ON THE ECONOMIC LOT SCHEDULING PROBLEM

SUN HAINAN
(B.SC, University of Science and Technology of China)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2007

ACKNOWLEDGMENTS

I would like to express my gratitude and appreciation to my two supervisors, Associate Professor Huang Huei Chuen and Assistant Professor Jaruphongsa Wikrom, for their invaluable advice and patient guidance throughout the whole course of my research. Without them, it would be impossible for me to finish the work reported in this dissertation. I also would like to thank all the other faculty members of the ISE Department, from whom I have learned a lot through the coursework.

A special thank to my parents. They gave me all the encouragements and supports I needed when I was in the low moments that inevitably occurred during the whole course of my study. This dissertation is dedicated to them.

Contents

SUMMARY

The Economic Lot Scheduling Problem (ELSP) has occupied researchers for more than fifty years. Scheduling production of multiple products on a single machine under capacity constraints is one of the classic problems in operations research.

As far as we know, no one has presented a procedure to determine the optimal lot sizes and production sequence for the general ELSP. Many policies have been proposed to reduce the complexity of the problem. This dissertation uses the Extended Basic Period (EBP) and Power-of-Two (PoT) policy for this problem and develops several algorithms under this policy.

The problem is formulated as a nonlinear integer programming problem. The optimal solution is found by treating one of the variables as a parameter and solving the problem by a series of integer linear programming problems. It is the first algorithm that can find the optimal solution under the EBP and PoT policy, although it takes a long time to determine the optimal solution. A heuristic based on insights drawn from the algorithm is developed. The heuristic yields solutions almost as good as the optimal solutions and reduces the running time dramatically. A genetic algorithm is also developed for this problem. This algorithm produces solutions better than those obtained by earlier genetic algorithms in the literature without the PoT restriction and it is very fast. It finds the optimal solutions under this policy for all the benchmark problems. In addition, it finds the optimal solutions under this policy for about 95% of all the randomly generated problems.

We also consider the Multiple-machine ELSP (MELSP). The MELSP schedules many products on multiple machines. It is assumed that the machines are identical and the products cannot be split on different machines. A genetic algorithm under the Common Cycle (CC) policy is presented with an integer encoding scheme. The solution dominates a previous heuristic under the CC policy for this problem and the running time does not vary much when the machines are heavily loaded, which is not guaranteed by the previous heuristic.

Based on an earlier study in the literature, the solution under the CC policy is quite close to the lower bound of the general version of this problem. However, we observe that the earlier study only tested the CC policy when there are either 5 or 10 machines. From our computational results, we see that the CC policy is not as good when there are less machines. A less restrictive policy, the EBP and PoT policy, is used for solving this problem. Again, a genetic algorithm is used and it is found that the solutions are a lot better than the genetic algorithm under the CC policy, especially when the number of machines is small. Probably due to the difficulty of finding a good encoding scheme, no one has applied genetic algorithms for solving the MELSP before. We find that the genetic algorithm works well for solving the MELSP with a good encoding scheme.

List of Figures

List of Tables

NOMENCLATURE

BP	Basic Period.
CC	Common Cycle.
EBP	Extended Basic Period.
EH	Efficient Search.
ELS	Equal Lot Sizes.
ELSP	Economic Lot Scheduling Problem.
GA	Genetic Algorithm.
GACC	Genetic Algorithm under the Common Cycle policy.
GAEBP	Genetic Algorithm under the Extended Basic Period and Power-of-Two policy.
HH	Haessler's Heuristic.
IS	Independent Solution.
LB	Lower Bound.
MELSP	Multiple-machine Economic Lot Scheduling Problem.
OPT	Optimal solution under the Extended Basic Period and Power-of-Two policy.
PoT	Power-of-Two.
SS	Small Step search algorithm.
ZIP	Zero Inventory Production.
I	Number of products.
T_i	Cycle time for product i.
C_i	Average cost per unit time for product i.
A_i	Setup cost for product i.
h_i	Inventory holding cost per unit time per unit for product i.

