Journal of the Operational Research Society (1999) 50, 810–825
http://www.stockton-press.co.uk/jor

Hybrid heuristics for the multi-stage capacitated lot sizing and loading problem

L Özdamar[1] and G Barbarosoğlu[2]

[1]*Istanbul Kültür University, and* [2]*Bogaziçi University, Istanbul, Turkey*

The multi-stage capacitated lot sizing and loading problem (MCLSLP) deals with the issue of determining the lot sizes of product items in serially-arranged manufacturing stages and loading them on parallel facilities in each stage to satisfy dynamic demand over a given planning horizon. It is assumed that regular time capacity decisions have already been made in the tactical level and allocated to the stages, but it is still an important decision problem whether to augment regular time capacity by overtime capacity. Each item may be processed on a technologically feasible subset of existing facilities with different process and setup times on each facility. Since the solution of the MCLSLP requires the design of a powerful algorithm, simulated annealing (SA) and genetic algorithms (GA) are integrated to enhance their individual performances. Furthermore, these global optimisation methods are incorporated into a Lagrangean relaxation scheme, hence creating a hybrid solution methodology. Numerical results obtained using these methods confirm the mutual benefits of integrating different solution techniques.

Keywords: Heuristics; simulated annealing; genetic algorithms; Lagrangean relaxation; multi-stage lot sizing

Introduction

The multi-stage capacitated lot sizing and loading problem (MCLSLP) addresses the issue of determining the production lot sizes of various items appearing in consecutive production stages over a given planning horizon. The production environment considered in this study can be best described as a flow-type process manufacturing such that some processing is done on the material in each stage as it proceeds through serially-arranged stages and that the item has no distinct identity as a finished product until it leaves the very last stage. Therefore the same product index is preserved for an item in each stage although the material undergoes transformation as it moves downstream. Examples of this type of manufacturing include steel mills, chemicals, plastics, pharmaceuticals, ceramics, glass plants and refineries. Although it can be argued that there usually exists one bottleneck which is stable over time in such plants, when there are parallel facilities in each stage so that an item may be produced on more than one facility, and especially if the associated processing times exhibit considerable variation, the loading scheme may lead to different capacity utilisation rates in each stage over time. Therefore, one stage which is bottleneck in a certain time period may not be bottleneck in another time period. This implies that bottlenecks are not necessarily stable and do change dynamically over time in case of parallel facilities. Furthermore, in accordance with the optimised production technology (OPT),[1] the synchronisation of all stages with the bottleneck can be best done by considering the bottleneck and all the down-stream stages which follow the bottleneck simultaneously. In this context, multi-stage capacitated lot sizing decisions are related with the determination of lot sizes in each stage so as to facilitate a synchronised flow in the presence of capacity constraints. The loading problem, on the other hand, arises from the existence of unrelated parallel facilities of different classes in each stage and requires the assignment of production lot sizes to various facilities. In the literature, lot sizing and loading decisions are usually treated independently for simplifying the overall decision-making problem. In the classical hierarchical decision making context, it is suggested that lot sizing decisions take place in the tactical medium-range planning level whereas loading decisions are dealt with in the strategic short-term planning level.[2] However, decoupling these problems and solving them hierarchically such that first lot sizing and then loading decisions are made may naturally lead to severe capacity infeasibility issues and will certainly generate sub-optimal solutions since capacity decisions are strongly affected by the particular loading schemes. Inevitably, lot sizes determined in a higher level of planning affect the solution to the loading problem and an a-priori decision to solve the lot sizing problem first is not justified unless the trade-offs between the loading and lot sizing components in the integrated objective function are thoroughly investigated. In view of these facts, much time and effort seem to be wasted to solve the decomposed subproblems indepen-

Correspondence: Dr. G Barbarosoğlu Bogaziçi University, Industrial Engineering Department, Bebek, Istanbul, Turkey.
E-mail: barbaros@boun.edu.tr

dently. Consequently, it is proposed to formulate the MCLSLP as an integrated problem, rather than decomposing it into two subproblems.

The first class of lot sizing models is known to be the economic lot scheduling problem (ELSP) which assumes continuous stationary demand over an infinite planning horizon. Dobson,[3] Philipoom *et al,*[4] and Gallego and Joneja[5] studied the capacitated single level multi item ELSP case whereas a multi stage analysis can be found in El-Najdawi.[6] The next generation of lot sizing models can be grouped together within the single stage/multi-stage capacitated lot sizing models (CLSP) which assume that demand is deterministic and dynamic over a finite planning horizon; limited capacity is shared by all items produced in each period and no backorders are permitted. Various extensions of the single stage CLSP model are proposed in the literature.[7,8,9,10] More details on the single/multi-stage CLSP can be found in Drexl and Kimms[11] and Özdamar and Birbil.[12]

In the single and multi stage CLSP, capacity constraints are generally modelled as aggregate capacity of a single facility, and loading issues are not considered. However, in practice loading decisions are at times crucial in the minimisation of total costs. The loading decisions discussed in this research involve the assignment of lots to facilities when there are parallel facilities in each production stage. The assignment problem becomes NP-hard when facilities are unrelated (item process times depend on facilities) and/or there exist different classes of facilities and each item cannot be processed on every facility resulting in a class scheduling problem.[13] For a general review on the parallel machines problem, the reader is referred to Cheng and Sin[14] where it is discussed that for most performance measures, the assignment problem is NP-hard in the feasibility and optimality problems even when the facilities are identical.

Recent literature on the integrated treatment of lot sizing and scheduling is mostly focused on discrete lot sizing and scheduling (DLSP). In the DLSP, only one item can be produced in each discrete time period and full capacity has to be utilised whenever a lot size is positive. DLSP is usually suitable for short term planning where planning periods may represent small time buckets such as shifts. Due to its special structure, rather large size problems can be solved to optimality.[15, 16] . Variants of the DLSP are considered in Cattrysse *et al*[17] and Jordan and Drexl.[18] De Matta and Guignard[19] dealt with the DLSP with no setup times and with splittable lots on multiple production lines of different classes.

In an effort to integrate loading and lot sizing problems, Özdamar and Bozyel[20] propose a single period lot sizing and loading model where the lot sizing and loading decisions are made simultaneously by using the lot sizing algorithm developed in a previous study.[21] The loading problem involves the bin packing problem with bins of different classes.

The integrated model of the MCLSLP analysed in this study is a multi-stage extension of the single stage CLSLP model in Özdamar and Birbil.[12] Furthermore, here, setup costs are included in the objective function and shortages are permitted for end items in the last stage. The multi stage CLSLP subsumes the multi-stage CLSP with set-up times, overtime capacity and a serial product structure, and the unrelated parallel facility loading problem with facilities of multiple classes. Because of the fact that the MCLSLP permits multiple items to be produced in one period and that lot sizes are not restricted to null or full capacity, the MCLSLP also subsumes the DLSP.

The MCLSLP is a synthesis of three different planning and loading problems, namely the capacitated lot sizing problem (CLSP) with overtime decisions and setup times, the problem of minimising total tardiness on unrelated parallel processors, and the class scheduling problem, each of which is NP-complete in feasibility and NP-hard in optimality. Therefore, the MCLSLP which incorporates both is NP in both issues. Hence, it is necessary to develop efficient solution procedures for the MCLSLP. Here, two hybrid heuristics are developed. The first heuristic integrates two meta-heuristics: simulated annealing (SA),[22] and the genetic algorithm (GA).[23, 24] The second heuristic integrates SA with Lagrangean relaxation (LR). As it is shown in Özdamar and Bozyel,[10] the performance of SA can be improved considerably by using a diversified set of initial solutions which can be obtained by the GA mechanism with its reproduction, crossover and mutation operators working on the solution populations. Secondly, the results obtained by Özdamar and Barbarosoğlu[25] for the multi-stage CLSP reveal that the performance of LR can be enhanced by using SA within the Lagrangean heuristic in each iteration. Consequently, SA is integrated within GA and LR techniques to result in two different hybrid procedures, namely (GA/SA) and (LR/SA). As it is well-known, the LR technique is used as a robust solution method for the single- and multi-stage CLSP.[7,9,26,27,28,29,30]

This paper is organised as follows: A mathematical formulation of the MCLSLP is provided in Section 2. The details of the SA procedure are explained in Section 3. The GA algorithm integrated with SA(GA/SA) is discussed in Section 4. Section 5 includes the main decomposition scheme of LR. In Section 6, the heuristics developed to solve the resulting sub-models and the Lagrangean meta-heuristic (SA) are described. The experimental design of test problems and the computational results are given in Section 7. Some concluding remarks are made in Section 8.

