Technical note: New results for the capacitated lot sizing problem with overtime decisions and setup times

Linet Özdamar , Şevket İlker Bilbil & Marie-Claude Portmann

Technical note: New results for the capacitated lot sizing problem with overtime decisions and setup times

LINET ÖZDAMAR, ŞEVKET İLKER BIRBIL and MARIE-CLAUDE PORTMANN

Keywords lot sizing, genetic algorithm, heuristics

Abstract. The capacitated lot sizing problem with overtime and setup times (CLSPOS) consists of planning the lot sizes of multiple families over a planning horizon with the objective of minimizing overtime and inventory holding costs. Each time that an item's lot size is positive, capacity is consumed by a setup. Capacity is limited and includes regular time capacity as well as overtime. It is assumed that setups do not incur costs other than lost production capacity and therefore, setups contribute to total costs implicitly via overtime costs whenever capacity bottlenecks occur. The CLSPOS is more complicated than the standard capacitated lot sizing problem (CLSP) which involves explicit setup costs, no capacity consuming setups and only regular time capacity. Here is described a genetic algorithm (GATA) integrated with tabu search (TS) and simulated annealing (SA) to solve CLSPOS. GATA integrates the powerful characteristics of all three search algorithms, GAs, TS and SA. The results are compared with the ones reported in a previous study and demonstrate that GATA outperforms other heuristics.

1. Introduction

The single stage capacitated lot sizing problem (CLSP) is a NP-hard problem (Bitran and Yanasse, 1982) in production planning. In the CLSP, demand is deterministic but variable over time, the planning horizon consists of a finite number of discrete periods, capacity is limited and no backorders are permitted. A solution to the CLSP consists of item lot sizes which satisfy demand throughout the planning horizon without exceeding the capacity limit in each period. Every time that an item is produced a setup cost is incurred. The objective function minimizes setup and inventory carrying costs. Heuristic solution methods proposed for this problem usually involve period-by-period heuristics (e.g., Maes and Van Wassenhove 1986, Günther 1987) where lot sizes for families are determined based on a cost saving criterion. Each period, future demand is satisfied until no further costs can be saved or the capacity of that period is exhausted. There are also heuristic approaches based on mathematical programming (e.g., Trigeiro *et al.* 1989, Cattrysse *et al.* 1990). As Trigeiro *et al.* (1989) remark, when setup times are included in the model, the CLSP becomes NP-Complete in the feasibility problem as well.

The model proposed here extends the CLSP in two respects: (i) it assumes two sources of capacity, regular

Authors: Linet Özdamar and Şevket İlker Birbil, Yeditepe University, Department of Systems Engineering, Istanbul, Turkey, E-mail: lozdamar@hotmail.com, and Maria-Claude Portmann, LORIA-INPL – Ecole des Mines de Nancy, France.

Linet Özdamar has BSc, MSc and PhD from Boðaziçi, University, Department of Industrial Engineering, Istanbul Turkey. Her research interests lie in hierarchical production planning, project scheduling, metaheuristics, global optimization and basic fuzzy theory. She is a member of the EURO Working Groups on Project Management and Scheduling, Industrial Engineering and Production Economics, and on Distributed Decision Making. She has published related articles in journals such as *European Journal of Operations Research, International Journal of Production Research, IEEE Transactions on Systems, Man and Cybernetics, IEEE Transactions on Automatic Control,* etc.

Production Planning & Control ISSN 0953–7287 print/ISSN 1366–5871 online © 2002 Taylor & Francis Ltd
http://www.tandf.co.uk/journals
DOI: 10.1080/0953728011004927 2

time and overtime, resulting in the loss of capacity homogeneity, (ii) setups consume capacity and they do not incur costs other than lost production capacity. Therefore, setup costs are not explicitly stated in the objective function, but they incur costs implicitly when capacity bottlenecks occur and overtime capacity is used. In the resulting problem, the objective function consists of inventory carrying and overtime costs while the constraints guarantee that demand is satisfied in each period and the sum of regular time and overtime capacities is not exceeded. This problem is denoted the capacitated lot sizing problem with overtime decisions and setup times (CLSPOS). The mathematical formulation of the CLSPOS is given in Table 1. Diaby *et al.* (1992) develop a Lagrangean relaxation based on loosened capacity limits and subgradient optimization for CLSPOS. However, they assume that all families have the same process times.

