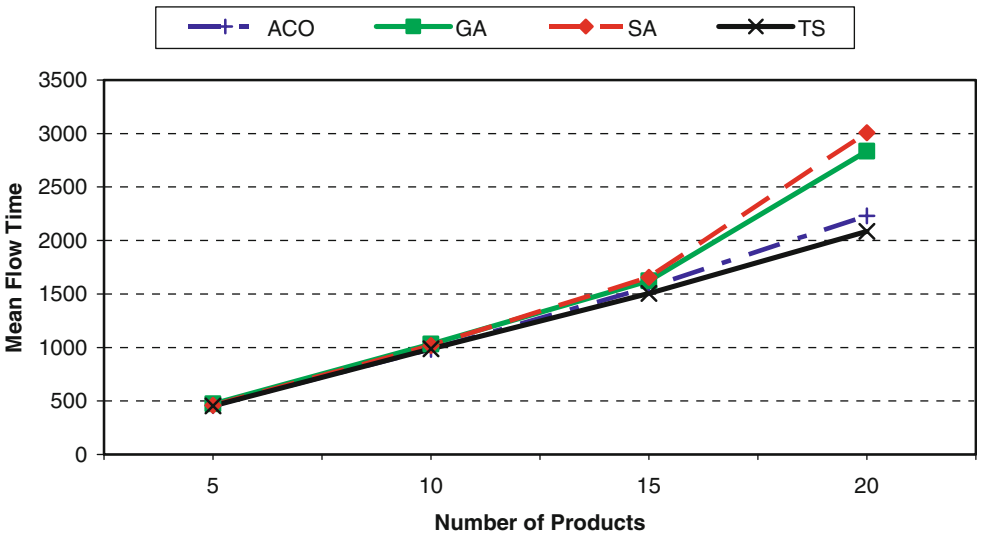

Fig. 2 The average performance of metaheuristics with respect to P, (D=1.0)

respect to the best result obtained is calculated. The results are summarized in Table 4. In the table, the mean and variance of error terms for each method are grouped with respect to the density parameter.

When D=0.5 the variance of error terms is the smallest and SA is the best performer among the four metaheuristics. As the density of customer orders increase (D=0.75 and 1.0) TS and ACO provide better solutions and their error variances are still small. Solution quality of GA and SA deteriorates, as the density of the problem data, D, increases. At this point, one has to note that at the limiting case, i.e., D=1/P, and thus *N*=C, when each customer asks for only one type of the product variety, the problem becomes trivial. Consequently, it is normal to accept D as the problem complexity measure.

SA is a very practical and efficient heuristic. But since it relies heavily on random moves, it is not surprising that its performance reduces as the problem complexity increases. Among the four metaheuristics, GA and ACO are population-based algorithms and as a search process they differ in one important aspect from SA and TS. At each iterative step, a number of different solutions are generated and information is carried over to the next step. Both GA and ACO are slower due to these extra processes involved in generating or updating a population rather than a single individual. In return, these algorithms utilize the information pooling in the population. Problem-specific information is incorporated into the solution improvement module of ACO and this gives ACO an advantage over GA. Therefore, for complex problem instances, ACO performs better than GA.

