An Evolutionary Computation Approach to the Economic Lot Scheduling Problem

Cenk Tunasar and *Jayant Rajgopal*

University of Pittsburgh, Department of Industrial Engineering, Pittsburgh, PA 15261

Abstract: The economic lot scheduling problem (ELSP) addresses the problem of scheduling several items - each with its own static, deterministic demand - at a single facility, where only one item can be produced at any time. This classical production planning problem has been studied extensively over the last three decades and various heuristic solution methods have been proposed, most of them based on the concept of a basic period. A common drawback of the majority of the existing heuristics is the lack of explicit, algorithmic mechanisms to develop feasible schedules, and the consequent use of ad hoc procedures. We propose an efficient evolutionary computation technique for this problem. Our heuristic searches the problem domain with a population of solutions and converges very quickly to low cost, feasible schedules. We present our results in comparison to the best known heuristics.

1. Introduction

The *Economic Lot Scheduling Problem* (ELSP) is a common production planning problem that has attracted researchers for a long time. It is the problem of scheduling *N* items at a single production facility where the setup times, production and demand rates, and the associated setup and inventory holding costs for each item are allowed to be different. Each item is assumed to have a known static, continuous demand over an infinite planning horizon, which allows the

adoption of cyclical production patterns. For each item i =1,...,N ; the following are assumed to be given:

r_i : static demand rate (*units per unit time*)

p_i : static production rate (*units per unit time*)

S_i : sequence independent setup time per production lot (*unit time*)

A_i : setup cost per production lot (*$*)

h_i : inventory holding cost (*$ per unit per unit time*)

Additionally, we define the *intensity* of item i via $b_i=r_i/p_i$.

Suppose we define T_i as the cycle time for item i, i.e., the time that elapses between the commencement of two successive production runs for item i. The ELSP problem is to find feasible cycle times $T_1,...,T_N$ for which the average cost per unit time (C) of producing all N items is minimized. This cost C is equal to $\sum_i C_i$ where C_i is the cost per unit time for producing item i and is given by

$$C_i = A_i/T_i + h_i r_i(1-b_i)T_i/2\,. \qquad (1)$$

In the basic period approach, each T_i is some multiple n_i of a time interval W which we refer to as a basic period. Thus for each item:

$T_i = n_i W$ = the cycle time of item i,

$t_i = (r_i T_i)/p_i = b_i T_i$ = the processing time per lot of item i,

$\sigma_i = S_i + t_i = S_i + b_i T_i$ = the total production time per lot of item i.

A Lower Bound

Considering item i in isolation; the value of C_i is minimized when the cycle length T_i is given by

$$T_i^* = \sqrt{2A_i / h_i r_i (1 - b_i)}\,, \tag{2}$$

which yields a minimum cost of

$$C_i^* = \sqrt{2A_i h_i r_i (1 - b_i)}\,. \tag{3}$$

A solution having $T_i = T_i^* \ \forall \ i$ with a total cost of $\Sigma_i \ C_i^*$ is known as the *Independent Solution* (*IS*). This yields a lower bound on the cost of any feasible solution and clearly, in the rare case where the *IS* is feasible it is also the optimum solution to the ELSP. However, since the *IS* treats each item independently it is highly unlikely that it produces feasible solutions.

An Upper Bound

An upper bound on the optimum cost may be obtained by assuming a common cycle which is long enough to accommodate the production of every item, so that each item is produced exactly once during each cycle. This approach, developed by Hanssmann [1] is known as *Common Cycle* (*CC*) approach. Having all $T_i = T$, a necessary and sufficient condition for feasibility is

$$\sum_{i=1}^{N} S_i + b_i T \le T\,. \tag{4}$$

Denoting $\sum_i b_i$ by B, the minimum of the total average cost per unit time (C) subject to the feasibility equation (4) is obtained for a common cycle length of

$$T^* = Max\left[\left(2\sum_i A_i \Big/ \sum_i h_i r_i (1 - b_i)\right)^{1/2}\ ,\ \sum_i S_i \Big/ (1 - B)\ \right], \tag{5}$$

with an associated minimum cost of

$$C^* = \sum_{i=1}^{N} A_i / T^* + h_i r_i (1 - b_i) T^* / 2\,. \tag{6}$$

Proposed Approach

In this article we present a heuristic - local expansion search (LES) - for the ELSP. Our heuristic searches the problem domain for low cost solutions while allowing infeasible solutions throughout the search. The algorithm converges to a low cost, feasible schedule with the aid of a penalty function that penalizes infeasible solutions. In Section 2 we present some of the existing heuristics for this problem followed by a detailed discussion on the issue of feasibility in ELSP in Section 3. We then describe our heuristic in Section 4 and report some experimental findings in comparison to other heuristics in Section 5, followed by our concluding remarks.

2. ELSP Heuristics - A Brief Review

In this section, we examine some of the existing heuristics for the ELSP. Our aim is not to provide an exhaustive coverage of the literature; rather, we choose to briefly describe some of the better known heuristics in order to form a basis for comparison with the proposed method. Elmaghraby [2] was the first to provide a comprehensive overview of the heuristics for the ELSP that existed at that time; a more recent paper by Lopez and Kingsman [3] provides an overview and discusses later developments. As pointed out in these papers, the heuristics that have been proposed for this problem cluster around two approaches: the ones that assume a basic period W and represent the cycle times of all items as multiples of this W, and those which reject the concept of a basic period [4,5]. This study follows the first approach.

