List-Based Threshold-Accepting Algorithm for Zero-Wait Scheduling of Multiproduct Batch Plants

ARTICLE *in* INDUSTRIAL & ENGINEERING CHEMISTRY RESEARCH · NOVEMBER 2002

Impact Factor: 2.59 · DOI: 10.1021/ie010570n

3 AUTHORS, INCLUDING:

Jong Moon Park
Pohang University of Science and Technology
216 PUBLICATIONS **5,764** CITATIONS

List-Based Threshold-Accepting Algorithm for Zero-Wait Scheduling of Multiproduct Batch Plants

Dae Sung Lee,[†] Vassilios S. Vassiliadis,*[,†] and Jong Moon Park[‡]

Department of Chemical Engineering, University of Cambridge, Pembroke Street, Cambridge CB2 3RA, United Kingdom, and School of Environmental Science and Engineering, Department of Chemical Engineering, POSTECH, San 31, Hyoja-dong, Nam-Gu, Pohang, Kyungbuk, Korea 790-784

A novel list-based threshold-accepting (LBTA) algorithm is proposed for solving the zero-wait (ZW) scheduling problem. The LBTA algorithm belongs to the class of threshold-accepting algorithms, but the acceptance probability decreases based on a list that is rejuvenated and adapted according to the topology of the solution space of the problem. A probabilistic steepest optimization strategy was adapted to search the solution space effectively. The effectiveness of the LBTA method is illustrated through case studies from scheduling literature which were formulated as mixed-integer linear programming and mixed-integer nonlinear programming models. The performance of the LBTA algorithm is also compared with that of the simulated annealing (SA) algorithm for a large number of various problem sizes. The proposed algorithm gives optimal solutions for small- to moderate-size ZW scheduling problems within a very short time and shows much superior computational performance compared to SA for large-size problems.

Introduction

During the past decades, batch scheduling has been recognized as one of the most important operational planning issues in the chemical batch process industry. Because of the inherent flexibility of batch processes, their own productivities and economic effectiveness depend critically on the plant production schedule. Batch scheduling is the determination of production orders to units as well as timing of operations so as to optimize a specific performance criterion. In numerous special cases of the scheduling problem, this paper deals with the zero-wait (ZW) scheduling of multiproduct batch plants.

A ZW scheduling problem occurs in a production environment in which a product must be processed until completion without any interruption either on or between processing units. This situation is encountered in a variety of industrial processes, for example, in chemical and pharmaceutical industries where a series of processes must follow one another immediately owing to the chemical instability of intermediate products.

The ZW scheduling belongs to the NP-complete class of problems (Papadimitriou and Kanellakis, 1980), for which a large number of solution methods have been proposed (Hall and Sriskandarajah, 1996). The ZW scheduling problem can be formulated as an asymmetrical traveling salesman problem (TSP) (Reddi and Ramamoorthy, 1972; Wismer, 1972) and may be solved by a considerable body of optimization algorithms for the TSP. Pekny and Miller (1991) applied an exact branch and bound algorithm for the TSP to solve the ZW scheduling problem. The optimization algorithms, chiefly based on the branch and bound (Van Deman et al., 1974; Pekny and Miller, 1991), integer programming approach (Birewar and Grossmann, 1989; Jung et al., 1994; Moon et al., 1996), are always desirable because they can give a global solution. However, the disadvantage of these methods is that the required computation time increases exponentially or as a high degree polynomial for a linear increase in problem size.

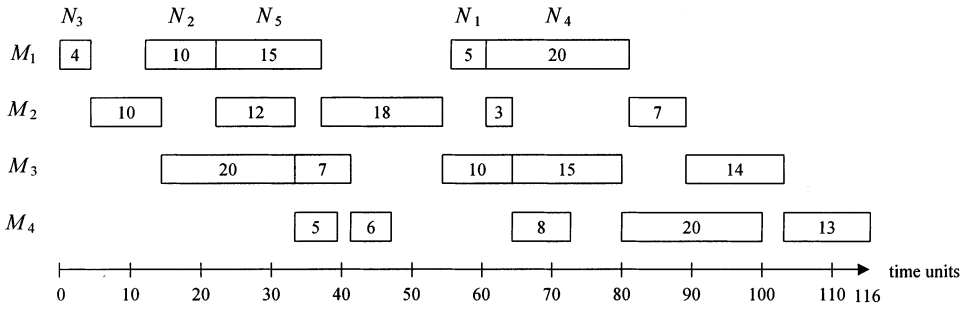
On the other hand, approximation methods, although they do not guarantee a globally optimal solution, can provide near optimal solutions within moderate computing times. Therefore, the approximation algorithms are more suitable for larger instances, particularly in practical applications of industrial interest. The approximation algorithms, especially iterated local search methods such as simulated annealing (SA) (Das et al., 1990) and genetic algorithms (GA) (Chen et al., 1996), have been successfully used to obtain good improving solutions. However, one of the main difficulties in applying these techniques is that many choices have to be made concerning parameter values, and tuning, which are not trivial and are quite often very poorly made by trial and error.

Recently, a meta-heuristic algorithm called the list-based threshold-accepting (LBTA) algorithm has been developed and has shown significant promise for computing improved solutions to combinatorial optimization problems that are NP-complete (Tarantilis et al., 2000). The advantage of LBTA over the majority of other neighborhood search-based meta-heuristic methods is that it is characterized by fewer user-controlled parameters that have to be tuned in order to produce the best possible solution.

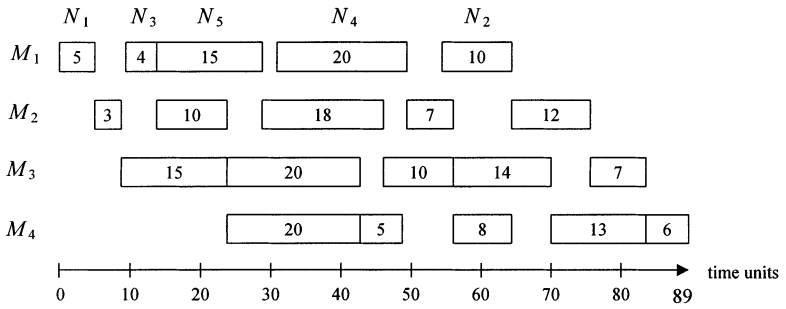
In this paper, we investigate the potential of a meta-heuristic approach based on the easy to implement LBTA algorithm for the problems of ZW scheduling. The performance of the proposed algorithms is examined by solving two examples in the literature that solved by mixed-integer linear programming (MILP) and mixed-integer nonlinear programming (MINLP). We

* To whom all correspondence should be addressed. Fax: +44-1223-330142. E-mail: vsv20@cheng.cam.ac.uk.
[†] University of Cambridge.
[‡] POSTECH.