General description of the mathematical model

This study is mainly concerned with analysing the lot sizing and loading decisions in the MCLSLP which may be encountered in a flow-type manufacturing system. The multi-stage nature of the problem, the existence of back-

orders in the last stage, facility dependent process and setup times are some of the features which further complicate the solution of the problem. The main assumptions underlying this study are stated below:

(i) Demand for the end items is known with certainty but varies dynamically over the planning horizon;
(ii) Backlogging is permitted for end items but upper bounded by a given percentage of demand in each period;
(iii) Flow manufacturing encountered basically in process industries is visualised as the context of the research;
(iv) Production in each stage involves identical, uniform or unrelated parallel facilities of the same or of different classes. Each item can be loaded on different facilities, and different items can be assigned to the same facility during the same period as long as capacity permits;
(v) Lot-splitting is not permitted in any stage;
(vi) Manufacturing lead-time in each stage is negligible, so omitted in the model;
(vii) Production in each facility requires a setup which in turn consumes capacity;
(viii) Regular capacities for the facilities in each stage are predetermined and known as a consequence of tactical planning, but they can be augmented by overtime capacity up to the overtime limit;
(ix) Inventory holding, backordering and overtime costs are covered in objective function minimisation.

The basic notation used in the mathematical model to follow is given below:

Parameters

d_{jt} = external demand of end item j in period t
h_{ejt} = unit inventory holding cost of item j in stage e carried from period t to $t+1$
π_{jt} = unit backorder cost of end item j carried from period t to $t+1$
SC_{ejk} = setup cost for item j on facility k in stage e
θ_{ekt} = overtime cost per manhour employed on facility k in stage e in period t
BP = permissible percentage of end item demand that can be backordered
p_{ejk} = unit process time of item j on facility k in stage e (hours)
s_{ejk} = setup time required for time j on facility k in stage e (hours)
C_{et} = regular time capacity for each facility in stage e in period t
CO_e = maximum overtime capacity that can be employed in stage e
F_{ej} = set of facilities on which item j can be loaded in stage e
L_{ek} = set of items which can be loaded on facility k in stage e
M = a very large number

Decision variables

X_{ejt} = production lot size of item j in stage e in period t
I_{ejt} = inventory level of item j in stage e at the end of period t
B_{jt} = backorder amount of end item j at the end of period t
Y_{ejkt} = production lot size of item j loaded on facility k in stage e in period t
O_{ekt} = overtime capacity employed on facility k in stage e in period t
W_{ejkt} = a binary variable indicating that item j is loaded on facility k in stage e in period t

A generic mathematical formulation for the MCLSLP is provided below:

Model P

$$z = \min \sum_t \sum_e \sum_j h_{ejt} I_{ejt} + \sum_t \sum_j \pi_{jt} B_{jt} + \sum_t \sum_e \sum_j \sum_{k \in F_{ej}} SC_{ejk} W_{ejkt} + \sum_t \sum_e \sum_k \theta_{ekt} O_{ekt} \quad (1)$$

subject to

$$I_{Ej,t-1} - B_{j,t-1} - I_{Ejt} + B_{jt} + X_{Ejt} = d_{jt} \quad \forall j, t \quad (2)$$

$$I_{ej,t-1} - I_{ejt} + X_{ejt} - X_{e+1,jt} = 0 \quad \forall e, j, t \quad (3)$$

$$B_{jt} \leqslant BP d_{jt} \quad \forall j, t \quad (4)$$

$$X_{ejt} - \sum_{k \in F_{ej}} Y_{ejkt} = 0 \quad \forall e, j, t \quad (5)$$

$$\sum_{j \in L_{ek}} (p_{ejk} Y_{ejkt} + s_{ejk} W_{ejkt}) - O_{eckt} \leqslant C_{et} \quad \forall e, k, t \quad (6)$$

$$\sum_{k \in F_{ej}} W_{ejkt} \leqslant 1 \quad \forall e, j, t \quad (7)$$

$$Y_{ejkt} \leqslant M W_{ejkt} \quad \forall e, j, k, t \quad (8)$$

$$O_{ekt} \leqslant CO_e \quad \forall e, k, t \quad (9)$$

$$X_{ejt}, I_{ejt}, B_{jt}, Y_{ejkt}, O_{ekt}, \geqslant 0 \quad W_{ejkt} = 0, 1 \quad \forall e, j, k, t \quad (10)$$

Here, the objective function (1) minimises the holding costs of the end items and the work in process, backordering costs of the end items, the setup costs involved in loading the facilities and the overtime capacity cost incurred during the whole planning horizon. Constraints (2) express the material balance for the end items where backorders are permitted while constraints (3) express the material balance for the work-in-process. The fact that the backorder quantities of the end items carried in each period are upper bounded by a certain percentage of the external demand in that period is stated in (4). The constraints (5) link the production lot sizing and loading decision variables. The capacity restric-

tions are imposed by constraints (6) while overtime capacity decisions for each facility and stage in each period are limited in constraints (9). It is important to note that an equal amount of regular time capacity is allocated to each facility in a stage (C_{et}), and not in aggregate terms for the considered stage. The parallel facilities become identical if p_{ejk} and s_{ejk} are defined independent of the facility index, k. The restrictions on lot splitting are expressed by constraints (7). Constraints (8) guarantee that a setup is undertaken if an item is loaded on a given facility.

The simulated annealing procedure (SA)

The first procedure developed to solve the MCLSLP is the stand alone simulated annealing (SA) approach which is initiated from an inventory feasible, but capacity infeasible solution and which tries to improve the objective function in a given number of moves. In generating neighbouring solutions, inventory feasibility is always preserved and moves which lead to capacity violations are penalized per unit capacity hour overused. In order to provide a complete characterization of any SA procedure, it is necessary to describe the generation of the initial solution, the cooling scheme, and the perturbance algorithm necessary to generate the next neighbouring solution.

Generation of an initial solution

An initial solution is generated by first considering the very last stage where there exists independent demand. A random lot size is assigned to every item in each period. If a randomly assigned lot size does not satisfy the demand in the considered period, it is satisfied by randomly augmenting the lot sizes of previous periods. Then, the lot sizes assigned in the last stage are propagated to the preceding stage as the dependent demand and the same random lot sizing procedure is repeated for each upstream stage. This hierarchical approach applied to consecutive stages is repeated until the very first stage is considered. Then, it becomes necessary to make the loading decisions. Positive lot sizes are loaded on randomly selected facilities of appropriate classes without permitting any lot splitting. This scheme guarantees to achieve non-negative inventory levels between consecutive stages, but since capacity constraints are overlooked, the capacity requirements implied by random lot sizing may certainly violate the capacity restrictions. In such a case, penalties per unit violation are added to the cost of the solution.

The cooling scheme

It is usually possible to save on inventory and/or overtime costs during a move. However, the cost difference Δ_{cost} between two consecutive moves may be positive and a move may result in a cost higher than the incumbent one. The SA procedure calculates a probability of acceptance (PA) which depends on the amount of deterioration caused by the move in the objective function and on the number of times a deteriorated cost has been obtained consecutively, because the temperature (tSA) is reduced everytime a non-improving move is executed. PA is calculated as follows:

$$\text{PA}(\Delta_{\text{cost}}, t\text{SA}) = \exp(-\Delta_{\text{cost}}/(\text{Cost} \times t\text{SA}))$$

where Cost is the cost of the incumbent solution. Here, PA is based on the relative cost difference rather than the absolute cost difference mainly because the exponential term most often drives PA close to near-zero values when the latter is used. Furthermore most SA procedures in literature use the relative cost difference. The temperature reduction scheme is continuous as given in Dowsland.[31] Namely, after each non-improving move, the temperature, tSA whose initial value is 1.0, is reduced by $t\text{SA} \leftarrow t\text{SA}/(1 + \beta \cdot t\text{SA})$ where β is a parameter less than 1.0. If tSA falls below a very small value, then it is reset to 1.0, so that PA does not become infinitesimally small after a large number of moves. Here, the temperature reduction parameter β is not taken as a constant, but it is adaptive in the sense that its value depends on the status of the search. If the last M number of consecutive moves result in improving cost values, then β is decreased by a step-size less than 1.0 (a convenient step-size is 0.01). On the other hand, if the last M number of consecutive moves result in non-improving cost values, then β is increased by the step-size. This dynamic temperature reduction scheme serves to update PA according to the specific part of the feasible region to which the search is heading. If the search is moving towards a favourable direction, β is decreased to encourage non-improving moves so that local optima can be skipped. Non-improving moves are discouraged when the search is visiting an unfavourable region of the feasible region, so that the search can escape from a region that is unlikely to store the global optimum solution.