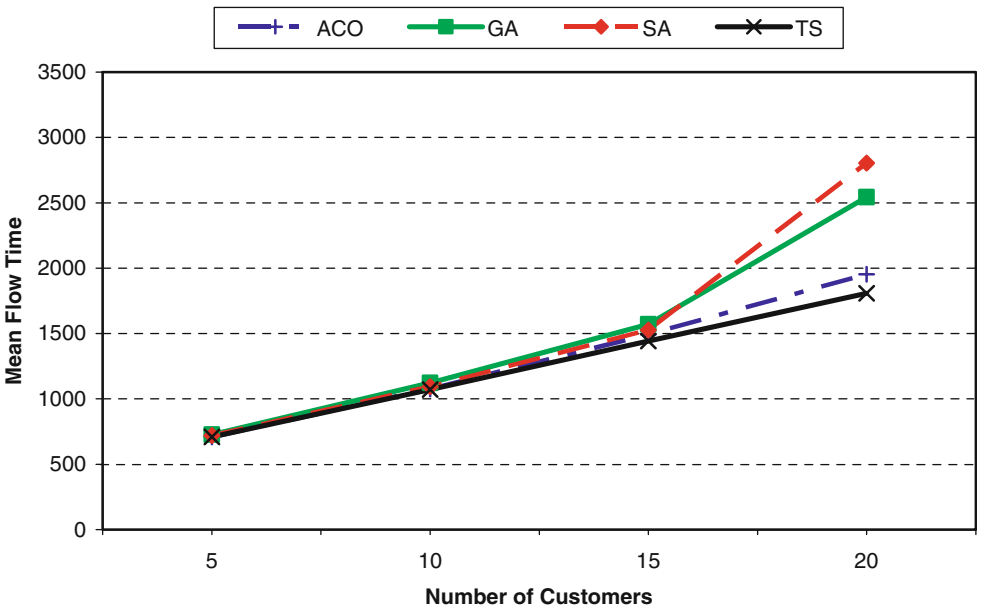

Fig. 3 The average performance of metaheuristics with respect to C, (D=1.0)

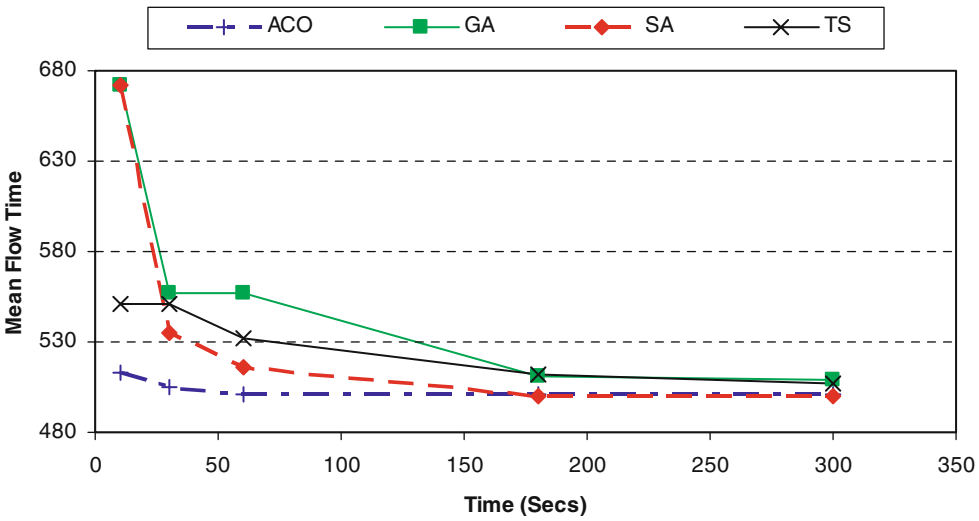

Fig. 4 Best solution found by each algorithm as computational time increases, for a 10-product type 10-customers (N=50) problem with setup times at high-mean, high-variance level, and where D=0.5

Along with ACO, TS is another intelligent metaheuristic that uses smart techniques to avoid local optimums. For the COSP, TS is the only method which constantly improves its performance as the problem complexity increases. Figures 1, 2, and 3 also verify this result, i.e., the performance of TS improves as the problem size, N, increases.

Another performance criterion of any algorithm is how fast it converges. We run each metaheuristic sufficiently long (3 h) to compare their convergence performances. Figures 4 and 5 depict the performance of each heuristic against time, for a small and large test problem, respectively. GA and SA start at a relatively poor solution, and then converge to the best result at a slow or fast rate depending on the problem complexity, whereas ACO and TS start at a better solution and improve faster irrespective of the problem size. Therefore in case of time shortage, it is more worthwhile to use TS if the problem is complex, and SA if the problem is simple.

