ELSEVIER

European Journal of Operational Research 110 (1998) 525–547

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Theory and Methodology

Hybrid heuristics for the capacitated lot sizing and loading problem with setup times and overtime decisions

Linet Özdamar *, Şevket İlker Birbil

Department of Systems Engineering, Yeditepe University, Gayrettepe, Emekli Subay Evleri 23/5, Istanbul, Turkey

Received 1 February 1997; accepted 1 July 1997

Abstract

The capacitated lot sizing and loading problem (CLSLP) deals with the issue of determining the lot sizes of product families/end items and loading them on parallel facilities to satisfy dynamic demand over a given planning horizon. The capacity restrictions in the CLSLP are imposed by constraints specific to the production environment considered. When a lot size is positive in a specific period, it is loaded on a facility without exceeding the sum of the regular and overtime capacity limits. Each family may have a different process time on each facility and furthermore, it may be technologically feasible to load a family only on a subset of existing facilities. So, in the most general case, the loading problem may involve unrelated parallel facilities of different classes. Once loaded on a facility, a family may consume capacity during setup time. Inventory holding and overtime costs are minimized in the objective function. Setup costs can be included if setups incur costs other than lost production capacity. The CLSLP is relevant in many industrial applications and may be generalized to multi-stage production planning and loading models. The CLSLP is a synthesis of three different planning and loading problems, i.e., the capacitated lot sizing problem (CLSP) with overtime decisions and setup times, minimizing total tardiness on unrelated parallel processors, and, the class scheduling problem, each of which is NP in the feasibility and optimality problems. Consequently, we develop hybrid heuristics involving powerful search techniques such as simulated annealing (SA), tabu search (TS) and genetic algorithms (GA) to deal with the CLSLP. Results are compared with optimal solutions for 108 randomly generated small test problems. The procedures developed here are also compared against each other in 36 larger size problems.

1. Introduction

In the literature, lot sizing and scheduling 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 medium range planning level whereas scheduling decisions are dealt with in the short term planning level (Hax and Candea, 1984).

Lot sizing models are grouped together within the single stage/multi-stage capacitated lot sizing model (CLSP) which assumes that demand is

* Corresponding author. Fax: 90 216 387 9108; e-mail: ozdamar@yeditepe.edu.tr.

0377-2217/98/$19.00
PII S0377-2217(97)00269-5

deterministic and dynamic over a finite planning horizon; limited capacity is shared by all items produced in each period and no backorders are permitted. Furthermore, in each period that an item/family is produced, a setup cost is incurred. The objective function minimizes setup and inventory carrying costs. In some single stage CLSP formulations setups also consume capacity (Trigeiro et al., 1989) and regular time capacity is expanded by overtime capacity (Dixon et al., 1983). The single stage CLSP with setup times and overtime decisions is investigated by Diaby et al. (1992) and Özdamar and Bozyel (1996).

Much computational effort is required for solving the single stage CLSP, because it has been shown to be NP-hard (Bitran and Yanasse, 1982). When setup times are included in the model, finding a feasible solution to the CLSP also becomes an NP-Complete problem (Garey and Johnson, 1979). Researchers developed period by period heuristics (Lambrecht and Vanderveken, 1979; Dixon and Silver, 1981; Maes and VanWassenhove, 1986; Günther, 1987) and mathematical programming based methods (Thizy and Van Wassenhove, 1985; Trigeiro et al., 1989; Cattrysse et al., 1990; Kirca and Kökten, 1994) for the single stage CLSP. Naturally, it is even harder to solve the multi-stage CLSP where different product structures exist and the lot sizing decisions have to be made for multiple stages of production. Researchers exclude certain modeling features from the multi-stage CLSP in order to devise efficient solution procedures. Maes et al. (1991) assume an assembly product structure and do not permit capacity sharing by mutiple levels. Setup times (but not overtime) are included in the model. Kuik et al. (1993) also assume an assembly product structure, but setup times and overtime are both excluded and only a single capacitated stage is assumed to exist among all stages. Billington et al. (1986) consider a general product structure. A single capacity bottleneck is assumed to exist and only setup times are included. Billington et al. (1994) simplify the latter model by excluding setup times and overtime, and by assuming a serial product structure and uniform process times. However, all stages are now capacitated. Tempelmeier and Helber (1994) consider a general product structure and permit capacity sharing among multiple levels and also restrict capacity at each stage. Tempelmeier and Derstroff (1996) extend the latter model by including setup times.

In the single and multi-stage CLSP, capacity constraints are aggregate and loading issues are not considered. In other words, capacity is assumed to belong to a single facility. However, in practice loading decisions are at times crucial in the minimization of total costs. The loading decisions discussed here involve the assignment of family lots to facilities when there are parallel facilities in a production stage. The assignment problem becomes NP-hard when facilities are unrelated (family process times depend on facilities) and/or there exist different classes of facilities and each job cannot be processed on every facility resulting in a class scheduling problem (Kolen and Kroon, 1991). For a general review on the parallel machines problem the reader is referred to Cheng and Sin (1990) 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.

Two assignment problems in the context of production planning are given here. Identical parallel facilities are assumed to exist, yet, an aggregate treatment of capacity might lead to infeasible/suboptimal solutions. (i) In the tiling industry the bottleneck on a production line is usually the kiln. Here, the number of active kilns is minimized over the planning horizon due to the high energy costs incurred by active kilns. In this case, the minimum possible number of active facilities is searched for, and the loading decisions have to be considered in detail as the problem solved becomes the bin packing problem (Johnson, 1974) which is known to be NP-hard (Garey and Johnson, 1979). (ii) When the capacity of a facility is not homogeneous, i.e., overtime as well as regular time capacity exists for each facility and overtime costs are to be minimized, given a set of nonsplittable lot sizes, the loading problem reduces to the parallel machine problem with a common due date (the due date for all jobs is equivalent to the regular time capacity) where total tardiness (the amount of work load on a facility in excess of regular time capacity) is minimized. This model

is valid when parallel production lines are operated in a factory and an overtime option exists to expand capacity. The two models considered just now become even more difficult when unrelated parallel facilities of multiple classes are considered instead of identical ones.

Regarding the discussion above we conclude that both the lot sizing and loading problems are difficult to solve even when decomposed into two problems. The solutions to the decomposed models might be considerably suboptimal for the integrated lot sizing and loading problem. Furthermore, they might even prove to be capacity infeasible. Inevitably, lot sizes determined in a higher level of planning affect the solution to the loading problem as 'job' process times are determined by the lot sizes. An apriori 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 independently. Consequently, we formally describe the CLSLP here and propose hybrid heuristics which attack the integrated problem, rather than decompose it into two subproblems.

Recent literature on the integrated treatment of lot sizing and scheduling is mostly focused on discrete lot sizing and scheduling (DLSP), economic lot sizing and scheduling (ELSP) and more complex models integrating the features of the job-shop scheduling problem and the lot sizing problem.

The main assumption of the DLSP is that only one item can be produced in each discrete time period and full capacity has to be utilized whenever a lot size is positive. A lot size may cover several time periods and a setup cost is incurred only at the beginning of the production batch, but setup times are ignored. The 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 (Fleischmann, 1990). Complexity analysis of the DLSP is investigated by Salomon et al. (1991) and variants of the DLSP including sequence-dependent and sequence-independent setup times are considered by Cattrysse et al. (1993) and Jordan and Drexl (1996). De Matta and Guignard (1994) deal with the DLSP with no setup times and with splittable lots on multiple production lines of different classes. However, the reported performance of their Lagrangian based heuristic is not compared with the optimal solution but it is measured against the gap between the feasible schedule found from the Lagrangean solution and the tightest lower bound over the best integer solution. Furthermore, all randomly generated test problems assume one of the two family/facility class patterns given in their article.

The ELSP assumes that time is continuous, demand is constant, capacity is limited and the length of the planning horizon is infinite. Kim and Mabert (1995) and Duplaga et al. (1996) consider models integrating the DLSP and ELSP whereas Dauzere-Peres and Laserre (1994) include lot sizing decisions in a job-shop environment. Özdamar and Bozyel (1997) deal with the single period lot sizing and bin packing problem where the lot sizing decisions are made simultaneously with the loading decisions by using a lot sizing algorithm developed in a previous study (Özdamar et al., 1996). Given a set of lot sizes, the loading decisions are made by using a hybrid search heuristic incorporating features from tabu search (TS) (Glover, 1989, 1990) and simulated annealing (SA) (Kirkpatrick et al., 1983).

The model of the Capacitated lot sizing and loading problem (CLSLP) considered here subsumes the CLSP with set up times and overtime and the unrelated parallel facility loading problem with facilities of multiple classes. Unlike the standard CLSP, properties involving the exact sequence of lots loaded on a facility (e.g., sequence-dependent setups) may be incorporated into the CLSLP, since the capacity constraints in the model require detailed information on the loading of lots on facilities. Hence, if required, the CLSLP may incorporate DLSP assumptions such as "a setup is required only in the starting period of a batch production run which continues during multiple consecutive periods". With respect to the latter property and the fact that the CLSLP permits multiple items to be produced in

one period and that lot sizes are not restricted to null or full capacity, the CLSLP also subsumes the DLSP.