10.1021/ie010570n CCC: $22.00
Published on Web 11/13/2002



(a) A feasible solution with makespan 116: N_3 - N_2 - N_5 - N_1 - N_4.



(b) The optimal solution with makespan 89: N_1 - N_3 - N_5 - N_4 - N_2.

Figure 1. Gantt charts of the instance in Table 1.

Table 1. Processing Times of a ZW Scheduling Instance

	units			
products	M_1	M_2	M_3	M_4
N_1	5	3	15	20
N_2	10	12	7	6
N_3	4	10	20	5
N_4	20	7	14	13
N_5	15	18	10	8

also show by computational experiments that the LBTA algorithm performs much better than the simulated annealing algorithm for randomly generated large-size problems. Qualitative arguments, used for its development and interpreting its subsequent good performance, will be highlighted in later sections of this paper.

ZW Scheduling Problem Description

A multiproduct batch plant with M batch processing units has N products to be processed. The processing time τ_{ij} of product i (i = 1, 2,..., N) on unit j (j= 1,2,..., M) is specified. Each product consists of a predetermined sequence of processing units, and once started, the processing of a product on a unit may not be interrupted. In ZW processing strategy, all products must be transferred immediately from unit j to the next unit (j + 1). The objective is then to find a sequence of batches of products that requires the shortest total time for producing all products, the so-called makespan. Table 1 provides processing times of a ZW scheduling problem, consisting of five products and four units. The times to transfer batches between units and to set up units when processing different products are assumed negligible here. The Gantt chart is a convenient way of representing visually a solution of the ZW scheduling problem. An example of a feasible solution and the optimal solution for the 5 × 4 problem are shown in Figure 1.

List-Based Threshold-Accepting Algorithm

During the past decade, many different types of local search algorithms for scheduling have been developed. The main variants can be divided into threshold algorithms (including simulated annealing), taboo search algorithms, and genetic algorithms, and some of them have proved to be very effective for scheduling problems (Vaessens et al., 1996). To implement such an algorithm to solve a scheduling problem, it is necessary to define a finite set S of feasible configurations, a cost function $C(\cdot)$: $S \rightarrow \mathbb{R}$, and a move function $N(\cdot)$: $S \rightarrow 2^S$, which is a subset of the solution space S and defines for each feasible configuration s_f. $N(s)$ is a function which chooses a simple transition from a solution s to another solution by inducing a small perturbation and then defining the neighborhood as the set of solutions which can be obtained from a given one by a single transition. $N(\cdot)$ must possess the property that its repeated application can take any starting configuration, s_o, to any other feasible configuration, s_f, thus being able to visit in theory any configuration in the solution space. Roughly speaking, the strategy of local search is to progressively perturb the current configuration through a succession of neighbors in order to direct the search to an improved solution, starting from an initial feasible solution.

The list-based threshold-accepting (LBTA) algorithm is a new meta-heuristic technique belonging to the class of threshold-accepting-based algorithms. The innovation of the LBTA algorithm over a typical threshold-accepting algorithm, such as simulated annealing (SA) and threshold algorithms, is based on the fact that the threshold values used in the implementation of the move acceptance criterion are normalized, and they are determined by a list that is rejuvenated and adapted according to the topology of the solution space of the problem. The introduction of the list in the concept of the LBTA algorithm adds to it two main advantagesover a typical threshold-accepting and other types of algorithms belonging to this general class, including SA: (1)

The search parameters involved in the threshold reduction strategy, such as the initial value of threshold and the percentage of the threshold reduction, are determined by the algorithm automatically, without the intervention of the user; (2) The normalization of the threshold values stored in the list of the algorithm and the implementation of the "conservation of threshold" criterion (see algorithm description below) during the optimization process help this new algorithm to accept more easily cost-increasing neighboring solutions of a current solution, facilitating escape from local minimum points, thus increasing the possibility of finding a better solution, even the global one

The construction of the topology of the solution space depends on the way the selection method of a neighbor from a current solution s is made. Specifically, in local search methods, the selection of a neighbor is dictated by the choice criteria. Several common selection strategies include the following: (1) Selecting one of cost-decreasing stochastically selected neighbors (probabilistic local optimization); this is applied in threshold algorithms; (2) Choosing the most cost-decreasing neighbor in the entire neighborhood (steepest descent − greedy algorithm approach); this is done by the shifting bottleneck procedure; (3) Choosing the most cost-decreasing neighbor from a number of neighbors selected stochastically among all the neighbors of the current solution (probabilistic steepest decent); this is the selection procedure applied in genetic algorithms.

Originally, the LBTA algorithm was proposed as a probabilistic local optimization meta-heuristic algorithm. However, the choice of good neighbors is often difficult by choosing the first lower cost neighbor found. With this move selection method, the LBTA methodology might spend a high proportion of the search in unattractive regions of the solution space and finally be trapped in a poor local minimum solution. If the best neighbor is selected in the entire neighborhood, the probability of finding a good neighbor may be higher, but its identification takes much time. Therefore, we adapted the probabilistic steepest neighbor selection strategy to the original LBTA method to compromise between searching time and neighborhood quality. We denote this modified LBTA algorithm as LBTA′. To direct the search to an improved solution effectively, the LBTA′ algorithm chooses the best of a sample of stochastically selected neighbors rather than choosing one of its cost-decreasing selected neighbors in the original algorithm. The LBTA algorithm is a two-phase algorithm. In the first phase, a list filling procedure with threshold values is conducted. Details of the algorithm follow.

PHASE 1 (A List Filling Procedure with Threshold Values)

In this phase, after a feasible solution is produced, local search is conducted in order to compute the threshold values (changes in the objective function value) that will initialize the list of the algorithm. Local search uses a blend of moves. The selection of the type of the move is done stochastically, generated by a stochastic move generation algorithm. The main property of these moves is that they are "biased uniformly distributed", that is to say, their selection follows the uniform distribution, but the percentage of the usage of each can be changed according to the user. Local search starts as follows: (1) A neighboring solution s' of a current solution s is selected by using one of the local search moves. (2) The change in objective function value, $C(s') - C(s)$, is calculated and then is normalized as

$$\Delta = C(s') - C(s)/C(s) \qquad (1)$$

If $\Delta > 0$, then $T_h = \Delta$, and the normalized threshold value is inserted in the list, otherwise no threshold value is set in the list. (3) The same procedure is repeated for each of the moves mentioned in the previous section, until the list of threshold values is filled; the only controlling parameter of LBTA being, effectively, the list size.