6 Concluding remarks

In this study, the performances of four well-known metaheuristics are compared on the COSP, which is a

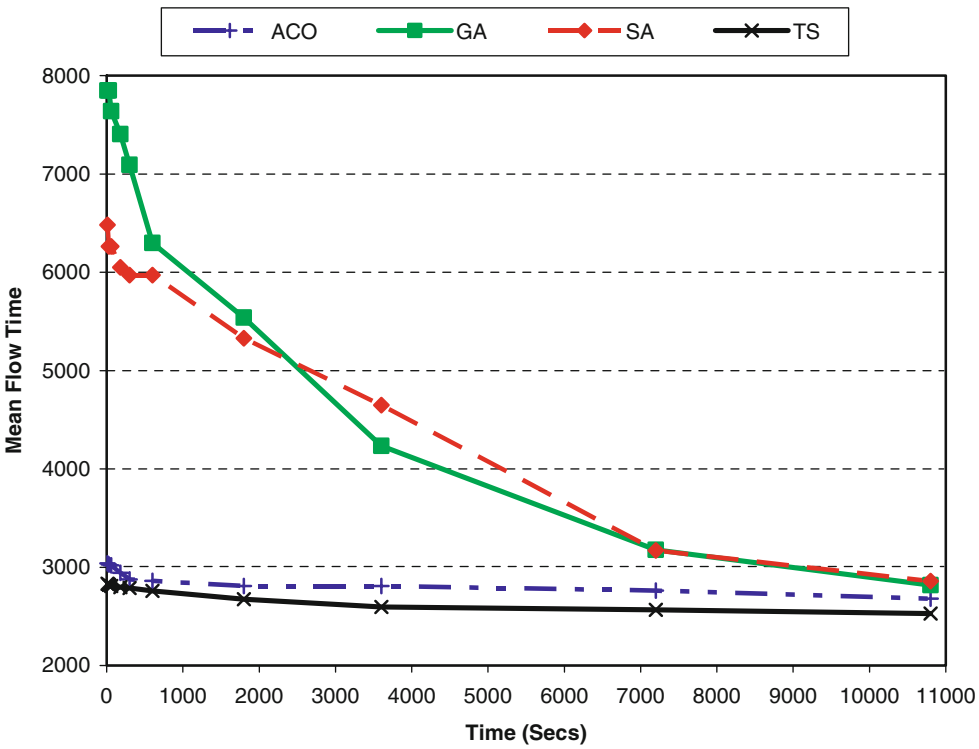

Fig. 5 Best solution found by each algorithm as computational time increases, for a 20-product type 15-customers (N=300) problem with setup times high-mean, low-variance level, and D=1

practically relevant but difficult combinatorial optimization problem. SA, GA, TS, and ACO were chosen for this study because these metaheuristics are not only the most frequently used ones, but they also present a good variety over several criteria, such as from simple to complex, from local search to nature-inspired methods, etc. From the numerical experimentation, the following observations can be drawn.

1. The output quality of a metaheuristics depends on the problem size (P, C, or D).
2. As the complexity of the problem (or the product density, D) reduces, all four methods perform better, but among them, the best algorithm is SA, even though it is the least intelligent one.
3. However, for dense problem instances, e.g., the D=1 case, TS and ACO outperform SA and GA. TS has slightly better results than ACO.
4. TS and ACO converge to their best results faster than SA and GA. Therefore in case of time limitations, they are more preferable.
5. Although the coding complexity of these methods is beyond the scope of this study with respect to the number of parameters that have to be fine tuned (see Table 2), one can say that implementation of TS or SA is easier than the others.

In light of these observations and experimentation, TS and ACO were chosen as the best alternatives for the COSP with the mean flow time as the objective function. Although the overall performance of TS in our experimentation is slightly better than ACO, we deliberately do not want to put ACO in second place, since this small difference can easily be reversed by improving the intelligent search module of ACO. SA is also proposed as a viable option for small and/or less complex problem instances (i.e., when D is small).

