Infeasibility handling in Genetic Algorithm using Nested Domains for Production Planning

Maristela Oliveira Santos\textsuperscript{a} \quad Sadao Massago\textsuperscript{b} \quad Bernardo Almada-Lobo\textsuperscript{c,*}

\textsuperscript{a} Universidade de S\~{a}o Paulo - Instituto de Ci\^{e}ncias Matem\^{a}ticas e de Computa\c{c}\~{a}o, Av. Trabalhador S\~{a}o-carlense, 400, 13560-970 S\~{a}o Carlos-SP Brasil

\textsuperscript{b} Universidade Federal de S\~{a}o Carlos -Departamento de Matem\^{a}tica, Rod. Washington Luiz, Km 235 - C.P. 676 - 13565-905 S\~{a}o Carlos, SP - Brasil

\textsuperscript{c} Faculdade de Engenharia da Universidade do Porto, Rua Dr. Roberto Frias s/n, Porto 4200-465, Portugal

Abstract

In this paper we present a genetic algorithm with new components to tackle capacitated lot sizing and scheduling problems with sequence dependent setups that appear in a wide range of industries, from soft drink bottling to food manufacturing.

Finding a feasible solution to highly constrained problems is often a very difficult task. Various strategies have been applied to deal with infeasible solutions throughout the search. We propose a new scheme of classifying individuals based on nested domains to determine the solutions according to the level of infeasibility, which in our case represents bands of additional production hours (overtime). Within each band, individuals are just differentiated by their fitness function. As iterations are conducted, the widths of the bands are dynamically adjusted to improve the convergence of the individuals into the feasible domain.

The numerical experiments on highly capacitated instances show the effectiveness of this computational tractable approach to guide the search toward the feasible domain. Our approach outperforms other state-of-the-art approaches and commercial solvers.

Keywords: capacitated lot sizing and scheduling, feasible solutions, nested domains, genetic algorithms

1 Introduction

A genetic algorithm (GA) is a method based on natural evolutionary processes observed in biology, which may be used to solve a variety of combinatorial optimization problems, as first proposed by Holland [1975]. The GA starts with an initial population of individuals generated randomly or by specialized operators. The individuals are evaluated by a given fitness function, which indicates their merit with respect to the overall population and it is used to sort the population. Then, by using simulated crossover and mutation operations, the individuals reproduce, creating the members of the new population. Usually, the new population is generated by one of the two most common population replacement techniques: steady-state and generational.

\textsuperscript{*}Corresponding author. \textit{E-mail address:} almada.lobo@fe.up.pt; tel:+351 225082133
For highly constrained problems, depending on the representation scheme of the individuals, feasibility may not be guaranteed when crossover and mutation operations are applied, since feasible parents may easily give rise to infeasible offsprings. An important GA design issue in this case is whether or not to allow infeasible individuals to be part of the population. If infeasible solutions are not discarded, not only the objective function but also the constraints of the problem have to be taken into account in evaluating the performance of the individuals. Michalewicz [1995] discusses the difficulty of how “to order” the individuals and measure the feasibility and infeasibility, namely how to establish the order in the elements of a group of infeasible individuals, or how to prioritize the feasible individuals. Various strategies have been applied by the research community to measure infeasible individuals:

- using a representation in order to ensure that all solutions are feasible;
- repairing algorithms in order to transform an infeasible solution into a feasible one (can be complemented by rejection);
- relying on penalty functions that penalize unfeasible individuals.

The penalty function is the most used approach to tackle constrained optimization problems. However, one of the hurdles of working with these functions is to establish the weights of the violations during the process, i.e. to determine how the penalty function is designed and applied to infeasible solutions (Deb [2000]). In Toledo et al. [2008], a multi-population hierarchical GA procedure is developed to solve a synchronized and integrated two-level lot sizing and scheduling problem that arises at the soft drink bottling industry. During the process, if the algorithm is not able to restore the feasibility, the authors consider a function that penalizes constraint violation. Ozdamar and Birbil [1998] develop hybrid GA to solve the capacitated lot sizing and loading problem with setup times and overtime decisions. Infeasible solutions are penalized in proportion to their degree of infeasibility.

In a recent work, Defersha and Chen [2008] presented a heuristic method based on a GA for integrated cell formation and lot sizing where the GA determines the value of the integer variables, while the corresponding values of the continuous variables are determined by solving a linear programming subproblem. In order to fix the integer variables, a repair algorithm tries to obtain integer feasible solutions for most of the constraints, while for the remaining a penalty function is considered. Kimms [1999] uses a special data representation and repair methods to considerably increase the performance of an approach to solve the multi-level multi-machine proportional lot sizing and scheduling problem. Chootinan and Chen [2006] use a gradient of constraint to create more efficient repair algorithms.

In order to avoid working with the penalty functions, Kimbrough et al. [2008] split the population into two groups: the feasible individuals group which tries to improve the objective function, and the group of infeasible member, which attempts to reduce the constraint violation. Once a new individual is generated, it is allocated in one of these two mutually exclusive groups. Chu and Beasley [1998] propose the following strategy to implement a GA for highly constrained problems and to find good solutions for the Set Partitioning problem: instead of classifying the population based on a single fitness measure (combining original objective and penalty terms), the authors separate the fitness (related to the
objective function value) and unfitness (related to the constraint violation degree) scores. Each member of the population has a pair of values (fitness, unfitness). Taking this into account, the problem of determining the weight of the penalty term is eliminated at the cost of having to deal with a multiobjective function afterwards. Using these scores of unfitness, the population is divided into several subgroups of distinct characteristics, which are useful for the selection and replacement processes. This strategy is used in the subsequent works of Beasley [2004] for the cutting stock problem and of Beasley et al. [2001] for the aircraft landing scheduling.