r_i	Demand rate for product i.
p_i	Production rate for product i.
ρ_i	r_i/p_i, production density for product i.
ρ	Sum of ρ_i for ELSP.
H_i	$h_i r_i(1-\rho_i)/2$, the holding cost factor for product i.
C	Total average cost per unit time for all products.
T_i^*	Optimal cycle time of the Independent Solution for product i.
C_i^*	Optimal cost of the Independent Solution for product i.
s_i	Setup time for product i.
n_i	Multiplier for product i.
j_i	Production position for product i.
L_i	Lower bound of n_i.
U_i	Upper bound of n_i.
W	Basic period.
$\boldsymbol{n}$	Multiplier vector $(n_1,\ \ldots,\ n_I)$.
$\boldsymbol{j}$	Production position vector $(j_1,\ \ldots,\ j_I)$.
K	Number of basic periods in a common cycle.
T	Common cycle time of all products.
T^*	Optimal common cycle time.
S_k	Set of products in the kth basic period.
t_i	$\log_2 n_i$.
μ_i	Lower bound of t_i.
ν_i	Upper bound of t_i.
x_{i,t_i,j_i}	0-1 Decision variable, 1 if the multiplier of product i is 2^{t_i} and the production position is j_i.

y_{i,t_i}	0-1 Decision variable, 1 if the multiplier of product i is 2^{t_i}.
$PoT(\cdot)$	Formulation of the ELSP under the EBP and PoT policy.
$PoT(W)$	Formulation of $PoT(\cdot)$ given W.
$f(W)$	Optimal objective value of $PoT(W)$.
$g(W)$	Optimal objective value of $PoT(W)$ relaxing capacity constraints.
W_{LB}	Lower bound of W.
W_{UB}	Upper bound of W.
w_j	jth junction point of $g(W)$.
I_p	pth interval of the curve of $g(W)$ divided by junction points.
$\underline{w}_p$	Left point of I_p.
$\overline{w}_p$	Right point of I_p.
$\hat{w}_p$	Minimum point of $g(W)$ in I_p.
$C(W, \boldsymbol{n})$	Total Cost given W and $\boldsymbol{n}$.
$\underline{C_p}$	Minimum value of $g(W)$ in I_p.
$\overline{C}$	Global upper bound of $f(W)$.
$\boldsymbol{n}^p$	Optimal multiplier vector of $g(W)$ in I_p.
$\boldsymbol{n}(W)$	Optimal multiplier vector for given W.
$\boldsymbol{x}(W)$	Optimal solution of $PoT(W)$.
W_R	A lower bound of W.
W^*	Optimal basic period.
M	Number of machines.
T_m	Cycle time of products on machine m.
W_m	Basic period of products on machine m.
$\underline{W_m}$	A lower bound of W_m.
a_i	Machine number on which product i is produced.
$\boldsymbol{a}$	Machine number vector $(a_1, a_2, \ldots, a_I)$.

K_m	Number of basic periods in a cycle on machine m.
S_{mk}	Set of products produced in period k on machine m.
T_m^*	Optimal cycle time of products on machine m.
W_m^*	Optimal basic period of products on machine m.
N	Number of chromosomes in a population.
f	Fitness value of a chromosome.
f_m	Fitness value of machine m in the chromosome.

Chapter 1

Introduction

1.1 Economic Lot Scheduling Problem

The Economic Lot Scheduling Problem (ELSP) has received attentions from researchers for more than fifty years as many companies produce several different products but own only one or a few production lines. Thus, it is important for the company to schedule the cycle times and the production sequence so that the demand of all products can be satisfied at the minimal cost.

In the literature, the ELSP is defined as a problem to schedule several products on a single machine over an infinite planning horizon. There are a setup cost and a setup time associated with producing each product and the objective is to determine the lot sizes and the production sequence so as to minimize the holding and setup cost per unit time. It can be characterized as follows:

- Only one product can be produced at a time on the single machine.

- A setup cost and a setup time associated with producing each product are known and constant.
- The setup cost and setup time depend only on the product.
- The demand rate and production rate for each product are known and constant over an infinite planning horizon and all demand must be met without backlogging.
- Holding costs are directly proportional to inventory levels.

Essentially, the ELSP arises from the desire to produce the products with a cyclical production pattern based on economic manufacturing quantity calculations for individual products. However, when two or more products compete for the machine's capacity, a compromising schedule needs to be developed so that the total production cost is minimized while the cyclical patterns of production are still maintained.

1.2 Multiple-Machine Economic Lot Scheduling Problem

The ELSP deals with the production that involves only a single machine. However, many real problems have two or more machines. Thus, it is of great importance to solve the Multiple-machine ELSP (MELSP). The MELSP has to determine the allocation of products to different machines, the lot sizes of the products on different machines and the production schedules of the products

on all the machines to minimize the total average inventory and setup costs for all the machines. In this dissertation, we discuss the MELSP with identical machines and it can be described as follows:

- Only one product can be produced at a time on a machine.
- There are a setup cost and a setup time associated with producing each product.
- The setup cost and setup time depend only on the product.
- The demand rate and production rate for each product are known and constant over an infinite planning horizon and all demands must be met without backlogging.
- Holding costs are directly proportional to inventory levels.
- The machines are identical with respect to the production costs and the production rates for each product.
- The production of a product cannot be split on different machines.