Although the authors try to extend the problem space used in this study, the results may not reflect the best performance of each algorithm, neither the best metaheuristics can be announced depending on the outcome of this study. Therefore the results must be used with caution. However, this study is valuable, especially to the practitioners, in that it shows that the choice of metaheuristics depends on both the problem structure and its parameters. Consequently, for every difficult problem, such a comparative study among several metaheuristics is useful, instead of choosing a well-known heuristic due to its performance on other problems. Extending this work by including hybrid metaheuristic algorithms and recent metaheuristics, e.g., variable neighbourhood search, particle swarm and electromagnetism, as well as considering different NP-hard problems, are subjects of future work.

References

1. Potts CN, Kovalyov MY (2000) Scheduling with batching: a review. Eur J Oper Res 120:228–249
2. Allahverdi A, Gupta JND, Aldowasian T (1999) A review of scheduling research involving setup considerations. Omega 27:219–239
3. Julien FM, Magazine MJ (1990) Scheduling customer orders: an alternative production scheduling approach. J Manuf Oper Man 3:177–199
4. Baker KR (1988) Scheduling the production of components at a common facility. IIE Trans 20:32–35
5. Coffman EG, Nozari A, Yannakakis M (1989) Optimal scheduling of products with two subassemblies on a single machine. Oper Res 37:426–436
6. Vickson RG, Magazine MJ, Santos CA (1993) Batching and sequencing of components at a single facility. IIE Trans 25:65–70
7. Daganzo CF (1989) The crane scheduling problem. Transp Res-B 23B:159–175
8. Yang J, Posner ME (2005) Scheduling parallel machines for the customer order problem. J Scheduling 8:49–74
9. Yang J (2005) The complexity of customer order scheduling problems on parallel machines. Comput Oper Res 32:1921–1939
10. Gerodimos AE, Glass CA, Potts CN, Tautenhahn T (1999) Scheduling multi-operation jobs on a single machine. Ann Oper Res 92:87–105
11. Erel E, Ghosh JB (2007) Customer order scheduling on a single machine with family setup times: complexity and algorithms. App Math Comput 185:11–18
12. Blum C, Roli A (2003) Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput Surv 35:268–308
13. Reeves CR (1993) Modern heuristic techniques for combinatorial problems. Blackwell Scientific Publishing, Oxford
14. Glover F (1986) Future paths for integer programming and links to artificial intelligence. Comput Oper Res 13:533–549
15. Pradhan S, Lam SSY (2007) Minimizing makespan during environmental stress screening using a genetic algorithm and an ant colony optimization. Int J Adv Manuf Technol 32:571–577
16. Baskar N, Asokan P, Saravanan R, Prabhaharan G (2005) Optimization of machining parameters for milling operations using non-conventional methods. Int J Adv Manuf Technol 25:1078–1088
17. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680
18. Johnson DS, Aragon CR, McGeoch LA, Schevon C (1989) Optimization by simulated annealing: an experimental evaluation: part I. Graph partitioning. Oper Res 37:865–892
19. Johnson DS, Aragon CR, McGeoch LA, Schevon C (1991) Optimization by simulated annealing: an experimental evaluation: part II. Graph coloring and number partitioning. Oper Res 39:378–406
20. Holland JH (1975) Adaption in natural and artificial systems. The University of Michigan Press, Ann Harbor, MI
21. Dorigo M, Gambardella LM (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Comput 1:53–66
22. Fox B, Xiang W, Lee HP (2007) Industrial applications of the ant colony optimization algorithm. Int J Adv Manuf Technol 31:805–814
23. Glover F, Laguna M (1997) Tabu search. Kluwer Academic Publishers, Dordrecht
24. Webster S, Jog PD, Gupta A (1998) A genetic algorithm for scheduling job families on a single machine with arbitrary earliness/tardiness penalties and an unrestricted common due date. Int J Prod Res 36:2543–2551
25. Davids L (1985) Applying adaptive algorithms to epistatic domains. Proceedings of the Ninth International Joint Conference on Artificial Intelligence, pp 162–164