Our work also relies on a GA based on the separation of fitness and unfitness scores. We consider $N$ products to be manufactured in the same capacitated machine over a discrete planning horizon of $T$ periods. Due to the sequence-dependency of setups in a product changeover, lot sizing and sequencing are simultaneously tackled. The objective is to find the strategy that meets demands without backlogging and minimizes both setup and holding costs. This problem is known as capacitated lot sizing with sequence dependent setup times and costs. The reader is referred to Goren et al. [2008] for an extensive review of evolutionary algorithms applied to lot sizing problems, and to Jans and Degraeve [2007] for a review of recent literature using a variety of meta-heuristics and other solution approaches to solve the lot sizing problem and extensions. An introduction of integer programming formulation of lot sizing problems can be found in Pochet and Wolsey [2006], as well as a good review on some variants of those problems.

The GA solutions are represented by the production sequence (binary variables). The values of the continuous variables (production and inventory amounts) are determined afterwards with an exact method. This procedure has similarities with those proposed by Defersha and Chen [2008], Meyr [2000] and Laguna [1999]. The latter two also consider production and inventory control problems in the presence of sequence-dependent setup times. Meyr [2000] develops two solution procedures based on the local search heuristic threshold accepting and simulated annealing to determine the production sequence and a minimum cost network problem is solved in order to determine the production variables. Laguna [1999] presents a Tabu Search method with short-term memory which coordinates linear programming problem and the traveling salesman problem. Information from the linear problem is used to obtain the production sequence.

In our problem, after the mutation and crossover operators, there is no guarantee that the corresponding linear sub-problem is feasible, due to capacity constraints. We develop a new scheme that relies on a feasibility classification using nested domains to determine the solutions according to the level of infeasibility, which in our case represents bands of additional production hours (overtime). Within each band, individuals are differentiated only by their fitness function. As iterations are conducted, the widths of the bands are adjusted to improve the convergence of the individuals into the feasible domain. Numerical experiments on highly capacitated instances, for which finding feasible solutions is a very difficult task, show the effectiveness and superiority of this procedure in guiding the search toward the feasible domain. This novel approach is able to find high-quality solutions on instances that state-of-the-art approaches fail to do so.

The paper is organized as follows. In Section 2, we present the model formulation of the capacitated lot sizing with sequence dependent setup times and costs. The concept of nested domains, as well as the solution representation are reported in Section 3. The
other components of our hybrid GA are detailed in Section 4. Numerical experiments are given in Section 5. Finally, conclusions drawn from the experimental results are provided in Section 6.

2 Model for CLSP with Sequence-dependent Setup Costs and Times

We consider a general single-stage model involving multiple items to be scheduled on a single machine, in which stockouts are not allowed (see Almada-Lobo et al. [2007]). The model size is defined by \((N, T)\) representing the number of products and the number of time periods, respectively. The index set \((i, j, t)\) is defined as: \(i, j \in \{1, 2, \ldots, N\}\) and \(t \in \{1, 2, \ldots, T\}\).

In order to define the input data required by the model, let \(d_{it}\) denote the demand of product \(i\) in period \(t\), \(s_{ij}\) and \(c_{ij}\) the time and cost incurred when a setup occurs from product \(i\) to product \(j\), \(h_i\) the capacity-unit inventory carrying cost for product \(i\) and \(C_t\) the machine capacity in period \(t\). In addition, the processing time of one unit of product \(i\) is given by \(p_i\), and the upper bound on the quantity of product \(i\) to be produced in period \(t\) by:

\[
M_{it} = \min \left\{ \frac{C_t}{p_i}, \sum_{u=t}^{T} d_{iu} \right\}.
\]

We assume that the triangle inequality with respect to the setup cost and time holds, i.e. \(c_{ik} \leq c_{ij} + c_{jk}\) and \(s_{ik} \leq s_{ij} + s_{jk}\) for all products \(i, j,\) and \(k\). This implies that it is not efficient to produce more than one batch of the same product in a given period.

The output of the model are the continuous decision variables \(X_{it}, I_{it}, V_{it}\) and the binary decision variables \(\alpha_{it}\) and \(T_{ijt}\). \(X_{it}\) denotes the quantity of product \(i\) produced in period \(t\), \(I_{it}\) computes the stock of product \(i\) at the end of period \(t\), and \(V_{it}\) ranks the production lot of product \(i\) in period \(t\), ensuring that the machine is set up for just one product on any given time. Finally, the setup carryover variable \(\alpha_{it}\) is set to 1 if the machine is set up for product \(i\) at the beginning of period \(t\), while \(T_{ijt}\) equals to 1 or 0 whether or not a setup occurs from product \(i\) to product \(j\) in period \(t\).

The model for CLSP with sequence-dependent setup costs and times, and setup carryover reads:

\[
\min \sum_{i} \sum_{j} \sum_{t} c_{ij} \cdot T_{ijt} + \sum_{i} \sum_{t} h_i \cdot I_{it}
\]  

(1)
\[ I_{it} = I_{i(t-1)} + X_{it} - d_{it} \quad \forall i, t \]  
\[ \sum_i p_i \cdot X_{it} + \sum_i \sum_j s_{ij} \cdot T_{ijt} \leq C_t \quad \forall t \]  
\[ X_{it} \leq M_{it} \cdot \left( \sum_j T_{ijt} + \alpha_{it} \right) \quad \forall i, t \]  
\[ \sum_i \alpha_{it} = 1 \quad \forall t \]  
\[ V_{it} + N \cdot T_{ijt} - (N - 1) - N \cdot \alpha_{it} \leq V_{jt} \quad \forall i \neq j, j, t \]  
\[ (X_{it}, I_{it}, V_{it}) \geq 0, (\alpha_{it}, T_{ijt}) \in \{0, 1\}. \]  