1.3 Contributions of Dissertation

The main contributions of the dissertation can be outlined as follows:

- Formulate and find the optimal solution of the ELSP under the Extended Basic Period (EBP) and Power-of-Two (PoT) policy. The EBP and PoT policy is a good policy for solving the ELSP. However, up to our knowledge, no one has characterized an optimal solution procedure under this

policy.

- Develop a good and fast heuristic method for the ELSP under the EBP and PoT policy based on insights drawn from the optimal solution procedure.

- Propose a Genetic Algorithm (GA) for the ELSP under the EBP and PoT policy, which is a lot faster than the previous methods while finding the optimal solutions for almost all the problems tested.

- Solve the MELSP under the Common Cycle (CC) policy with a GA, which is shown empirically to be better than an existing algorithm in the literature.

- Propose a GA for the MELSP under the EBP and PoT policy, which performs better than the GA under the CC policy. It is also shown that the EBP and PoT policy improves the solution quality significantly compared with the CC policy when the number of machines is small.

1.4 Organization of Dissertation

The dissertation consists of six chapters. A brief description of the next chapters is listed below:

- Chapter 2 reviews some of the related works done on the ELSP and the MELSP.

- Chapter 3 discusses the EBP and PoT policy in detail. It formulates the problem with a mathematical program and introduces a parametric search algorithm to solve the problem optimally. Based on insights drawn from the search algorithm, a heuristic is presented to solve the problem in shorter time. Computational results are also reported for the optimal and heuristic algorithms.

- Chapter 4 tests the application of GA to the ELSP where the EBP and PoT policy is used. In designing the GA, an encoding scheme that is lean, efficient and natural to the problem structure is applied. The heuristic is shown to be able to find the optimal solution under the EBP and PoT policy for most tested problems.

- Chapter 5 focuses on the MELSP. First, a GA is developed to solve the problem under the CC policy, which is shown to be better than a previous heuristic. Next, a GA under the EBP and PoT policy is proposed and shown to be better in performance.

- Chapter 6 concludes this dissertation and gives some advice on future research.

Chapter 2

Literature Review

The ELSP is a classic scheduling problem to determine the lot sizes and production sequence for several products on a single machine. The tradeoff between holding inventory for the product and frequent production setups on a single machine makes it necessary to determine good lot sizes for all the products. The MELSP is an extension of the ELSP, which schedules on multiple machines.

2.1 ELSP

Most papers on this problem make two assumptions. Firstly, the lot sizes of each product are assumed to be equal, which is called the Equal Lot Size (ELS) assumption. Secondly, the production of each product starts and only starts when its inventory is zero, which is called the Zero Inventory Production (ZIP) assumption. These two assumptions are also used in this dissertation. Since the feasibility of a schedule is of prime concern, policies have been taken to guarantee feasibility from the outset, by imposing some constraints on the

cycle times. The following notations are used in this chapter:

I = Number of products;

A_i = Setup cost for product i;

s_i = Setup time for product i;

r_i = Demand rate for product i;

p_i = Production rate for product i;

ρ_i = r_i/p_i, utilization for product i;

h_i = Inventory holding cost for product i;

n_i = Multiplier for product i, $n_i \in \{1, 2, 4, \ \ldots\}$;

k_i = Production frequency for product i in a cycle;

K = Number of basic periods in a cycle;

j_i = Production position for product i;

W = Basic period;

The problem of ELSP is to determine the cycle time for product i, denoted by T_i, with the objective of minimizing the total average setup cost and inventory holding cost while satisfying the demands. The average cost per unit time when product i is produced in a cycle of length T_i is given by:

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

where A_i/T_i is the average setup cost over a cycle and $h_i r_i(1 - \rho_i)T_i/2$ is the average inventory holding cost over a cycle. We let $H_i = h_i r_i(1 - \rho_i)/2$ and rewrite the expression of C_i as follows:

$$C_i = A_i/T_i + H_i T_i.$$

Thus, ELSP needs to find T_i's to minimize the total average costs:

$$\min C = \sum_{i=1}^{I} C_i.$$

It is not difficult to see that the minimum of C can be derived from the following equations:

$$\partial C/\partial T_i = -A_i/T_i^{*2} + H_i = 0,\ i = 1,\ 2,\ \ldots,\ I.$$

The min-cost cycle is given by:

$$T_i^* = \sqrt{A_i/H_i},\ i = 1,\ 2,\ \ldots,\ I,$$

with the corresponding minimum cost

$$C_i^* = 2\sqrt{A_i H_i},\ i = 1,\ 2,\ \ldots,\ I.$$

T_i^* is called the Independent Solution (IS), which is optimal when the restriction on feasibility is ignored. However, following the IS, two products may be required to produce at the same time on the machine, which is not possible physically. The ELSP is proved to be NP-hard (Hsu, 1983; Gallego and Dong, 1997) and so far no one has characterized an optimal policy for solving the

general ELSP. The most common approach is to make assumptions on the cycle times and solve the restricted version of this problem.

One of the most popular policies is the CC policy (Hanssmann, 1962), which assumes that all products have a common cycle time. That is, $T_1 = T_2 = \cdots = T_I = T$. With this assumption, the schedule is feasible as long as the following condition is met:

$$\sum_{i=1}^{I} (s_i + \rho_i T_i) \leq T. \tag{2.1}$$

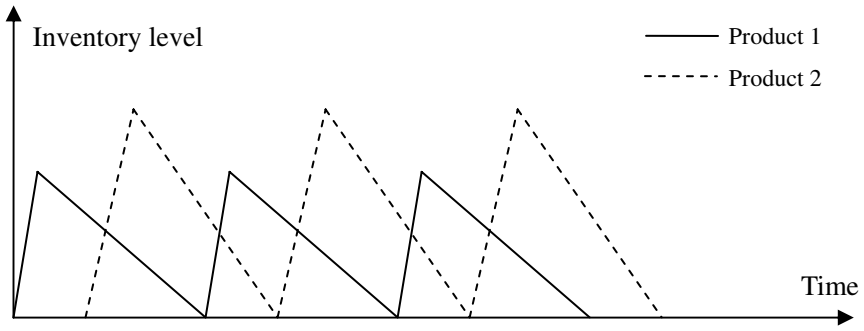

Figure 2.1 A 2-product example under the CC policy

Figure 2.1 shows a 2-product example. Normally, the ELSP assumes that the production begins and only begins when a product has zero inventory, so the amount of product i produced in one cycle is $r_i T_i$ and the production time for product i is $r_i T_i / p_i$ (or $\rho_i T_i$). In each cycle, it is required to produce each product once, so the feasibility constraint is that the sum of setup time and production time for all the products should be less than or equal to T, which is inequality (2.1).

The problem under the CC policy can be formulated as follows:

$$\min C = \sum_{i=1}^{I} \left(\frac{A_i}{T} + H_i T \right)$$

subject to

$$\sum_{i=1}^{I} (s_i + \rho_i T) \leq T. \tag{2.2}$$

Ignoring constraint (2.2), the min-cost T is found by letting:

$$\partial C / \partial T = -\frac{\sum_{i=1}^{I} A_i}{T^2} + \sum_{i=1}^{I} H_i = 0,$$

which is

$$T^* = \sqrt{\frac{\sum_{i=1}^{I} A_i}{\sum_{i=1}^{I} H_i}}.$$

Observing inequality (2.2), the optimal solution under the CC policy is

$$T^* = \max \left\{ \sqrt{\frac{\sum_{i=1}^{I} A_i}{\sum_{i=1}^{I} H_i}}, \frac{\sum_{i=1}^{I} s_i}{1 - \sum_{i=1}^{I} \rho_i} \right\}.$$

It is assumed that $\sum_{i=1}^{I} \rho_i < 1$ for the ELSP. Although it is easy to solve the ELSP under the CC policy, in general this policy will not give the optimal solution to the original problem. Actually, as pointed out by Maxwell (1964), this policy can only be defended on the basis of convenience in analysis and implementation. Jones and Inman (1989) showed that the solution under the CC policy can be near optimal under certain conditions. However, in practice to get good quality solutions, more flexible policies are required to be used.

It is interesting to note that the CC policy has been used frequently for solving the extension problems of the ELSP. Galvin (1987) applied the CC policy in the second part of a model dealing with a sequence-dependent setup problem. Gupta (1992) presented an analysis on the ELSP under the CC policy when backlogging is allowed. Hahm and Yano (1995) introduced the economic lot delivery scheduling problem under the CC policy of which an optimal polynomial time algorithm was given by Jensen and Khouja (2004), and this was further improved by Clausen and Ju (2006) with an efficient hybrid algorithm. On the other hand, Khoury et al. (2001) considered the 2-product ELSP with insufficient capacity under the CC policy and presented a mathematical model with a solution procedure. Lately, Lin et al. (2006) determined a near optimal cycle time for the ELSP with deteriorating products under the CC policy.