Here is developed a powerful genetic algorithm (GATA) integrated with a local search procedure (TSSA) incorporating features from Tabu Search (TS) and Simulated Annealing (SA). GATA involves a direct encoding of the CLSPOS, and as it is difficult to maintain the feasibility of all constraints, it usually conducts the search in the infeasible solution space. While GATA imitates the evolutionary process, TSSA is activated once in a while depending on the diversity of the current population. TSSA reduces the range of the population's performance by encouraging moves that render feasibility to the chromosomes that it works on. Then, GATA expands it by using crossover and mutation operators which may destroy capacity feasibility. The consecutive contraction/expansion phases result in a fast convergence to the optimal solution. GATA also benefits from migration and executes a simultaneous search on parallel populations. In the computational results, GATA's performance is compared with a GA involving an indirect encoding of the CLSPOS (IGA) developed in a previous study (Özdamar and Bozyel 2000) as well as with other methods cited in the latter reference.

Table 1. Mathematical formulation of the capacitated lot sizing problem with overtime decisions and setup times.

Model P:

$$\text{Min} \sum_t \sum_j I_{jt} h_j + \sum_t ov\ O_t$$

$$\begin{aligned}
\text{s.t.}\quad & I_{j,t-1} + Y_{jt} - I_{jt} = d_{jt} && \forall\, j, t \\
& \sum_j (p_j Y_{jt} + w_{jt} s_j) \le C_t + O_t && \forall\, t \\
& O_t \le CO_t && \forall\, t \\
& Y_{jt} \le M w_{jt} && \forall\, j, t
\end{aligned}$$

where d_{jt} = demand of item j in period t
p_j = process time of item j (hr)
ov = overtime cost per hour
h_j = holding cost per unit of item j per period
C_t = regular time capacity in period t
CO_t = overtime capacity in period t
I_{jt} = inventory of item j held at the end of period t ($I_{jt} \ge 0$)
Y_{jt} = production lot size of item j in period t ($Y_{jt} \ge 0$)
O_t = overtime capacity used in period t ($O_t \ge 0$)
s_j = setup time required for item j
w_{jt} = binary variable (= 1, if item j is produced in period t; = 0, otherwise)
M = a large number

2. The Genetic Algorithm – GATA

Genetic algorithms (GAs) are powerful search tools often used in optimization. Extensive reviews of GAs are given by Portmann (1996), and Alexandre *et al.* (1997). GAs emulate natural selection and diversity by applying crossover, reproduction and mutation operators to a population of solutions. Another probabilistic search method is Simulated Annealing (SA) (Kirkpatrick *et al.* 1983) which is a single solution search technique where worse solutions are accepted by a cooling schedule similar to an annealing process. In this manner the search may escape from local optima. Tabu search (TS) also works on a single solution, but a new solution is generated by investigating all of the immediate neighbours to the current solution and the best neighbour becomes the new current solution even if it is non-improving. To prevent revisiting the same solutions, a Tabu List keeps track of a number of previous moves.

As a hybrid metaheuristic where all three search approaches are merged together, the GATA procedure developed here, incorporates many of their advantages. GATA works on multiple populations and migrates chromosomes by executing a crossover operation which accomplishes an interpopulation mix. Thus, population diversity is increased and premature convergence is avoided. Furthermore, the freedom to visit solution spaces which are very far from containing the optimal solution is restricted by an occasional execution of TSSA which improves population performance and pushes chromosomes towards interesting feasible regions.

Both GATA and TSSA procedures are capable of conducting the search in infeasible regions as well as in feasible ones. The crossover and mutation operators in GATA preserve only the nonnegativity of the inventory levels and therefore, they are fast operators. The capacity infeasibilities implied by a set of lot sizes are penalized in the objective function in proportion to the total capacity excess. Consequently, GATA usually conducts the search in the infeasible space during the initial phases of the search. This feature does not only contribute to the computational efficiency of GATA, but also adds extra diversity to the population.

GATA activates the TSSA procedure when the population is likely to get stuck to the same area of the feasible/infeasible solution space. TSSA carries out local search on randomly selected chromosomes in the current population. A chromosome randomly selected by TSSA is replaced by the best solution obtained during local search. Of course, there is no guarantee that starting from an infeasible chromosome, TSSA will end up in a capacity feasible chromosome. The number of chromosomes that are processed by TSSA is equal to *PopSize***r*, where *PopSize* is the population size, and r is a constant less than one.