Generation of a neighbour solution

A move is generated as the result of perturbing the lot size of an item in a given solution. A neighbouring solution is one where exactly one item in a given stage has its lot size modified by either backward or forward shifting. In this study, a procedure called Perturb is developed to generate moves on which an implicit tabu restriction is applied in order to prevent cycling. Here a move is considered for execution if it is not found in a tabu list which stores only the costs associated with a certain recent number of moves. The tabu list prevents the search from shifting the lot size of an item if the shift under consideration has a cost equal to one of the costs stored in the list. The tabu list is updated with each executed move by adding the most recently obtained total cost to the end of the list and deleting the

oldest one from the list. The size of the tabu list is specified according to the problem dimensionality in the algorithms.

The main idea in a move is to determine a stage/item/period triplet for which a possible forward or backward lot size shifting can be done. In a forward shift, a lot is wholly or partially transferred to the next period whereas in a backward shift it is transferred to the previous period. Furthermore, a lot can be transferred to another facility in the same period. Firstly, the stage/item/period triplet (e^*, j^*, t^*) is selected randomly. The lot size of item j^* in stage e^* will be decreased in period t^*. Then, it is randomly decided whether the selected lot size will be forward $(t = t^* + 1)$ or backward-shifted $(t = t^* - 1)$, or transferred to another facility in the same period $t(t = t^*)$. Finally, the facility f^* on which item j^* will be loaded in period t is determined by searching over all possible facilities in period t and selecting the best facility with minimum incremental cost. The lot transfer will be made to an appropriate facility on which item j^* can be loaded and lot-splitting will be avoided in period t. When the lot is shifted to a future period, the shift may lead to backorders (permitted only in the last stage) and the maximum amount that can be shifted is limited both by the lot size and the backorder limit of period t^*. If $e^* \neq$ E, then the shifted lot size is limited by the inventory level in period t^*. However, if the lot is to be shifted backward, in order to preserve inventory feasibility between successive stages, the inventory level in the previous stage, $e^* - 1$, in period t should be taken into account, and the maximum amount that can be shifted to an earlier period is determined by the minimum of the lot size itself and its inventory level in the previous stage. If the cost of the move does not coincide with any of the costs in the tabu list and if it improves the incumbent solution, then the move is immediately executed. Otherwise, the probability of acceptance (PA) is computed and the move is executed only if it is probabilistically chosen. In both cases, the state values of the system parameters, the tabu list and the annealing temperature parameters are updated accordingly. The details of the procedure Perturb are given in the pseudocode in Figure A1 in the Appendix.

Although the SA procedure is used as a stand alone solution approach for the MCLSLP, it is also incorporated into a GA and a LR procedure as a Lagrangean heuristic. In fact, it is imbedded into each one of them not only to improve the performance of the latter procedures mentioned, but also to enhance its own search capability. Previous experimentation with a pure SA procedure developed for the single stage single facility CLSP[10] demonstrates that encouraging results are obtained (an average deviation from the optimum is reported to be about 1% and 4% for a SA procedure starting from a good solution and a random solution, respectively) by running the SA procedure numerous times with a new initial solution and finally reporting the best solution from among all runs. Unfortunately, the initial random seed affects the performance of SA within a limited number of moves, and diversity has to be achieved by rerunning SA with different random seeds. If SA is coupled with another procedure, the other procedure should serve as a natural diversification tool for SA. In the hybrid procedures developed in this study (GA/SA, LR/SA), the GA population and the iterative LR mechanism provide different good starting points for SA.

On the other hand, the performance of the Lagrangean heuristic is well-known to influence the convergence and the solution quality of any LR approach. As a Lagrangean heuristic, the impact of the SA procedure on the performance of LR depends on its ability to restore the feasibility of a relaxed solution in any LR iteration and on the tightness of the upper bound it provides. Since the upper bound is to be deployed in updating the Lagrangean parameters for the next iteration, it strongly affects the convergence of the algorithm. In the SA design proposed here, the geometric cooling scheme results in a fast convergence to a feasible solution with a tight upper bound and satisfies the requirements of a Lagrangean heuristic.

The hybrid genetic algorithm (GA/SA)

GA is known to involve reproduction, crossover and mutation operators which help the initial population of random solutions (chromosomes) to converge to the global optimum. While reproduction creates new replica of good solutions replacing the bad ones, the crossover operator aims to generate new offspring from two good parent chromosomes. Mutation introduces random changes in some genes, preventing the population from getting stuck to the same area of the feasible space. Hence, with these three operators working on the population, the population evolves in each generation with an improved average performance. The GA implementation for the MCLSLP is similar to the one described in Özdamar and Birbil[12] for the single stage CLSLP. Similar to the SA, the GA acts in a solution space which is inventory-feasible, but capacity-infeasible. Capacity infeasibility is penalised per unit excess over the sum of regular and overtime capacities in each period. Since the crossover operator is bound to destroy inventory-feasibility during the swap of a block of genes, a repair procedure acts upon the offspring to restore inventory-feasibility. The mutation preserves inventory-feasibility while changing genes in a chromosome. The pseudocode for the GA/SA is given in Figure A2 in the Appendix. The GA/SA procedure conducts reproduction, crossover and mutation on a number parallel independent populations (PopNo). After a given number of generations (GenNo) are carried out independently in each population, GA applies crossover to chromosomes across all populations, so genes from good chromosomes migrate to various populations to result in good quality offspring and in a population merge.

During the independent processing of each population, the GA activates the SA procedure Perturb in Figure A1 when the population tends to get stuck to the same region of the solution space. The SA is applied to a number of randomly selected chromosomes bounded by (frac) (PopRange) (PopSize) where frac is a fraction of the population size defined by the user, PopRange is the ratio of the minimum objective function value among the chromosomes to the maximum one, and PopSize is the number of chromosomes in the population. For the SA, each selected chromosome is a starting solution and it is replaced by the best solution obtained during the search.

When the diversity of the population is lost (PopRange is close to one), the SA processes more chromosomes, because the selected chromosomes are expected to change considerably and the population will be renewed. After the SA completes the search on the selected chromosomes, the GA resumes applying its reproduction, crossover and mutation operators. The SA procedure has a diversification effect on the solution space considered by the GA, because the solution obtained at the end of the SA is usually far away from that described by the initial chromosome and frequently has a much better objective function value. On the other hand, SA has an intensification effect in terms of population performance, because it introduces more feasible solutions into the population. Consecutive GA/SA iterations diversify/intensify population performance which helps the chromosomes converge to the optimum solution.

Chromosome encoding

A solution to the MCLSLP is encoded by indicating Y_{ejkt} variables in two structures: the structure X_{ejt} concerning the lot sizes, where X_{ejt} is the lot size of family j in stage e in period t in the given solution and the structure a_{ejt} indicating the specific facility k on which a positive lot size of family j is loaded in stage e in a specific period t.

Generating the initial population

The generation of the initial solution is carried out as in the SA procedure by using the procedure Perturb.

The reproduction operator

The reproduction strategy used in the GA/SA procedure is a pure selection one.[32] Each chromosome in the population is reproduced a number of times proportional to its objective function value.

The crossover operator

The crossover operator is a two-point operator which randomly selects two parents (1) and (2) from the population, and two periods, t1 and t2. For all families j and stages e, the structures $X_{ejt}(1), a_{ejt}(1)$ and $X_{ejt}(2), a_{ejt}(2)$ are swapped for all $t1 \leqslant t \leqslant t2$. The two-point crossover described here may lead to infeasibility with respect to inventory balance equations. These infeasibilities are repaired starting from the last stage by adding or subtracting the necessary amounts to or from family lot sizes. The backorder levels permitted for the last stage are also taken into account. Details on this repair procedure are found in Özdamar and Birbil.[12]

The MixCross operator

The MixCross operator is similar to the crossover operator, but aims at merging genes of chromosomes in different populations.

The mutation operator

Mutation is carried out on each chromosome, but only one stage/item/period triplet (with positive lot size) in each chromosome is selected randomly for mutation. A random amount is deducted from the lot size, but the reduction neither exceeds the item's lot size nor its inventory level. The same amount is added to the lot size of the next period. Therefore inventory feasibility is preserved.

Crossover and mutation probabilities

The crossover/mutation probability for a chromosome c is calculated by $pc = (\text{maxobj-objc})(\text{maxobj-minobj})$[33], if minobj $\neq$ maxobj and $p_c = 1.0$ otherwise. Here minobj is the least(best) objective function value in the current population, maxobj is the worst one and obj_c is the objective function value of chromosome c. A chromosome's crossover/mutation probability is dependent both on its own objective function value and on the population's performance range. The adaptive crossover/mutation probabilities are demonstrated to be superior to fixed probabilities by Srinivas and Patnaik[33] and Özdamar.[34]

The Lagrangean relaxation procedure (LR)

LR is a technique known to decompose a combinatorial optimisation problem into a number of easy-to-solve subproblems by dualizing the "difficult" coupling constraints. As it is discussed in Fisher,[35] it can be efficiently used in place of linear programming relaxation in any branch and bound algorithm. At each node of the branch and bound tree, it is used to obtain a feasible solution and a lower bound (for minimisation problems) on the optimal objective function value. In this study, LR is used as a stand-alone procedure, not within branch and bound, to decouple the lot sizing and loading decisions inherent in the model (P) defined by (1)–(10). Therefore, the relaxation design is

obtained by relaxing the coupling constraints (5) and including them into the objective function:

$$\boldsymbol{P(u)} \quad z(u) = \min \sum_t \sum_e \sum_j h_{ejt} I_{ejt} + \sum_t \sum_j \pi_{jt} B_{jt} + \sum_t \sum_e \sum_j \sum_{k \in F_{ej}} SC_{ejk} W_{ejkt} + \sum_t \sum_e \sum_k \theta_{ekt} O_{ekt} + \sum_t \sum_e \sum_j u_{ejt} \left(X_{ejt} - \sum_{k \in F_{ej}} Y_{ejkt} \right) \tag{11}$$

subject to constraints: (2)–(4), (6)–(10).