A tactic commonly followed by many of the heuristics is to impose a condition that ensures the existence of a feasible schedule, and to then optimize individual cycle times. In the original *Basic Period* (BP) algorithm of Bomberger [6], feasibility is ensured by choosing the basic period W such that it is long enough to accommodate the production of all items. This leads to a dynamic programming formulation where the item multipliers are determined recursively. BP has several drawbacks. First, it does not simultaneously solve for the multiplier vector $\boldsymbol{n}$ and the

basic period *W*; rather, it finds the best multipliers for a given basic period *W*. One still has to search for the *W* that yields the lowest cost, and Bomberger suggests a rather inefficient trial and error method where interpolation / extrapolation is used to search for the best *W* value. Second, and more seriously, BP only solves the problem within its defined feasible domain as determined by the restriction on the value of *W*. There may of course be lower cost solutions that are outside this feasible region. A third drawback comes in its implementation; the dynamic programming formulation requires a discrete representation of the basic period, which is continuous. Bomberger suggests a discretization by equal intervals. Such a scheme may provide inaccurate results, and in some cases may result in suboptimal solutions. The actual implementation is dependent on the discretization chosen, which accounts for the fact that different values of *W* and the optimum cost are reported in the literature for the same test problem of Bomberger.

An extension of Bomberger's method - the *Extended Basic Period* (EBP) algorithm - was provided by Elmaghraby [2]. It also assumes a basic period *W* and represents the cycle times by integer multiples of *W*. The difference is that the dynamic programming formulation now considers two consecutive basic periods each of duration *W* and loads the items into these two basic periods. Doing so, the feasibility constraint of BP approach is relaxed, allowing for the dynamic programming formulation to explore a somewhat larger feasible region. EBP clearly dominates BP in solution quality but requires more effort in developing the solution. While this approach could be generalized to loading more than two consecutive basic periods, the increase in modeling and computational complexity rule this out.

Stankard and Gupta [7] (S&G) devised a heuristic where they group items based on their cycle times. They present a necessary and sufficient feasibility condition which is less restrictive than that of Bomberger and yields a larger feasible region to explore. The major objection to this

approach is the forming of the groups. The grouping procedure is rather arbitrary and the number of possible groupings could in general, be very large.

An alternative to requiring feasibility at the start is to try and attain feasibility when a desirable solution becomes infeasible. Madigan's [8] approach (MDG) is based on the independent solution and the common cycle approach. It starts with the CC solution and modifies the item multipliers so as to minimize the deviations of the costs of individual items from the independent solution cost, while checking for feasibility at each step. Although this procedure is capable of producing good solutions, major drawbacks exist. First, the algorithm does not suggest any *systematic* way to modify the item multipliers. This makes it very hard to code this approach since individual judgment is required in updating the multipliers. Second, the basic period W is not updated based on the new multipliers and the procedure deviates from a cyclic production pattern. Elmaghraby [2] showed that Madigan's solution can be significantly improved by computing the optimum duration of W once the multiplier vector $\boldsymbol{n}$ is known.

A more efficient procedure for the ELSP is suggested by Doll and Whybark [9] (D&W) who use an iterative procedure to simultaneously determine item multipliers $\boldsymbol{n}$ and the basic period W. The starting solution is based on the independent solution, and the procedure iterates until the basic period W and item multipliers $\boldsymbol{n}$ are the same at two successive iterations. Once again, the major drawback of their algorithm is in ensuring feasibility, since there is no guarantee that the final solution will be feasible. If the solution is infeasible they vary the multipliers to try and achieve a feasible solution, but once again, do not provide any algorithmic procedure for this. Furthermore, if one develops a heuristic procedure to restore feasibility and links it to this method, it is possible that convergence may never be achieved. Goyal [10],[11] (GOY) developed

a very similar iterative procedure with a slight difference in how the item multipliers are determined. The same drawbacks of D&W also apply to Goyal's procedure.

Finally, we come to three procedures that claim to *explicitly* ensure feasibility of the final solution without resorting to any *ad hoc* steps. The earliest and best known of these is based on the extended basic period of Elmaghraby, and was proposed in Haessler and Hogue [12] and Haessler [13] (HAE). It is an iterative algorithm that successively determines $\boldsymbol{n}$ and W values until they converge, similar to the D&W algorithm. However, Haessler explicitly provides a method of testing the feasibility along with a systematic procedure for obtaining feasible solutions from infeasible ones. Feasible solutions are obtained by restricting the item multipliers to powers of two. In order to restore feasibility HAE systematically increase the basic period W until a feasible solution is found for the current values of $\boldsymbol{n}$. However, it should be noted that the value of $\boldsymbol{n}$ is fixed during the course of the search for feasibility, so that the feasible solution obtained may not be a good solution anymore. A more recent and simpler algorithm suggested by Geng and Vickson [14] (G&V) is very similar to Haessler's algorithm in the way it restores feasibility, although it is also more likely to have higher costs. This method has drawbacks that are similar to those of HAE. Lastly, we have the method of Larraneta and Onieva [15] (L&O) which presupposes that a reference subset of the products with relatively small independent cycle times have multipliers of one, while the frequencies of the others are related to their cycle times. The algorithm then tries to find the best reference subset and the multipliers of the other products. While this method is fairly simple, it is also rather restrictive in terms of limiting itself to a portion of the feasible space.

3. Feasibility of ELSP

The previous section shows that only three out of the large number of algorithms in the literature have explicit mechanisms for providing a final solution that is feasible, and all of these have their drawbacks. Checking for feasibility is a major problem when developing any solution technique for the ELSP. Hsu [16] has shown that the problem of finding a feasible schedule for the ELSP, or even proving that none exists, is NP-complete. Vemuganti [17] has shown that for the two product case, feasibility may be determined through a fairly simple necessary and sufficient condition. However, this is mostly of theoretical interest since it does not apply to problems having more than two items ($N \geq 2$). For these, he suggests a mixed integer linear program (MILP) with $N(N\text{-}1)$ constraints, $(N\text{-}1)$ continuous variables and $N(N\text{-}1)/2$ integer variables. Elmaghraby [2] presents another integer linear programming (ILP) formulation with fewer variables and constraints. However, from a practical perspective these are not of much use since these MILP formulations can be quite large, and the effort required to solve these could be much higher than the effort required to come up with the solution whose feasibility is being verified. Thus heuristic procedures are intuitively much more appealing.