After TSSA completes the search, GA operators take over to continue with the evolutionary process. The TSSA procedure diversifies the population similar to the mutation operator, because the best solution obtained at the end of the search differs considerably from the initial solution. On the other hand, the objective function value of the best solution is at least as good as and most probably much lower than that of the initial solution. Therefore, as TSSA operates on a number of chromosomes in the population, the amount of infeasibility in the population diminishes. In terms of the population's performance, TSSA intensifies the search while having a diversification effect on the solution space spanned by the population, provided that the improved chromosomes do not converge to a too small set of local optima.

A pseudocode for GATA is given in Table 2. GATA starts with a predetermined number (*PopNo*) of randomly generated initial populations and proceeds with each of them in parallel by applying consecutive GA/TSSA phases for a number of generations (*GenNo*). TSSA procedure is activated when the population's performance range (*PopRange*), i.e., the ratio of the minimum objective function value among the chromosomes to the maximum one, almost reaches the value of one despite the fact that the mutation operator has just been activated. Then, parents from different populations are crossed over and the offspring inserted into their corresponding parent populations (*Migrate* operator). Each population is reprocessed independently until the next migration. The procedure goes on until a stopping condition is verified.

Table 2. Pseudocode for GATA procedure.

```
Procedure GATA;
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;
        calculate PopRange;
        if $PopRange > \alpha$ then mutation;
        calculate PopRange;
        if $PopRange > \lambda$ then
          begin
            Do TSSA on r*PopSize different chromosomes;
            SA := true;
          end;
      end;
    Migrate;
  until a stop condition is verified;
end;
```

Definitions:
PopNo: number of parallel populations
GenNo: number of generations to go without migration
PopSize: number of chromosomes in population
SA: boolean indicating if TSSA has just been implemented
r: fraction of population undergoing TSSA
PopRange: ratio of the minimum objective function value to the maximum in the population
α, λ: constants less than one.

3. Chromosome encoding in GATA and the initial population

The chromosome encoding in GATA is direct (Portmann 1996) in the sense that it simply consists of the lot size variables y_{jt} indicated in the mathematical model. The objective function value of a solution represented by a chromosome c is indicated by obj_c.

A chromosome in the initial population is generated by assigning a random lot size 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 augmenting the lot sizes of the previous periods randomly. Thus, a chromosome is guaranteed to have nonnegative inventory, but the capacity requirements implied by the lot sizes may violate the limits.

4. GA operators: reproduction, crossover, mutation

The reproduction operator multiplies each chromosome in proportion 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. The probability for reproducing a given chromosome k is given by,

$$\text{prob}_k = \frac{(1/\text{obj}_k)^2}{\sum_{k \epsilon \text{population}} (1/\text{obj}_k)^2}$$

However, reproduction is not executed immediately after TSSA is implemented, so that critical blocks of genes in the chromosomes not processed by TSSA are preserved in the population, because the objective function values of these chromosomes are comparably very large (due to infeasibility penalties) with respect to the feasible ones processed by TSSA.

The crossover operator is a two-point operator which randomly selects two parents from the population, and two periods, t_1 and t_2 randomly. y_{jt} are swapped across the two parents for all $t_1 \leq t \leq t_2$. The crossover operator may destroy the feasibility of inventory balance constraints. That is, there may be over production or under production in the offspring. These infeasibilities are repaired by adding/subtracting the necessary amounts to/from the lot sizes. The repair operation is carried out first by calculating the cumulative demand for each item, starting from the first period on to the last and checking if the cumulative lot sizes are greater than or equal to cumulative demand in each period. In the case that the latter condition is violated for any item, 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 item's inventory is checked in the last period and if inventory is positive, then the minimum between that item's lot size and its inventory level is deducted from the lot size. If the inventory level in the last period is still positive after deduction, 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.