2. Mathematical formulation of the CLSLP

The CLSLP is concerned with determining the lot sizes of multiple product families or end items over a planning horizon. Demand is known with certainty but varies over time. Lot sizes should be assigned within available capacity limits which may include overtime capacity as well. Each time a lot size is positive, capacity is consumed by a set-up. Setup costs may take place in the objective function if setups incur costs other than that of capacity consumption. The loading environment may involve identical, uniform or unrelated parallel facilities of the same or of different classes. The objective function may involve inventory holding, overtime and setup costs as well as costs specific to the loading problem.

The generalized formulation of the CLSLP is given in Table 1. The parameters and variables used in the table are as follows.

Parameters:

ST_{jk}	setup cost of family j on facility k
d_{jt}	external demand of family j in period t
p_{jk}	process time of family j on facility k (h)
co	overtime cost per hour
C_{kt}	regular time capacity of facility k in period t
CO_{kt}	overtime capacity of facility k in period t
h_j	holding cost per unit of family j per period (cost of work in process in stages prior to the last)
s_{jk}	setup time required for family j on facility k
$E1$	cost of keeping a facility active per period
$E2$	cost of activating a facility after a shut down
M	a large number
F_j	set of facilities on which family j can be loaded
g_{jx}	gozinto factor (number of units of family j required to produce one unit of family x; can be a real number)
$S(j)$	set of families which receive family j as input
L_k	set of families which can be loaded on facility k

Variables:

O_{kt}	overtime capacity used on facility k in period t
I_{jt}	inventory of family j held at the end of period t (work in process inventory in stages prior to the last stage)
y_{jkt}	production lot size of family j loaded on facility k in period t
w_{jkt}	binary variable indicating that family j is loaded on facility k in period t
z_{kt}	Binary variable indicating that facility k is active in period t
v_{kt}	Binary variable indicating that facility k is active in period t and inactive in period $t-1$

The formulation in Table 1 is flexible in the sense that it can be easily adapted to a model designed for a specific production environment. For instance, the model can be converted to a single stage model by omitting from Eq. (2) the component with the gozinto factor, g_{jx}. In the multi-stage case, families in a stage are recognized by their indices j. For example, if three families are produced in the first stage, the indices of the families in the second stage start from four. The product structure is described by the gozinto factors and is not restricted to any particular design, e.g., it may be serial, assembly type or generalized, including intrees and outtrees. Families in between stages may also have independent demand. In Eq. (3) the capacity of each facility may be augmented by overtime if the overtime capacity limit, CO_{kt} is positive. Furthermore, the loading problem becomes a bin packing problem if z_{kt} are free to adopt a value of zero. Otherwise, z_{kt} are set permanently to a value of one and the costs of active facilities and reactivation costs, $E1$ and $E2$, become constant. If the bin packing formulation is adopted, then a positive value for $E2$ and Eq. (7) are not compulsory, because activating a facility after shutting it down may not involve extra costs. The parallel facilities become identical if

Table 1
Mathematical formulation of the general CLSLP

Model P

$$\min \quad \sum_j \sum_t I_{jt} h_j + E1 \sum_t \sum_k z_{kt} + E2 \sum_k \sum_t v_{kt} + \sum_k \sum_t \mathrm{co} O_{kt} + \sum_k \sum_t \sum_j \mathrm{ST}_{jk} w_{jkt} \tag{1}$$

$$\text{s.t.} \quad I_{jt-1} + \sum_{k \in F_j} y_{jkt} - I_{jt} = \sum_{x \in S(j)} g_{jx} \sum_{k \in F_x} Y_{xkt} + d_{jt} \quad \forall j, t \tag{2}$$

$$\sum_{j \in L_k} p_{jk} y_{jkt} + s_{jk} w_{jkt} \leqslant (C_{kt} + O_{kt}) z_{kt} \quad \forall k, t \tag{3}$$

$$O_{kt} \leqslant \mathrm{CO}_{kt} \quad \forall k, t \tag{4}$$

$$\sum_{k \in F_j} w_{jkt} \leqslant 1 \quad \forall j, t \tag{5}$$

$$y_{jkt} \leqslant M w_{jkt} \quad \forall j, k, t \tag{6}$$

$$z_{kt} - z_{kt-1} \leqslant v_{kt-1} \quad \forall k, t \tag{7}$$

$$z_{kt}, v_{kt}, w_{jkt} = 0, 1 \quad \forall k, t, j \tag{8}$$

p_{jk} are independent of the facility index k. If L_k consists of all families for every facility k, then all facilities belong to the same class. Note that families belonging to different levels in the product structure may share the capacity of the same stage.

The relaxation of Eq. (5) permits lots to be split among different facilities in the same period. Eq. (6) may be omitted if setup times do not consume capacity.

Here, we assume that all facilities are active in each period, i.e., we do not solve the bin packing problem. The model is assumed to have a single stage and facilities have both regular and overtime capacity. Lots are not divisible and setup times consume capacity, but they do not incur costs other than capacity consumption costs. The resulting problem becomes more difficult than the explicit setup costs case, because setups are penalized implicitly through overtime hours rather than explicitly. The facilities are unrelated and of different classes. The model considered here is briefly given in Table 2.

Bozyel and Özdamar (1996) deal with the single stage CLSLP with a bin packing type of loading problem (excluding overtime capacity) with positive $E1$ and $E2$ parameters and facility independent family process times. A hybrid TS/SA heuristic is developed for the latter problem.

Table 2
Mathematical formulation of the CLSLP considered here

Model P1

$$\min \quad \sum_j \sum_t I_{jt} h_j + \sum_t \sum_k \mathrm{co}\ Okt \tag{9}$$

$$\text{s.t.} \quad I_{jt-1} + \sum_{k \in F_j} y_{jkt} - I_{jt} = d_{jt} \quad \forall j, t \tag{10}$$

$$\sum_{j \in L_k} p_{jk} y_{jkt} + s_j w_{jkt} \leqslant (C_{kt} + O_{kt}) \quad \forall k, t \tag{11}$$

$$O_{kt} \leqslant \mathrm{CO} \quad \forall k, t \tag{12}$$

$$\sum_{k \in F_j} w_{jkt} \leqslant 1 \quad \forall j, t \tag{13}$$

$$y_{jkt} \leqslant M w_{jkt} \quad \forall j, k, t \tag{14}$$

$$w_{jkt} = 0, 1 \quad \forall k, t, j \tag{15}$$

Results are compared against optimal solutions for both nonsplittable and splittable lots as well as for facilities of a single class and of multiple classes. The authors report that the problem to be solved becomes most difficult when lots are splittable and all facilities are of the same class.

3. A general outline of the solution procedures

The solution procedures developed here are based on compu-search methods (Portmann, 1996b) such as SA, TS and GA (Holland, 1975; De Jong, 1980). Extensive reviews of GAs are given by Goldberg (1989) and Michalewicz (1994). Here, we integrate into a GA a local search scheme which incorporates features from TS and SA.

In the GA implementation for the CLSLP, crossover and mutation operators which preserve both inventory and capacity feasibility are difficult to conceive and furthermore, computationally expensive. For instance, if the lot sizes of two parent chromosomes are crossed over, then the swapped genes are likely to lead to infeasible inventory levels. If these infeasibilities are repaired, the solution may end up with capacity infeasibilities.

We develop GA operators which repair only inventory level infeasibility. The capacity infeasibilities implied by a set of lot sizes are penalized in proportion to the work load in excess of the sum of regular and overtime capacities. Consequently, the GA usually conducts the search within the infeasible space in the initial phases of the search. Occasionally, when the population tends to get stuck to the same area of the feasible/infeasible solution space, the GA activates a hybrid nonrestrictive TS/SA procedure (NTSSA) which carries out local search on randomly selected chromosomes in the current population. (The adjective nonrestrictive is assigned to this procedure, because capacity feasibility is not considered while making moves.) A chromosome randomly selected by NTSSA is replaced by the best solution obtained during local search.

A dynamic proportion of the population is processed by NTSSA. The NTSSA works on *frac*PopRange*PopSize* number of chromosomes with *different* objective function values, where *frac* is a fraction of the population size defined by the user and *PopRange* is the ratio of the minimum objective function value among the chromosomes to the maximum one. Thus, when the diversity of the population is lost, NTSSA processes more chromosomes, because the selected chromosomes are expected to change considerably. (To save on computation time, the NTSSA procedure is only activated when *PopRange* is near the value of one despite the fact that the mutation operator has just been activated.) If many chromosomes lead to the same objective function value (diversity in the population has been lost), only one of them is worked on by NTSSA in order to avoid computations which are not likely to be beneficial. Therefore, the number of chromosomes undergoing NTSSA are bounded by *frac*PopRange*PopSize*, but they might be less than the latter upper bound.

After NTSSA completes the search, the GA resumes applying its reproduction, crossover and mutation operators. The NTSSA procedure has a diversification effect on the solution space considered by the GA, because the solution obtained at the end of NTSSA is usually far away from that described by the initial chromosome. Since the initial solution is replaced by the best solution obtained in NTSSA, a proportion of the GA population undergoes through an operation analogous to the mutation operator. However, unlike the mutation operator, NTSSA usually results in a new solution at least as good as the initial one and mostly, much better. Since NTSSA conducts the search to improve the initial solution, and since infeasible solutions are penalized in proportion to their degree of infeasibility, the number of infeasible solutions in the population diminishes as a result of the execution of NTSSA. Consequently, in terms of the population performance, NTSSA has an intensification effect.