The objective function (1) is to minimize the sum of sequence-dependent setup costs and holding cost. Constraints (2) ensure the demand supply in each period without backlogging and (3) ensure that the machine is used for no longer than its available capacity. Requirements (4) ensure that a product is produced only if the machine has been set up for it. Constraints (5) impose the machine to be set up for exactly one product at the beginning of each period. Constraints (6) ensure a balanced network flow of the machine configuration states and carry the setup configuration state of the machine into the next period. Constraints (7) cut disconnected subtours off. In the presence of one subtour, they force the machine to be set up at the beginning of that period to one of the products belonging to the subtour (Almada-Lobo et al. [2008]). Finally, constraints (8) define the variables domain. Alternative formulations are known for this problem (see, for example, Gupta and Magnusson [2005] and Pochet and Wolsey [2006]).

3 Representation and fitness function

The problem of minimizing \( f : D \subset \Gamma \times \mathbb{R}^n \to \mathbb{R} \) such that \( D = \bigcup_{\lambda \in \Gamma} \{\lambda\} \times D_\lambda \) is decomposed in the standard way as \( \min_{(\lambda, X) \in D} \{ f(\lambda, X) \} = \min_{\lambda \in \Gamma} \{ \min_{X \in D_\lambda} \{ f_\lambda(X) \} \} \) where \( f_\lambda(X) = f(\lambda, X) \). Thus, one complex minimization problem is converted as combination of two simpler minimization problems, and appropriate methods are chosen for internal and external minimizations. In our problem, \( \lambda \) represents all binary setup-related variables and it is controlled by the GA due to its complex relationship with the function \( f \). For the problem \( \min_{X \in D_\lambda} \{ f_\lambda(X) \} \), only the continuous variables (production and inventory amounts) remain to be determined in \( X \) and it is relatively simple to be solved. Given a fixed set \( \lambda \) of parameters, the remaining linear subproblem is solved optimally using the following reformulation:

\[ z_\lambda = \min \sum_i \sum_{s=1}^{T} \sum_{t=1}^{s-1} \frac{h_{ij}}{p_i} \cdot \xi_{its} + \bar{g}(\lambda) \]
\[
\sum_{s=1}^{t} \xi_{ist} = p_i \cdot d_{it} \quad \forall i, t 
\]  
(10)

\[
\sum_{i} \sum_{s=t}^{T} \xi_{its} \leq C'_t \quad \forall t 
\]  
(11)

\[
\xi_{its} \geq 0, 
\]  
(12)

where \( \xi_{its} \) defines the amount (in capacity units) of product \( i \) produced in period \( t \) to meet the demand in period \( s \), \( C'_t \) the available capacity for production (\( C'_t = C_t - \sum_{i,j} s_{ij} \cdot T_{ijt}^* \)) and \( \bar{g}(\lambda) \) a fixed setup-related cost (\( \bar{g}(\lambda) = \sum_{i,j} c_{ij} \cdot T_{ijt}^* \)). It is well known that this transportation problem can be solved in polynomial time. In our work, it is solved using a state-of-the-art commercial solver. For each \( \lambda \), the value \( z_\lambda = \min_{X \in D_\lambda} \{ f_\lambda(X) \} \) is used to classify \( \lambda \) in the GA.

Now, the challenge is the proper classification of the infeasible parameters \( \lambda \) when \( \min_{X \in D_\lambda} \{ f_\lambda(X) \} \) is infeasible. In this case, \( D_\lambda \) is empty and no suitable value is returned to score \( \lambda \). One possible strategy to overcome this issue is to construct a sequence of broader nested domains \( D_\lambda = D^1_\lambda \subset D^2_\lambda \cdots \subset D^K_\lambda \) and then try to minimize \( f_\lambda : D^K_\lambda \rightarrow \mathbb{R} \). The infeasible parameter \( \lambda \) is scored as \( \ell_\lambda = \ell \) with \( \ell \leq K \), if \( \ell \) is the first index such that \( f_\lambda : D^K_\lambda \rightarrow \mathbb{R} \) yields a solution. We set \( z_\lambda = \min_{X \in D^K_\lambda} \{ f_\lambda(X) \} \) if \( \ell_\lambda \) exists. Using \( \ell_\lambda \) as the score of \( \lambda \), we establish the order of the parameters as the lexicographical order of the pair \((\ell_\lambda, z_\lambda)\). In case the sequence of nested domains is well defined, this method allows us to classify a significant amount of infeasible parameters.

### 3.1 (In) Feasibility classification using nested domains

In our problem, the additional overtime over a given capacity is used to define the sequence of nested domains. Let \( D^K_\lambda = \{ X: g_t(\lambda, X) \leq C_t + O_\ell \} \) where \( g_t(\lambda, X) \) gives the capacity restriction of the problem in period \( t \) and \( O_\ell \) the additional overtime. If \( O_\ell < O_m \leq O_K \) for \( \ell < m \), then the domains in the sequence are nested. To ensure that a significant amount of different sets \( \lambda \) are scored, we set the last (maximum) overtime (denoted by \( W \)) so that the lot-for-lot scheduling capacity can be attended as follows:

\[
W = \max_i \left\{ 0; \sum_i d_{it} + N \cdot \max_{i,j} s_{ij} - C_t \right\} 
\]  
(13)

As we will see later, at least one initial individual is generated by the lot-for-lot based heuristic. The lot-for-lot heuristic allocates to each period the demand for that period \( (X_{it} = d_{it} \text{ for every } i,t) \), without considering the capacity constraints.