For any problem, a necessary condition for feasibility is that $p_i > r_i$ for all i (the production rate must be higher than the demand rate). In addition, it is also necessary to ensure that there is enough time to produce all items at the facility. Consider a time interval T which is the product of individual item cycle times; $T=T_1*T_2* \ldots *T_N$. Over the interval T, there are T/T_i production runs of item i, each of length $\sigma_i = S_i + b_i T_i$. This leads to the following additional necessary condition for feasibility:

$$\sum_{i=1}^{N} \sigma_i \cdot T/T_i \leq T \quad \Rightarrow \quad \sum_{i=1}^{N} [S_i/T_i + b_i] \leq 1 \tag{7}$$

Unfortunately, the above conditions are not sufficient to ensure feasibility. Equation 7 is only concerned with the *total* time and implicitly assumes that setup times are divisible, but this is an invalid assumption since setups cannot in general, be preempted. Bomberger's BP approach avoids this problem by requiring $T_i=n_iW$ for all i and W to be long enough to accommodate the production of *all* items. Since the production run for i is of length $S_i+b_i(n_iW_i)$, we have

$$\sum_{i=1}^{N} S_i+b_i n_i W \le W \,. \qquad (8)$$

In this case the smallest cycle time is at least W, so that setups would never need to be preempted and hence feasibility is guaranteed. However, this feasibility condition is very restrictive since the basic period W could be very large, and there could easily be other cyclic policies with smaller yet feasible W values for which the total costs are much lower. Although one could come up with more relaxed versions of the above by considering two or more successive basic periods (e.g. Elmaghraby's EBP approach), it is impossible to express the general feasible region for the ELSP through a set of simple mathematical inequalities.

There is another major issue with respect to feasibility, which has received surprisingly little attention in the literature. This has to do with the fact that even when a feasible schedule is *known* to exist, it could in general be very difficult to determine. In other words, even if one were given values of W and n_i which were somehow guaranteed to be feasible, determining which items are produced in a specific basic period is not a trivial task. For example, suppose we have N=3 with n_1=1, n_2=3, n_3=3, so that the cyclic pattern will repeat every 3 basic periods (the least common multiple of n_1, n_2 and n_3), with three runs of product 1, and one each of products 2 and 3. Two possible patterns over a sequence of three basic periods are shown below in Table 1.

Table 1. Two Possible Patterns for $\boldsymbol{n}$=[1,3,3]

Basic Period	*1*	*2*	*3*
Pattern 1	1,2,3	1	1
Pattern 2	1,2	1,3	1

For the corresponding value of W it is possible that the second production pattern is feasible while the first one is not despite the fact that the values of n_i are identical for both. It is interesting that the results reported in the literature typically just provide values for W and n_i; reconstructing the actual production pattern from these is not necessarily an easy task. This is especially true when the values of N, and the LCM of the n_i are relatively large.

In summary, two conclusions may be drawn. First, with the difficulty of representing the feasibility constraints mathematically it appears that heuristic procedures provide the best option for deriving feasible cyclic production schedules for the ELSP. Second, it is also desirable that the algorithm provide the actual production pattern in addition to W and n_i. The approach detailed in this paper is based upon these two conclusions. An evolutionary algorithm is used to find the best values for the multiplier vector $\boldsymbol{n}$ and the basic period W. For each candidate solution a simple heuristic is used to check for feasibility by assigning products to time slots as follows:

Step 1 Define a complete rotation cycle $T=W*n_L$ where n_L is the least common multiple of the item multipliers $n_1,n_2,..,n_N$. The production sequence repeats after each complete rotation cycle.

Step 2 Partition the complete rotation cycle T into n_L slots, each having W units of time available.

Step 3 Compute the production time $\sigma_i=S_i+b_in_iW$ for each item.

Step 4 Sort the items in ascending order of their n_i values. Those items with the same n_i values are further sorted in a descending order of their production times (σ_i).

Step 5 Assign the item at the top of the list to the first basic period slot that has sufficient time available for its production, and in a similar fashion to subsequent slots with intervening gaps of n_i-1 slots, until the requisite n_L/n_i assignments for that item have been made. If such an assignment cannot be completed go to Step 7.

Step 6 Remove the item from the list and update the time available in each slot. If the list is non empty return to Step 5; otherwise a feasible assignment has been made and the heuristic stops.

Step 7 The heuristic has failed to find a feasible assignment - stop.

The logic behind steps 4 and 5 is that items with smaller multipliers have to be produced more frequently and thus allow us lesser flexibility with respect to their assignment to some uniform sequence of basic periods. Hence we try to assign these early while there is more space available. Similarly, for two items with the same multiplier, the one with the larger production time is more likely to have trouble fitting into a basic period, and so we assign these earlier when more space is available. This is a very simple heuristic and based upon our experimental work, it does quite well at finding a feasible assignment when one exists. By incorporating it into the procedure for finding $\boldsymbol{n}$ and W, one can be sure that in addition to the final values for the latter an exact pattern of production will also be specified. While one could possibly develop a more sophisticated heuristic for determining feasibility (in the extreme case, one could even solve the integer programs of Vemuganti or Elmaghraby), the benefits of doing so need to be balanced against the computational requirements, especially when the procedure is used as one step in an

outer algorithm for solving the ELSP. The above heuristic is fast and easy to code but tends to lose some of its efficiency when n_L is very large. However, this may be avoided by restricting n_i to powers of two as done by Haessler [12], in which case n_L is simply the maximum value of n_i.