The above paragraph discusses the effects of a TS/SA type of search on GAs. Now we can analyze the effects of a GA on a TS/SA procedure. Previous experimentation with a pure SA procedure developed for the CLSP (Özdamar and Bozyel, 1996) 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 random initial solution and finally reporting the best solution from among all runs. Unfortunately, the initial random seed affects the performance of SA and diversity has to be achieved by rerunning SA with different random seeds. If SA is coupled with a GA, the population of the GA serves as a natural diversification tool. In the integrated GA/NTSSA procedure described here the GA population provides different good starting points for the NTSSA procedure.

From the viewpoint of the GA, NTSSA imbedded in GA serves two purposes: the population undergoes mutation and the overall performance level of the population ameliorates. The consecutive GA/NTSSA iterations have an accordion effect on the performance level of the population as observed in Fig. 1. The hybrid GA/NTSSA approach which we denote as GATA converges rapidly due to the overall improvement achieved in every activation of NTSSA. In Fig. 1 every new geometric figure in subsequent generations demonstrates new chromosomes replacing the old ones. The NTSSA replaces infeasible/feasible old chromosomes by new and usually feasible ones. When GA takes over from NTSSA, some feasible chromosomes drop out of the feasible region but some improve in their objective value while remaining in the feasible region due to the GA's own reproduction, crossover, mutation operators and adaptive crossover/mutation probabilities which enable the survival of the fittest.

A discussion on the connection between TS and GAs via the concept of strategic oscillation takes place in Glover et al. (1995). Strategic oscillation is concerned with intensifying and diversifying the search for finding near optimal solutions while preventing premature convergence. Glover et al. (1995) claim that one of the ways to achieve diversification in a GA is to permit infeasible solutions in the population. On the other hand, local search methods applied to some of chromosomes during the GA application intensify the search in the sense that the search digs deeper into a promising solution space. In the literature, GA approaches combined with local search are reported to have success in solving combinatorial problems (Huntley and Brown, 1996; Mühlenbein, 1993; Uckun et al., 1993).

Three different GATA procedures are developed here. The first one, denoted as GATA-1, starts with a randomly generated initial population and proceeds with consecutive GA/NTSSA iterations as in Fig. 1 until no improvement is observed in the best solution obtained so far.

GATA-2 starts with an initial population which is *partially* made of randomly generated chromosomes. A number of chromosomes are generated by applying a restrictive TS/SA procedure (RTSSA) which starts with a random *feasible* chromosome and preserves inventory and *capacity* feasibility until the end of the search. Once a predetermined number of different feasible chromosomes are generated by RTSSA, GATA-2 inserts them in the initial population and generates the rest of the population randomly. Then, GATA-2 acts as GATA-1 throughout the search. GATA-2 is implemented so that the performance of the stand-alone RTSSA procedure which carries out a combined TS/SA search in the feasible solution space is evaluated and furthermore, the effects of starting with a number of good solutions are observed for GATA.

Finally, GATA-3 simultaneously processes a number of parallel independent populations (*PopNo*). Each population is acted upon by GATA-1 independently for a number of generations (*GenNo*) and then a crossover operator (*MixCross*) crosses over parents from different populations and inserts the offspring into their corresponding parent populations, to achieve a population merge. GATA-3 is developed to extend the restricted range of the solution space covered by the initial random population and exchange good genes among populations. The pseudocodes for the three GATA procedures are given in Table 3.

4. Chromosome encoding in GATA procedures

The encoding adopted here is direct (Portmann, 1996a) in the sense that it consists of the

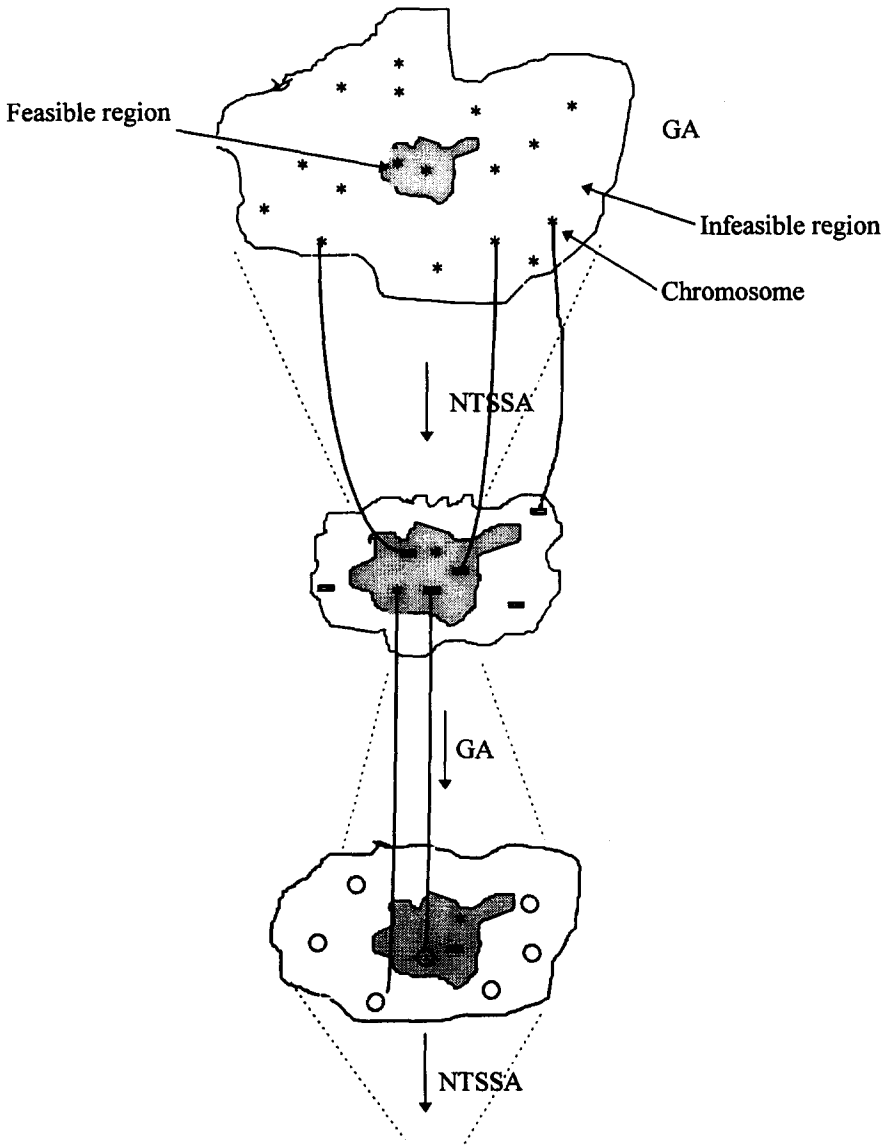

Fig. 1. Accordion motion of the hybrid GATA approach with respect to the overall performance of the population.

lot size variables indicated in the mathematical model. A solution c to the CLSLP is encoded by indicating y_{jkt}variables in two structures: the structure q_{cjt} concerning the lot sizes, where q_{cjt} is the lot size of family j in period t in the solution c and the structure a_{cjt} indicating the specific facility k on which a positive lot size of family j is loaded in a specific period t. For instance, q_{cjt} might be 75, 50, 0 and 25 units and a_{cjt} might be 1, 3, 0, and 1 for a family j which can be loaded on the first and third facilities, and whose demand is 50, 0, 75 and 25 units for the next four periods. Thus, instead of having a cubic structure of size $J \times K \times T$, we have two square structures of size $J \times T$ (J is the number of families, T is the number of periods in the planning horizon, K is the num-

Table 3
Pseudocodes for GATA procedures

Procedure GATA-1;
begin
generate *PopSize* chromosomes;
repeat
if not(*SA*) then reproduce;
SA := false;
crossover;
if *PopRange* > α then mutation;
calculate *PopRange*;
if *PopRange* > λ then
begin
Do NTSSA for (*frac***PopSize** *PopRange*) different chromosomes;
SA := true;
end;
until no improvement;
end;
Definitions:
PopSize: no. of chromosomes in population
SA:boolean indicating if NTSSA has just been implemented
frac: fraction of population undergoing NTSSA.
PopRange: ratio of minimum obj. func. value to the maximum in the population.
α, λ: constants less than one.

Procedure GATA-2;
begin
Do (RTSSA) *P* times;
Insert *P* solutions in initial pop.;
generate *PopSize-P* chromosomes;
repeat
if not(SA) then reproduce;
SA := false;
crossover;
if *PopRange* > α then mutation;
calculate *PopRange*;
if *PopRange* > λ then
begin
Do NTSSA for (*frac***PopSize** *PopRange*) different chromosomes;
SA := true;
end;
until no improvement;
end;

Procedure GATA-3;
begin
for *i* = 1 to *PopNo* do
generate *PopSize* chromosomes;
repeat
for *i* = 1 to *PopNo* do
for *j* = 1 to *GenNo* do
begin
if not(*SA*) then reproduce;
SA := false;
crossover;
if *PopRange* > α then mutation;
calculate *PopRange*;
if *PopRange* > λ then
begin
Do NTSSA for (*frac***PopSize** *PopRange*) different chromosomes;
SA := true;
end;
end;
MixCross;
until no improvement;
end;
Definitions:
PopNo: number of parallel populations
GenNo: number of generations to go without migration

ber of facilities). The objective value of a solution represented by a chromosome *c*, is indicated by obj_c.

