GENETIC ALGORITHM FOR SOLVING ECONOMIC LOT SIZE SCHEDULING PROBLEM

Ruhul Sarker and Charles Newton
OR/MS Group, School of Computer Science
University of New South Wales, ADFA, Canberra, Australia

ABSTRACT

The purpose of this research is to determine an optimal batch size for a product, and purchasing policy of associated raw materials. The mathematical model for this problem is a constrained nonlinear integer program. Considering the complexity of solving such model, we investigate the use of genetic algorithms (GAs) for solving this model. We develop genetic algorithm code with three different penalty functions usually used for constraint optimizations. The model is also solved using an existing commercial optimization package to compare the solution. The detail computational experiences are presented.

1. INTRODUCTION

Consider a batch-manufacturing environment that procures raw materials from outside suppliers and processes to convert them into finished products for retailers. If the raw material purchasing policy is determined independently, then the optimal raw material purchasing quantity may not be equal to the raw material requirement for an optimal manufacturing batch size. To reduce the un-matching problem, it is necessary to optimise the activities of both raw material purchasing and production batch sizing simultaneously. Unfortunately, until recently, most of the analytical studies do not take all the costs for both sub-systems into consideration [1]. Therefore, the overall optimality of the system may be ignored.

The products produced in one manufacturing batch may be delivered to the retailer in \(m\) small lots (ie \(m\) retailer cycles per one manufacturing cycle) at fixed time intervals. This delivery system is very common in batch manufacturing environment [2]. In the pharmaceutical and chemical industries, the critical products are not delivered to the retailers until the whole lot is finished and quality certification is ready [3 & 4]. The inventory level of such products increases linearly at the production rate during the production up-time, and it forms a stair case pattern during production down time in each manufacturing cycle. In this paper, we also consider that a larger lot of raw material will be purchased that will be consumed in \(n\) manufacturing cycles. To simplify the modelling approach, we assume that both \(m\) and \(n\) are integers.

We have developed a total cost equation with respect to the production quantity, finished product delivery frequency for a manufacturing cycle and the number of production runs for a raw material purchasing cycle. Since the finished product delivery frequency per manufacturing cycle and the number of production runs per purchasing cycle are integer, the total cost function is nondifferentiable. There is always a natural limitation on storage space for raw materials and finished products. In addition to storage limitation, it is logical to consider the minimum truck-load for each delivery instead of unit transportation cost. So the resulting batch scheduling model is a constrained nonlinear integer program. Considering the complexity of solving such a model, we use genetic algorithms to solve this model. We use static [5], dynamic [6] and adaptive [7] penalty functions to handle the constraints within the
GAs approach. We use binary coding, to generate the population of candidate variables in GAs, with different crossovers and mutations. The results of three penalty function methods are analysed and discussed.

2. INTRODUCTION TO GA

Genetic algorithms (GAs) are optimization techniques that use the principles of evolution and heredity to arrive at near-optimum solutions to difficult problems. GAs follow a step-by-step procedure that closely matches the story of the rabbits discussed above; they mimic the process of natural evolution, following the principles of natural selection and "survival of the fittest". In the algorithm, a population of individuals (potential solutions) undergoes a sequence of unary (mutation type) and higher order (crossover type) transformations. These individuals strive for survival. A selection scheme, biased towards fitter individuals, selects the next generation. This new generation contains a higher proportion of the characteristics possessed by the "good" members of the previous generation; in this way good characteristics are spread over the population and mixed with other good characteristics. After some number of generations, the program either converges or is terminated, and the best individual is taken as the solution.