4. Local Expansion Search Heuristic

In this section, we describe the evolutionary computation heuristic that is used to find the values of W and n_i. With computers becoming increasingly more powerful and less expensive, stochastic search techniques (of which genetic algorithms and simulated annealing are the two most well known) have become very popular in the field of combinatorial optimization. Genetic Algorithms (GA) are a class of machine-learning techniques that are so named due to their resemblance to natural evolution. They were pioneered by Holland [18] and Goldberg [19], and have been applied to combinatorial problems by various authors (e.g., Biegel and Davern [20], Muhlenbein et al. [21]). These iterative solution techniques maintain a population of candidate solutions, and at each iteration improvements are made by using information from good solutions in the current candidate population. Simulated Annealing (SA) is another stochastic search technique, that is derived from statistical mechanics, and is used to search for global minima in large optimization problems. Kirkpatrick et al. [22] were the first to introduce this technique and apply it to combinatorial optimization problems, primarily in response to the problems of applying traditional hill-climbing methods to global optimization. While SA algorithms employ a greedy search mechanism, they differ from traditional methods in that they allow some moves that worsen the cost function. In SA, if a solution increases the cost function it is still accepted, but with a probability that is inversely related to the magnitude of the cost increase.

A recent book edited by Reeves [23] is a very good reference for various applications of modern heuristic techniques such as the two mentioned above to combinatorial optimization

problems. The algorithm described in this paper is based on a stochastic neighborhood search and bears resemblance to both genetic algorithms and simulated annealing. It has the advantages of being robust, predictable, easy to understand and code, and very efficient, both with respect to the quality of the solution as well as the solution time. This particular technique has also been applied successfully by the authors to other areas, such as lotsizing in multistage assembly systems [24].

The algorithm (LES) is an iterative search technique and similar to GA, the search proceeds with an evolving population of solutions. The evolutionary mechanism employs information from the current population, but is biased towards the better members. The population of solutions at each iteration contains good solutions as well as some poorer ones. The latter have a small but finite role in the evolutionary mechanism so that like SA, the algorithm can get past local optima. Finally, as in traditional hill-climbing methods the improvement step is implemented with a local search which is expanded within a suitably defined small neighborhood of a selected solution. In this instance, every member of this neighborhood is allowed to become a member of the new population. While the search is stochastic in the selection mechanism with better solutions having a higher probability of selection, the logic used to select a local neighborhood is deterministic and predefined on the basis of the problem structure. In addition, the algorithm also allows for the possibility of looking at different regions of the feasible space by randomly generating (with some suitably low probability) a completely new solution at each iteration. A generic Local Expansion Search (LES) algorithm applied to a general optimization problem may be summarized as follows:

Step 0 *Initialization*

Create an initial population of candidates by generating p (p=population size) initial solutions, and sort this candidate set in descending order of solution quality.

Step 1 *Generation*

Create a new candidate set of *p* elements as follows:

Step 1A: Copy the best solution in the current candidate set to the new candidate set.

Step 1B: Randomly select an individual solution (*e*) from the current candidate set by means of a *biased selection scheme*,

Step 1C: Obtain a suitable *neighboring* solution set, *N(e)*, by perturbing *e*,

Step 1D: Copy the members of *N(e)* to the new candidate set one by one; if the population of the new set reaches *p* at any time then stop and go to Step 2.

Step 1E: With some small predetermined probability β, randomly generate a completely new solution and copy this to the new candidate set. Again, if the population reaches *p* go to Step 2. If not, go to Step 1B.

Step 2 *Termination Check*

Sort the newly created candidate list in descending order of solution quality. If the *termination criteria* are not met go to Step 1. Otherwise exit with the best solution in the current candidate list.

Specific differences between applications will arise in how the solution is represented (the solution encoding) and how the neighborhood is defined. The initial population could be generated in a completely random fashion, or it may consist of intelligently chosen candidates. A combination of both types of candidates is usually a good strategy. The iterative step generates a new population of solutions from the current one. The new set consists of the best solution from the current population, solutions that are neighbors of selected individuals from the current population, and occasionally, some completely new solutions as well. The individuals whose neighborhoods are explored are selected by means of a biased selection rule that favors better

population members. To do this the population is first sorted in descending order of quality. Then the population size p is multiplied by a squared uniform U(0,1) random number and the j^{th} member of the current population is selected, where $j=\text{integer}\{p*(U[0,1])^2\}$. Note that the expectation of a U[0,1] random number is 0.5 but a squared U[0,1] random number has an expectation of only 0.33. Since the population is kept sorted in increasing order of costs, this rule will select better members from the population with a higher probability but will also allow poor members to be selected occasionally. Thus the local expansion search will be performed primarily around better members of the population which gives the algorithm the strength of a directed search. At the same time, the inclusion of some poor members in the population helps the algorithm avoid premature convergence to a local optimum. The occasional introduction of completely new solutions to the population is also important since this allows the algorithm to look at different regions of the solution space. Conventional local search techniques typically lack this property since they only have local information.

The termination criterion may be defined as desired, e.g. stop after N generations, stop if no improvement is observed in the best solution for k successive generations, stop if the improvement in the best solution is less than α% over the last k generations, etc. The only user controlled parameters are the probability β of introducing a completely random solution and the population size p. Smaller values for β cause the search to be more focused, while relatively large values allow for a wider search space. Larger values for p enable the algorithm to evaluate more neighborhoods as well as more solutions within these neighborhoods, but at the cost of increased computation times.