5. Generating the initial population

A chromosome in the initial population is generated by assigning a random lot size to every family in each period. If a randomly assigned lot size does not satisfy the demand in the considered period, it is satisfied by augmenting the lot sizes of the previous periods randomly. Positive lot sizes are loaded on randomly selected facilities of appropriate classes. In this scheme, a chromosome is guaranteed to have nonnegative inventory, but the capacity requirements implied by the lot sizes may violate the limits. The GATA procedures avoid repetitive chromosomes in the initial population by enforcing the creation of chromosomes different from existing ones.

5.1. The reproduction operator

The reproduction strategy used in GATA is a pure selection one (Grefenstette, 1986). Each chromosome in the population is reproduced a number of times proportional to its objective function value. The fitness function used for reproducing a chromosome is a second degree polynomial of the inverse of the chromosome's objective function value. Reproduction is not carried out immediately after NTSSA has been implemented, because critical alleles of the chromosomes which have not been processed by NTSSA can be lost during reproduction due to the fact that there is usually a big gap between their objective function values (due to infeasibility penalties) and the ones

belonging to the feasible chromosomes processed by NTSSA.

5.2. The crossover operator

The crossover operator is a two-point operator which randomly selects two parents from the population, and two periods, $t1$ and $t2$. For all families j, the structures q_{1jt} (a_{1jt}) and q_{2jt} (a_{2jt}) 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 by adding/subtracting the necessary amounts to/from family lot sizes. The repair operation is carried out first by calculating the cumulative demand for each family, starting from the first period on to the last and checking if the cumulative lot sizes are greater than or equal to the cumulative demand in each period. In case the latter condition is violated for any family, only the minimal amount is added to the lot size of the currently considered period. This forward repair pass prevents backorders. In the backward pass, each family's inventory is checked in the last period and if inventory is positive, then the minimum between that family's lot size and its inventory level is deducted from the lot size. If the inventory level in the last period is still positive, then the lot sizes of the prior periods are deducted so as to retract the final period's inventory completely. The backward repair pass prevents excess production. While deducting or augmenting lot sizes, a_{cjt} are also updated. If a lot size is augmented from zero level, then a facility of appropriate class is randomly selected and the lot size is loaded on that facility. If a lot size is reduced to zero level, then the corresponding a_{cjt} is set to null. An example will clarify the repair operation. Suppose we have the two parent chromosomes given in Table 4 (for simplicity we assume there exists a single family loaded on two facilities for a planning horizon of 4 periods). The swap is to be made during periods 2 and 3.

Table 4
Example for the crossover operation with repair

Period	1	2	3	4	Period	1	2	3	4
Demand	50	0	75	25					
Parent 1					Parent 2				
Lot size	75	50	0	25	Lot size	50	0	75	25
Facility assigned	1	3	0	1	Facility assigned	1	0	3	3
Inventory	25	75	0	0	Inventory	0	0	0	0
Offspring 1					Offspring 2				
Lot size	75	0	75	25	Lot size	50	50	0	25
Facility assigned	1	0	3	1	Facility assigned	1	3	0	3
Inventory	25	25	25	25	Inventory	0	50	−25	−25
Repaired Offspring 1 (Forward pass)					Repaired Offspring 2 (Forward Pass)				
Lot size	75	0	75	25	Lot size	50	50	25	25
Facility assigned	1	0	3	1	Facility assigned	1	3	1	3
Inventory	25	25	25	25	Inventory	0	50	0	0
Repaired Offspring 1 (Backward pass)					Repaired Offspring 2 (Backward Pass)				
Lot size	75	0	75	0	Lot size	50	50	25	25
Facility assigned	1	0	3	0	Facility assigned	1	3	1	3
Inventory	25	25	25	0	Inventory	0	50	0	0

After the swap, offspring 1 has an excess production of 25 units while the second one has backorders. In the forward pass backorders are repaired and in the backward pass excess production is eliminated.

The crossover operation in all GATA procedures carry out partial replacement. That is, the number of new offspring is equivalent to *PopSize*PopRange*. Thus, the crossover rate increases with *PopRange*, i.e., as the diversity of the population is lost the crossover rate increases. However, when the diversity of the population is large, *PopRange* may sometimes be too small so that very few new offspring are created. To avoid the latter situation, a lower bound of (*PopSize/3*) is enforced on the number of new offspring.

The crossover implemented here is elitist in the sense that the offspring replace chromosomes in the parent population in descending order of objective function value. That is, the worse performing parent chromosomes die out.

5.3. The MixCross operator

The MixCross operator aims at merging genes of chromosomes in different populations on which search has been conducted simultaneously. In MixCross, two randomly selected parents of different populations are crossed over and repaired similar to the crossover operator. However, the repaired offspring are each inserted to their own parent populations. In MixCross, the parent populations are totally replaced by offspring so that a thorough merge is achieved. Similar concepts on merging parallel populations are discussed by Potts et al. (1994) where the loss of critical alleles due to reproduction and the disruption of schemata due to crossover are prevented by keeping a static population where no reproduction, crossover and mutation are carried out and the best chromosomes found in other populations are collected. Occasionally, chromosomes from the static population are migrated to the other populations. Migration among dynamic populations is also permitted. In GATA procedures, a static population does not exist, because NTSSA, which reintroduces chromosomes of good quality into the population, is frequently implemented.

5.4. The mutation operator

Mutation is carried out on each chromosome, but only *a single* family/period pair (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 family's lot size nor its inventory level. The same amount is added to the lot size of the next period. Thus, inventory feasibility is preserved. Naturally, reduction/augmentation is coupled with the necessary update in the facility assignments. For instance, if the lot size in the second period is to be mutated in Parent 1 given in Table 4, the maximum deductable amount is $\min\{50, 75\} = 50$ units. If the amount is randomly selected as 35, then the corresponding lot size becomes 15 units in period 2 and the lot size in period 3 is increased to 35 units. The inventory level in period 2 becomes 40 units. a_{113} is randomly selected as the first facility.

6. Crossover and mutation probabilities

The crossover/mutation probability for a chromosome c is calculated as follows:

$$pc = \begin{cases} \text{max obj-objc} & \text{if minobj} \neq \text{maxobj}, \\ \text{max obj-min obj} & \\ 1.0 & \text{otherwise}, \end{cases}$$

where minobj is the least (best) objective function value in the current population and maxobj is the worst one. A chromosome's crossover/mutation probability is dependent both on its own objective function value and on the population's performance range. The lower a chromosome's objective function value, the higher is its probability for crossover, so that 'good' genes get a higher opportunity to recombine in the next generation. Furthermore, a chromosome's crossover/mutation probability keeps changing according to the convergence of the population's performance range adapting itself to every generation's performance.

The crossover/mutation probabilities increases when the population's performance range tends to get stuck at a local optimum. The crossover/mutation probabilities defined here are a variation of the adaptive probabilities described by Srinivas and Patnaik (1994) and they have been successfully implemented for a NP-hard discrete optimization problem (Özdamar, 1996).

7. The restrictive TS/SA procedure

The RTSSA is utilized in GATA-2 to generate a number of feasible solutions to be inserted into the initial population. Comparing the results of RTSSA with the final results of GATA-2, we can observe the improvement achieved by the diversity of GA's use of populations during the search rather than a single point. Furthermore, we can deduce if adding high quality initial chromosomes to the initial population affects the performance of the GA when the two procedures GATA-1 and GATA-2 are compared.

The CLSLP involves a large number of binary family/facility assignment variables as well as floating lot size variables. Therefore, cycling among previously visited solutions occur frequently. RTSSA incorporates features from TS in order to prevent cyclic visits to the same solutions. RTSSA starts the search from an initial feasible solution and perturbs the solution for a given number of times. Perturbation is based upon going from one neighbouring solution to the next. A neighbouring solution is one where exactly one family has its lot size reduced and augmented by an amount "*DeltaLot*" in two consecutive periods $t1$ and $t2$, respectively, or in the same period ($t1 = t2$). When the lot size is reduced and augmented in the same period, the quantity of the lot size remains the same, but the facility on which the family is loaded changes.

The pseudocode for procedure Perturb which constitutes the main body of the procedure RTSSA is given in Appendix A. In the procedure Perturb, a move is considered for execution if it is not found among the *size* (Tabulist1) moves in Tabulist1 which stores a number of selected family/facility/period triplets $(j^*, k^*, t1)$ whose lot size is to be deducted/augmented by a random amount during the search. Tabulist1 prevents the search from reducing/augmenting a family lot size which has recently been augmented/reduced. Tabulist1 is updated with each executed move by adding the most recently selected triplet to the list and deleting the oldest one from the list. The size of Tabulist1 is dynamic. If the incumbent solution does not change in the last N moves, then the size of Tabulist1 is truncated randomly conserving only the most recent moves. However, size (Tabulist1) is not permitted to drop below the minimum tabulist size, *MinSize*. If the last M moves are all deteriorating moves, then size (Tabulist1) is extended randomly without exceeding the maximum list size, *MaxSize*. The dynamic list size is devised for helping the search skip over local minima and preventing it from visiting worse areas of the feasible region.