Here $u = \{u_{ejt}/u_{ejt}\text{free}\}$ is a vector of Lagrange multipliers. Rearranging the terms in (11), it is easy to obtain

$$z(u) = \min \sum_t \sum_e \sum_j h_{ejt} I_{ejt} + \sum_t \sum_j \pi_{jt} B_{jt} + \sum_t \sum_e \sum_j u_{ejt} X_{ejt} - \sum_t \sum_e \sum_j u_{ejt} \sum_{k \in F_{ej}} Y_{ejkt} + \sum_t \sum_e \sum_j \sum_{k \in F_{ej}} SC_{ejk} W_{ejkt} + \sum_t \sum_e \sum_k \theta_{ekt} O_{ekt} \tag{12}$$

subject to constraints: (2)–(4), (6)–(10).

Since this relaxation decouples the lot sizing and loading variables, it is easy to observe that the model $P(u)$ is decomposable into two submodels such that the first one addresses the issue of production lot sizing while the capacitated loading decisions are covered in the second. The multi-item lot sizing model is given by

$$\textbf{LOT(u)} \quad \text{zlot}(u) = \min \sum_t \sum_e \sum_j h_{ejt} I_{ejt} + \sum_t \sum_j \pi_{jt} B_{jt} + \sum_t \sum_e \sum_j u_{ejt} X_{ejt} \tag{13}$$

subject to:

$$I_{Ej,t-1} - B_{j,t-1} - I_{Ejt} + B_{jt} + X_{Ejt} = d_{jt} \quad \forall j, t \tag{2}$$

$$I_{ej,t-1} - I_{ejt} + X_{ejt} - X_{e+1,jt} = 0 \quad \forall e, j, t \tag{3}$$

$$B_{jt} \leqslant BPd_{jt} \quad \forall j, t \tag{4}$$

$$X_{ejt}, I_{ejt}, B_{jt} \geqslant 0 \quad \forall e, j, k, t \tag{10}$$

The model **LOT(u)** is further decomposable into multi-stage single-item submodels since there is no coupling among the items. Thus each single-item submodel can be described by

$$\textbf{LOT(u(j))} \quad \text{zlot(u(j))} = \min \sum_t \sum_e h_{ejt} I_{ejt} + \sum_t \pi_{jt} B_{jt} + \sum_t \sum_e u_{ejt} X_{ejt} \tag{14}$$

subject to constraints: (2), (3), (4) and (10) for each j.

Each model **LOT(u(j))** turns out to be a linear program without any discrete variables and can be classified as a single-item multi-stage production planning problem. Although it is very easy to solve this model optimally by using a commercial software, the interfacing between such a code and LR procedure would inhibit an efficient computational design. Consequently, a heuristic procedure is developed in this study to obtain an approximate solution which will satisfy the needs of the LR. In this procedure, a hierarchical design first determines the production lot sizes of end item j over the planning horizon considering the backordering possibility as well and then propagates these lot sizes recursively through upstream stages to determine the lot sizing decisions in each stage. The procedure is repeated for each of the end items independently. The details of the procedure will be provided in the next section.

The second submodel given below aims to make the inherent capacitated loading decisions:

$$\textbf{LOAD(u)} \quad \text{zload}(u) = \min - \sum_t \sum_e \sum_j u_{ejt} \sum_{k \in F_{ej}} Y_{ejkt} + \sum_t \sum_e \sum_j \sum_{k \in F_{ej}} SC_{ejk} W_{ejkt} + \sum_t \sum_e \sum_k \theta_{ekt} O_{ekt} \tag{15}$$

subject to:

$$\sum_{j \in L_{ek}} (p_{ejk} Y_{ejkt} + s_{ejk} W_{ejkt}) - O_{ekt} \leqslant C_{ekt} \quad \forall e, k, t \tag{6}$$

$$\sum_{k \in F_{ej}} W_{ejkt} \leqslant 1 \quad \forall e, j, t \tag{7}$$

$$Y_{ejkt} \leqslant M W_{ejkt} \quad \forall e, j, k, t \tag{8}$$

$$O_{ekt} \leqslant CO_e \quad \forall e, k, t \tag{9}$$

$$Y_{ejkt}, O_{ekt}, \geqslant 0 \quad W_{ejkt} = 0, 1 \quad \forall e, j, k, t \tag{10}$$

Again it is obvious that no coupling exists among the stages and across the time periods since there is no capacity sharing. Consequently the model **LOAD(u)** can be decomposed into single-period single-stage submodels as shown below:

$$\textbf{LOAD(u(e,t))} \quad zload(u(e,t)) = \min - \sum_j u_{ejt} \sum_{k \in F_{ej}} Y_{ejkt} + \sum_j \sum_{k \in F_{ej}} SC_{ejk} W_{ejkt} + \sum_k \theta_{ekt} O_{ekt} \tag{16}$$

subject to constraints: (6), (7), (8), (9), and (10) for each e and t.

The model **LOAD(u(e,t))** turns out to be a static single-stage single period capacitated loading problem where lot sizes of different items should be loaded on some facilities in a given time period. Since the process and setup times are dependent on the facilities, loading decisions will certainly affect the capacity requirements which will ignite the capacity decisions. Capacity decisions are mainly concerned with augmenting regular time capacity by overtime capacity whenever it is necessary. Another heuristic algorithm is developed to solve each of these models independently.

An important issue in LR is to obtain the values for the Lagrange multipliers which are generated in many applications by solving the dual problem:

MODEL D:

$$z_D = \max_u z(u)$$

At this point subgradient optimization is used without loss of generality to solve the model **D** where the multipliers will be updated systematically according to the implementation given in Held *et al.*[36] At any iteration r, the values of the Lagrange multipliers are determined by

$$u_{ejt}^{r+1} = u_{ejt}^{r} + ss^{r}\left[X_{ejt}^{r} - \sum_{k\in F_{ej}} Y_{ejkt}^{r}\right] \quad \forall e, j, t \qquad (17)$$

where ss^r is the step size in iteration r and given by

$$ss^{r} = \lambda^{r}|z_{up}^{r} - z^{r}| \Bigg/ \sum_{e}\sum_{j}\sum_{t}\left(X_{ejt}^{r} - \sum_{k\in F_{ej}} Y_{ejkt}^{r}\right)^2 \qquad (18)$$

Here X_{ejt}^r and Y_{ejkt}^r are the solutions obtained by solving the submodels **LOT(u(j))** for all j and **LOAD(u(e,t))** for all e and t, respectively. Furthermore, z_{up}^r is an upper value on z, z^r is the objective function value associated with the approximate solutions of the submodels as computed in iteration r, and $|\cdot|$ denotes the absolute value.

Since it is well-known that the solution of the dual problem **D** is not necessarily feasible to the original problem **P**, it is essential to design a procedure which will generate a feasible solution out of the near feasible dual solution. In each subgradient optimization iteration the dual solution is input to a so-called Lagrangean heuristic to obtain a feasible solution whose objective function value can be used as an upper bound for the problem **P**. This bound is expected to improve during the LR procedure, so that at termination one gets a feasible solution with the best upper bound. A Lagrangean heuristic based upon SA principles is developed in this research in order to serve this purpose. The initial value for z_{up} is found by considering the worst-case performance

$$z_{up}^{0} = \sum_{t}\sum_{j}\left(\pi_{jt}d_{jt} + \sum_{e} h_{ejt}d_{jt} + \sum_{e}\max(SC_{ejk})\right) + \sum_{t}\sum_{e}\sum_{k}\theta_{ekt}CO_{e} \qquad (19)$$

and the upper value is updated whenever the Lagrangean heuristic obtains a feasible solution with an objective function value lower than the incumbent.