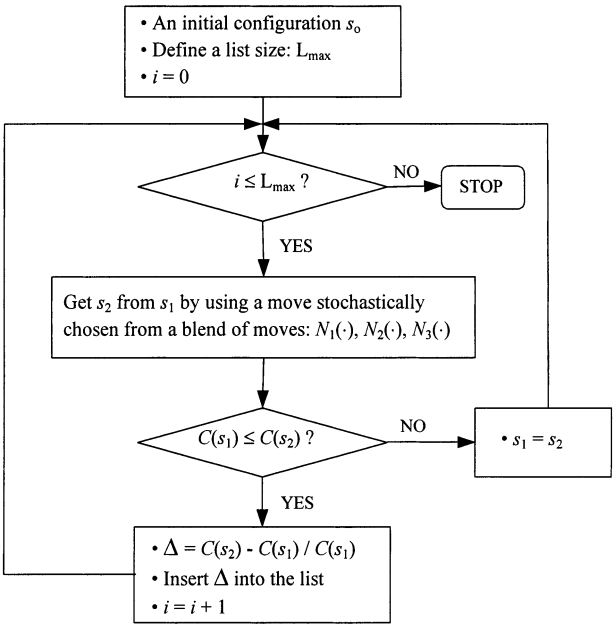


Figure 2. Flowchart of the list filling algorithm.

The flowchart of a list-construction procedure is shown in Figure 2. The list serves as a memory of the variability of local function values, stored as normalized value changes from each old configuration. The storage scheme employs up to L_{max} (user defined parameter) values in a binary tree list. Storage and retrieval from such a scheme have logarithmic complexity (for a perfectly balanced binary tree), as opposed to linear complexity per operation in a simple linear-storage scheme.

PHASE 2 (Main Optimization Algorithm)

In this phase, the main optimization algorithm is described and a threshold controlling procedure is conducted in the list of the algorithm. The local search method is applied as follows: (1) An initial solution is produced in the same way as in Phase 1. (2) Using one of the local search moves selected stochastically, a neighboring solution s' of a current solution s is produced. (3) The acceptance condition given by the following simple formula

$$\Delta = [C(s') - C(s)]/C(s) < T_{hmax} \qquad (2)$$

is checked for satisfaction, where T_{hmax} is the largest threshold value stored in the list. (4) If it is satisfied, then s is set to s', and the stored value of T_{hmax} is rejected from the list, with the new value of Δ being appropriately inserted, otherwise the proposed new con-

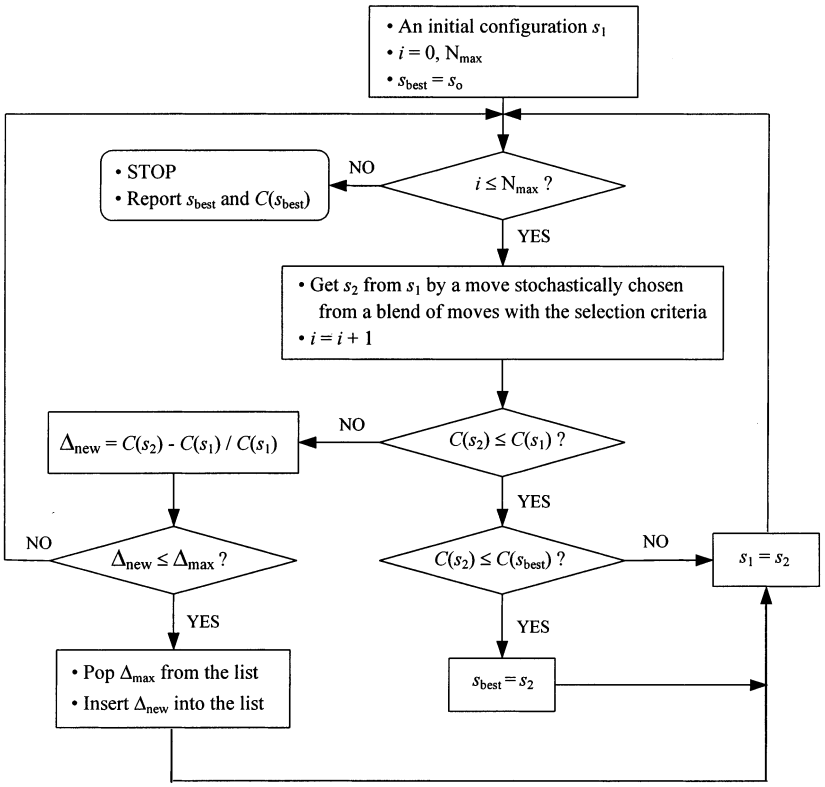


Figure 3. Flowchart of the main LBTA algorithm.

figuration is rejected. (5) The same procedure is repeated for each of the above configurations produced, until no value of Δ is found to be better than the highest value of threshold T_{hmax}, for a number of feasible moves (stopping criterion).

The detailed flowchart of the main optimization search algorithm is shown in Figure 3. To facilitate escape from local minimum points, and increase the possibility of finding a better one, the algorithm attempts to keep the values of threshold of the list high by placing in the list the highest of the normalized threshold values found for every 10 feasible moves. This strategy is termed as conservation of threshold, representing a delay factor in the list values adaptation. Together with the list size, they comprise the main tuning parameters of the LBTA.

Application of the LBTA Algorithms to the ZW Scheduling Problem

To implement the LBTA algorithms to the ZW scheduling problems, some problem-specific principles such as the initial solution, move, cost function, and user-controlled tuning parameters have to be defined. Hereinafter, these items are discussed in more detail.

Initial Solution. The application of local search heuristics requires the construction of initial feasible solutions. The initial solution can be generated in a number of different ways such as the identity permutation, a random permutation, or some heuristically determined starting solution such as the insertion technique. The methods selected to initialize the LBTA algorithms are a modification of the insertion algorithm of Fink and Voss (1999). The proposed insertion technique starts with the product that has the largest total processing time and ranking the remaining products in order of decreasing processing time. Then we build successively a complete permutation by choosing in each iteration $k = 2, ..., N$ the best combination under consideration of the remaining $N - k - 1$ products and all k insertion positions.

Cost Function. Let C_{ij} denote the completion time at which the ith product in a feasible solution s finishes processing unit j. To compute C_{ij}, we should consider both the time at which the ith product finishes processing on the previous unit $C_{i(j-1)}$ and the time at which the $(i-1)$th product leaves unit j. Because any consecutive processing units should be processed without any delay under the ZW strategy, the consecutive $(i-1)$th and ith products should be processed immediately at least at one of the units j ($j = 1, 2, ..., M$) in order to minimize makespan.