Now, the appropriate values of the overtime need to be established in order to increase the power of the GA to evolve good feasible solutions. Since the overtime is represented by a real number, we work in one dimension. As the solution search progresses, we want to move infeasible \( \lambda \) towards the feasible region. The unfitness axis (that takes values from 0 to \( W \)) depicted in Figure 1, is divided into a certain number \( K \) of bands.
Instead of ranking the infeasible individuals according to a continuous overtime function, the individuals belonging to the same “unfitness band” are not differentiated by their unfitness, but solely by their fitness values. Experience shows that parents with similar infeasibility degrees are equally likely to generate children closer to the feasibility domain, avoiding the greediness of the usual ranking of unfit solutions. Moreover, it avoids a premature convergency to low/medium quality feasible solutions, by favoring solutions from the same band with the lowest objective function values (also called fitness score), even if they show a higher unfitness score.

The boundaries of each band are dynamically adjusted throughout the search, in such a way that, by excluding the feasible individuals of the population, the best unfitness value is always placed into the second band. This band is edged by goal limit and best limit. Only the first two bands may be of different widths. The interval between the best limit and \( W \) is subdivided into \( k - 2 \) bands with the same width in order to classify the elements which are further away from the feasibility domain.

At the end of each iteration, if a member of the current generation is classified in the first band, both the goal limit and best limit are updated, so that the member with the best unfitness value stays in the second band. The best limit is set to goal limit and the new goal limit becomes smaller (see Figure 2), moving the second band nearer to the feasibility region.

As iterations are done, the width of the first band (or the goal limit) tends to zero, and infeasible solutions converge to the feasible domain. The unfitness is only zero if the solution is feasible. Clearly, the goal limit has to be adjusted each time in an effective
way, so that unfit individuals are able to reach the first band, and then transform into feasible solutions. In other words, the nested domain boundaries are changed in order to promote the feasibility movement direction. In order to increase performance, if no element becomes a member of the first band during a fixed number of iterations, the goal limit is increased to reduce its distance to the best limit.

3.2 Solution Representation

Consider the Gantt chart of Figure 3 which illustrates a feasible solution for the model described in Section 2 for a given instance. The machine is set up for product 3 at the beginning of the planning period, i.e. $\alpha_{31} = 1$. The machine is then set up in period 1 to product 1, followed by 2 and, finally, by product 3, forming a cycle. In terms of the setup variables, this sequence is represented by $T_{311} = T_{121} = T_{231} = 1$. Observe that the last setup in period 1 is empty, since the idle time at the end of that period is used to set up product 3, the first one to be produced at the beginning of period 2, but no production of 3 occurs at the end of period 1. In period 2, the solution entails the production sequence $3 \rightarrow 1 \rightarrow 4$. The setup state of the machine for product 4 is carried from period 2 to period 3 and then, finally, there is a setup from product 4 to 1. At the end of the planning horizon, the machine is ready to produce product 1.

![Figure 3: Gantt chart of a solution to CLSP-based model](image)

Each individual is decoded with $T$ integers for initial setup ($i0[t]$) and $T$ variable length integer-strings, each containing the production sequence of one planning period ($sequence[t]$). Therefore, the length of the each string varies from one up to $N$. Figure 4 shows the individual representation of the solution presented in Figure 3.

![Figure 4: Individual representation example](image)

The first string does not consider product 3 in its last position, since no production occurs. Note that equivalent feasible schedules (same objective function) could be found by distributing the batch of product 3 in period 1 into two lots, respectively produced in the first
and the last positions of the sequence. Nevertheless, the original model does not differenti-
ate these schedules (actually the decision variables account for the same values), and so in
our work we consider that in case of a cycle, the production is concentrated in one batch.
By representing an individual as a set of production sequences instead of setup sequences,
gives us the flexibility to implement genetic operators in consistent way.

This representation defines the \( \lambda \) presented previously. The production (and inventory)
quantities are then determined by the remaining linear subproblem solved until optimally.
As the individual may not be feasible, its infeasibility degree is measured through the
unfitness band it belongs to. Consider the model displayed in Section 3, but replace re-
quirements (11) by the following set:

\[
\sum_{i} \sum_{s=t}^{T} \xi_{its} \leq C_{i}^{r} + O_{k} \quad \forall t, \tag{14}
\]

where \( O_{k} \) denotes the upper limit of band \( k \) and \( O_{0} \) equals to zero, limiting the feasibility
domain. The nested domain \( D_{K}^{\lambda} \) (set of feasible solutions given parameter \( \lambda \)) is described
by constraints (10), (12) and (14). As referred to before, we have to find the minimum \( \ell \)
(with \( \ell \leq K \)) for which the model returns a solution (and the individual belongs to band
\( \ell \)). If \( \exists k \leq K : D_{k}^{\lambda} \neq \emptyset \), then the individual does not belong to any of the nested domains
and it is discarded.

4 Selection, Mutation and Crossover

4.1 Parent Selection technique

Two individuals are compared according to the following rules:

- between two feasible solutions, the one having the better objective function value is
  preferred;
- between two infeasible solutions, the one belonging to the lower band is preferred; in
  case both are in the same band, the one having a better objective function value is
  preferred;
- a feasible solution is preferred over an infeasible one.

Our parent selection approach relies on the positional roulette that is implementated with
a pseudo-exponential distributed random number. A “bias” redistribution function
\( f_{\alpha}(t) = \frac{t}{(1 - \alpha - 2) \cdot 1 - t} + 1 \), (Schlick [1994]) is applied over a uniform distributed random number \( t \in [0, 1] \).
In our computational tests, \( \alpha \) is set to 0.25.