First, it is randomly decided whether a selected lot size is going to be reduced or augmented. Then, the family/facility/period triplet $(j^*, k^*, t1)$ is selected randomly. However, if the lot size is to be increased from zero level, the facility on which the move is going to be carried out, k^*, assumes the index of the facility which would add the least overtime cost to the objective function if family j^* is assigned a positive lot size in period $t1$. Thus, a simple local search over available facilities determines k^*. Once the triplet $(j^*, k^*, t1)$ is determined, $t2$ randomly assumes one of the following values: $t1 - 1$, $t1$ or $t1 + 1$. When the lot size is reduced, for every possible facility, d, that family j^* can be loaded on in period $t2$, *DeltaLot* and the corresponding possible change in the objective function value, *CostDif*, are calculated. The facility, d^*, with the minimum *CostDif* is selected for an increase in lot size if *CostDif* is not on Tabulist2 which stores the objective function values of the solutions obtained in the last *size* (Tabulist2) moves. (Thus, for identifying d^* a brief local search is also carried out.) Tabulist2 has a longer memory than Tabulist1 and it is kept to prevent repetitive moves as well as impede the search from considering redundant solutions which result in the same objective function value. The length of Tabulist2 is also dynamic, but *MaxSize* for Tabulist2 is greater than that for Tabulist1.

When the lot size is decreased, *DeltaLot* is bounded from above by the minimum of the capacity limit of the facility d in period $t2$ to which it is going to be transferred, the inventory level in period $t1$(if $t2 > t1$), and the lot size in period $t1$. *DeltaLot* assumes a random value less than or equal to the latter upper bound. When the lot size is augmented, d^* is known apriori since the lot size in period $t2$ is already positive (it is going to be reduced and a lot is going to be transferred to period $t1$) and loaded on a facility d^*. In this case, *DeltaLot* is bounded by the lot size in period $t2$, the inventory level in period $t2$ (if $t2 < t1$) and the available capacity of facility k^* in period $t1$.

When $t2$ equals $t1$, the lot size has to be transferred to another facility as a whole. If the idle time of facilities do not permit the latter transfer, then *DeltaLot* becomes zero. Furthermore, whenever a lot size is augmented, it can be loaded to any appropriate facility if it is augmented from zero level; else it may only be loaded on the facility on which a lot of that family is already loaded. The latter conditions guarantee the indivisibility of lots.

It is usually possible to save on inventory and/or overtime costs during a move. However, *CostDif** may be positive and a move may result in a cost higher than the incumbent one. Unlike a TS procedure, rather than accepting the best disimproving move, the RTSSA procedure calculates a probability of acceptance, PA which depends on the amount of the deterioration caused by the move in the objective function and on the number of times a deteriorated cost has been obtained consecutively (tSA). Furthermore, in RTSSA not all the neighbourhood is searched for the best solution, since the latter would be computationally inefficient. So, in these respects, RTSSA acts like a SA procedure which keeps tabulists to prevent cycling. PA is calculated as follows:

$$\mathrm{PA}(CostDif, t\mathrm{SA}) = \exp(-CostDif/(Cost^* t\mathrm{SA})),$$

where *Cost* is the cost of the incumbent solution. The temperature reduction scheme is continuous (Dowsland, 1993). Namely, after each disimproving move, the temperature, tSA, is reduced as follows:

$$t\mathrm{SA} \leftarrow t\mathrm{SA}/(1 + \beta\, t\mathrm{SA}),$$

where β is a parameter less than one. Initially tSA is equal to one. If tSA falls below a very small value, then it is reset to one, so that PA does not become infinitesimally small after a large number of moves.

The temperature reduction parameter β is not constant here, but dynamic and it depends on the status of the search, similar to dynamic tabulist lists. If the last M number of consecutive moves have resulted in improving cost values, then β is decreased by a *Stepsize* less than 1 (a convenient stepsize is 0.01). On the other hand, if the last M number of consecutive moves have resulted in disimproving cost values, then β is increased by the *Stepsize*. The 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, then β is decreased to encourage disimproving moves so that local optima can be skipped. Disimproving 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.

8. An example move carried out by RTSSA

Suppose the lot size of a family j^*, in period $t1 = 2$ has been selected for deduction and it is currently loaded on the second facility. Family j^* may also be loaded on the first facility. In Table 5 we convey a partial solution incorporating information on inventory levels and lot sizes of family j^* only among other families, as well as the idle times

Table 5
Example data for describing a move made by RTSSA

Period	1	2	3	4
Inventory	15	60	20	0
Lot size	30	80	0	10
Loaded on facility	1	2	–	1
Idle time of facility 1	42	55	61	90
Idle time of facility 2	16	0	60	70

of the two facilities. Suppose also that the inventory holding cost is 2$/unit-period and the overtime cost is 0.5$/h. A unit of family j^* is made in 3 h on the first facility and in 2 h on the second one. The setup time is 10 h on both facilities. The overtime limit per facility is 40 h/period. The idle times of facilities indicated in Table 5 implicitly specify the overtime used. For instance, in the first period, the first facility does not accrue overtime costs since its idle time is greater than the overtime limit. On the other hand, the second facility uses an overtime of 24 h.

Suppose $t2$ is selected randomly and it turns out to be period one. *DeltaLot*, which is to be transferred from period 2 to period 1, is bounded from above by the lot size in period 2, 80 units. Next, we have to find out d^*, the facility which would accrue the minimum *CostDif* in period one. $42/3 = 14$ units can be loaded on facility 1 and since there is a positive lot size on facility 1, the second facility does not represent a feasible choice due to the indivisibility of lot sizes. So, *DeltaLot* is now bounded by 14 units. Suppose *DeltaLot* is randomly assigned a value of 10 units. *CostDif* is then equal to added inventory holding cost plus added overtime cost on the first facility in period one minus the overtime cost saved on the second facility in period two $(= 10 \times 2 + (30 - 2) \times 0.5 - 20 \times 0.5) = 24\$$. *CostDif* is positive and before the move is executed the probability of acceptance has to be checked as well as Tabulist1.

Now suppose $t2$ is selected as period 2. The only facility that can be considered is facility 1. However, its idle time is not sufficient from the whole lot size to be transferred $(10 + 80 \times 3)$, so *DeltaLot* turns out to be zero.

Finally, if $t2$ is selected as period three, then both facilities should be considered since the lot size is zero in period three. For facility 1, *DeltaLot* can at most be $(61 - 10)/3 = 17$ units, and for facility 2 at most $(60 - 10)/2 = 25$ units. If *DeltaLot* is randomly set to 10 units for facility one, then *CostDif* for facility one becomes $(-10 \times 2 - 20 \times 0.5 + (40 - 21) \times 0.5) = -20.5$ \$. For a *DeltaLot* of 20 units, *CostDif* $(-20 \times 2 - 40 \times 0.5 + (50 - 20) \times 0.5) = -45$ \$ for facility two. Thus, facility two becomes d^* if 'Cost–45' is not on Tabulist2. In Table 6, a summary of the moves for the supposedly selected *DeltaLot* values are given in the case of three different $t2$ periods.

Table 6
Summary of moves which can be carried out for different values of $t2$ and *DeltaLot*

$t2 = 1, d^* = 1$		Periods			*DeltaLot*	*CostDif*
Value after move	1	2	3	4		
Inventory	25	60	20	0	10	24
Lot size	40	70	0	10		
Loaded on facility	1	2	–	1		
Idle time of facility 1	12	55	61	90		
Idle time of facility 2	16	20	60	70		
$t2 = 2$						
Inventory	15	60	20	0	0	0
Lot size	30	80	0	10		
Loaded on facility	1	2	–	1		
Idle time of facility 1	42	55	61	90		
Idle time of facility 2	16	0	60	70		
$t2 = 3, d^* = 2$						
Inventory	15	40	20	0	20	–45
Lot size	30	60	20	10		
Loaded on facility	1	2	2	1		
Idle time of facility 1	42	55	61	90		
Idle time of facility 2	16	40	10	70		

9. Nonrestrictive simulated annealing procedure

The nonrestrictive simulated annealing procedure (NTSSA) is utilized in all the three GATA procedures as a diversification as well as a performance improvement tool. NTSSA differs from RTSSA in two respects: First, it does not check on Tabulist1 when selecting a family/facility/period triplet to work on. Tabulist1 is eliminated for computational efficiency. Second, it enables moves which are infeasible with respect to capacity constraints by omitting the capacity bound from the Preserving Feasibility equation in procedure Perturb, i.e., UbLot $= \min\{y_{j*k*t1}, I_{j*t1}\}$ or UbLot $= y_{j*k*t1}$. Inventory feasibility is always preserved and moves which lead to capacity violations are penalized per unit overused capacity hour while *CostDif* is calculated. For instance, in the example given in Table 5, for $t2 = 1$, the upper bound of 14 units imposed on *DeltaLot* by the available idle capacity level of facility one would not exist. The upper bound would simply be 80 units. If the randomly selected *DeltaLot* exceeds 14 units, then every hour consumed on facility one in excess of the overtime limit adds a penalty to the objective function. NTSSA stores the best feasible/infeasible solution obtained in the search and replaces the parent chromosome from which the search has started with the best solution.