Given a feasible sequence s, the completion time C_{ij} is obtained recursively by the following equations:

$$C_{1j} = \sum_{k=1}^{j} \tau_{1j} \qquad j = 1, ..., M \tag{3}$$

$$C_{iM} = \max_{j=1,...,M} [C_{(i-1)j} + \sum_{k=j}^{M} \tau_{ik}] \qquad i = 2, ..., N \tag{4}$$

$$C_{ij} = C_{iM} - \sum_{k=j}^{M} \tau_{ij} \qquad i = 2, ..., N \qquad j = 1, ..., M-1 \tag{5}$$

The length, or the makespan, of a feasible schedule s is defined by $C_{NM}(s)$. Therefore, the problem is to find an optimal schedule, s^*, i.e., a feasible schedule of minimum length:

$$s^* = \arg\min_{s \in S} C_{NM}(s) \tag{6}$$

Move Functions. A crucial ingredient of the LBTA algorithm is the definition of a feasible move function. Several move functions have been proposed in the literature; the implementation of LBTA is based on a blend of three different kinds of move functions: $N_1(s)$, $N_2(s)$, and $N_3(s)$. Specifically, the move function used in our implementation randomly selects among three move functions. $N_1(s)$ is a pairwise exchange move that selects a pair of consecutive products and exchanges their order. $N_2(s)$ is a shift move that takes a product from its current position and reinserts it to another position. $N_3(s)$ is a swap move that selects randomly two products and exchanges their position. All of these moves together have the ability to reach iteratively any point in the configuration space.

User Controlled Parameters. Stochastic metaheuristic algorithms are governed by several user-controlled parameters that express the generic decisions that have to be made when implementing them. The advantage of LBTA over the majority of these methods is that it is characterized by fewer user-controlled parameters that have to be tuned in order to produce the best possible solution. The LBTA effectively depends on the size of the list only, because the other generic decisions are made by the LBTA itself. For a given list size, our LBTA implementation produced its best solutions over the benchmark instances, using the following standard settings: (1) The percentage of $N_1(s)$ moves is set equal to 40%; (2) The percentage of $N_2(s)$ moves is set equal to 30%; (3) The percentage of the $N_3(s)$ moves is set equal to 30%; (4) The number of l samples of neighbors in LBTA′ algorithm is set equal to 4; (5) The number of maximum evaluations, $N_{\max}$, in the LBTA and LBTA′ are set to $3N^3$ and N^3, respectively.

It is also interesting to note that the LBTA is comprised of a single loop, merging thus the so-called inner and outer loops of SA and other threshold acceptance (TA) methods.

Computational Experience

The applicability and effectiveness of the proposed approach for the ZW scheduling problems is illustrated with the case studies presented in this section. The LBTA algorithms have been implemented in C++ on a personal Pentium III computer (500 MHz).

Case Study 1: Birewar and Grossmann (1989). This is a modest-sized scheduling problem where a multiproduct batch plant with four units (stages) processes 30 batches ($N = 30$) belonging to six different products (Np = 6), A − F. A schedule is to be determined to manufacture five batches of A, seven of B, three of C, five of D, four of E, and six of F. The batch processing times are shown in Appendix I. Birewar and Grossmann proposed an approximate solution method to solve the problems for productions for single product campaign (SPC) and mixed product campaign (MPC). Moon et al. (1996) also presented mixed-integer linear programming (MILP) models to solve the same scheduling problems.

For MPC production, the LBTA algorithms can be easily applied with the completion time calculated by eqs 3−5. The solution of SPC can be obtained by specifying that the optimal sequence will consist of one campaign per product. To compute the completions times for SPC, let τ'_i be the SPC times defined as follows:

$$\tau'_i = \max_{j=1,\ldots,M} [(n_i - 1)\tau_{ij}] \qquad i = 1, \ldots, N_p \tag{7}$$

Then, the completion time for the SPC scheduling problem can be computed through the following recursive equations:

$$C_{1j} = \sum_{k=1}^{j} \tau_{1k} + \tau'_i \qquad j = 1, \ldots, M \tag{8}$$

$$C_{ij} = C_{i(j-1)} + \tau_{ij} \qquad i = 2, \ldots, N; \quad j = 1, \ldots, M \tag{9}$$

$$C_{iM} = \max_{j=1,\ldots,M} [C_{(i-1)j} + \tau'_i + \sum_{k=j}^{M} \tau_{ik}] \qquad i = 2, \ldots, N \tag{10}$$

where $C_{ij} = 0$, if $j = 0$.

The LBTA algorithm has been applied to both the MPC and SPC scheduling problems. The comparison of the makespan and computational time by different approaches is shown in Table 2. The globally optimal solutions for MPC and SPC were reproduced as being 145 and 177 h, respectively, as reported in the references cited. All three methods give global minimum solutions for both SPC and MPC scheduling problems. The standard deviations of LBTA results in the table are computed from five runs on the same problem. The probabilistic nature of the algorithm makes it necessary to carry out multiple runs on the same problem instance in order to get meaningful results. All of the five solutions obtained for both MPC and SPC problems give the same global values. Figure 4 shows the Gantt charts for both the SPC and MPC scheduling problems. In comparison with both the approximate MILP models by Birewar and Grossmann (1989) and the MILP models by Moon et al. (1996), the LBTA algorithm gives alternative solutions but with the same value of the global optimal solutions. These results show that the LBTA algorithm can be effectively used to find the global solutions with makespan minimization for small-sized ZW scheduling problems. Of course, being a stochastic search method, no guarantee of global optimality can be provided, other than a statistical measure of the quality of the solutions derived.

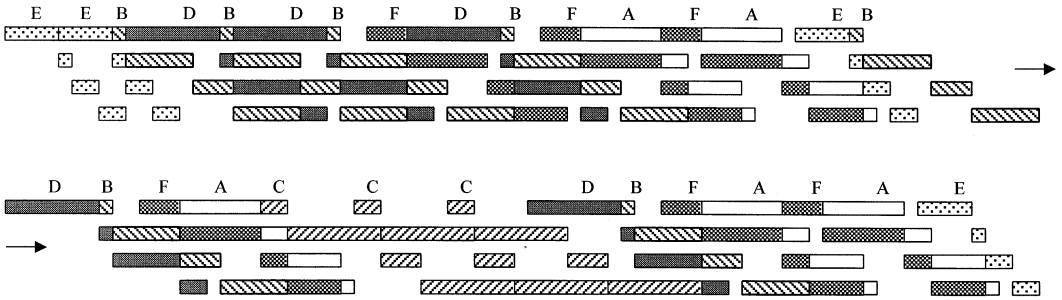
Case Study 2: Kim et al. (1996). In this example, a multiproduct plant of four processing units produces eight products with different types of intermediate storage policies such as ZW, unlimited intermediate storage (UIS), no intermediate storage (NIS), and finite intermediate storage (FIS). Nonzero transfer times and sequence-dependent setup times are considered. Kim et al. (1996) presented mixed integer−nonlinear programming (MINLP) models for each storage policy in order to get minimum makespan.