Also it is important to note that (17) differs from the conventional step-size s found in most subgradient optimization studies. Whenever the submodels are solved optimally, $z(u)$ is known to be a lower bound for z. However, since the model **P(u)** is not solved to optimality in this research, but only approximate solutions are generated for the resulting submodels, the objective function defined by

$$z^{r} = \sum_{j} z\text{lot}(u^{r}(j)) + \sum_{e}\sum_{t} z\text{load}(u^{r}(e,t)) \qquad (20)$$

is not necessarily a lower bound for z and may turn out to be greater than the upper bound in any iteration. Thus the absolute value of the distance between the upper value and z^r is used in order to be able to continue with the Lagrangean iterations.

In (18), λ is a scalar which is initiated from 2.0 and reduced by $\lambda^{r+1} = 0.75, \lambda^r$ whenever the objective function value z^r does not improve in 4 iterations. The overall LR procedure can be summarized as follows:

Step 1: Solve the submodels **LOT(u^r(j))** for all j and the submodels **LOAD(u^r(e, t))** for all e and t by using u^r. Update z^r as computed by (20).

Step 2: Using the solutions X_{ejt}^r and Y_{ejkt}^r obtained in Step 1, carry out subgradient optimization to update the values of Lagrangean multipliers by (17) and (18).

Step 3: Execute the Lagrangean heuristic to generate a feasible solution out of the solution obtained in Step 1. Update the upper value z_{up}. Repeat Steps 1 and 2 until a maximum number of iterations is reached.

At this point it is necessary to clarify the coordination effort of the LR via the Lagrange multipliers. Since dualizing constraints (5) decouples the lot sizing and loading decisions, the respective decisions are made independently and infeasibility will naturally arise from the inconsistency between the solutions of **LOT(u)** and **LOAD(u)**. The subgradient given by $[X_{ejt}^r - \sum_{k\in F_{ej}}, Y_{ejkt}^r]$ measures the inconsistency using the most recent solutions. A positive subgradient indicates that lot sizes are greater than the loaded quantities and X_{ejt}^r should be decreased while Y_{ejkt}^r should be encouraged. This information is conveyed to the next iteration by the subgradient engine. A positive value for the subgradient will tend to increase the updated multipliers in the next iteration through (17). This increase will have a decreasing effect upon the lot size decisions while an increasing effect upon the loading decisions. The opposite argument is true when the subgradient turns out to be negative. Thus, it can be concluded that the process of reducing the inconsistency between the lot sizing and loading models is monitored by the Lagrange multipliers generated by the subgradient coordination.

Heuristics in the Lagrangean relaxation procedure

At this point, it is necessary to explain the heuristic procedures developed to solve the submodels and the design of the Lagrangean heuristic. As it is stated previously, the main effort is focused on obtaining only approximate solutions to the resulting submodels since a global search will be carried out in each Lagrangean iteration both to maintain feasibility and to improve the incumbent objective function value. The fact that solving the model **(P(u))** only approximately will cause **z(u)** to lose its lower bound property is remedied easily within the frame-

work of subgradient optimization by defining the step size as in (18).

The lot sizing heuristic

In order to solve the model **LOT(u)**, it is necessary to solve the submodels **LOT(u(j))** independently for each of the items. Each **LOT(u(j))** is a single-item multi-stage production planning problem where the production and holding costs vary from one period to another and limited backordering is permitted only for the end items. Since it suffices to generate an approximate solution, a heuristic procedure is designed to be implemented in a hierarchical manner so that inventory feasibility between the stages given by (3) is guaranteed. Firstly, the production plan of item j in the very last stage is determined for the planning horizon using the external demand requirements for the end item under consideration and considering the possibility of upper-bounded backorders. Then, this plan is downloaded to the previous stage as the dependent demand pattern, and the production plan for that stage is obtained without permitting backorders. This is repeated recursively by propagating the successive stage's production lot sizes to the previous stage as dependent demand until the very first stage is reached. The solution procedures for obtaining the production lot sizes are the same for all the stages except the very last one, since, although limited, backlogging is permitted only in the last stage.

The algorithm developed to determine the lot sizes of a single product to be produced in the last stage is based upon backward and forward shifting principle. The procedure is initiated by defining the lot size in each period equal to external demand on a lot-for-lot basis; that is, $X_{Ejt} = d_{jt}$ for each t. Then, the marginal cost of shifting one unit of a lot backward to a previous period and forward to a future period is computed. The consequence of shifting one unit of a lot in period t to a previous period k is to carry inventory from period k to period t. Thus, the marginal cost associated with this change is given by

$$\text{marcos}_{tk} = \sum_{q=k}^{t} h_{Ejq} - u_{Ejt} + u_{Ejk} \qquad t \geqslant k$$

However, shifting one unit of a lot from period t to a future period k will result in backordering and the marginal cost is given by

$$\text{marcos}_{tk} = \sum_{q=t}^{k} \pi_{jq} - u_{Ejt} + u_{Ejk} \qquad t < k$$

The algorithm performs the most cost-effective shift by considering these marginal costs. The (t, k) pair with minimum and negative marcos_{tk} over all pairs is selected for a shift. Once the (t, k) pair and the type of shift is thus determined, the next issue is to decide on the quantity of the lot shift. Although no capacity consideration is accomodated in the model **LOAD(u(j))**, if shift quantities are determined without considering the capacity availability, it becomes very difficult to restore feasibility between lot sizing X^r_{Ejt} and loading Y^r_{Ejkt} decision variables in any Lagrangean iteration r within the framework of the Lagrangean heuristic. Thus, some capacity information is implicitly incorporated into the procedure in determining the lot sizes to be shifted. Thus, the maximum amount that can be backward-shifted from period t to period k (denoted as transbac) is limited by either the lot size in period t or the remaining capacity in period k, and is given by

$$\text{transbac} = \max\{0, \min\{X_{Ejt}, \text{aloc}(E, j) - X_{Ejk}\}\} \qquad (21)$$

Here, $\text{aloc}(E, j)$ is an upper bound for the capacity that can be allocated to item j in the loading problem and is given by

$$\text{aloc}(E, j) = \left(C_{Ek} + CO_E - \min_{m \in Fej}\{s_{Ejm}\}\right) \Big/ \min_{m \in FEj}\{p_{Ejm}\} \qquad (22)$$

Similarly, the amount that can be shifted from period t to period k (denoted as transfor) is restricted by the lot size in period t, by the remaining capacity in period k and more importantly by the backordering limitation given in (4). Obviously, forward shifting will cause backordering and, in any period, backordered amount can not exceed a certain percentage of that period's demand. Thus, it is necessary to determine the minimum backordering limit between t and k-1 by

$$\begin{aligned} mini &= \min_{1=t \ldots k-1}\{\text{backquota}_j\} \text{ where back quota}_j \\ &= BP \times d_{jl}, \text{ initially} \end{aligned}$$

and the maximum forward-shifted amount is defined by

$$\text{transfor} = \max\{0, \min\{\text{mini}, X_{Ejt}, \text{aloc}(E, j) - X_{Ejk}\}\} \qquad (23)$$

This shifting procedure is tried for t, k pairs which have negative marginal costs, starting with the most negative $\text{marcos}_{t,k}$ and proceeding in ascending order until the marginal costs of all remaining possible shifts turn out to be positive, which implies that no improvement can be expected by any further shifting. The whole algorithm is summarized in the pesudocode given in Figure A3 in the Appendix. The same procedure is repeated for all intermediate stages by using the lot sizes of successive stages as the dependent demand
and not permitting backorders in (23). The objective function value **zlot(u)** is obtained by summing **zlot(u(j))** for all j.

The loading heuristic

The solution of the submodel **LOAD(u(e,t))** which is a static and a single-stage problem involves the minimum-cost

assignment of lot sizes of a number of items to non-identical facilities without lot splitting such that regular time and over time capacity limitations are met. It is obvious that when all u_{ejt} are negative so that all objective function coefficients are positive, the solution is trivially not to produce. However, when some u_{ejt} are positive, it will be more cost effective to allocate capacity to some items. Of course, these items will be among those which possess a promising cost factor. Since information about lot sizing decisions is not included into the loading model, a bounding convention is needed to determine the load quantities. One main assumption made in the design of the heuristic is to allocate total regular time capacity of a particular facility to a single item which deserves it most. Thus, all cost_{ejm} factors for possible combinations of item j and facility $m \in F_{ej}$ are computed by

$$\text{cost}_{\text{ejm}} = -u_{ejm}\left(C_{et} - s_{ejm}\right)/p_{ejm} + SC_{ejm} \qquad (24)$$

and only those items with $\text{cost}_{\text{ejm}} < 0$ will be considered as candidates to be assigned to facilities. The most deserving assignment then is made by

$$Y_{ejmt} = \left(C_{et} - s_{emt}\right)/p_{emt} \qquad (25)$$

Letting $J = \{ j : \text{cost}_{\text{ejm}} \leqslant 0 \text{ for some } m \in F_{ej}\}$, the procedure can be summarized as follows:

Step 1: Select item $j^* \in J$ and $m^* \in F_{ej^*}$ with minimum cost_{ejm} over all $j^* \in J$ and $m \in F_{ej^*}$ and load item j^* on facility m^* by determining its lot size by (25).