10. Testing environment

Two sets of test problems are generated, the first set of problems consist of 5 families, 3 facilities and 4 periods while the second set consists of 20 families, 5 facilities and 6 periods. The first and second sets of problems consist 108 and 36 test instances, respectively. The first set of small size problems are generated to test the performance of GATA procedures against optimal solutions which are obtained by using the GAMS optimization package and through the second set we expect to observe more conspicous differences in the performance of GATA-1, GATA-2 and GATA-3. Here, a remark on the complexity of the problem should be made. For finding optimal solutions, tight upper bounds identified by the RTSSA procedure are imposed on the GAMS solution procedure. Had we not started the optimization process with the latter upper bounds, it would not have been possible to determine the optimal solutions of even small test problems within a couple of hours. The CPU times required by GAMS for solving the small test problems to optimality range between 0.1 and 2320 CPU s on a Pentium 133 PC. Excessive CPU times are required for problems where all facilities are of the same class.

10.1. Test problem characteristics

All test problems are specified by four characteristics. The first one is the ratio of the average holding cost/unit-period to the unit cost of overtime (*H/O*), in other words the average overtime cost/unit. The average overtime cost/unit is estimated as *co*average process time*, where the average process time is the mean of family process times over all appropriate facilities. *H/O* ratio is held at the two levels of 1.5 and 0.5.

The second characteristic is the standard deviation of demand (STD). Family demands are generated from a normal distribution with a mean of 100 units per period with a STD held at two levels (10,50). Furthermore, demand has a seasonal trend over the planning horizon (with a seasonal constant of 1.5) and not all families' demand patterns need to match.

The maximum number that a facility can be loaded on, MaxPos, is the third characteristic. In the first set of problems there are three of facilities. So, we hold MaxPos at the three levels of 1, 2 and 3. With a MaxPos of one, every family can be loaded on a single facility. A MaxPos of three indicates that all facilities are of the same class. In the second set of problems, MaxPos is held at the level of three only in order to provide a class scheduling environment.

The fourth characteristic is defined by the tightness of facility capacity (CS). The required cumulative capacity in each period is calculated by multiplying the average family process times with their corresponding demands and accumulating the latter number each period. Then, the required

capacity for each period is calculated by dividing that cumulative requirement by (MaxPos*number of periods considered up to then). The result of the latter division is stored and the maximum among all stored values is adopted as required capacity per facility per period. (Notice that this approach assumes that lots are splittable.) Then, the maximum required capacity per facility per period is multiplied by a constant held at the three levels of 1.2, 1.4 and 1.6, to result in the capacity limit (regular + overtime) per facility per period. Overtime capacity is assumed to be one third of the overall facility capacity.

To summarize, there exist ($2 \times 2 \times 3 \times 3 = 36$) factor combinations and for each combination 3 replicates are generated resulting in 108 problems. The second set of problems consist of 3 replicates of the three factor combinations (CS, *H/O*, STD) because MaxPos is always equal to 3.

The remaining problem parameters are family process times (uniformly distributed between 1 and 5), setup times (uniformly distributed between 100 and 200), holding costs (uniformly distributed between 1 and 20) and overtime costs calculated according to the *H/O* ratio.

11. Computational results

First, we investigate the effects of different parameters on the performance of RTSSA procedure which provides initial feasible chromosomes for GATA-2. In Table 7, the results of six such tests are given. In the first test, we rerun RTSSA 10 times (NRS = 10), each time with a new random seed, and we limit the number of moves carried out by 15,000. Next, we lower NRS and increase the number of moves. In the third test, the SA parameter β is not dynamic and it is equal to 0.005 throughout the search. In the next two tests, we change the value of β. In the next test, β is dynamic, but Tabulist1 and Tabulist2 are both discarded. So, the procedure becomes a pure SA procedure. Finally, we have a pure TS procedure where the best disimproving move among all moves is accepted with a probability of one. RTSSA is converted into a pure TS procedure by looking over all families in the search for the best move, rather than randomly selecting a single family. Thus, all the neighbourhood of an incumbent solution is investigated rather than a restricted area in the neighbourhood. In these experiments we observe that the best results are obtained with the RTSSA procedure which integrates both TS and SA concepts in the search and works with dynamic β and Tabulists.

All GATA procedures are implemented on the first set of small test problems and compared against optimal results. In Table 8, we observe the results of the three GA procedures with varying population sizes and different *frac* values (percentage of population that undergoes NTSSA procedure). In all experiments, α, the conditional value for *PopRange* in mutation, and λ, the conditional value for NTSSA execution, are respectively held at 0.5 and 0.7. The number of moves in

Table 7
Comparison of RTSSA results (first set of problems) with different sets of parameters

RTSSA	Avg	STD	Max
NRS = 10, No. of Moves = 15,000, dynamic β and dynamic Tabu Lists	2.77	4.99	31.35
NRS = 5, No. of Moves = 30,000, dynamic β and dynamic Tabu Lists	2.51	4.12	28.28
NRS = 10, No. of Moves = 15,000, fixed $\beta = 0.005$, dynamic Tabu Lists	4.48	7.13	52.20
NRS = 10, No. of Moves = 15,000, fixed $\beta = 0.01$, dynamic Tabu Lists	3.64	6.72	42.16
NRS = 10, No. of Moves = 15,000, fixed $\beta = 0.02$, dynamic Tabu Lists	3.71	6.59	38.44
NRS = 0, No. of Moves = 15,000, No Tabu Lists; Pure SA procedure, dynamic β	3.56	5.18	37.91
NRS = 10, No. of Moves = 15,000, PA = 1; Pure TS procedure, dynamic Tabu Lists	60.04	104.00	615.85

NRS: # of different random seeds; Avg: average percentage deviation from the optimum; STD: standard deviation of the percentage deviation from optimum; Max: maximum percentage deviation from optimum.

Table 8
Performance comparison of GATA procedures (first set of problems) under different parameters

GATA procedures	Avg	std	Max	Avg RTSSA	std RTSSA	Max RTSSA	Avg generation	Avg NTSSA
GATA-1, NRS = 4 *Popsize* = 30, *frac* = 0.15	3.87	9.7	77.05				33	8
GATA-1 , NRS = 4 *Popsize* = 30, *frac* = 0.30	1.83	5.2	46.97				44	15
GATA-1 , NRS = 4 *Popsize* = 90, *frac* = 0.15	1.81	3.37	20.41				34	12
GATA-1, NRS = 4 *Popsize* = 90, *frac* = 0.30	1.36	3.12	22.64				48	36
GATA-2, NRS = 4 *Popsize* = 30, *frac* = 0.15	1.21	2.33	13.22	2.41	3.66	21.27	16	6
GATA-2, NRS = 4 *Popsize* = 30, *frac* = 0.30	1.10	2.09	14.61	2.97	5.79	38.50	18	12
GATA-2, NRS = 4 *Popsize* = 90, *frac* = 0.15	1.06	2.17	13.29	3.03	4.45	19.04	31	13
GATA-2 , NRS = 4 *Popsize* = 90, *frac* = 0.30	0.86	1.58	8.69	2.87	4.63	28.66	40	31
GATA-3 , *PopNo* = 4 *Popsize* = 30, *frac* = 0.15	1.45	2.71	15.57				12	5
GATA-3 , *PopNo* = 4 *Popsize* = 30, *frac* = 0.30	1.24	2.37	13.04				14	8
GATA-3 , *PopNo* = 4 *Popsize* = 90, *frac* = 0.15	0.64	1.47	9.16				21	7
GATA-3 , *PopNo* = 4 *Popsize* = 90, *frac* = 0.30	0.53	1.41	10.17				25	16
GATA-3, *PopNo* = 4 *Popsize* = 30, *frac* = 0.00	369	1159	9138				–	–

NTSSA is limited by 15,000. In GATA-2, the number of feasible chromosomes P in the initial population is 10. In GATA-3, the number of generations to go without migration, *GenNo*, is 25 and the number of parallel populations, *PopNo*, is 4. To provide a fair comparison among GATA-1/GATA-2 versus GATA-3, we rerun GATA-1 and GATA-2 with four different random seeds and report the minimum objective values. In both RTSSA and NTSSA implementations, the maximum sizes of Tabulist1 and Tabulist2 are 10 and 100, respectively.

In Table 8, Avg, std and Max are the average percentage deviations from the optimum, the percentage standard deviation and the maximum percentage deviations of the best solutions obtained

Table 9
Performance comparison of GATA procedures (second set of problems)

GATA procedures	Avg (%)	std (%)	Max (%)	Avg RTSSA (%)	std RTSSA (%)	Max RTSSA (%)	Avg generation	Avg NTSSA	# Best Solutions	# Infeasible problems
GATA-1, NRS = 3, *Popsize* = 100, *frac* = 0.05	1194.3	6236.7	35,931				103	37	1	3
GATA-2, NRS = 3, *Popsize* = 100, *frac* = 0.05	46.88	91.71	542.11	2739.9	15,497.5	93,136.4	99	64	5	0
GATA-3, *PopNo* = 3, *Popsize* = 100, *frac* = 0.05	5.57	15.84	67.93				805	552	35	0

by the GATA procedures. The best performance is obtained by GATA-3. GATA-3 carries out more or less the same number of evaluations as GATA-1 and GATA-2, because the latter two procedures are repeated NRS = 4 times. However, GATA-3 outperforms the other two procedures through the power of migration which achieves continual diversity in the populations worked on. We note that the increase in *PopSize* affects GATA-3 procedure the most. However, as indicated in the last row of Table 8, without the activation of the NTSSA procedure, even GATA-3 results in exorbitant solutions. We remark that without the aid of NTSSA, GATA-3 could find feasible solutions to 87 problems only. Naturally, the performance of all GATA procedures improves with increasing *PopSize* and *frac*.