4.2 Generating the initial population

Four different methods are used to generate the solutions of the first generation to obtain
a diversified representation of the search space.
a) lot-for-lot pass ($X_{it} = d_{it}$, for every $i,t$), followed by a random lot sequencing;

b) lot-for-lot pass, followed by backward sequencing and amending procedures; the sequence is the $\text{minmax}$ algorithm detailed in Almada-Lobo et al. [2007], which places feasibility over optimality by trying to minimize the sequence-dependent setup times. The amending pass, in case of a capacity violation, tries to push overtime of a given period towards the previous period;

c) lot-for-lot pass, followed by a random lot sequencing and by the amending procedure of the previous method;

d) forward pass: all products are allocated in the first period to the machine, and random products are assigned to it in the following periods; lots are sequenced randomly.

Remark that the initial population is not allowed to contain duplicates (identical solutions) in order to promote diversification. All its solutions meet the demand requirements. Thus, in case of infeasibility, it is related to machine capacity violation. The implementation ensures that at least one individual of the population is computed by heuristic a). In other words, at the end of the first step of GA, at least one individual belongs to the nested domain (recall the definition of maximum overtime $W$, in Section 3).

4.3 Crossover operators

Crossover operators play a very important role since they are responsible for creating new solutions, hopefully better than their parents, and in this way to effectively search the solutions space. In our work, the following crossover operators were considered:

a) The one-point crossover works by randomly choosing one period $t^{*}$ (the crossover point) and then swapping segments of the two parent strings to produce a child string. Let $\text{ind}_1.\text{sequence}$ and $\text{ind}_2.\text{sequence}$ be the parent strings $\text{ind}_1.\text{sequence}[1], \ldots, \text{ind}_1.\text{sequence}[T]$ and $\text{ind}_2.\text{sequence}[1], \ldots, \text{ind}_2.\text{sequence}[T]$ (see Figure 4). Given a generated crossover period $t^{*}$ (with $1 \leq t^{*} \leq T - 1$), the child string (C) reads

$$C := \text{ind}_1.\text{sequence}[1], \ldots, \text{ind}_1.\text{sequence}[t^{*}] \cup \text{ind}_2.\text{sequence}[t^{*}+1], \ldots, \text{ind}_2.\text{sequence}[T].$$

b) The intersection operator constructs an offspring $C$ from parents $\text{ind}_1.\text{sequence}$ and $\text{ind}_2.\text{sequence}$ in the following way. Each of the $T$ child-strings is produced by checking if any integer (that represents the batch of a certain product) of the string of one of the parents is common to any integer of the string of the other parent. If affirmative, the integer is pushed back to the child string.

c) The union crossover operator merges two or more solutions as follows. First each string of the child is replicated from the respective string of one of the parents randomly selected. Let $R_{\text{ind}_1.\text{sequence}}$ and $R_{\text{ind}_2.\text{sequence}}$ be the ranking positions of parents 1 and 2 in the population, respectively. The selection of sequence is performed by roulette with weight $R_{\text{ind}_1.\text{sequence}}$ and $R_{\text{ind}_2.\text{sequence}}$, i.e. the probability of selecting $\text{ind}_1.\text{sequence}$ equals to $R_{\text{ind}_1.\text{sequence}}/(R_{\text{ind}_1.\text{sequence}} + R_{\text{ind}_2.\text{sequence}})$. Then, if the string of the non-selected parent contains integers that are not present in the recent-string, they are pushed back to the recently formed child string.
d) The last operator starts with a $T$-point crossover operator working with segments of the two parents. $C[t] := ind_1.sequence[t]$ or $C[t] := ind_2.sequence[t]$, according to roulette with same weight as the union crossover.

Except for the union crossover, we use a repair mechanism to meet the demand. For each period, the algorithm adds the items with positive gross requirements.

4.4 Mutation operators

The role of the mutation operator is to provide a higher level of diversification, in order to ensure that the whole solution space is searched. We develop and implement the following mutation operators:

a) Scramble: selects a random period and scrambles the productions in it. Note that the initial production of each period cannot be shuffled, since it would influence the production sequence of the previous period (which is given).

b) Insertion: chooses a period randomly (a string of one individual) and a product that is not part of that string, and inserts the product in a random position along the string, increasing the length of the string.

c) Displacement: picks a period randomly (a string of one individual) and removes a random product from the sequence, decreasing the length of the string.

d) Replacement: combines an insertion and a displacement procedure.

e) Setup Carryover: looks for improvements and diversification in the links between adjacent periods, by changing the initial state of the machine in a randomly selected period.

f) Minmax feasibility (random): re-sequences every string according to the minmax random version of the algorithm detailed in Almada-Lobo et al. [2007] who tries to minimize the setup times and use the random version of the amending procedure from the same authors.

g) Minmax feasibility (static): similar to above, but with static minmax and amending procedures.

h) Minmax feasibility: applies only the random version of the minmax procedure in a period per period fashion.

i) Shifting Forward: tries to reduce the inventory holding costs by shifting forward a fraction or an entire lot of every product from a source period to the following period.

After some operators are performed, repair algorithms might be necessary to keep the consistency of a production sequence. The percentage of elitism, mutation and crossover are determined by parameters. For initializer, mutation and crossover, the selection of their operators is performed by a weighted roulette with given weights.
4.5 Population Replacement Strategy

The replacement of the existing members by the new children is a fundamental stage of the evolution of a GA and there are various different strategies which determine how individuals are replaced at each generation. In our work, we adopt a generational replacement (Banzhaf et al. [1998] or Davis [1991]) due to auto combination of genetic operators, but here the elitism is required and clearing recommended. At each iteration, we retain into the next generation the best (fittest) \#Elite individuals from the previous one, and the rest of the members are replaced by the new individuals. The \#Elite implicitly defines the so-called generation gap (the percentage of population which is replaced at each generation).

The selection of the parents for mutation and crossover relies on the positional based roulette.