3. MATHEMATICAL MODEL

The unconstrained model, for manufacturing batch sizing, without transportation cost is developed by Sarker and Khan [3] and with transportation cost by Sarker and Newton [4]. Our problem is very much similar to Sarker and Khan [3] except the constraints imposed to make the situation more realistic. The notations used in the model are given below.

\[ D_p = \text{demand rate of a product } p, \text{ units per year} \]
\[ P_p = \text{production rate, units per year (here, } P_p > D_p) \]
\[ Q_p = \text{production lot size} \]
\[ H_p = \text{annual inventory holding cost, }$/\text{unit/year} \]
\[ A_p = \text{setup cost for a product } p (\$/setup) \]
\[ r = \text{amount/quantity of raw material required in producing one unit of a product} \]
\[ D_i = \text{demand of raw material for the product } p \text{ in a year, } D_i = rD_p \]
\[ Q_i = \text{ordering quantity of raw material} \]
\[ A_i = \text{ordering cost of a raw material} \]
\[ H_i = \text{annual inventory holding cost for raw material} \]
\[ P_{R_i} = \text{price of raw material} \]
\[ Q_{*i} = \text{optimum ordering quantity of raw material} \]
\[ x = \text{shipment quantity to customer at a regular interval (units/shipment)} \]
\[ L = \text{time between successive shipments} = x/D_p \]
\[ T = \text{cycle time measured in year} = Q_p/D_p \]
\[ m = \text{number of full shipments during the cycle time} = T/L \]

The mathematical model is as follows:

Minimize \( TC = \frac{D_p}{mx} \left( A_p + \frac{A_i}{n} \right) + \frac{mx}{2} \left( \frac{D_p}{P_p} + 1 \right) H_p + \frac{r_i H_i (D_p/P_p + n - 1)}{2} \cdot \frac{x}{H_p} \)

Subject to:
\[ n m x \leq \text{Raw\_S\_Cap} \]
\[ n m x \geq \text{Truck\_Min\_load} \]
\[ m x \leq \text{Fin\_S\_Cap} \]
\[ m \text{ and } n \geq 0 \text{ and integer.} \]
Constraint 1 is for raw material storage capacity limitation, constraint 2 for a lower limit on truck load and the last constraint is for finished product storage capacity. This is clearly a nonlinear integer program, where two out of three constraints are nonlinear.

4. PENALTY FUNCTION

Penalty function method converts the constrained model into an equivalent unconstrained model and then solves using suitable search algorithm. GA uses the same conversion procedure. The key issue in the penalty function approach is the choice of penalty coefficient in each iteration. There are three different approaches in GA to setting the penalty coefficient: the static one where the coefficient is a constant, dynamic where the coefficient is a predetermined monotonically nondecreasing sequence and adaptive which rely on population information to adjust the coefficient adaptively during the optimization process.

The method of static penalty [5] assumes that for every constraint we establish a family of intervals that determine the appropriate penalty parameters. The results are parameter dependent. As reported by Michalewicz [8], we use three level of fixed penalty coefficient for each constraints: 0.5, 0.5x10^3 and 0.5x10^6.

Joines and Houck [6] proposed dynamic penalties. They assumed that the penalty coefficient $\mu_k = (Ck)\alpha$, where $C$ and $\alpha$ are constants. This method requires much smaller number of parameters than the first method. Also, instead of defining several levels of violation, the pressure on infeasible solutions is increased due to the $(Ck)^\alpha$ component of the penalty term: towards the end of the process (for high values of $k$), this component assumes large values.

The adaptive penalty method uses feedback from the search process (see [8]). This method allows either an increase or a decrease of the imposed penalty during evolution. This involves the selection of two constants to adaptively update the penalty function multiplier, and the evaluation of the feasibility of the best solution over successive intervals of given generations. As the search progresses, the penalty function multiplier is updated every $N$ generations based on whether or not the best solution was feasible during that interval.

5. IMPLEMENTING GA

It is generally accepted that any GA to solve a problem must have five basic components.

5.1 Problem Representation

The GA representation often relies on binary coding. GAs work with a population of competing strings, each of which represents a potential solution for the problem under investigation. The individual strings within the population are gradually transformed using biological based operations. For example, two strings might 'mate' and produce an offspring that combines the best features of its parents. In accordance with the law of the survival of the fittest, the best-performing individual in the population will eventually dominate the population.

5.2 Initialize the Population

There are no strict rules for determining the population size. Larger populations ensure greater diversity but require more computer resources.
Once the population size is chosen, then the initial population must be randomly generated.

5.3 Calculate fitness

The fitness function is the penalty objective \( f_p \) function in our case. Each string is decoded into its decimal equivalent. This gives us a candidate value for the solution. This candidate value is used to calculate the fitness value.