While the proposed method resembles simulated annealing as well as genetic algorithms, it has some unique features that make it different. First, unlike simulated annealing we search with

populations of candidate solutions and employ no temperature parameter. Second, unlike genetic algorithms, we have no crossover operator which is a distinguishing feature of GA. Furthermore, the randomness in the evolution is not accomplished via mutation (a genetic algorithm operator that introduces randomness into the search), but rather by the generation of a completely new solution (some recent genetic algorithms have also started to employ this strategy and term it *migration*). Finally, the primary mechanism for generating new members of the population is not through crossover but via an expansion operator that generates a small set of neighboring solutions around a selected solution. The expansion operator is powerful because the definition of the neighborhood could be based upon problem specific features such as its solution structure. Moreover, in many problems, feasibility or a specific desirable solution structure may be lost during crossover, and genetic algorithms often require some sort of post processing to restore the structure. With the local expansion search algorithm, an appropriate definition of the neighborhood can obviate this problem.

Solution encoding

In the LES heuristic a candidate solution is represented by the vector of positive integer multipliers $\boldsymbol{n}$; a population of individual solutions is a collection of such integer vectors. The vector $\boldsymbol{n}$ is sufficient to represent a solution since the duration of the basic period W that is optimum for a given vector $\boldsymbol{n}$ can be determined from the following equation (e.g., [25])

$$W = \sqrt{\sum_i (A_i / n_i) \Big/ \left(\sum_i h_i r_i (1 - b_i) n_i / 2 \right)} . \qquad (9)$$

The individual item cycle times can then be computed as $T_i = n_i W$. Unfortunately, this solution is not necessarily feasible for any $\boldsymbol{n}$. In order to test for feasibility we use the heuristic procedure described in Section 3. If the current solution is found to be feasible then the solution quality Z is

simply the cost C of producing all items and is computed via $C=\sum_i C_i$ where C_i is given by (1). On the other hand, the heuristic may pronounce the solution infeasible; recall that this happens the first time the heuristic is unable to assign an item j to any uniform sequence of n_j basic periods within the cycle of n_L basic periods. Also recall that all items below item j in the sorted list generated in step 4 of the heuristic remain unassigned as well. If this is the case, we penalize the current solution by adding an appropriate penalty to the cost. Specifically, the penalty is an amount proportional to the sum of the processing times of item j and all other unassigned items:

$$Z = C + F * [\lambda * \sum_{i \in Q} \sigma_i] \tag{10}$$

where $F = \begin{cases} 0 & \text{if the solution is feasible} \\ 1 & \text{if the solution is infeasible} \end{cases}$,

Q = the set of all items not assigned to any basic period,

λ = penalty constant.

Furthermore, as in traditional penalty function algorithms for general optimization problems, the penalty constant λ is varied throughout the search by setting it to be proportional to the generation counter. During the earlier stages of the search λ is relatively low, allowing infeasible solutions within the population. The value of λ is gradually increased as the search proceeds, and reaches its maximum value at the end of the search when infeasible solutions are pushed out of the population and only good feasible solutions remain. A linear increase in the penalty constant value did well for the problems considered here, but one could just as easily describe alternative procedures for updating the penalty constant.

A sample generation step of the LES heuristic is depicted in Figure 1. In this example, the LES algorithm randomly selects the fourth individual solution from the current population by the

biased selection rule discussed earlier and generates two new neighboring solutions by increasing / decreasing the multiplier for the sixth item (randomly selected among items 1 through *N*). A completely new solution is also introduced to the new population with a pre-specified probability.

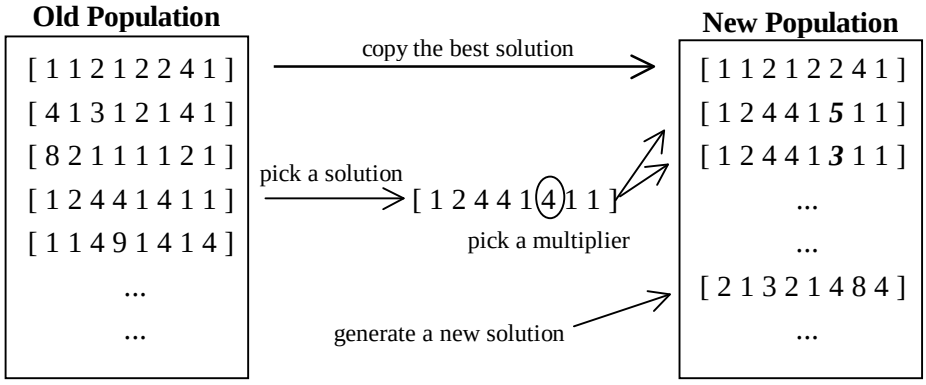

Figure 1. A Sample Generation Step of Local Expansion Search

We used a value of p=20 for the population size and β=0.1 for the probability of introducing a completely new solution. The stopping criterion is no improvement in the best solution for more than 20 consecutive generations. The parameter values used for the LES algorithm were chosen after an initial set of experiments; the effect of the values of p and β on the search have been discussed earlier. It was observed that LES is quite robust with respect to these values. We have experimented with population sizes of 10 to 50 with a probability of introducing a new member between 0.1 and 0.5, and the results in all instances have been quite good.

The LES algorithm is applied in two passes - a required and an optional one - for any given problem. In the first pass a search is performed for "powers-of-two" solutions only. In general, it takes less effort to converge to a powers-of-two solution. By restricting the search, the feasibility check of Section 3 becomes easier due to the fact that the lowest common multiple of the n_i's is simply the maximum of their values, whereas with general multipliers this value could

become quite large and unwieldy. Moreover, there are various arguments in the literature that support powers-of-two solutions, and our results on the problems tested seem to bear out the fact that in many cases improvements obtained by going to general multipliers are quite marginal. If such improvements are desired, the second (optional) pass may be applied where the search is extended from the previous point to look for non powers-of-two solutions as well.