Step 2: Eliminate the loaded facility m^* and family j^* from further consideration. Repeat Step 1 if all facilities and all items in J are not considered and there still remain some unloaded facilities with idle regular time capacity.

As it is indicated above, only regular time capacity is allocated to the items and overtime capacity is left reserved simply because it is intended to maintain idle capacity so that the search undertaken in the Lagrangean heuristic can execute moves more freely. Since the search is basically geared towards generating a feasible solution, allocating overtime capacity in solving the submodels **LOAD(u(e,t))** will tighten the available capacity levels and make it very difficult to restore feasibility.

This procedure is repeated independently for each stage and time period, and **zload(u)** is obtained by summing **zload(u(e,t))** associated with individual submodels.

The Lagrangean heuristic

An essential design issue in the Lagrangean relaxation approach is to develop a heuristic procedure which will generate a feasible solution from the relaxed solutions of the submodels. It is a rare situation that the solutions of the submodels will ever be feasible with respect to the relaxed constraint. The Lagrangean heuristic developed in this study considers only the solutions of the lot sizing models **LOT(u(j))** as the initial solution and generates a loading plan out of these without taking into account the solutions of the submodels **LOAD(u(e,t))**. This convention is used in order to facilitate the attempts of the Lagrangean heuristic to maintain feasibility.

The loading decisions are made in accordance with the parallel machine scheduling principles. Given the solutions of the models **LOT(u(j))**, a stage/item/period triplet with positive X_{ejt} is chosen randomly and X_{ejt} is assigned to an eligible facility k in F_{ej} which has the least amount of workload already assigned. Then, the associated loading variables are made active (that is, $W_{ejkt} = 1$ and $Y_{ejkt} = X_{ejt}$). This is repeated until all positive X_{ejt} are considered. The resulting initial solution is of course inventory feasible, but most probably capacity infeasible. However the SA procedure incorporated into the heuristic will serve to restore this capacity infeasibility.

Here, the procedure Perturb in Figure A1 in the Appendix is implemented with a modification: A neighbour is identified as in the procedure Perturb, but the principle of executing a move differs from the stand alone SA approach: In the LR/SA procedure, only improving moves are executed unless a number of consecutive moves, denoted by hilcount, result in non-improving solutions. In that case, a single SA iteration is carried out for the next non-improving move where the execution of the move depends on the PA. Then, hilcount is reset to zero and the hill-climbing phase is repeated. Hence, in the LR/SA, one SA iteration alternates with a given number of hill-climbing moves. This method is proposed in order to accelerate the efforts of the Lagrangean heuristic to obtain a feasible solution. With a very limited number of moves as permitted in one Lagrangean iteration, SA does not converge to a feasible solution and the need for a hill-climbing search arises.

From the viewpoint of LR, SA is integrated with LR in order to enhance its performance; in fact, SA imbedded in each iteration not only generates a feasible solution from a considerably hard-to-resolve infeasibility issue, but also serves to provide a good upper bound to be used in the step-size and consequently in the update of Lagrangean multipliers. From the SA point of view, LR provides promising starting solutions and accelerates the search for the optimum.

Experimental design

The solution procedures for the model (P) discussed in the preceding sections are compared by generating a number of hypothetical problems. It is important to note that in any solution to the model **P** setup cost works against holding

and backordering costs while backordering and inventory holding costs act against overtime cost. Therefore, it is crucial to analyze the effect of these parameters by varying their relative magnitudes. Furthermore, the difference between the capacity requirement imposed by the demand process and the available capacity clearly affects the performance of all solution procedures. In fact, important problem attributes which influence the computational effort in locating the optimal solution can be listed as capacity tightness, demand dispersion around the mean, the relative magnitude of holding and overtime costs, the flexibility of the facilities in each stage (that is, the possible number of facilities on which a particular item can be loaded) and the degree of value-added upon the item as it moves through the stages (that is, the relative magnitude of holding costs between consecutive stages). Therefore, different scenarios related with each of these five attributes are designed and possible combinations are generated to test the overall performance. The first five parameters described below are generated randomly compatible with the data of a tiling company in Turkey. The remaining parameters are calculated based on these five. The manner in which problem parameters are generated is described below:

(i) Demand is assumed to possess a seasonal trend with a base level multiplied with the seasonality factor. The base level is generated from two different Normal distributions: the first distribution is normal distribution with a mean of 100 units and a standard deviation (STD) of 10 (that is, N(100,10)) and the second one is N(100,50). Here, the main idea is to reveal the effect of dispersion from the mean demand. The seasonality factor is assumed to depend upon the cycle time of a sine wave and generated from uniform distribution over 1.0 and 1.5 (that is, $U(1.0,1.5)$ for both base level cases.

(ii) Processing times in hours are generated from identical uniform distributions $U(1,5)$.

(iii) The possible number of facilities denoted as Maxfac on which a particular item can be loaded in each stage is taken as 2 and 3 in small problems and as 3 and 5 in large problems. A Maxfac equal to the number of facilities in a stage indicates that all facilities are of the same class resulting in maximum flexibility for the loading problem.

(iv) Set-up cost is randomly generated from U(300,500).

(v) Since it is assumed without loss of generality that each successive operation in successive stages contributes added-value to the material, the holding cost of the work-in-process is designed to increase between successive stages. Therefore, the inventory costs are generated successively. First h_{ljt} is generated randomly from U(1,20). Then, the holding costs in each stage are generated by $h_{ejt} = VAP.h_{e-I,jt}$ for $e \geqslant 2$ where VAP is the consecutive value-added percentage. In one scenario, VAP is taken as 1.1 while in a more accelerated one it is taken as 1.3.

(vi) The backordering cost for all items are defined by $\pi_{jt} = 1.25.h_{Ejt}$.

(vii) Set-up times are related to total processing time and are defined by

$$s_{ejk} = S\left[\sum_t \sum_j \sum_{k \in F_{ej}} p_{ejk} d_{jt}\right] \Big/ T.\text{Maxfac}$$

where S is a random number generated from U(0.05, 0.10) and T is the length of the planning horizon.

(viii) An important attribute directly affecting the performance of the procedures is the capacity tightness. First of all, the average capacity requirement per each facility in each stage is determined by

$$C_e = \max\left\{\sum_{t=1}^{\tau} \sum_j \sum_{k \in F_{ej}} p_{ejk} d_{jt}/\tau.\text{Maxfac}\right\}.$$

Then, the capacity in each stage in each period is found by $C_{et} = CAT.C_e$ where CAT is the capacity tightness percentage. In a "tight" capacity scenario CAT is taken as 1.2 whereas in a loose capacity situation CAT is taken as 1.6.

(ix) Overtime capacity limit CO_e is taken as one-third of regular time capacity C_{et}.

(x) The overtime cost θ_{ekt} is determined by solving the holding cost/overtime cost ratio HOR_e target value in stage e for

$$HOR_e = \sum_j h_{ejt} \Big/ \theta_{ekt}\left(\sum_j \sum_{k \in F_{ej}} p_{ejk}\right)/\text{Maxfac}.$$

Two levels of HOR_e are considered as 0.5 and 1.5.

Considering the five attributes mentioned above (HOR_e, CAT, VAP, MaxFac, STD) there arise ($2 \times 2 \times 2 \times 2 \times 2 = 32$) factor combinations of manufacturing scenarios to be tested for each problem. Then two different sets of test problems with different dimensionality are generated. Smaller problems include 5 items, 3 stages, 3 facilities in each stage, and 6 time periods. Each small test instance involves 270 binary and 534 continuous variables, whereas each large problem has 20 items, 4 stages, 5 facilities in each stage, and 6 periods, which result in 2400 binary and 3600 continuous variables. Both sets of problems are generated according to the factorial design described above. 3 replications are generated for small problems while only one replication is generated for the larger problems. Therefore, $32 \times 3 = 96$ small and $32 \times 1 = 32$ large problems are generated.

SA is applied 10 times to small problems in a stand alone fashion (the number of random seeds is equal to 10) each time with a new initial solution and the number of moves in each run for small problems is limited by 300,000. However,

Table 1 Values for the GA/SA parameters

GA/SA parameters	*Small problems*	*Large problems*
PopSize	40	70
Frac	0.10	0.06
Number of random seeds	3	1
PopNo	4	3
MaxNo	3,000,000	12,000,000
Number of SA moves/chromosome	15,000	50,000

the number of moves for large problems is limited by 1,200,000 while the number of seeds is again taken as 10.

The values of the GA/SA parameters used in the experimentation are given in Table 1.