To implement the LBTA algorithm to the scheduling problems with nonzero transfer times and setup times, we first redefine completion time C_{ij} to be the time at which the product i finishes processing on unit j and transferred out from it. We will denote this new completion time by C'_{ij}. Then, given fixed transfer times T_{ij} for transferring product i out of unit j to $(j + 1)$ and setup times $S_{i(i+1)j}$ for preparing unit j between the ith product and the $(i + 1)$th product, the completion times

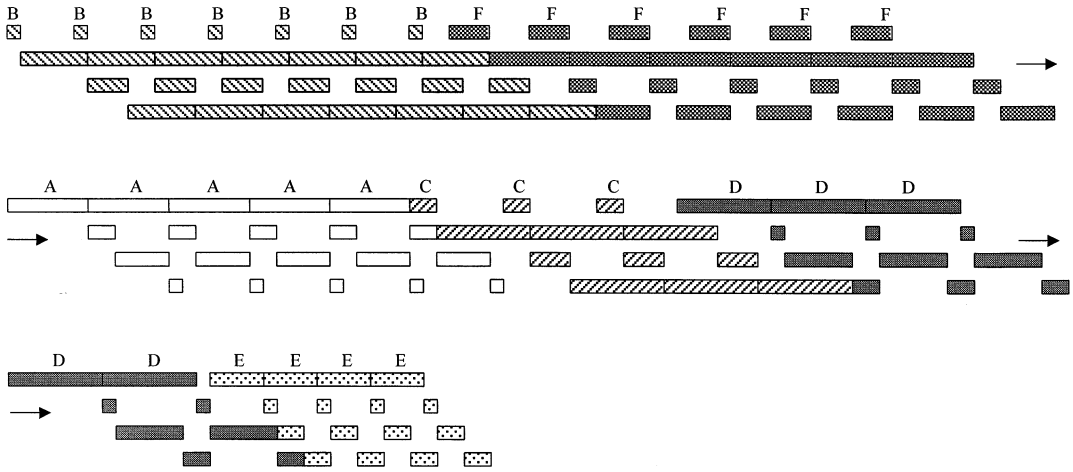
Table 2. Computational Results for the Case Study 1

	global makespan solution (h)	approximate MILP models Bire and Grossmann (1989)		MILP models Moon et al. (1996)[a]		LBTA algorithms			
		C_{MILPa}	t_{MILPa} (s)	C_{MILP}	t_{MILP} (s)	L_{max}	C_{LBTA}	T_{LBTA} (s)	s_c[b]
SPC	177	177	–	177	2.44	100	177	0.03	0.0
MPC	145	145	–	145	3.05	200	145	37.85	0.0

[a] The CPU time on a SUN/SPARC. [b] s_c: standard deviation over five runs.



(a) Optimal sequence for mixed product campaigns (makespan: 145 hrs).



(b) Optimal sequence for single product campaigns (makespan: 177 hrs).

A: B: C: D: E: F:

Figure 4. Optimal schedules for both MPC and SPC.

can be obtained from the following equations:

$$C'_{1j} = \sum_{k=1}^{j} \tau_{ik} + \sum_{k=0}^{j} T_{ik} \qquad j = 1, ..., M \tag{11}$$

$$C'_{iM} = \max_{j=1,...,M} [C'_{(i-1)j} + S_{i(i-1)j} + \sum_{k=j}^{M} \tau_{ik} + \sum_{k=j-1}^{M} T_{ik}] \quad i = 2, ..., N \tag{12}$$

$$C'_{ij} = C'_{iM} - \sum_{k=j}^{M} \tau_{ik} - \sum_{k=j}^{M} T_{ik} \quad i = 2, ..., N; j = 1, ..., M-1 \tag{13}$$

A Gantt chart of ZW policy for a four-product and three-unit instance is shown in Figure 5.

In UIS policy, at least $(N - 1)$ intermediate storage units are available between every two adjacent batch units. That means that a storage unit is always available whenever a product batch needs it. Under NIS policy, the intermediate storage is not used in the batch process plant like ZW mode but a batch, after its completion in a processing unit, may be held temporarily in the unit. For FIS policy, finite storage tanks are available between batch processing units. In the case of the FIS system, it is assumed that one storage tank is available between unit three and unit four. The computational time algorithms for UIS, NIS, and FIS policies were developed (Kim et al., 1996) and are presented in Appendix III.

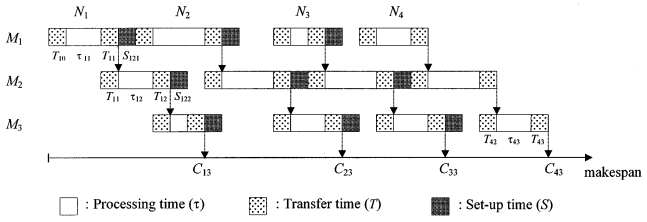


Figure 5. Gantt charts of ZW policy with nonzero transfer times and setup times.

Table 3 shows the computational results of MINLP models and the LBTA algorithm. *The LBTA algorithms give the same optimal makespan values with the MINLP*

Table 3. Computational Results for the Case Study 2

	MINLP (Kim et al., 1996)			LBTA			
	optimal sequence	C_{MINLP}	t_{MINLP} (s)[a]	L_{max}	optimal sequence	C_{LBTA}	t_{LBTA} (s)
UIS	$N_5-N_7-N_1-N_2-N_6-N_8-N_4-N_3$	173	120.68	100	$N_7-N_4-N_5-N_1-N_2-N_6-N_3-N_8$	173	0.17
FIS	$N_5-N_7-N_4-N_1-N_3-N_6-N_2-N_8$	178	201.34	350	$N_5-N_7-N_4-N_1-N_3-N_6-N_2-N_8$	178	0.19
NIS	$N_5-N_7-N_2-N_4-N_1-N_6-N_8-N_3$	185	154.55	200	$N_5-N_7-N_2-N_4-N_1-N_6-N_8-N_3$	185	0.100
ZW	$N_5-N_1-N_6-N_3-N_7-N_4-N_2-N_8$	195	203.93	300	$N_5-N_1-N_6-N_8-N_7-N_4-N_2-N_3$	195	0.19

[a] The CPU time on a IBM RS/6000.

models within extraordinarily short computational times (less than 0.2 s). The LBTA method gives the same optimal sequence with MINLP models for FIS and NIS policies but alternative optimal solutions for UIS and ZW policies.

Case Study 3: Randomly Generated Test Problems. In this section, we present computational experiments using randomly generated test problems of a wide range of products and units. We tested two versions of the LBTA algorithms on the ZW scheduling problems: LBTA with the probabilistic local optimization method and LBTA′ with the probabilistic steepest descent method. The computational performance of the LBTA algorithms is also compared with that of SA algorithm.

Simulated Annealing. Simulated annealing (SA) has been by far the most popular threshold algorithm and has been applied successfully to scheduling problems (Ku and Karimi, 1991; Raaymakers and Hoogeveen, 2000). SA is a random local search technique that was introduced as an analogy from statistical physics of the computer simulation of the annealing process of a hot metal until its minimum energy state is reached. In SA, the thresholds are positive and stochastic. SA accepts all better solutions and will accept inferior solutions with some probability, decreasing exponentially as the change in the objective function increases.