5.4 Selection and Genetic Operators

Only the best-performing members of the population will survive in the long run. This can be done in a variety of ways. We generate a ranking of the competing strings based upon their fitness. Higher probability of choosing better solutions. The chromosomes, which survive the selection process, undergo genetic operations, crossover and mutation. Crossover permits two strings to 'mate' to produce offspring. The offspring may be fitter than their parents. Mutation introduces random deviations into the population. Mutation zaps a 0 to a 1 and vice versa. Mutation is usually performed with low probability, otherwise it would defeat the order building being generated through selection and crossover. Mutation attempts to bump the population gently into a slightly better course.

5.5 Parameters of GA

All GA runs has the following standard characteristics:
- Probability of crossover: 1.0
- Probability of mutation: \( 1/(\text{string length of the chromosome}) \)
- Two point crossover and bit-wise mutation.
- Population size: 50
- Number of generations in each run: 200
- Number of independent runs: 300

6. COMPUTATIONAL EXPERIENCE

The mathematical model presented in an earlier section is solved using three different penalty functions based GA described earlier. The data used for the model are:

\[ D_p = 4 \times 10^6 \text{ units, } P_p = 5 \times 10^6 \text{ units, } A_s = $50.00 \text{ per setup, } A_i = $3,000 \text{ per order, } H_i = $1.00 \text{ per unit per year, } H_p = $1.20 \text{ per unit per year, } r = 1.00 \text{ and } x = 1,000 \text{ units.} \]

The RHS values are:

\[ \text{Raw}_S\_\text{Cap} = 400,000, \text{Truck}_\text{Min}_\text{load} = 200,000 \text{ and } \text{Fin}_S\_\text{Cap} = 20,000 \]

The best solution found: \( m = 13 \) and \( n = 10 \) with \( TC = 1.8483E5 \). The comparison of the three methods are presented below:

<table>
<thead>
<tr>
<th>Method</th>
<th>Minimum</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Median</th>
<th>Maximum</th>
<th>Optimal %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic</td>
<td>1.8483</td>
<td>1.8484</td>
<td>0.0014</td>
<td>1.8483</td>
<td>1.8651</td>
<td>99</td>
</tr>
<tr>
<td>Static</td>
<td>1.8483</td>
<td>1.8484</td>
<td>0.0010</td>
<td>1.8483</td>
<td>1.8651</td>
<td>100</td>
</tr>
<tr>
<td>Adaptive</td>
<td>1.8483</td>
<td>1.8484</td>
<td>0.0014</td>
<td>1.8483</td>
<td>1.8651</td>
<td>99</td>
</tr>
</tbody>
</table>

As we can see the static penalty method provide optimal solutions in all test runs within the set 200 generations (see Parameters of GA). However the performances of other two methods are similar and close to static penalty method. All three methods have the same minimum and
maximum fitness function values. This problem can be solved using a commercially available optimization package like LINGO. However there is a possibility of having a local optimum. For example, the best solution we found with LINGO5: m = 17 and n = 12 with TC = 1.8865E5. This is a minimization problem. The objective is 2.07% higher than GAs solutions, and even 1.15% higher than maximum GAs objective function value recorded. The GAs code is developed in MATLAB on an UNIX machine. It takes few minutes to get the solutions. However LINGO takes less than 30 seconds on a PC.

7. CONCLUSIONS

The purpose of this research was to determine an optimal batch size for a manufacturing system. Like any other practical situation, the constraints like limited storage space and transportation fleet capacity are imposed. This problem forms a constrained nonlinear integer program. Considering the complexity of the model, the well known genetic algorithms (GAs) was applied. The genetic algorithms codes were developed with three different penalty functions usually used for constraint optimizations in evolutionary computation. The model is also solved using an existing commercial optimization package. The results of three penalty functions are compared. Their performances are almost similar in terms of number of times hitting the optimal values. As compared to the solution of the commercial package, the GAs solution is acceptable.

Acknowledgments

This work is supported by UC special research grant, ADFA, UNSW, Australia, awarded to Dr. R. Sarker. We like to thank Mr. T. Runarsson for developing GA code and experimentation.

References