In the LR/SA, 100 and 150 Lagrangean iterations are permitted for small problems and large problems, respectively. In small problems, the Lagrangean heuristic within each Lagrangean iteration is executed with 13,500 moves while the number of hill-climbing/SA moves within each Lagrangean iteration is limited by 50,000 in large problems. The number of seeds for the LR/SA is 6 and 4 in small and large problems, respectively. The size of the tabu list used in the procedure Perturb in small problems is 15 while it is 30 in large problems. The number of consecutive non-improving moves (given by hilcount) in the LR/SA is taken as 25 in small problems and as 50 in large problems.

Then, it is intended to compare the performances of the solution procedures against a commercial optimizer. Although any specialized branch and bound or cut procedure could be used to serve this purpose, without loss of generality, GAMS XA-Solver is utilized here since almost all mixed integer linear programming codes use a branch and bound framework with linear programming relaxation (Nemhauser and Wolsey[37]). The test problems are solved with a maximum limit of 1,000,000 GAMS iterations for small problems and 5,000,000 iterations for large problems. All procedures are coded in C++ and the computation times are observed on a Pentium 233 MMX computer. The results for the small and large problems are provided in Tables 2 and 3, respectively.

In Tables 2 and 3, avg, std, and max denote the average percentage deviation from the best solution obtained by any one of the procedures, the percentage standard deviation and the maximum percentage deviations from the best solution,

Table 2 Comparison of different solution procedures (small problems)

	avg	*std*	*max*	*NoBest*	*NoInfeas*	*CPUsec*
GAMS	0.708	0.502	2.311	3	0	2402.663
GA/SA	0.062	0.045	0.197	9	5	211.355
SA	0.066	0.066	0.431	16	5	49.933
LR/SA	0.016	0.038	0.177	68	4	49.815

Table 3 Comparison of different solution procedures (large problems)

	avg	*std*	*max*	*NoBest*	*NoInfeas*	*CPUsec*
GAMS	0.828	0.553	1.827	0	0	45159.071
GA/SA	0.304	0.048	0.366	0	0	533.812
SA	0.205	0.050	0.259	0	0	307.687
LR/SA	0.000	0.000	0.000	32	0	302.593

respectively. The number of best solution instances found by each procedure is expressed by NoBest while the number of problem instances in which a feasible solution is not found is given by NoInfeas for each procedure.

Comparing the results in Tables 2 and 3 respectively, we can conclude that the LR/SA contributes significantly to the performance of the stand alone SA despite the fact that its average computation time is of the same order. This is mainly due to the fact that LR, being a convergent algorithm by itself, provides a good starting solution for SA in each iteration. It is important to note that the number of moves permitted in each SA procedure in LR/SA is very small as compared to the stand alone SA (13500 versus 300000 in small problems and 50000 versus 1200000 in large problems). The negative effect of limiting the number of moves is compensated by incorporating hill-climbing moves. The effect of LR on SA is most prominently observed in Table 3 where the results of the larger problems are displayed. Here, the LR/SA finds the best solution in all of the problems.

The important parameters in GA/SA are set based on previous experimentation done on the single stage version[12] of the problem considered here and on some preliminary work. In general the solution quality of pure GA procedures improves as the population size and the number of generations increases. However the population sizes are chosen rather small in GA/SA since GA is integrated with the SA procedure which compensates the latter and the procedure converges to a feasible solution. Moreover, CPU times of GA/SA are prohibitive as is. As far as the GA/SA is concerned, it does not contribute significantly to the stand alone SA and its computation time is significantly larger. However, its performance with respect to both criteria is better as compared to GAMS. In fact, in both Tables 2 and 3, the ranking of GAMS and the three heuristics demonstrate the general performance trend for difficult combinatorial optimization problems. Branch and bound procedures have the worst performance due to their exponential complexity when employed for large problems (240 0–1 variables in a small problem has a high complexity order). GAs are robust techniques, but they have a slow convergence rate. On the other hand, SA requires a minimal number of calculations and its memory requirements are very low. Thus, SA enhanced with LR, has an accelerated convergence rate and results in the best performance.

Statistical analysis is conducted to reveal the impact of key problem parameters (HOR_e, CAT, VAP, MaxFac, STD) upon the performance of the algorithms. The *t*-tests show that HOR_e, VAP, and STD have no significant effect upon the solution quality; however, CAT and MaxFac turn out to be the problem attributes with significant impacts on all four procedures. As CAT and MaxFac are increased expanding the feasible region, the performances of GAMS and LR/SA improve while those of GA/SA and stand-alone SA deteriorate. This observation justifies the fact that search procedures may diverge as the feasible space expands.

Tables 2 and 3 could also display the results of stand alone GA and stand alone LR procedures with a Lagrangean heuristic which does not incorporate global search approach. In fact, in preliminary experimentation, the authors also employed stand alone versions of both methods which ended up with disappointing results. Without SA, neither GA nor LR converged; namely, LR identified feasible solutions to at most 20% of the small test problems whereas GA could provide none. Therefore, these results are not displayed.

Conclusion

In this research, a mathematical model is developed to describe the multi stage capacitated lot sizing and loading problem often encountered in flow type manufacturing environments. Within this model, one can explicitly consider the loading issue related to multiple facilities of different classes (representing different machine types/technologies) existing within a department. The model is of practical use particularly because loading issues are considered in an aggregate planning framework.

Efficient hybrid heuristics are developed here to solve this difficult problem. The first hybrid consists of integrating two meta-heuristics, GA and SA, while the second one consists of integrating a meta-heuristic SA and Lagrangean relaxation LR. The performance of these hybrid heuristics as well as stand alone SA on hypothetical test problems of different sizes demonstrate that the hybrid LR/SA procedure provides the best results both with respect to solution quality and computation time. LR approaches are problem-dependent since they are coupled by a Lagrangean heuristic which is specifically designed for the problem under question, and the performance of LR is strongly dependent on this heuristic. Imbedding a meta-heuristic which is general in nature into LR provides a global perspective for the LR technique and eliminates the problem-dependent nature of the Lagrangean heuristic by enhancing the relaxation capability of the LR. As future work, the hybrid LR/SA approach can be imbedded within a branch and bound algorithm and potential improvements can be reckoned. In such a design, information gathered during branching at each node can be included into the SA procedure to direct the search towards better moves.

Appendix

Procedure **Perturb**

MaxMove: Permitted number of moves
Counter: move counter
facno: number of facilities
minΔ: smallest incremental cost over all facilities
Cost: cost of the current solution
tabu: a function which stores the costs of last Tabusize solutions
Δ: amount to be shifted
Δ_{cost}: incremental cost implied by the shift (for $e < E$, $BP = 0.0\%$)
$idle_{kt}$: idle capacity of facility k in period t

Initialize tSA = 1; tabulist = []; counter = 0;
Generate Initial Solution; (Calculate Cost);
do {
  counter = counter + 1;
  minΔ = a big number;
  Randomly select e^*, j^*, t^* where $X_{e^*j^*t^*} > 0$ is to be reduced;
  Randomly select period t where $X_{e^*j^*t^*}$ is to be augmented ($t = t^*$, $t = t^* - 1$, or $t = t^* + 1$);
  Identify the facility k on which $X_{e^*j^*t^*}$ is loaded;
  for $f = 1$ to facno do
    if ($f \in F_{ej^*}$) AND ($\Sigma_{i=f} Y_{e^*j^*t} = 0$){
      if $t > t^*$, $\Delta = \min\{X_{e^*j^*t}, I_{e^*j^*t} + BP.d_{e^*j^*t}\}$;
      if ($t < t^*$) AND ($e^* > 1$), $\Delta = \min\{X_{e^*j^*t}, I_{e^*-1,j^*t}\}$;
      if $t \neq t^*$, $\Delta \leftarrow$ random(Δ);
      Calculate Δ_{cost};
      if ($\Delta_{cost} < \min\Delta$) AND (Cost + $\Delta_{cost} \neq$ tabu){
        $f^* \leftarrow f$; $\min\Delta \leftarrow \Delta_{cost}$; $\Delta^* \leftarrow \Delta$;
        }
      }// if
  if ($\min\Delta > 0$), calculate p, PA;
  if ($\min\Delta < 0$) or ($\min\Delta > 0$ and $p < PA$){
    update tabulist (remove first element and add Cost as last element);
    if ($\min\Delta > 0$)tSA$t \leftarrow t$SA$/(1 + \beta.\ t$SA);
    if (last M moves are improving) reduce β;
      else augment β;
    Cost = Cost + minΔ;
    if ($t = t^* - 1$){
      Increase $I_{e^*j^*t}$ by Δ^*;
      Decrease $I_{e^*-1j^*t}$ by Δ^*;
      };
    If ($t = t^* + 1$){
      Decrease $I_{e^*j^*t}$ by Δ^*;
      Increase $I_{e^*-1j^*t}$ by Δ^*;
      };
    if ($X_{e^*j^*t} = 0$) decrease $idle_{f^*t}$ by $[s_{e^*j^*f} + p_{e^*j^*f}\Delta^*]$
      else decrease $idle_{f^*t}$ by $[p_{e^*j^*r}\Delta^*]$;
    Decrease $X_{e^*j^*t}$ by Δ^*;
    Increase $X_{e^*j^*t}$ by Δ^*;
    if ($X_{e^*j^*t^*} = 0$) decrease $idle_{kt^*}$ by $[s_{e^*j^*k} + p_{e^*j^*k}\Delta^*]$
      else decrease $idle_{kt^*}$ by $[p_{e^*j^*fk}\Delta^*]$;
    }// end move
  } while (counter < MaxMove);

Figure A1 Pseudocode for the procedure **Perturb**.