We applied the SA algorithm with the Metropolis algorithm for move acceptance and the exponential annealing schedule to the ZW scheduling problems. The Metropolis algorithm is the simplest and has been used the most (Aarts and Korst, 1989). The initial temperature is chosen to be about 10 times the maximum value of ΔC between two consecutive configurations when all moves are accepted. The temperature is decreased by multiplying it by a cooling factor, α, where $0 < \alpha < 1$. After trying a number of different values of α in the annealing schedule ranging from 0.6 to 0.98, the best value was selected for each test problem. On the basis of the experiment done with different combinations of the moves, $N_1(s)$, $N_2(s)$, and $N_3(s)$, the stochastic mixture of three move functions was found to be the most appropriate. The number of attempted moves at each temperature level is set equal to 0.05 times the maximum number of iterations (comprising the inner search loop for fixed pseudotemperature). The number of maximum evaluations was set equal to $5N^3$ in the SA algorithm. The algorithm is made to terminate after a maximum number of moves are evaluated.

Small-Size Problems. A total of 450 small-size problems were generated for 15 different combinations of products ranging from 6 to 8 with the number of processing units ranging from 5 to 25. The processing times were randomly generated from a uniform distribution ranging from 1 to 99. These problems were solved by both LBTA and SA algorithms. All of the LBTA results are obtained only by varying the size of the list ranging from 100 to 5000 elements. Using the optimal solutions obtained by complete enumeration, we used the mean deviation from the optimum as the criteria for evaluation. Table 4 presents the computational results of the LBTA and SA algorithms on the test bed. Both the LBTA and SA methods give optimal solutions for all of the problems within a very short computational time.

Table 4. Performance of SA and LBTA Algorithm for Small-Size ZW Scheduling Problems

			SA		LBTA	
N	M	number of problems	mean dev. from optimal [a]	mean CPU time (s)	mean dev. from optimal	mean CPU time (s)
6	5	30	0.0	0.05	0.0	0.05
6	10	30	0.0	0.05	0.0	0.05
6	15	30	0.0	0.05	0.0	0.05
6	20	30	0.0	0.05	0.0	0.05
6	25	30	0.0	0.05	0.0	0.05
7	5	30	0.0	0.05	0.0	0.05
7	10	30	0.0	0.06	0.0	0.05
7	15	30	0.0	0.07	0.0	0.05
7	20	30	0.0	0.11	0.0	0.05
7	25	30	0.0	0.13	0.0	0.06
8	5	30	0.0	0.05	0.0	0.05
8	10	30	0.0	0.06	0.0	0.05
8	15	30	0.0	0.11	0.0	0.05
8	20	30	0.0	0.16	0.0	0.05
8	25	30	0.0	0.21	0.0	0.06

[a] Dev. from optimal = 100[(algorithm makespan − optimal makespan)/optimal makespan].

Large-Size Problems. Computational experiments were carried out for 320 large-size problems with 10, 20, 30, 50, and 100 products and the number of units ranging from 5 to 20. The processing times were, again, randomly sampled from a uniform distribution ranging from 1 to 99. These problems were solved by SA, LBTA, and the LBTA′ algorithm. The results of LBTA and LBTA′ are obtained only by varying the size of the list ranging from 1000 to 250 000 elements depending on the problem sizes.

For large-size problems, we cannot expect to find globally optimal solutions within a reasonable time, because the ZW multiproduct scheduling problems is NP-complete in the strong sense. Therefore, relative deviations and proportions of best solutions are used as the criteria for evaluation (Ku and Karimi, 1991). Relative deviation is calculated with respect to the best of solutions obtained from the SA, LBTA, and LBTA′ algorithms. The proportion of best solutions is obtained as the percentage of problems for which each algorithm gives the best solution. The comparative results in Table 5 clearly show the superiority of the LBTA′ algorithm over SA and LBTA. The performance of LBTA′ is far better than that of both SA and LBTA in the proportions of best solutions. The SA algorithm shows very poor results for large-size problems even though it gives optimal solutions for the small-size problems. As the problem size increases, the SA algorithm seems to spend much searching time in unattractive regionsand not transcending poor local minima. The LBTA algorithm

Table 5. Performance of SA, LBTA, and LBTA′ Algorithms for Large-Size ZW Scheduling Problems

		number of	proportion best (%) [a]			mean dev. from best (%)[b]			mean CPU time (s)		
N	*M*	problems	SA	LBTA	LBTA′	SA	LBTA	LBTA′	SA	LBTA	LBTA′
10	5	20	100	100	100	0.0	0.0	0.0	0.05	0.06	0.05
10	10	20	100	100	100	0.0	0.0	0.0	0.17	0.22	0.15
20	5	20	75	90	100	0.09	0.05	0.0	0.82	1.04	0.65
20	10	20	40	75	100	0.42	0.46	0.0	2.18	3.75	2.00
20	15	20	5	10	100	0.33	0.23	0.0	3.41	5.67	3.20
20	20	20	0	0	100	0.51	0.43	0.0	5.23	7.36	4.50
30	5	20	0	10	100	0.30	0.31	0.0	3.84	4.86	3.46
30	10	20	0	0	100	0.38	0.71	0.0	9.67	12.53	9.61
30	15	20	0	0	100	0.68	0.56	0.0	18.86	22.25	17.19
30	20	20	0	0	100	0.46	0.38	0.0	28.20	34.93	26.85
50	5	20	0	10	100	0.39	0.32	0.0	32.95	37.46	28.00
50	10	20	0	5	100	0.83	0.53	0.0	97.88	110.73	74.64
50	15	20	0	0	100	1.20	1.38	0.0	160.95	185.17	151.64
50	20	20	0	0	100	1.58	1.91	0.0	234.56	260.94	207.41
100	5	20	0	0	100	1.84	1.53	0.0	543.59	580.63	493.12
100	10	20	0	0	100	2.18	1.85	0.0	1579.42	1611.24	1431.58

[a] Proportion best (%) = 100(number of best solutions/number of test problems). [b] Dev. from best (%) = 100[(algorithm makespan − best makespan)/best makespan.]

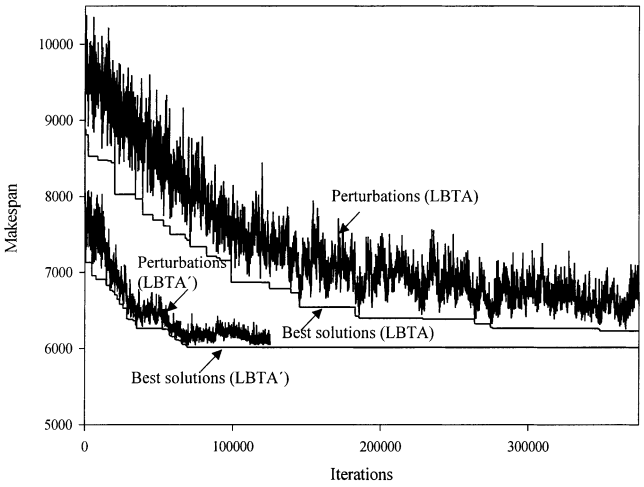


Figure 6. Perturbation and best makespan values produced by the LBTA algorithms for a 50 × 20 problem.