An example will clarify the repair function in the crossover operator. Suppose in the first parent chromosome an item's lot size is (15, 35, 0, 70, 40) for a planning horizon of five periods. The demand for the item is (5, 40, 5, 25, 85) units. The second parent has lot sizes of (5, 60, 20, 75, 0) units. When $t_1 = 2$ and $t_2 = 4$, the offspring will have (15, 60, 20, 75, 40) units and (5, 35, 0, 70, 0) units, respectively. The first offspring has an overproduction of 50 units and the second one has an underproduction of 50 units. The forward pass identifies in the second offspring a backorder of 5 units in the second period and repairs it by adding 5 units to the second period's lot size. Similarly, 5 and 40 units of backorders are repaired in the third and fifth periods. The resulting modified lot sizes are (5, 40, 5, 70, 40) units. The backward pass repairs the positive inventory of the first offspring in the last period, by subtracting 40 units from the lot size of the last period and 10 units from the lot size of the fourth period to result in the lot sizes (15, 60, 20, 65, 0).

In GATA the number of offspring to replace the parent population is given by *PopSize*PopRange*. Thus, as the diversity of the population is lost, the number of new offspring 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 for the number of new offspring. Chromosomes in the parent population die out in descending order of objective function value. That is, the best chromosomes are preserved to carry on in the next generation.

The probability of selection for crossover is given by $(\text{maxobj} - \text{obj}_c)/(\text{maxobj} - \text{minobj})$, where minobj is the least objective function value in the current population and maxobj is the worst one. When maxobj equals minobj, then the crossover probability is set to one. Thus, a parent's selection probability depends both on its own objective function value and 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.

Each chromosome is a candidate for mutation with a mutation probability calculated in a manner similar to the crossover probability. However, only *a single* family/period pair (with positive lot size) in each chromosome is mutated. A selected y_{jt} is reduced by a random amount limited by the inventory level of family j in period t, and y_{jt+1} is augmented by the same amount, always preserving inventory nonnegativity.

5. Migration

The migration operator aims to cross chromosomes over from parallel populations and return the offspring to their respective parent populations. Chromosomes are repaired in case they violate inventory balance equations. Migrated chromosomes replace parent populations completely in order to obtain a good merge.

6. TSSA procedure

TSSA starts the search from an initial solution represented by the lot sizes in a randomly selected chromosome and perturbs the solution for a given number of times. Each time the current solution is perturbed the lot size of one item is reduced (augmented) in period t_1 and augmented (reduced) in period t_2 by a random amount *DeltaLot* where t_1 and t_2 are two consecutive periods.

The pseudocode for TSSA is given in Table 3. Here, a move is considered for execution if it is not found among the *size*(Tabulist1) moves in Tabulist1 which stores a number of selected item/period couples (j^*, t_1) whose lot size have been 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. The size of Tabulist1 is dynamic. In other words, if the current 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 tabulist size, *MaxSize*. The dynamic tabulist size is devised for helping the search leave 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 couple (j^*, t_1) not on Tabulist1 is selected randomly. t_2 randomly assumes one of the following values: $t_1 - 1$ or $t_1 + 1$. Then, a random amount, *DeltaLot*, which preserves inventory and lot size nonnegativity is selected, and the corresponding change in the objective function value, *CostDif*, is calculated.

A second tabulist, Tabulist2, stores a number of objective function values achieved in the last *size*(Tabulist2) iterations. 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. Tabulist1 has a short-term memory whereas Tabulist2 has a long-term one. A move can be executed if $Cost + CostDif$ is not on Tabulist2.

When *CostDif* is positive, a probability of acceptance, *PA*, is calculated to decide if the move is going to be executed. Unlike a TS procedure, rather than accepting the best nonimproving move in the whole neighbourhood, TSSA procedure calculates *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*). *PA* is calculated as follows:

$$PA(CostDif, tSA) = \exp(-CostDif/(Cost^* tSA))$$

where *Cost* is the cost of the current solution. After each nonimproving move, the temperature, *tSA*, is reduced as follows

$$tSA \qquad tSA/(1 + \beta\ tSA)$$

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. Rather, the value of β is dynamic and depends on the status of the search, similar to dynamic tabulists. 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.

To summarize, in TSSA, the SA move selection rule is combined with a short-term memory and this results in the so-called probabilistic TS. Further enhancements on TS and SA are implemented through the use of adaptive tabu list sizes and an adaptive cooling scheme.