Procedure **GA/SA**

```
PopNo: number of parallel populations
GenNo: number of generations to go without migration
Popsize: number of chromosomes in a population
frac: fraction of population undergoing SA
PopRange: ratio of minimum objective function value to the maximum in the population
MaxNo: total number of SA moves permitted
α, λ: constants less than 1.0

for i = 1 to PopNo do
  generate PopSize chromosomes;
do {
  for i = 1 to PopNo do
    for j = 1 to GenNo do {
      crossover;
      if PopRange < α then mutation;
      calculate PopRange;
      if PopRange < λ then
      do SA for (frac) (PopSize) (PopRange} different chromosomes;
      }
    MixCross;
    } while (number of SA moves < Maxno);
```

Figure A2 Pseudocode for the procedure **GA/SA**.

Procedure **FindX**

```
X_Ejt = d_jt for ∀j, t;
Calculate the cost for the solution: Cost = u_Ejt × X_Ejt;
Calculate the backorder quota for all t: backquota_t = BP × d_jt;
Calculate the marginal cost of meeting one unit of d_jt in period k
  If k ≥ t, marcos_tk = Σ_{q=t}^{k} π_jq − u_Ejt + u_Ejk
  else kt, marcos_tk = Σ_{q=k}^{t} h_Ejq − u_Ejt + u_Ejk
Get the sorted list of (t, k) pairs by arranging marcos_kt in ascending order and call it smarc_tk;
Set counter i = 1;
while (smarc[i]_tk < 0)} {
  If X_Ejt > 0 {
    If t ≥ k{// move the lot to period k, keep inventory
    Increase X_Ejk by transbac;
    Cost=Cost+transbac×smarc[i]_tk
    Decrease X_Ejt by transbac;
    }
        else{// move as much as possible to period k, carry backorder
        Increase X_Ejk by transfor;
        Decrease X_Ejt By transfor;
        Decrease backquota_l, l = t, ..., k − 1 by transfor;
        Cost=Cost+transfor smarc[i]_tj;
        }
  }//X_Ejt > 0
    i = i + 1;
  }// while
Return(Cost);
```

Figure A3 Pseudocode for the procedure **FindX**.

References

1 Jones G and Roberts M (1990). *Optimised Production Technology (OPT).* IFS Publications. Bedford, UK.
2 Hax AC and Candea D (1984). *Production and Inventory Management,* Prentice-Hall, Englewood Cliffs, NJ.
3 Dobson G (1987). The economic lot scheduling problem: achieving feasibility using time-varying lot sizes. *Ops Res* **35**: 764–777.
4 Philipoom PR, Rees LP and Taylor III BW (1989). Solving the economic lot scheduling problem using the method of prime subperiods. *Dec Sci* **20**: 794–810.
5 Gallego G and Joneja D (1994). Economic lot scheduling problem with raw material considerations. *J Opl Res Soc* **42**: 92–101.
6 El-Najdawi MK (1994). A job-splitting heuristic for lot size scheduling in multi-stage multi-product production processes. *Eur J Opl Res* **75**: 365–377.
7 Trigeiro WW, Thomas LJ and McLain JO (1989). Capacitated lot sizing with setup times. *Mgmt Sci* **35**: 353–366.
8 Dixon PS, Elder MD, Rand GK and Silver EA (1983). A heuristic algorithm for determining lot sizes of an item subject to regular and overtime production capacities. *J Ops Mnt* **3**: 121–130.
9 Diaby M, Bahl HC, Karwan MH and Zionts S (1992). A Lagrangean relaxation approach for very large scale capacitated lot sizing. *Mgmt Sci* **38**: 1329–1340.
10 Özdamar L and Bozyel MA (1997). The capacitated lot sizing problem with overtime decisions and setup times. Working Paper, Istanbul Kültür University.
11 Drexl A and Kimms A (1997). Lot-sizing and scheduling-Survey and extensions. *Eur J Op Res* **99**: 221–235.
12 Özdamar L and Birbil IS (1998). Hybrid heuristics for the capacitated lot sizing and loading problem with setup times and overtime decisions. *Eur J Op Res* **110**: 525–547.
13 Kolen AWJ and Kroon LG (1991). On the computational complexity of the (maximum) class scheduling. *Eur J Opl Res* **54**: 23–38.
14 Cheng TCE and Sin CCS (1990). A state of the art review of parallel machine scheduling research. *Eur J Opl Res* **47**: 271–292.
15 Fleischmann B (1990). The discrete lot sizing and scheduling problem. *Eur J Opl Res* **44**: 337–348.
16 Salomon M, Kroon LG, Kuik R and Van Wassenhove LN (1991). Some extensions of the discrete lot sizing and scheduling problem. *Mgmt Sci* **37**: 801–811.
17 Cattrysse D, Salomon M, Kuik R and Van Wassenhove LN (1993). A dual ascent and column generation heuristic for the discrete lot sizing and scheduling problem with setup times. *Mgmt Sci* **39**: 477–486.
18 Jordan C and Drexl A (1996). Discrete lot sizing and scheduling by batch sequencing. Working Paper, University of Kiel.
19 De Matta R and Guignard M (1994). Dynamic Production Scheduling for a Process Industry. *Ops Res* **42**: 492–503.
20 Özdamar L and Bozyel MA (1998). Simultaneous lot sizing and loading of product families on parallel facilities of different classes. *Int J Prod Res* **36**: 1305–1324.
21 Özdamar L, Ath AÖ and Bozyel MA (1996). Heuristic family disaggregation techniques in hierarchical production planning systems. *Int J Prod Res,* **34**: 2613–2628.
22 Kirkpatrick A, Gelatt CD and Vechi MP (1983). Optimization by simulated annealing. *Mgmt Sci* **30**: 671–680.
23 Holland JH (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI.
24 Michalewicz Z (1994). *Genetic Algorithms + Data Structures = Evolution Programs*, Springer: Verlag, Berlin.
25 Özdamar L and Barbarosoğlu G (1998). An integrated Lagrangean relaxation-simulated annealing method for the multi-level multi-item capacitated lot sizing problem. In: Grubbström RW (ed). *Proceedings of the International Working Seminar on Production Economics*. Austria. 603–617.
26 Thizy JM and Van Wassenhove LN (1985). Lagrangean relaxation for the multi-item capacitated lotsizing problem: A heuristic implementation. *IIE Trans* **17**: 308–313.
27 Chen WH and Thizy JM (1990). Analysis of relaxations for the multi-item capacitated lot sizing problem. *Annals Ops Res* **26**: 29–72.
28 Lozano S, Larrancta J and Onieva L (1991). Primal-dual approach to the single level capacitated lot sizing problem. *Eur J Opl Res* **51**: 354–366.
29 Billington PJ (1983). Multi-level lot sizing with a bottleneck work center. Ph.D. Thesis. Cornell University.
30 Tempelmeir H and Derstroff M (1996). A Lagrangean-based heuristic for dynamic multi-item multi-level constrained lot sizing with setup times. *Mgmt Sci* **42**: 738–757.
31 Dowsland KA (1993). Some experiments with simulated annealing techniques for packing problems. *Eur J Opl Res* **68**: 389–399.
32 Grefenstette J (1986). Optimization of control parameters for genetic algorithms, *IEEE Trans on Sys, Man and Cyber,* SMC-16.
33 Srinivas M and Patnaik LM (1994). Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Trans on Sys, Man and Cyber* **24**: 656–667.
34 Özdamar L (1999). A genetic algorithm approach to a general category project scheduling problem. *IEEE Trans on Sys, Man and Cyber* **29**: 44–59.
35 Fisher ML (1981). The Lagrangean relaxation method for solving integer problems. *Mgmt Sci* **27**: 1–18.
36 Held M, Wolfe P and Crowder HP (1974). Validation of subgradient optimization. *Mathe Program* **6**: 62–88.
37 Nemhauser GL and Wolsey LA (1988). *Integer and Combinatorial Optimization.* John Wiley and Sons. New York.

Received June 1998;
accepted March 1999 after one revision