5. Experimental Comparison

The general approach to the ELSP has been based on the single objective of cost minimization, although other aspects such as the length of the basic period, the length of the planning horizon, and the percentage of idle time could also be important considerations. In this study, we consider cost minimization as our only objective. The quality of the solutions yielded by the LES heuristic is analyzed in comparison with some of the existing solution techniques. Table 2 provides an overview of the methods selected for comparison. These methods were discussed earlier in Section 2. While this table is not necessarily exhaustive, we believe that it is adequate for our comparisons.

Table 2. Selected Methods for Comparison

Method	*Author*	*Reference No.*	*Type*	*Feasibility*
CC	Hanssmann	1	Analytical	Guaranteed
BP	Bomberger	6	Analytical	Guaranteed
EBP	Elmaghraby	2	Analytical	Guaranteed
MDG	Madigan	8	Heuristic	Not Guaranteed
S&G	Stankard & Gupta	7	Heuristic	Not Guaranteed
D&W	Doll & Whybark	9	Heuristic	Not Guaranteed
GOY	Goyal	10,11	Heuristic	Not Guaranteed
HAE	Haessler	12,13	Heuristic	Guaranteed
G&V	Geng & Vickson	14	Heuristic	Guaranteed
L&O	Larraneta & Onieva	15	Heuristic	Guaranteed

We consider a total of nine test problems each with N=10 products. Four of them are from Bomberger [6], and have been extensively used as test problems in the literature. These four problems are identical except that four different levels of demand are used. The base problem (*Bom1*) has the lowest demand rate and the remaining three problems are obtained by considering cases where the demands are two, three and four times that in *Bom1*. We call these problems *Bom2*, *Bom3* and *Bom4* respectively. Problem *Bom4* in particular has been extensively used as a benchmark problem in the literature. Next, we consider the six test problems presented by Fujita [26]. Fujita's first example is actually Bomberger's original problem which we have already included here as *Bom1*. Fujita's problems are denoted as *Fuj2*, *Fuj3*, *Fuj4*, *Fuj5* and *Fuj6*. It should be noted that all of these problems have been addressed in the literature, e.g., Lopez and Kingsman [3] use these problems in comparing alternative solution techniques for the ELSP.

Our decision to choose test problems that have appeared in the literature stems from two main reasons. First, we wished to report LES performance in comparison to existing heuristics and the *ad hoc* nature of most of these makes it hard to code them and do a comparative analysis on the same computing platform. Most of these methods also require too much user interaction and experience in order to converge to a feasible solution. In fact, Lopez and Kingsman [3] in their comparative study make this same point, and they reduced their choices to only HAE, G&V and L&O. Given this fact, we chose to compare our solutions to the ones that have already been reported in the literature. A second reason for our decision is the difficulty in generating random test problems. A purely random test problem may not be appropriate for testing different algorithms since the feasible space may be too tight and almost all of the algorithms can converge to the common cycle solution. This makes the test problem rather uninteresting, and there are really no guidelines in the literature for generating meaningful test problems for the ELSP.

In the experiment, the LES heuristic is replicated 10 times for each of the nine different problem instances. Multiple replications are made for the LES heuristic to account for the randomness built into it. For a given problem instance, it starts with a different initial population and generates a different sequence of populations each time, both of which could affect its performance. All computational runs were performed on a 90 Mhz Intel Pentium based microcomputer. Table 3 presents the results of our heuristic for the nine problems considered. For each problem instance, we present the best solution found among the 10 replications. We report the basic period W, the item multipliers $\boldsymbol{n}$ and the corresponding cost of production. Also note that although we have not listed it here, each solution also has a specific production schedule associated with it. Two solutions are reported for each problem instance, the powers-of-two solution and an unrestricted solution where the item multipliers are not restricted in any way.

Recall that in the first pass LES searches for the best powers-of-two solution and then in the second pass we relax this restriction and search for a better solution. The unrestricted search yields better solutions on all test problems except *Bom4* and *Fuj3*; for these two problems, we conjecture that the powers-of-two solutions found are optimal for the feasible region considered.

Table 3. Best LES Solutions

Problem	*Search Type*	*Cost*	*W*	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
Bom1	powers of 2	4130.8	31.7	8	2	2	1	4	8	16	1	4	2
	unrestricted	4078.2	36.4	9	2	2	1	3	6	11	1	3	2
Bom2	powers of 2	6104.5	25.6	8	2	2	1	2	4	8	1	4	2
	unrestricted	6056.5	27.4	9	2	2	1	2	5	7	1	3	2
Bom3	powers of 2	6843.6	26.4	8	2	2	1	2	4	8	1	2	2
	unrestricted	6773.6	21.9	9	3	2	1	3	6	10	1	3	2
Bom4	powers of 2	7697.1	23.4	8	2	2	1	2	4	8	1	2	2
	unrestricted	7697.1	23.4	8	2	2	1	2	4	8	1	2	2
Fuj2	powers of 2	4730.6	15.4	8	8	2	16	2	8	4	8	1	1
	unrestricted	4727.0	15.6	8	8	2	16	2	8	3	8	1	1
Fuj3	powers of 2	8800.8	11.0	8	2	8	2	1	2	2	4	8	16
	unrestricted	8800.8	11.0	8	2	8	2	1	2	2	4	8	16
Fuj4	powers of 2	21716.8	17.3	8	2	8	2	4	2	16	2	1	4
	unrestricted	21484.4	15.6	8	2	8	2	4	3	16	2	1	6
Fuj5	powers of 2	4194.4	19.0	8	8	8	8	1	4	1	8	8	8
	unrestricted	4172.2	18.8	6	7	10	10	1	4	1	7	11	8
Fuj6	powers of 2	21519.1	14.9	2	4	4	8	4	4	1	2	2	2
	unrestricted	21508.3	14.8	2	4	4	8	6	4	1	2	2	2