7. Computational results

The performance of GATA is compared with the optimal solutions of 90 test instances generated by Özdamar and Bozyel (2000). The problems consist of four families to be scheduled on a planning horizon of six periods. The characteristics of these problems are provided in detail in Özdamar and Bozyel (2000).

The second set of 90 problems consist of 14 families and a planning horizon of six periods. The first 90 problems are solved to optimality by the optimization package HyperLindo, but the second set of problems cannot be solved optimally, so results are compared with the best solution found among all solution methods presented here.

Özdamar and Bozyel (2000) have developed for the CLSPOS the following approaches: the classical Hierarchical Production Planning approach and its extensions, an iterative approach (IAOP) solving the relaxed LP of model P given in Table 1 where setup times are omitted, a Genetic Algorithm approach with an indirect encoding (IGA) where a solution is represented by priorities for scheduling the items in each

Table 3. Pseudocode for TSSA.

```
**Procedure TSSA**;

{$ID_t$: remaining idle capacity in period $t$;
$UbLot$: upper bound on the lot size to be transferred;
$DeltaLot$: transferred lot size;
$Cost$: cost of current solution;
$CostDif$: cost difference caused by the move;
$BestCost$: cost of the best solution obtained so far;
$t_1, t_2$: time index; $j*$: family index
Accept: a positive outcome of the random test for a non-improving move}

begin {TSSA}
  if Decrease Lot then
    begin
      Select $t_1, j*$ such that $y_{j*,t_1} > 0$ and $(j*, t_1)$ is not on Tabulist1;
      Update Tabulist1;
      randomly select $t_2$; $(t_2 = t_1 - 1$ or $t_1 + 1)$
      if $t_1 < t_2$ then $UbLot = \min \{y_{j*,t_1}, I_{j*,t_1}\}$   {Preserving Inventory Feasibility}
      else $UbLot = y_{j*,t_1}$;
      Select a random quantity for $DeltaLot$ such that $0 < DeltaLot \leq UbLot$;
      Calculate $CostDif$ with respect to $ID_{t_1}, ID_{t_2}$ and $I_{j*t}$;
      If $(((CostDif \geq 0)$ and $(Accept))$ or $(CostDif < 0))$ and $(Cost + CostDif$ not on Tabulist2) then
        begin
            Update Tabulist2;
            Update $size$(Tabulist1) and $size$(Tabulist2) if necessary;
            Update $SA$ parameter $\beta$ if necessary;
            Transfer $DeltaLot*$ from period $t_1$ to period $t_2$;
            {$y_{j*,t_1} = y_{j*,t_1} - DeltaLot$; $y_{j*,t_2} = y_{j*,t_2} + DeltaLot$; Adjust $ID_{t_1}, ID_{t_2}, I_{j*,t}$}
            $Cost = Cost + CodtDif$;
            If $Cost < BestCost$ then begin $BestCost = Cost$; $Save\ y_{jt}$; end;
        end;
      end; {Decrease Lot}
  else {Increase Lot}
    begin
      Select $t_1, t_2, j*$ s.t. $(y_{j*,t_2} > 0)$ and $((j*, t_1)$ not on Tabulist1);
      Update Tabulist1;
      randomly select $t_2$; $(t_2 = t_1 - 1$ or $t_1 + 1)$
      if $t_1 > t_2$ then $UbLot = \min \{y_{j*,t_2}, I_{j*,t_2}\}$   {Preserving Inventory Feasibility }
      else $UbLot = y_{j*,t_2}$;
      Select a random quantity for $DeltaLot$ such that $0 < DeltaLot \leq UbLot$;
      Calculate $CostDif$ with respect $ID_{t_1}, ID_{t_2}$ and $I_{j*t}$;
      If $(((CostDif \geq 0)$ and $(Accept))$ or $(CostDif < 0))$ and $(Cost + CostDif$ not on Tabulist2) then
        begin
            Update Tabulist2;
            Update $size$(Tabulist1) and $size$(Tabulist2) if necessary;
            Update $SA$ parameter $\beta$ if necessary;
            Transfer $DeltaLot$ from period $t_2$ to period $t_1$;
            {$y_{j*,t_1} = y_{j*,t_1} + DeltaLot$; $y_{j*,t_2} = y_{j*,t_2} - DeltaLot$; Adjust $ID_{t_1}, ID_{t_2}, I_{j*,t}$}
            $Cost = Cost + CostDif$;
            If $Cost < BestCost$ then begin $BestCost = Cost$; $Save\ y_{jkt}$; end;
        end;
      end; {Increase Lot}
end; {TSSA}
```