Table 6. Process Data for the Case Study 1 (Birewar and Grossmann, 1989)

	units				
products	M_1	M_2	M_3	M_4	number of products (n_i)
N_{p1}	6	2	4	1	5
Np_2	1	5	3	5	7
N_{p3}	2	7	3	7	3
N_{p4}	8	1	5	2	5
N_{p5}	4	1	2	2	4
N_{p6}	3	6	2	4	6

is not competitive to the SA algorithm. The LBTA algorithm with the selection criteria of choosing the first lower cost neighbor found shows only slightly better performance compared to the SA algorithm but requires more computational time than SA. However, the strategy of selecting the best move from a sample of the neighborhood makes the LBTA′ algorithm to navigate to more attractive regions in the search space, thereby generating better and faster solutions. All of the best solutions were found by the LBTA′ algorithm. Figure 6 shows a comparison of typical profiles of makespan values with the LBTA and LBTA′ algorithms. As can be seen, the LBTA and LBTA′ algorithms seem to follow quite different search paths. For the LBTA algorithm, the makespan values exceed quite often the initial value (=9236), with high amplitude, during the perturbation process. On the other hand, the best makespan of the LBTA′ algorithm decreases very rapidly at the early stage of perturbation and gives better values than the LBTA algorithm throughout perturbations.

Table 7. Process Data for the Case Study 2 (Kim et al., 1996)

(a) Processing Times and Transfer times

	processing times (τ_{ij}) for units				transfer times (T_{ij}) for units				
prod	M_1	M_2	M_3	M_4	M_0	M_1	M_2	M_3	M_4
N_1	10	20	5	30	2	2	2	2	3
N_2	15	8	12	10	3	3	3	3	1
N_3	20	7	9	5	2	4	2	2	1
N_4	13	7	17	10	2	2	1	4	2
N_5	8	3	16	7	1	1	1	3	2
N_6	6	9	22	7	3	3	2	2	2
N_7	7	5	15	12	1	2	1	2	1
N_8	4	13	6	4	2	4	1	1	2

(b) Setup Times for Units and Storages

prod seq	unit				storage	prod seq	unit				storage
	M_1	M_2	M_3	M_4			M_1	M_2	M_3	M_4	
S(1−2)	3	1	2	4	2	S(5−1)	2	2	1	4	2
S(1−3)	2	2	1	3	2	S(5−2)	2	2	3	2	1
S(1−4)	1	4	2	2	2	S(5−3)	2	3	3	3	2
S(1−5)	1	3	2	3	1	S(5−4)	1	1	3	4	1
S(1−6)	3	2	1	1	1	S(5−6)	4	3	3	1	1
S(1−7)	3	1	2	1	2	S(5−7)	2	3	2	3	1
S(1−8)	2	3	3	2	1	S(5−8)	4	3	2	2	2
S(2−1)	4	1	2	3	1	S(6−1)	1	2	1	2	2
S(2−3)	1	1	4	3	2	S(6−2)	3	3	2	3	1
S(2−4)	3	2	3	2	1	S(6−3)	4	1	1	2	1
S(2−5)	3	1	2	2	1	S(6−4)	2	3	4	1	2
S(2−6)	1	1	2	4	1	S(6−5)	2	1	1	4	2
S(2−7)	2	2	1	1	2	S(6−7)	2	2	3	2	2
S(2−8)	3	2	1	3	2	S(6−8)	1	3	3	3	2
S(3−1)	2	1	4	3	2	S(7−1)	2	3	2	1	2
S(3−2)	1	2	3	2	3	S(7−2)	1	1	1	1	2
S(3−4)	2	2	2	2	3	S(7−3)	2	1	2	2	1
S(3−5)	2	1	2	1	2	S(7−4)	1	2	1	3	1
S(3−6)	2	1	1	3	2	S(7−5)	3	2	2	2	2
S(3−7)	1	4	2	2	1	S(7−6)	1	2	4	1	2
S(3−8)	2	2	3	1	1	S(7−8)	2	1	1	1	1
S(4−1)	4	3	4	3	1	S(8−1)	1	1	2	1	1
S(4−2)	1	4	3	3	1	S(8−2)	4	1	4	4	2
S(4−3)	3	2	2	1	3	S(8−3)	2	1	3	2	2
S(4−5)	1	3	3	2	2	S(8−4)	3	4	1	1	1
S(4−6)	3	1	2	2	2	S(8−5)	3	1	3	2	1
S(4−7)	1	1	3	1	3	S(8−6)	3	3	1	1	1
S(4−8)	2	3	1	3	1	S(8−7)	2	3	1	1	1
S(1−2)	3	1	2	4	2	S(5−1)	2	2	1	4	2

Conclusions

We have proposed a novel LBTA meta-heuristic algorithm approach for the problem of finding the minimum makespan for the ZW scheduling problem. For small- and medium-size problems, the LBTA algorithm gives optimal solutions for various types of ZW scheduling problems within a short computation time. The LBTA method is shown to be very flexible in that it can be readily modified to address different problems, such as the scheduling of a single product campaign, scheduling with nonzero transfer times, and setup time under different storage policies. For large-size test problems, the LBTA′ algorithm with a probabilistic steepest decent method gives superior computational performance to the simulated annealing algorithm and the LBTA algorithm with a probabilistic local optimization method. This result shows that the method of selecting the neighborhood is a crucial factor to navigate in the search space effectively and to transcend poor local minima. The proposed algorithm is very effective, simple, and easy to implement, and it appears to be applicable to different types of scheduling problems. The advantage of minimal tuning is very attractive, allowing almost complete automation without much intervention or experimentation by the user.