Next, we examine the robustness of the LES procedure. It was observed that for most of the test problems LES converges to the same solution. Table 4 reports the average and the standard deviation of the minimum cost across the ten replications for each problem instance. Computational requirements in CPU seconds per run are also reported in this table. For all the test problems except problem *Fuj6*, each of the ten runs in the powers-of-two search converges to the same solution and even for *Fuj6* the coefficient of variation is very low. For the unrestricted search, LES always converges to the same solution for 4 of the 9 test problems. For the

Table 4. Average LES Performance

Problem	*Search Type*	*Cost*		*CPU Seconds*	
		Avg.	*Std.*	*Avg.*	*Std.*
Bom1	powers of 2	4130.8	0	1.0	0.16
	unrestricted	4078.2	0	13.7	2.05
Bom2	powers of 2	6104.5	0	0.75	0.05
	unrestricted	6056.5	0	9.41	0.77
Bom3	powers of 2	6843.6	0	0.76	0.07
	unrestricted	6816.8	31.13	3.91	1.85
Bom4	powers of 2	7697.1	0	0.75	0.12
	unrestricted	7697.1	0	1.84	0.24
Fuj2	powers of 2	4730.6	0	0.97	0.18
	unrestricted	4730.3	4.32	4.32	1.67
Fuj3	powers of 2	8800.8	0	1.04	0.09
	unrestricted	8800.8	0	3.78	0.94
Fuj4	powers of 2	21716.8	0	0.94	0.05
	unrestricted	21546.4	57.31	6.15	1.35
Fuj5	powers of 2	4194.4	0	0.91	0.09
	unrestricted	4177.7	2.25	14.9	1.14
Fuj6	powers of 2	21549.9	49.75	1.52	0.11
	unrestricted	21540.8	42.40	13.49	1.37

remaining 5 problems one or more of the ten runs converged to other solutions. However, once again the coefficient of variation is still quite low. In summary, with powers-of-two it is very likely that a single run will suffice to find the best solution. With the general search, the search space becomes much larger and it is possible that LES will converge to different solutions. Given the small average CPU times, one strategy might be to run the LES algorithm ten (or even twenty) times and to adopt the best solution found.

We also experimented with LES by considering a large population size of 100 rather than 20. It was observed that both for the powers-of-two and the unrestricted search, the method converges to the best solution found across the ten replications of our earlier approach, but took a significantly longer time to converge. Thus one also has the option of running the LES heuristic much longer with a large population and using the single solution obtained as an alternative to the approach suggested in the previous paragraph. Table 4 also indicates that computational requirements are minor and do not vary much between replications. This is especially true for the powers-of-two search where convergence is achieved in about one second with almost negligible standard deviations.

In order to assess the performance of the LES heuristic, we compare it with the best solutions reported in the literature. In Table 5, we present the best reported solutions in the literature for nine of the test problems considered. We also include the independent solutions and the common cycle solutions which are the lower and the upper bounds respectively. For all except one of the test problems (*Bom1*), the LES algorithm with powers-of-two policies found the best reported solutions and in one case (*Bom3*) it found a better solution. Moreover, the unrestricted LES clearly outperformed all the other methods for seven of the nine test problems and equaled the best performance for the two others. Recall that Haessler's method which

Table 5. Best Reported Solutions for the Test Problems

	Bom1	*Bom2*	*Bom3*	*Bom4*	*Fuj2*	*Fuj3*	*Fuj4*	*Fuj5*	*Fuj6*
IS - LB	4075	6047	6739	7589	4663	8743	21418	4170	21219
CC - UB	5424	7627	8844	9880	6894	11536	26195	6302	22726
Best Solution	4090	6105	7184	7697	4731	8801	21717	4194	21519
Method	BP	HAE G&V	BP	HAE D&W G&V	HAE	HAE	HAE	HAE	HAE
LES powers-of-two	4131	6105	6844	7697	4731	8801	21717	4194	21519
LES unrestricted	4078	6057	6774	7697	4727	8801	21484	4172	21508

appears to be the best of the lot is restricted to powers-of-two solutions and it is therefore not unexpected that the unrestricted LES search finds better solutions.

Finally, it is worth emphasizing once again that along with W and $\boldsymbol{n}$ the procedure described here also comes up with an actual cyclic production schedule (with assignments of products to basic periods) which repeats itself. As an example, consider the solution to *Bom4* with item multipliers $[n_1,n_2,\ldots,n_{10}]$=[8 2 2 1 2 4 8 1 2 2] and W=23.4 and C=7697. The schedule provided for this is shown below in Table 6. Note that the cycle length is 8 consecutive basic periods and the production schedule repeats itself.

Table 6. Production Schedule for *Bom4*

Basic Period	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Items Produced	4,8,9,2	4,8,3,5, 10,6	4,8,9,2	4,8,3,5, 10,7	4,8,9,2	4,8,3,5, 10,6	4,8,9,2	4,8,3,5, 10,1

One point to be considered is that although the unrestricted LES results are very appealing with costs lower than any reported thus far in the literature, they could generate solutions whose cycle length is very long because the least common multiple of the item multipliers could be large. As an example, for *Fuj4* the minimum cost is found to be 21484.4 with a basic period W equal to

15.6 days and item multiplier vector ***n***=[8 2 8 2 4 3 16 2 1 6]. The lowest common multiple of these item multipliers is 48 which calls for a cycle length of 48*15.6=748.8 days, which is approximately 2 years! Although this solution is better in terms of the minimum cost, it assumes that the policy stays in effect for a relatively long time. In some instances, this may not be practical since the demand though static is likely to change during a long interval of time. However, there is no reason why the policy should not be followed until demand changes, at which time one could come up with a new schedule.