period (inventory and capacity feasibility is always preserved by the constructive algorithm which evaluates the chromosome) and a pure Simulated Annealing (PSA) approach similar to TSSA. Among these approaches, PSA, which preserves capacity and inventory feasibility while executing moves, is reported as the best performing method with respect to the quality of solutions and computational efficiency. PSA does not keep tabu lists, it starts from a capacity and inventory feasible solution and in calculating upper bounds for *DeltaLot*, it considers the remaining idle capacity in a period where a lot size is augmented. A fixed value of 0.01 is assigned to the parameter β in PSA.

The computational results given in Table 4 consist of the results obtained by the three methods, IAOP, PSA, and IGA, reported as best by Özdamar and Bozyel (2000) as well as the results obtained by GATA. The average percentage deviation of the solutions from the optimum and their standard deviations are indicated. The simple algorithm IAOP works as well as the IGA which was run with a population size of 80 and a maximum generation number of 1500 for small problems. Significantly better results are obtained with PSA which has been re-run with 15 times each executing 10 000 moves. PSA starts from the initial IAOP solution. PSA is quite efficient with respect to computation times and is remarkably fast. The CPU times provided in Table 4 are observed on a *Pentium* 200 Pro PC with 32 MB RAM. Due to its heavy computational burden, IGA's performance is reported to be considerably poor in the second set of problems consisting of 14 families and six periods (Table 5).

To compare TSSA procedure fairly against PSA, the PSA procedure is re-run starting with a random initial solution (denoted as PSA-I in Table 4). Now, these results can be fairly compared with the TSSA* procedure, which works in exactly the same way as TSSA does, but it preserves capacity feasibility all the time. Although it starts from a random initial solution, its performance is considerably better than that of PSA. The latter result is due to the dynamic SA parameter, β, which is updated after five ineffective (not changing the current solution or deteriorating) moves, and the two dynamic tabu lists supporting TSSA. The minimum(-maximum) tabulist sizes for Tabulist1 and Tabulist2, are 5(10) and 10(100), respectively.

To demonstrate the effects on performance, GATA is run with different parameters. GATA-I has a single population (no migration occurs), but in order to compare with GATA-III which has three migrating populations, GATA-I is run three times, each time with a different random initial population. The same goes for GATA-II where a single population exists, but additionally, GATA-II uses 10 chromosomes provided by TSSA* procedure. GATA-III applies migration at every 25 generations ($GenNo = 25$). All GATA runs are carried out with $r = 0.1$, i.e., 10% of the population undergoes TSSA procedure whenever TSSA is activated. In all GATA variants, mutation is applied when *PopRange* is greater than 0.7 and TSSA is applied when *PopRange* is greater than 0.5. All of these parameters are set by testing GATA preliminarily on harder problems within the test set where all methods found difficulty in converging to the optimum solution.

An observation which is not indicated in Table 4 is that GATA-I, GATA-II and GATA-III find their best solutions in 25, 22, and 12 generations on the average, respectively. So, GATA converges quite fast. GATA-III terminates when a better solution is not obtained in the next migration whereas GATA-I and GATA-II terminate when a better solution is not obtained in three consecutive generations. The computation time required for GATA is also modest, because TSSA is activated once in every 10 generations on the average and its moves are

Table 4. Results for the 90 test problems (with four families, six periods) generated by Özdamar and Bozyel (2000).

Results for 90 problems (4 it.x6per.)	Previous results [Özdamar, Bozyel (1996)]			New results				
Solution procedure	IAOP	IGA	PSA	PSA-I	TSSA*	GATA I	GATA II	GATA III
% Deviation from the optimum (standard deviation)	5.46 (5.43)	5.62 (10.08)	1.08 (4.02)	4.37 (6.34)	0.20 (0.38)	0.15 (0.30)	0.04 (0.04)	0.0
Number of optimal solutions	15	25	39	34	86	84	89	90
Number of iterations	3–4 LP solutions	1500	15×10^4	15×10^4	15×10^4	3 times re-run	3 times re-run	1 run
Popsize/*PopNo*	–	80/1	–	–	–	50/1	50/1	50/3
Property of initial solution	LP	random	IAOP	random	random	random	random + 10 TSSA* solutions	random
Average CPUsecs/problem	2	420	1	1	1.3	1.8	2.6	5.4

Table 5. Results for the 90 test problems (with 14 families, 6 periods) generated by Özdamar and Bozyel (2000).