4.6 An illustrative example of the nested domain strategy

In this section, we present a numerical example showing some of the basic features of the nested domains, and illustrate the effectiveness of overtime bands of variable widths to encourage the search to move toward the feasible region. Consider the example for a fifteen-product, ten-period problem. We do not present here the complete instance, but it is worth mentioning that the approximate machine capacity utilization, without taking into account setups which eat into production capacity, is above 90% and the coefficient of variation of demand is 57%. The commercial solver CPLEX 11.2 from Ilog is not able to find any feasible solution in one hour of running time.

The population size was set to 100 individuals. Figures 5 and 6 show the convergence of the GA with fixed and variable bands, respectively. The $y$-axis represents the sum of setup and holding costs of the individuals (their fitness value), while the overtime (unfitness) is specified in the $x$-axis. Recall that overtime is less or equal to $W$ (defined in expression (13)), which in this example is equal to 88. A cross in the plot corresponds to the pair of values (fitness, unfitness) of one individual.

Note that the feasible solutions lie in the cost-axis. These plots show the way the population evolves over a certain number of iterations. Clearly, there is no initial feasible solution. In the fixed band case, the individuals tend to be projected to the upper boundary of their band, as the cost is likely to be lower and individuals from the same band are differentiated based on their fitness values. This makes their movement to a lower overtime band more difficult. The population of the 1000-th iteration does not contain any feasible individual.

Regarding the procedure with adjustable bands, the width of the second band is gradually reduced to project solutions to the nearest lower unfixed boundary, i.e. solutions are pushed to the first band toward the feasible region. As before, solutions tend to be concentrated at the upper limit of their band. Nevertheless, by controlling the size of the nested domains, one may ease the movement to a space close to the feasibility region. The Figure 6 illustrates the gradual approach of individuals to feasibility over iterations. After 200 iterations, 43 feasible solutions are already part of the population.
Figure 5: Fixed band
Figure 6: Variable width band
Table 1: Parameters used for generation of uniformly-distributed test data and characteristics of the instances

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut (Capacity utilization per period)</td>
<td>0.9</td>
</tr>
<tr>
<td>Capacity per period (C_t)</td>
<td>C_t = \sum_{i} d_{it}/Cut</td>
</tr>
<tr>
<td>Demand (d_{it})</td>
<td>U[0, 100] (coefficient of variation of 0.58)</td>
</tr>
<tr>
<td>Holding costs (h_i)</td>
<td>U[2, 10]</td>
</tr>
<tr>
<td>Setup times (s_{ij})</td>
<td>U[5, 10]</td>
</tr>
<tr>
<td>Setup costs (c_{ij})</td>
<td>c_{ij} = 100 \cdot s_{ij}</td>
</tr>
<tr>
<td>Processing time for one unit (p_i)</td>
<td>1 (one unit of time)</td>
</tr>
<tr>
<td>Classes of instances</td>
<td>N = 15/T = 10, N = 20/T = 10, and N = 20/T = 15</td>
</tr>
<tr>
<td>Number of instances of each class</td>
<td>10</td>
</tr>
</tbody>
</table>

5 Computational Experiments

We tested the three different nested domain strategies of our GA (variable band denoted as var.b., fixed band as fix.b. and no band as no b.) on the random data sets generated by the approach of Almada-Lobo et al. [2007], but only considering highly capacitated scenarios, with a high variability of demand figures and setup costs. The parameter values used to generate random test instances are shown in Table 1, where U[a, b] is the uniform distribution. It is our main driver to test instances for which finding a feasible solution is a hard task.

The default parameter settings for all problems were a population size equal to 100, mutation rate is set to 9, static crossover rate to 9, elitism rate to 3 and new individuals rate to 1. The total number of overtime bands is set at five. Because GA is reliant on randomization, we ran it five different times on the same problem instance using different random number seeds.

Computational experiments were performed on a Pentium T7700 CPU running at 2.4 GHz with 2GB of random access memory. A parallel CPLEX 11.2 from ILOG was used as the mixed integer programming solver, while GA was coded in C++ language, compiled using Visual C++ NET 2005. The maximum time for the overall search is set at one hour. To evaluate the quality of heuristics, we use the lower bound given by CPLEX after 3600 seconds when solving the Simple Plant Location reformulation of the model in Section 2.

Tables 2 to 4 summarize the computational results of the algorithms proposed for N = 15/T = 10, N = 20/T = 10, and N = 20/T = 15, respectively. The instances are not reported in the tables (e.g. in Table 2, the instance ex.1 for the fixed bands scenario) when the algorithms were not able to generate any feasible solution.