6. Conclusions

This paper presents a simple approach that finds good, feasible cyclic schedules for the ELSP. It is based on an evolutionary programming algorithm that uses concepts from stochastic as well as directed search. The algorithm is combined with a simple heuristic for checking feasibility. Infeasible solutions are allowed throughout the search but with a standard penalty mechanism that eventually forces them out of the search space. The technique is unique in combining the two objectives of minimum cost and feasibility for the ELSP into a single measure that is then optimized with an evolutionary computation technique. The final solution is not only feasible but also has an actual cyclic production schedule associated with it.

References

[1] Hanssmann, F., *Operations Research in Production and Inventory Control*, New York, John Wiley and Sons, 1962, pp. 158-160.

[2] Elmaghraby, S. E., "The Economic Lot Scheduling Problem (ELSP): Review and Extensions," *Management Science*, Vol. 24, No. 6 (1978), pp. 587-598.

[3] Lopez, M.A.N. and B.G. Kingsman, "*The Economic Lot Scheduling Problem: Theory and Practice*," *International Journal of Production Economics*, 23 (1991), pp. 147-164.

[4] Rogers, J., "A Computational Approach to the Lot Scheduling Problem," *Management Science*, Vol. 3, No. 3 (1958), pp. 264-291.

[5] Delporte, C. M. and Thomas, L.J., "Lot Sizing and Sequencing for N Products on One Facility," *Management Science*, Vol.. 23, No. 10 (1977), pp. 1070-1079.

[6] Bomberger, E. E., "A Dynamic Programming Approach to a Lot Size Scheduling Problem," *Management Science*, Vol. 12, No. 11 (1966), pp. 778-784.

[7] Stankard, M. F and Gupta, S K., "A Note on Bomberger's Approach to Lot Size Scheduling: Heuristic Proposed," *Management Science*, Vol. 15, No. 7 (1969), pp. 449-452.

[8] Madigan, J. C., "Scheduling a Multi-Product Single-Machine System for an Infinite Planning Period," *Management Science*, Vol. 14, No. 11 (1968), pp. 713-719.

[9] Doll, C. L. and Whybark, D. C., "An Iterative Procedure for the Single-Machine Multi-Product Lot Scheduling Problem," *Management Science*, Vol. 20, No. 1 (1973), pp. 50-55.

[10] Goyal, S. K., "Scheduling a Multi-Product Single-Machine System," *Operational Research Quarterly*, Vol. 24, No. 2 (1973), pp. 261-266.

[11] Goyal, S. K., "Scheduling a Multi-Product Single-Machine System-A New Approach," *International Journal of Production Research*, Vol. 13, No. 5 (1975), pp. 487-493.

[12] Haessler, R. W. and Hogue, S. L., "A Note on the Single-Machine Multi-Product Lot Scheduling Problem," *Management Science*, V. 22, N. 8 (1976), pp. 909-912.

[13] Haessler, R. W., "An Improved Extended Basic Period Procedure for Solving the Economic Lot Scheduling Problem," *AIIE Transactions*, V. 11, N. 4 (1979), pp. 336-340.

[14] Geng, P. C and Vickson, R. G., "Two Heuristics for the Economic Lot Scheduling Problem: An Experimental Study," *Naval Research Logistics*, Vol. 35 (1988), pp. 605-617.

[15] Larraneta, J., and L. Onieva, "The Economic Lot-Scheduling Problem: A Simple Approach," *Journal of the Operational Research Society*, Vol. 39, No. 4 (1988), pp. 373-379.

[16] Hsu, W. L., "On the General Feasibility Test of Scheduling Lot Sizes for Several Products on One Machine," *Management Science*, Vol. 29, No. 1 (1983), pp. 93-105.

[17] Vemuganti, R. R., "On the Feasibility of Scheduling Lot Sizes for Two Products on One Machine," *Management Science*, Vol. 24, No. 15 (1978), pp. 1668-1673.

[18] Holland, J. H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, (1975).

[19] Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing, (1989).

[20] Biegel, J. E. and J. J. Davern, "Genetic Algorithms and Job Shop Scheduling," *Computers and Industrial Engineering*, Vol. 19 (1990), pp. 81-91.

[21] Muhlenbein H., M. Gorges-Schleuter and O. Kramer, "Evolution Algorithms for Combinatorial Optimization," *Parallel Computing*, Vol. 7 (1988), pp. 65-85.

[22] Kirkpatrick, S., C.D. Gelatt Jr. and M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, 220 (1983), pp. 671-680.

[23] Reeves, C.R., *Modern Heuristic Techniques for Combinatorial Problems*, John Wiley & Sons, Inc., New York, (1993).

[24] Tunasar, C., and J. Rajgopal, "A Local Expansion Search Heuristic with an Application to an Assembly System," Technical Report No. 94-3A, Department of Industrial Engineering, University of Pittsburgh, Pittsburgh, PA 15261 (under review in *Journal of Operational Research Society*).

[25] Haessler, R., "A Note on Scheduling a Multi-Product Single-Machine System for an Infinite Planning Period," *Management Science*, Vol. 18, No. 4 (1971), pp. B-240-241.

[26] Fujita, S., "Marginal Analysis to the Economic Lot Scheduling Problem," *AIIE Transactions*, Vol. 10, No. 4 (1978), pp. 354-361.