Solution procedure	IAOP	IGA	PSA	TSSA*	GATA I	GATA II	GATA III
% deviation from the optimum (standard deviation)	3.59 (2.59)	78.70 (77.73)	1.33 (1.73)	2.48 (2.68)	0.88 (1.05)	0.50 (0.86)	0.17 (0.36)
Number of iterations	3-4 LP solution	5000	150 000	150 000	3 times re-run	3 times re-run	1 run
PopSize/PopNo	–	40/1	–	–	150/1	150/1	150/3
Property of initial solution	LP	random	IAOP	random	random	random + 10 TSSA* solutions	random
Average CPUsecs/ problem	3	1176	1.25	3.2	4.8	6.7	8.3

limited by 20 000. Therefore, GATA-III executes TSSA on 5 chromosomes on the average until the best solution is obtained. GATA-III is run on parallel processors emulated on a local area network of PCs. Thus, a test problem is solved in a few seconds. The average percentage deviation from the optimum also show that GATA, which is a hybrid GA imbedding TS and SA features, outperforms all methods developed previously.

In Table 5 the results obtained on the second set of 90 problems are given. Here, performance is evaluated against the best solution obtained by all of the methods presented here. In this experimentation, 5% of the population undergoes TSSA procedure which executes 40 000 moves. When *t*-tests are applied to check the statistically significant differences between the means of each pair of methods we obtain the following results for the first set of 90 problems. Statistically significant differences exist between the means of all pairs of methods at all levels of significance except TSSA/GATA-I (insignificant), PSA/GATA-I (5% level of significance), and TSSA/PSA (2.5% level of significance). In the second set of 90 problems, except for PSA/GATA-I which is significant at 2.5% level of significance, all pairs of methods have statistically significant mean differences at all levels of significance. These results show that in smaller problems, GATA-I is not effective as compared with TSSA*. Furthermore, the ranking in the performance of TSSA* and PSA are swapped when problems of larger size are considered. Tabu lists seem to limit the search capabilities of SA in larger problems. Observing the results of both test problems one can conclude that it is certainly worthwhile to use GATA-II and GATA-III procedures despite the additional computational effort.

8. Conclusion

A GA procedure (GATA) is developed integrated with Tabu Search and Simulated Annealing (TSSA) to solve a production planning problem which is quite common in practice. GATA finds optimal results for test problems on which other methods have been tested extensively in a previous study (Özdamar and Bozyel 2000). In the latter reference, computational experience demonstrates that a GA which uses indirect coding is incapable of coping with the CLSPOS. GATA's direct coding preserves the good schemata that are implicitly inside the chromosomes. However, as it is difficult to build feasible solutions in direct coding, TSSA helps to visit as much as possible, feasible solutions in the neighbourhood of the infeasible solutions corresponding to most of the chromosomes in the population. The encouraging results obtained by GATA suggest that hybrid search heuristics which integrate GA, TS and SA may succeed in dealing with other NP-hard problems in production planning and control.

References

Alexandre, F., Cardeira, C., Charpillet, F., Mammeri, Z., and Portmann, M. C., 1997, Compu-Search Methodologies II: scheduling using genetic algorithms and artificial neural networks. In A. Artiba and S. E. Elmaghraby (eds). *The Planning and Scheduling of Production Systems: Methodologies and Applications* (Chapman and Hall) Chapter 10, pp. 301–336.

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

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

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

Günther, H. O., 1989, Planning lot sizes and capacity requirements in a single stage production system. *European Journal of Operational Research*, **31**, 223–231.

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

Özdamar, L., and Bozyel, M. A., 2000, The capacitated lot sizing problem with overtime decisions and setup times. To appear in the *IIE Transactions*.

Portmann, M. C., 1996, Genetic algorithms and scheduling: a state of the art and some propositions, *Proceedings of the Workshop on Production Planning and Control*, Mons, Belgium, 9–11 September, pp. I–XIV.

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