The first two columns refer to the nested domain strategy (dom.) and to the number of the instance to be solved. The next columns present the best feasible solution obtained (b.so) on the 5 trials of the algorithm, the number of iterations and the running times that the GA takes to first reach the final best solution (#it.b.so and t.b.so, respectively) but only for the run in which this best solution was obtained, the relative gap (%) between the best solution and the lower bound, the worst solution (w.so) on the 5 trials of the algorithm, the average fitness of the best feasible solution obtained in each run (av.so), the number of runs (#runs) out of five that were able to find a feasible solution after one hour, the average number of iterations (#av.it.) on the 5 trials of the algorithm, the average number of iterations to find the first feasible solution (#av.it.f.so), the average of the first feasible solution (#av.f.so) and the time average to find it (av.t.f.so). In addition, Table 2 reports the best solution obtained by CPLEX 11.2 in the last column.
| var.b. | ex1 | 68395 | 654 | 2805 | 5.7% | 69789 | 68914 | 832 | 5 | 38 | 75985 | 175 | 67508 |
| ex2 | 68147 | 402 | 1996 | 5.6% | 70286 | 69194 | 749 | 4 | 55 | 75667 | 302 | 67615 |
| ex3 | 62988 | 756 | 3000 | 4.7% | 66619 | 64982 | 735 | 5 | 11 | 77950 | 59 | 61860 |
| ex4 | 62637 | 797 | 3127 | 7.2% | 61289 | 63463 | 923 | 2 | 10 | 75828 | 55 | — |
| ex5 | 65818 | 723 | 3547 | 8.2% | 69742 | 68356 | 708 | 5 | 247 | 73739 | 1195 | — |
| ex6 | 67608 | 489 | 1890 | 5.7% | 68448 | 68117 | 906 | 4 | 90 | 74910 | 351 | — |
| ex7 | 68785 | 772 | 3597 | 8.4% | 70788 | 69445 | 788 | 5 | 13 | 80762 | 68 | — |
| ex8 | 68385 | 918 | 3588 | 8.4% | 70788 | 67181 | 901 | 5 | 9 | 80111 | 40 | — |
| ex9 | 63215 | 684 | 2735 | 7.8% | 66888 | 64708 | 837 | 5 | 28 | 75459 | 133 | — |
| ex10 | 75191 | 685 | 3422 | 24.0% | 77186 | 76430 | 760 | 4 | 149 | 81148 | 667 | — |

| fix.b. | ex2 | 69138 | 986 | 3479 | 7.2% | 75801 | 72470 | 939 | 2 | 583 | 76945 | 2283 | 67615 |
| ex3 | 64060 | 841 | 3411 | 6.5% | 67492 | 65223 | 873 | 5 | 66 | 78298 | 264 | 61860 |
| ex4 | 64309 | 1091 | 3552 | 10.1% | 64309 | 64309 | 1106 | 1 | 317 | 76685 | 980 | — |
| ex7 | 69156 | 914 | 3099 | 9.0% | 75534 | 70616 | 943 | 5 | 245 | 80143 | 988 | — |
| ex8 | 64666 | 953 | 3117 | 6.4% | 75534 | 64645 | 1088 | 5 | 132 | 80183 | 431 | — |
| ex9 | 63585 | 852 | 3517 | 8.4% | 66607 | 64982 | 1016 | 5 | 258 | 76490 | 837 | — |
| ex10 | 79521 | 896 | 3588 | 31.1% | 79521 | 79521 | 899 | 1 | 792 | 80500 | 3123 | — |

| no b. | ex1 | 79077 | 4143 | 3553 | 13.0% | 79077 | 79077 | 4163 | 1 | 3995 | 76735 | 3232 | — |
| ex2 | 65400 | 1294 | 3310 | 10.8% | 67114 | 66072 | 1488 | 5 | 101 | 77919 | 110 | 61860 |
| ex3 | 65121 | 1794 | 2566 | 7.2% | 66361 | 65689 | 2288 | 5 | 933 | 80758 | 752 | — |
| ex9 | 62454 | 1119 | 3553 | 6.5% | 71886 | 65833 | 2380 | 5 | 258 | 76490 | 837 | — |

Table 2: Computational results for $N = 15/T = 10$

| var.b. | ex1 | 88374 | 405 | 3409 | 9.4% | 84155 | 87498 | 429 | 5 | 42 | 100591 | 374 |
| ex2 | 88361 | 344 | 3534 | 9.6% | 90443 | 89601 | 386 | 5 | 4 | 103470 | 45 |
| ex3 | 89759 | 384 | 3375 | 9.1% | 93479 | 91442 | 402 | 5 | 9 | 102682 | 90 |
| ex4 | 87405 | 413 | 3354 | 9.0% | 90667 | 88905 | 420 | 5 | 9 | 104974 | 89 |
| ex5 | 91702 | 373 | 3355 | 10.3% | 93038 | 92930 | 355 | 3 | 143 | 102973 | 1492 |
| ex6 | 85358 | 353 | 3393 | 12.5% | 99754 | 89502 | 423 | 5 | 156 | 100972 | 1060 |
| ex7 | 95600 | 389 | 3563 | 8.8% | 97807 | 97144 | 399 | 4 | 107 | 103058 | 933 |
| ex8 | 90544 | 458 | 3604 | 12.4% | 95067 | 92513 | 447 | 5 | 39 | 89350 | 306 |
| ex9 | 98806 | 426 | 3441 | 16.8% | 100360 | 99765 | 447 | 3 | 207 | 102283 | 1584 |

| fix.b. | ex2 | 87900 | 400 | 3556 | 8.8% | 102000 | 92027 | 465 | 4 | 203 | 102587 | 1495 |
| ex3 | 90285 | 457 | 3416 | 11.6% | 92938 | 91843 | 473 | 5 | 3 | 102562 | 25 |
| ex4 | 87502 | 513 | 3588 | 9.4% | 89946 | 88348 | 495 | 5 | 2 | 108682 | 14 |
| ex5 | 91347 | 366 | 3286 | 10.1% | 91366 | 91357 | 407 | 2 | 155 | 109350 | 1318 |
| ex8 | 89778 | 522 | 3579 | 11.8% | 94573 | 92433 | 524 | 5 | 29 | 101530 | 210 |
| ex9 | 98841 | 503 | 3518 | 17.4% | 98841 | 98941 | 515 | 1 | 113 | 102740 | 757 |

| no b. | ex2 | 89914 | 726 | 3429 | 11.3% | 89914 | 89941 | 767 | 1 | 154 | 99971 | 501 |
| ex3 | 88601 | 739 | 3577 | 9.3% | 92668 | 91049 | 790 | 5 | 15 | 106522 | 36 |
| ex4 | 88704 | 741 | 3593 | 11.0% | 90667 | 90448 | 709 | 5 | 15 | 106522 | 36 |
| ex8 | 98914 | 901 | 3556 | 11.3% | 92492 | 90766 | 908 | 5 | 28 | 102410 | 59 |
| ex9 | 95153 | 782 | 3235 | 12.9% | 97300 | 96227 | 1075 | 2 | 354 | 102125 | 610 |
Table 4: Computational results for $N = 20 / T = 15$