Additional information on GATA-2 is provided as Avg RTSSA, std RTSSA and MaxRTSSA. The latter three abbreviations denote the corresponding percentage deviations obtained by RTSSA among 10 initial chromosomes which are inserted in GATA-2 population. In Table 8, we observe that GATA-2 procedure finally obtains percentage deviations which are nearly one third of the RTSSA percentage deviations.

In Table 8, the average number of generations within which the best solution is obtained, Avg generation, and the average number of NTSSA applications up to then, Avg NTSSA, are indicated to demonstrate the convergence of the procedures. In accordance with their quality of solutions, GATA-1, GATA-2 and GATA-3 find their best solutions in descending order of generations. The initial implementation of RTSSA which provides GATA-2 with feasible solutions improves convergence. Yet, GATA-3 which starts off with highly infeasible solutions achieves a much better performance both in solution quality and convergence.

The results of 36 larger test instances are given in Table 9. Results are compared against best solutions since it is not possible to obtain optimal solutions for this problem size. The number of best solutions are indicated for three GATA procedures. A *PopSize* of 100 and a *frac* value of 0.05 are used in all GATA procedures. NRS = 3 for GATA-1 and GATA-2 while *PopNo* = 3 for GATA-3. The number of moves for NTSSA and RTSSA are limited by 50,000. The remaining parameters are as given in the experiments for the first set of problems. In larger size test problems, the difference in the performance of the three GATA procedures become relatively exaggerated as compared to small test problems. No doubt, GATA-3 is the best procedure among all. The

effect of starting with some feasible solutions has an effect on the performance of GATA-2 when compared with GATA-1. Only 10 feasible chromosomes are generated by RTSSA in the initial population of GATA-2, and although the average performance of these 10 chromosomes looks bad when compared with the end results of GATA-1, these initial feasible chromosomes do carry valuable alleles which help the final convergence of GATA-2 (as compared with the final convergence of GATA-1). A common sense consequence of these results is that GATA-3 might converge much faster if it starts with a number of feasible chromosomes in the initial populations. As to the computational times of the procedures, GATA-3 is 4 times slower than GATA-2 because it finds much better results and converges after 800 generations on an average. The RTSS adds 40% more computation time to GATA-2 as compared to GATA-1, but obviously this is worthwhile. On the whole, GATA-3, which is most computationally cumbersome among all procedures, finds its solution within 3 min on an average (on a Pentium 133 PC) for a large test problem whereas GATA-2 finds it within a minute. However, for small problems there is not such a big difference, because GATA-3 converges to the best solution faster than GATA-2.

The single facility CLSP with overtime decisions and setup times (CLSPOS) is a special case of the CLSLP. The performance of GATA is also tested on the CLSPOS and compared with previously proposed methods (Özdamar et al., 1997). In the latter reference, GATA-3 is reported to locate the optimum solution in all of the 90 test instances consisting of 4 families and 6 planning periods. The other methods reported in the article (including an iterative LP relaxation approach, a GA approach with indirect encoding, a pure SA approach, RTSSA approach, and other constructive algorithms) are all outperformed significantly by the GATA procedure. Apart from the three GATA procedures, the RTSSA procedure is the method having the next best performance with an average deviation of 0.20% from the optimum whereas the pure SA method's average deviation is 4.37%. Results on the larger 90 problems with 14 families and 6 periods also indicate the same ratios between performances of different methods.

12. Conclusion

We developed a hybrid solution procedure GATA incorporating GAs, SA and TS for a difficult production planning problem which integrates loading and lot sizing decisions. Denoted here as the CLSLP, the problem considered is practically valid in planning situations where identical/uniform/unrelated parallel facilities (production lines) exist and a medium range plan involving inventory accumulation and overtime decisions is to be prepared for the next few planning periods. Lot sizing and loading decisions inevitably affect each other and it is important to consider the integrated problem in order to avoid capacity infeasibility issues and suboptimal solutions.

The CLSLP is challenging to researchers in terms of complexity, because it subsumes more than one NP-hard problems such as the capacitated lot sizing problem with overtime and setup times and parallel machine problem with total tardiness objective. It is not possible to conceive constructive heuristics which guarantee to find feasible solutions to this problem let alone optimal ones. The hybrid search heuristic GATA developed here demonstrates the power of compu-search methods in solving such complex problems. However, the empirical results emphasize that neither a stand-alone GA nor a stand-alone TS/SA procedure is sufficient to deal with this problem. GATA provides encouraging results and may be used to solve other NP-hard problems in production planning and control.

Acknowledgements

We thank Professor Marie Claude Portmann for her valuable suggestions related to the management of Genetic Algorithms and Professor Eralp Özil, Dean of Engineering Faculty, Yeditepe University for providing the physical means for research and encouraging us to carry on.

Appendix A. Pseudocode for the maind body of RTSSA (Lot Reduction part)

```
Procedure Perturb;
{ID_kt : remaining idle capacity of facility k in period t;
 UbLot: upper bound on the lot size to be transferred;
 DeltaLot: transferred lot size;
 Cost: cost of incumbent solution;
 CostDif: cost difference caused by the move;
 BestCost: cost of best solution obtained so far ;
 t1,t2: time index;
 k*,m, d, d*: facility index;
 j*: family index;
 K: number of facilities}

begin {Perturb}
 if Decrease Lot then
  begin
   Select t1, j* such that y_j*k*,t1 >0 and (j*,k*,t1) is not on Tabulist1;
   Update Tabulist1;
   randomly select t2; (t2=t1-1, or t1+1, or t1)
   MinCost=∝;
   for d=1 to K do {Local Search to identify d*}
   if (d ε F_j*) and ((t2=t1) or (y_j*m,t2=0 for m≠d)) and ((t1<>t2) or (d<>k*)) then
     begin
       if t1<t2 then UbLot=min {(ID_dt2)/ p_j*d, y_j*k*,t1 , I_j*,t1}  {Preserving Feasibility }
        else UbLot=min {(ID_dt2)/ p_j*d, y_j*k*,t1}; {s_j* subtracted from ID_dt2 if y_j*d,t2 =0}
       if (t1=t2) and (UbLot< y_j*k*,t1) then UbLot= 0;
       DeltaLot= random (UbLot);
       Calculate CostDif w.r.t ID_k*,t1, ID_d,t2 and I_j*t;
       if (CostDif+Cost<MinCost) and (Cost+CostDif not on Tabulist2) then
            begin
                d*=d;
                DeltaLot*=DeltaLot;
                MinCost=CostDif+Cost;
                CostDif*=CostDif;
            end;
     end;
       If CostDif*>0 then Calculate PA, tSA;
       If ((CostDif* ≥ 0) and (Accept)) or (CostDif*<0) and then
         begin
           Update Tabulist2;
           Update size(Tabulist1) and size(Tabulist2) if necessary;
           Update SA parameter β if necessary;
           Increment move count;
            Transfer DeltaLot* from period t1 to period t2;
           {y_j*k*,t1=y_j*k*,t1-DeltaLot*; y_j*d*,t2=y_j*d*,t2+DeltaLot*;Adjust ID_k*,t1,ID_d*,t2,I_j*,t}
           Cost = MinCost;
           If Cost < BestCost then begin BestCost = Cost; Save y_jkt; end;
         end;
end; {Decrease Lot}
```

```
else {Increase Lot}
 begin
    d*,k*=0;
    repeat
      Select t1, t2, j*; (t2=t1-1, or t1+1, or  t1)
      d*=d, s.t. y_{j*d,t2} >0
      if d*>0 then
        if y_{j*k,t1} =0 ∀ k then
        begin  {Local Search to identify k*}
              MinCost=∞;
              for k=1 to K do
                    begin
                      if t1<t2 then UbLot=min {(ID_{kt1}- s_{j*})/ p_{j*k}, y_{j*d*,t2} , I_{j*,t2}}  {Preserving Feasibility }
                      else UbLot=min {(ID_{kt1}- s_{j*})/ p_{j*k}, y_{j*d*,t2}};
                      if (t1=t2) and (UbLot< y_{j*d*,t2}) then UbLot= 0;
                      DeltaLot= random (UbLot);
                      Calculate CostDif  w.r.t ID_{k,t1}, ID_{d*,t2} and I_{j*t};
                      if (CostDif+Cost<MinCost) and ((j*, k, t1) not on Tabulist1) then
                          begin
                              MinCost=CostDif+Cost;
                              CostDif*=CostDif;
                              DeltaLot*=DeltaLot;
                              k*=k;
                          end;
                    end,
          end;
        else
          if (j*, k, t1) not on Tabulst1 then   { y_{j*k,t1} > 0}
            begin
                k*=k where y_{j*k,t1} > 0;
                if t1<t2 then UbLot=min {(ID_{k*t1})/ p_{j*k*}, y_{j*d*,t2} , I_{j*,t2}}  {Preserving Feasibility }
                else UbLot=min {(ID_{kt1})/ p_{j*k*}, y_{j*d*,t2}};
                if (t1=t2) and (UbLot< y_{j*d*,t2}) then UbLot= 0;
                DeltaLot*= random (UbLot);
                Calculate CostDif*  w.r.t ID_{k*,t1}, ID_{d*,t2} and I_{j*t};
            end;
    until k*>0;

    if (CostDif*+Cost not on Tabulist2) and (((CostDif* ≥ 0) and (Accept)) or (CostDif*<0))  then
            begin
              Update Tabulist2;
              Update size(Tabulist1) and size(Tabulist2) if necessary;
              Update SA parameter β if necessary;
               Increment move count;
               Transfer DeltaLot* from period t2 to period t1;
               {y_{j*k*,t1}=y_{j*k*,t1}+DeltaLot*; y_{j*d*,t2}=y_{j*d*,t2}-DeltaLot*;Adjust ID_{k*,t1},ID_{d*,t2},I_{j*,t}}
               Cost = Cost+ CostDif*;
               If Cost < BestCost then begin BestCost = Cost; Save y_{jkt}; end;
            end;
 end;{Increase Lot}
end; {Perturb}
```

References

Billington, P.J., McClain, J.O., Thomas, L.J., 1986. Heuristics for multilevel lot-sizing with a bottleneck. Management Science 32, 989–1006.