Nomenclature

C = cost function
C_{ij} = time at which the product i finishes processing on unit j
C'_{ij} = time at which the product i finishes processing on unit j and transferred out from the unit j
i = index for product
j = index for unit
l = number of a sample of neighbors
L_{max} = user-defined list size
M = number of units
n_i = number of batches of product i
N = total number of products
N_{max} = number of maximum evaluations
N_p = number of different products
$N(s)$, $N_1(s)$, $N_2(s)$, and $N_3(s)$ = move function
s_{best}, s_1, s_2, and s_f = feasible configurations
s_o = initial configuration
s^* = optimal schedule
S = a set of feasible configurations
S_{ijk} = fixed setup time for successive products I, j in unit k
SC_{ikj} = setup time required for product k after product I at the storage between unit j and $j + 1$
t = CPU time required (sec)
T_h = normalized threshold value
T_{hmax} = normalized maximum threshold value
T_{ij} = transfer time of product I out of unit j to unit $j + 1$
z_j = number of storage units between unit j and unit $j + 1$
α = cooling factor in exponential annealing schedule
τ_{ij} = processing time of product i in unit j
τ'_i = processing time of different product i for SPC scheduling
Δ = normalized threshold value

Abbreviations

FIS = finite intermediate storage
LBTA = list-based threshold-accepting
MILP = mixed-integer linear programming
MINLP = mixed-integer nonlinear programming
MPC = mixed production campaign
NIS = no intermediate storage
NP-complete = nondeterministic polynomial-time complete
SA = simulated annealing
SPC = single product campaign
TA = threshold-accepting
TSP = traveling salesman problem
UIS = unlimited intermediate storage
ZW = zero wait

Appendix I: Completion Time Algorithm with Nonzero Transfer Times and Set-Up Time (Kim et al., 1996)

UIS Policy. With the assumption that the transfer time of product i from unit j to unit $(j + 1)$ is equal to that form unit j to an available storage and that from the used storage to unit $(j + 1)$, completion time for UIS multiproduct plant can be calculated as follows:

$$C'_{ij} = \max [C'_{i(j-1)}, C'_{(i-1)j} + S_{(i-1)ij} + T_{i(j-1)}] + \tau_{ij} + T_{ij} \qquad i = 1, ..., N;\ j = 1, ..., M$$

NIS Policy. When the next unit $(j + 1)$ for a product i in unit j is busy, the product should be held in unit j. Under NIS policy, completion time is as follows:

$$C'_{ij} = \max [C'_{i(j-1)}, C'_{(i-1)j} + S_{(i-1)ij} + T_{i(j-1)}, C_{(i-1)(j+1)} + S_{(i-1)i(j+1)} - \tau_{ij}] + \tau_{ij} + T_{ij} \qquad i = 1, ..., N;\ j = 1, ..., M$$

FIS Policy. Under FIS policy, the number of finite storage between unit j and unit $(j + 1)$, z_j $(j = 1, ..., M - 1)$, is fixed and given. If we define SC_{ikj} as setup time required for product k after i at the storage which is introduced between unit j and $(j + 1)$, completion time is as follows:

$$C'_{ij} = \max [C'_{i(j-1)} + \tau_{ij} + T_{ij}, C'_{(i-1)j} + S_{(i-1)ij} + T_{i(j-1)} + \tau_{ij} + T_{ij}, C_{(i-zj)} - 1(j + 1) + S_{(i-zj-1)(i-zi)(j+1)} + z_j(SC_{(i-zj)ij} + T_{(i-zj)j}) + T_{ij}] \qquad i = 1, ..., N;\ j = 1, ..., M$$

Literature Cited

(1) Aarts, E. H. L.; Korst, J. H. M. *Simulated Annealing and Boltzmann Machines*; Jone Wiley and Sons: New York, 1989.

(2) Birewar, D. B.; Grossmann, I. E. Efficient Optimization Algorithms for Zero-wait Scheduling of Multiproduct Batch Plants. *Ind. Eng. Chem. Res.* **1989**, *28*, 1333.

(3) Chen, C.; Neppalli, R. V.; Aljaber, N.; Genetic Algorithms Applied to the Continuous Flow Shop Problem. *Comput. Ind. Eng.* **1996**, *30*(4), 919.

(4) Das, H.; Cummings, P. T.; Le Van, M. D. Scheduling of Serial Multiproduct Batch Processes via Simulated Annealing. *Comput. Chem. Eng.* **1991**, *14*(12), 1351.

(5) Fink, A.; Voss, S. Applications of Modern Heuristic Search Methods to Pattern Sequencing Problems. *Comput. Oper. Res.* **1999**, *26*, 17.

(6) Hall, N. G.; Sriskandarajah, C. A Survey of Machine Scheduling Problems with Blocking and No-wait in Process. *Oper. Res.* **1996**, *44*(3), 510.

(7) Jung, J. H.; Lee, H.; Yang, D. R.; Lee, I.-B. Completion Times and Optimal Scheduling for Serial Multi-product Processes with Transfer and Setup Times in Zero-wait Policy. *Comput. Chem. Eng.* **1994**, *18*, 537.

(8) Kim, M. S.; Jung, J. H.; Lee, I.-B. Optimal Scheduling of Multiproduct Batch Processes for Various Intermediate Storage Policies. *Ind. Eng. Chem. Res.* **1996**, *35*, 4058.

(9) Ku, H.; Karimi, I. An Evaluation of Simulated Annealing for Batch Process Scheduling. *Ind. Eng. Chem. Res.* **1991**, *30*, 163.

(10) Moon, S.; Park, S.; Lee, W. K. New MILP Models for Scheduling of Multiproduct Batch Plants under Zero-Wait Policy. *Ind. Eng. Chem. Res.* **1996**, *35*, 3458.

(11) Papadimitriou, C. H.; Kanellakis, P. C. Flowshop Scheduling with Limited Temporary Storage. *J. Assoc. Com. Mach.* **1980**, *27*, 533.

(12) Pekny, J. F.; Miller, D. L. Exact Solution of the No-wait Flowshop Scheduling Problem with a Comparison to Heuristic Methods. *Comput. Chem. Eng.* **1991**, *15*(11), 741.

(13) Raaymakers, W. H. M.; Hoogeveen, J. A. Scheduling Multipurpose Batch Process Industries with No-wait Restrictions by Simulated Annealing. *Eur. J. Oper. Res.* **2000**, *126*, 131.

(14) Reddi, S. S.; Ramamoorthy, C. V. On the Flowshop Sequencing Problem with No-wait in Process. *Oper. Res. Q.* **1972**, *23*, 323.

(15) Tarantilis, C. D.; Kiranoudis, C. T.; Vassiliadis, V. S. Efficient Solutions of the General Heterogeneous Fleet Vehicle Routing Problem via Novel Threshold Acceptance Metaheuristics. Presented at the AIChE Annual Meeting, Los Angeles, CA, 2000.

(16) Van Deman, J. M.; Baker, K. R. Minimisation Mean Flow Time in Flowshop with No Intermediate Queues. *AIIE Trans.* **1974**, *6*, 28.

(17) Vaessens, R. J. M.; Aarts, E. H. L.; Lenstra, J. K. Job Shop Scheduling by Local Search. *INFORMS J. Comput.* **1996**, *8*(3), 302.

(18) Wiser, D. A.. Solution of the Flowshop Scheduling Problem with No Intermediate Queues. *Oper. Res.* **1972**, *20*, 689.

Received for review July 3, 2001
Revised manuscript received April 9, 2002
Accepted April 25, 2002

IE010570N