<table>
<thead>
<tr>
<th>dom.</th>
<th>prob.</th>
<th>var.b</th>
<th>fix.b.</th>
<th>no b.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex1</td>
<td>137600</td>
<td>20</td>
<td>3613</td>
<td>18.8%</td>
</tr>
<tr>
<td>ex2</td>
<td>142200</td>
<td>20</td>
<td>3605</td>
<td>18.7%</td>
</tr>
<tr>
<td>ex3</td>
<td>131600</td>
<td>206</td>
<td>3587</td>
<td>14.9%</td>
</tr>
<tr>
<td>ex4</td>
<td>139120</td>
<td>205</td>
<td>3558</td>
<td>22.9%</td>
</tr>
<tr>
<td>ex5</td>
<td>134300</td>
<td>197</td>
<td>3612</td>
<td>19.8%</td>
</tr>
<tr>
<td>ex6</td>
<td>143600</td>
<td>184</td>
<td>3435</td>
<td>24.3%</td>
</tr>
<tr>
<td>ex7</td>
<td>138530</td>
<td>197</td>
<td>3600</td>
<td>19.1%</td>
</tr>
<tr>
<td>ex8</td>
<td>135650</td>
<td>197</td>
<td>3566</td>
<td>17.5%</td>
</tr>
<tr>
<td>ex9</td>
<td>139130</td>
<td>192</td>
<td>3568</td>
<td>23.1%</td>
</tr>
<tr>
<td>ex10</td>
<td>141370</td>
<td>186</td>
<td>3592</td>
<td>26.1%</td>
</tr>
<tr>
<td>ex1</td>
<td>150680</td>
<td>253</td>
<td>3575</td>
<td>30.1%</td>
</tr>
<tr>
<td>ex2</td>
<td>138230</td>
<td>222</td>
<td>3541</td>
<td>15.3%</td>
</tr>
<tr>
<td>ex3</td>
<td>141140</td>
<td>237</td>
<td>3557</td>
<td>23.3%</td>
</tr>
<tr>
<td>ex4</td>
<td>140120</td>
<td>240</td>
<td>3472</td>
<td>23.8%</td>
</tr>
<tr>
<td>ex5</td>
<td>132750</td>
<td>229</td>
<td>3545</td>
<td>18.4%</td>
</tr>
<tr>
<td>ex7</td>
<td>140710</td>
<td>239</td>
<td>3570</td>
<td>21.0%</td>
</tr>
<tr>
<td>ex8</td>
<td>144250</td>
<td>234</td>
<td>3578</td>
<td>16.7%</td>
</tr>
<tr>
<td>ex1</td>
<td>144270</td>
<td>35</td>
<td>3587</td>
<td>18.8%</td>
</tr>
<tr>
<td>ex3</td>
<td>132300</td>
<td>599</td>
<td>3558</td>
<td>15.6%</td>
</tr>
<tr>
<td>ex4</td>
<td>136320</td>
<td>402</td>
<td>3581</td>
<td>20.5%</td>
</tr>
<tr>
<td>ex5</td>
<td>135520</td>
<td>722</td>
<td>3563</td>
<td>22.5%</td>
</tr>
<tr>
<td>ex7</td>
<td>142500</td>
<td>501</td>
<td>3557</td>
<td>22.5%</td>
</tr>
</tbody>
</table>

Only for some of the smallest instances ($N = 15 / T = 10$) CPLEX 11.2 outperforms our heuristic approach. As instances increase, the superiority of the GA becomes clear. CPLEX 11.2 is only able to find a feasible solution for three instances of $N = 15 / T = 10$ within the one hour time limit. In all other seven cases, as well as for all the instances of types $N = 20 / T = 10$ and $N = 20 / T = 15$, it fails to do so.

Clearly, the variable band width strategy performs better than the other domain strategies. From the 50 trials on $N = 20 / T = 10$, the algorithm with variable band width found feasible solutions in 40 of them, contrasting to 22 of the fixed band algorithm and 18 of the procedure without overtime bands (see Table 4). In the class $N = 20 / T = 15$, var.b. variant succeeded in 48 out of 50 trials. It is worth mentioning that the best solutions were generated near the time limit stoppage criteria (see column t.b.so in tables 2-4). The best solutions were reported on instances for which the first feasible solution was obtained earlier. Similar results were obtained for the other instances. Regarding the quality of the solution for instances successfully solved by the three methods, the variable band has beaten the other two methods for all but one case ($N = 20 / T = 10$ against no b.), however the differences are quite small, as shown in Table 5. Furthermore, the var.b. approach finds a feasible solution in a lower average time and number of iterations than the other approaches.
6 Conclusion

In this paper, we propose a GA based on the separation of fitness (related to the objective function value) and unfitness (related to the constraint violation degree) scores to deal with a capacitated lot sizing and scheduling problem with sequence-dependent setup times and costs. In practical applications, it is important to find a good feasible solution. We developed a new scheme that relies on the feasibility classification using dynamic adjustable nested domains according to the level of infeasibility. A numerical experiment on highly capacitated instances, for which finding feasible solutions is a very difficult task, shows the effectiveness of this procedure in guiding the search toward the feasible domain.

Acknowledgments

This research was partially funded by the Fundação de Amparo a Pesquisa do Estado de São Paulo, Conselho Nacional de Desenvolvimento Científico e Tecnológico and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (Bolsista CAPES BEX-1870-09-2), Brazil.

References