Billington, P., Blackburn, J., Maes, J., Millen, R., Van Wasssenhove, L.N., 1994. Multi item lot sizing in capacitated multi-stage serial systems. IIE Transactions 26, 12–18.

Bitran, G.R., Yanasse, H.H., 1982. Computational complexity of the capacitated lot size problem. Management Science 28, 1174–1186.

Bozyel, M.A., Özdamar, L., 1996. A heuristic approach to the capacitated lot sizing and scheduling problem (with parallel facilities). Yeditepe University, Department of Systems Engineering (Working Paper).

Cattrysse, D., Maes, J., Van Wassenhove, L.N., 1990. Set partitioning and column generation heuristics for capacitated lotsizing. European Journal of Operational Research 46, 38–47.

Cattrysse, D., Salomon, M., Kuik, R., Van Wassenhove, L.N., 1993. A dual ascent and column generation heuristic for the discrete lot sizing and scheduling problem with setup times. Management Science 39, 477–486.

Cheng, T.C.E., Sin, C.C.S., 1990. A state of the art review of parallel machine scheduling reserach. European Journal of Operational Research 47, 271–292.

Dauzerre-Peres, S., Laserre, J.B., 1994. Integration of lot sizing and scheduling decisions in a job-shop. European Journal of Operational Research 75, 413–426.

De Jong, K.A., 1980. Adaptive system design: a genetic approach. IEEE Transactions on Systems, Man and Cybernetics SMC-10, 566–574.

De Matta, R., Guignard, M., 1994. Dynamic production scheduling for a process industry. Operations Research 42, 492–503.

Diaby, M., Bahl, H.C., Karwan, M.H., Zionts, S., 1992. A lagrangiean relaxation approach for very large scale capacitated lot sizing. Management Science 38, 1329–1340.

Dixon, P.S., Elder, M.D., Rand, G.K., Silver, E.A., 1983. A heuristic algorithm for determining lot sizes of an item subject to regular and overtime production capacities. Journal of Operations Management 3, 121–130.

Dixon, P.S., Silver, E.A., 1981. A heuristic solution procedure for the multi-item single level limited capacity lot sizing problem. Journal of Operations Management 2, 23–39.

Dowsland, K.A., 1993. Some experiments with simulated annealing techniques for packing problems. European Journal of Operational Research 68, 389–399.

Duplaga, E.A., Hahn, C.K., Watts, C.A., 1996. Evaluating capacity change and setup time reduction in a capacity constrained joint lot sizing situation. International Journal of Production Research 34, 1859–1873.

Fleischmann, B., 1990. The discrete lot sizing and scheduling problem. European Journal of Operational Research 44, 337–348.

Garey, M., Johnson, D., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco, CA.

Glover, F., 1989. Tabu search – Part I. ORSA Journal of Computing 1, 190–206.

Glover, F., 1990. Tabu search – Part II. ORSA Journal of Computing 2, 4–32.

Glover, F., Kelly, J.P., Laguna, M., 1995. Genetic algorithms and tabu search:Hybrids for optimization. Computers and Operations Research 22, 111–134.

Goldberg, D., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA.

Grefenstette, J., 1986. Optimization of control parameters for genetic algorithms. IEEE Transactions on Systems, Man and Cybernetics SMC-16.

Günther, H.O., 1987. Planning lot sizes and capacity requirements in a single stage production system. European Journal of Operational Research 31, 223–231.

Hax, A.C., Candea, D., 1984. Production and Inventory Management. Prentice-Hall, Englewood Cliffs, NJ.

Holland, J.H., 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI.

Huntley, C.L., Brown, D.E., 1996. Parallel genetic algorithms with local search. Computers and Operations Research 23, 559–571.

Johnson, D.S., 1974. Fast Algorithms for Bin Packing. Journal of Computer and System Sciences 8, 272–314.

Jordan, C., Drexl, A., 1996. Discrete lot sizing and scheduling by batch sequencing. University of Kiel, Kiel (Working Paper).

Kim, D., Mabert, V.A., 1995. Integrative versus separate cycle scheduling heuristics for capacitated discrete lot sizing and sequencing problems. International Journal of Production Research 33, 2007–2021.

Kirkpatrick, A., Gelatt Jr., C.D., Vechi, M.P., 1983. Optimization by simulated annealing. Science 220, 671–680.

Kirca, Ö., Kökten, M., 1994. A new heuristic approach for the multi-item dynamic lot sizing problem. European Journal of Operational Research 75, 332–341.

Kolen, A.W.J., Kroon, L.G., 1991. On the computational complexity of the (maximum) class scheduling. European Journal of Operational Research 54, 23–38.

Kuik, R., Salomon, M., van Wassenhove, L.N., Maes, J., 1993. Linear programming, simulated annealing and tabu search heuristics for lot sizing in bottleneck assembly systems. IIE Transactions 25, 62–72.

Lambrecht, M.R., Vanderveken, H., 1979. Heuristic procedure for the single operation multi-item loading problem. IIE Transactions 11, 319–326.

Maes, J., McClain, J.O., Van Wassenhove, L.N., 1991. Multilevel capcitated lot sizing complexity and LP-based heuristics. European Journal of Operational Research 53, 131–148.

Maes, J., Van Wassenhove, L.N., 1986. A simple heuristic for the multi-item single level capacitated lot sizing problem. Operations Research Letters 4, 265–273.

Michalewicz, Z., 1994. Genetic Algorithms + Data Structures=Evolution Programs, Springer, Berlin.

Mühlenbein, H., 1993. Evolutionary algorithms: Theory and applications. In: Narb, E.H.L., Lenstra, J.K. (Eds.), Local Search in Combinatorial Optimization. Wiley, New York.

Özdamar, L., 1996. A genetic algorithm approach for the multi-mode resource-constrained project scheduling problem under general resource categories. Proceedings of the Fifth International Workshop on Project Management and Scheduling, EURO WG-PMS, Poznan, Poland.

Özdamar, L., Atli, A.Ö., Bozyel, M.A., 1996. Heuristic family disaggregation techniques in hierarchical production planning systems. International Journal of Production Research 34, 2613–2628.

Özdamar, L., Birbil, İ., Portmann, M.C., 1997. New results for the capacitated lot sizing problem with overtime decisions and setup times. Yeditepe University, Department of Systems Engineering, Istanbul, Turkey (working paper).

Özdamar, L., Bozyel, M.A., 1996. The capacitated lot sizing problem with overtime decisions and setup times. Yeditepe University, Department of Systems Engineering, Istanbul, Turkey (working paper).

Özdamar, L., Bozyel, M.A., 1997. Simultaneous lot sizing and loading of product families on parallel facilities of different classes. International Journal of Production Research (to appear).

Portmann, M.C., 1996a. Genetic algorithms and scheduling: A state of the art and some propositions, Proceedings of the Workshop on Production Planning and Control, Mons.

Portmann, M.C., 1996b. Scheduling methodology: Optimization and compu-search approaches I. In: Artiba, A., Elmaghraby, S.E., (Eds.), Production and Scheduling of Manufacturing Systems, Chapman & Hall, London.

Potts, J.C., Giddens,T.D., Yadav, S.B., 1994. The development and evaluation of an improved genetic algorithm based on migration and artificial selection. IEEE Transactions on Systems, Man and Cybernetics 24, 73–86.

Salomon, M., Kroon, L.G., Kuik, R., Van Wassenhove, L.N., 1991. Some extensions of the discrete lot sizing and scheduling problem. Management Science 37, 801–811.

Srinivas, M., Patnaik, L.M., 1994. Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Transactions on Systems, Man and Cybernetics 24, 656–667.

Tempelmeier, H., Derstroff, M., 1996. A lagrangean based heuristic for dynamic multi item multi level constrained lot sizing with setup times. Management Science 42, 738–757.

Tempelmeier, H., Helber, S., 1994. A heuristic for dynamic multi item multi level capacitated lot sizing for general product structures. European Journal of Operational Research 75, 296–311.

Thizy, J.M., Van Wassenhove, L.N., 1985. Lagrangian relaxation the multi-item capacitated lotsizing problem: A heuristic implementation. IIE Transactions 17, 308–313.

Trigeiro, W.W., Thomas, L.J., McLain, J.O., 1989. Capacitated lot sizing with setup times. Management Science 35, 353–366.

Uckun, S., Bagchi, J.C., Proust, C., 1993. Managing genetic search in job shop scheduling. IEEE Expert 8, 15–24.