A more flexible policy, the BP policy, was proposed by Bomberger (1966), which allows different cycle times for different products. It is assumed that $T_i = n_i W$, where W is a basic period of all products and n_i's are positive integers. We denote n_i's by $\boldsymbol{n} = (n_1, \ \ldots, \ n_I)$ in this dissertation. To guarantee the feasibility of a schedule, it is required that W is long enough to accommodate the production of all products. Bomberger (1966) solved the problem by a dynamic programming approach for a given W. It is not easy to find the optimal W and $\boldsymbol{n}$ simultaneously. Under the BP policy, the problem can be formulated as follows:

$$\min \sum_{i=1}^{I} \left(\frac{A_i}{n_i W} + H_i n_i W \right)$$

subject to

$$\sum_{i=1}^{I}(s_i + \rho_i n_i W) \leq W, \tag{2.3}$$
$$W > 0,\ n_i \geq 1 \text{ and integer.}$$

After the BP policy was introduced, the use of W and $\boldsymbol{n}$ becomes very popular in representing the solution of the problem. However, the feasibility condition (2.3) is quite stringent. Maxwell and Singh (1983) discussed its implication and restriction imposed on the cycle times. To circumvent this restriction, most researchers relax the condition and propose heuristic methods to solve the problem. These heuristic methods are discussed in the next paragraph.

A heuristic proposed by Madigan (1968) is to use the optimal cycle time T_i^* and the optimal cost $C_i^* = A_i/T_i^* + H_i T_i^*$ of the IS solution as the reference point. Given the cost C_i under the CC policy for product i, its multiplier is revised if the difference $C_i - C_i^*$ is found to be rather significant. Also, each time a multiplier is changed, a check on feasibility is made. However, no guide on changing the multipliers is given in Madigan (1968). Another heuristic is proposed by Stankard and Gupta (1969) who divided the set of products in $K+1$ groups $G,\ G^1,\ \ldots,\ G^K$ in which G has cycle time W and any other G^i has cycle time $K \times W$ in a round robin pairing with G, as shown in Figure 2.2. In other words, the heuristic restricts the multiplier of each product to be either 1 or K. Doll and Whybark (1973) suggested an iterative procedure for the simultaneous determination of the individual multipliers $\boldsymbol{n}$ and W, which eliminates the restriction on the multipliers. But the procedure does not guarantee to find a feasible solution. Goyal (1973) also presented an iter-

ative procedure for the simultaneous determination of $\boldsymbol{n}$ and W. However, it was found later by Schweitzer and Silver (1983) that the mathematical model is ill-posed.

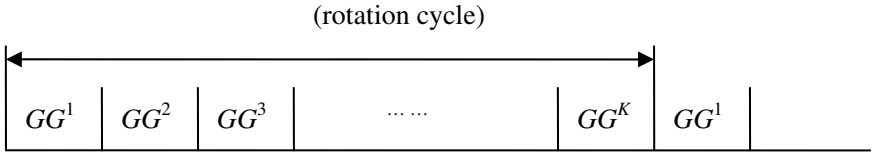

Figure 2.2 Schedule of production in S & G's procedure

The determination of $\boldsymbol{n}$ and W simultaneously under the BP policy is finally solved by Grznar and Riggle (1997). Subsequently Khouja et al. (1998) proposed an effective GA, which is found to be able to solve the high utilization problems very well. As for the extension problem, Soman et al. (2004) applied the BP policy to the ELSP with shelf life considerations.

Elmaghraby (1978) made an excellent review of all the early methods for the ELSP and presented a dynamic programming method for the problem under a less restrictive policy than the BP policy. This policy considers two consecutive basic periods at the same time. The products with even multipliers are divided into two sets of products B_1 and B_2, which are produced in the two basic periods respectively. Under this policy, the feasibility conditions can be expressed as two inequalities (2.4) and (2.5). It is not difficult to see that this policy is more flexible than the BP policy. Fujita (1978) then applied the policy to solve the problem using the marginal analysis method. This policy is known as the EBP policy. It reserves the capacity for the product with an odd

multiplier in each basic period, but only reserves the capacity for the product with an even multiplier in one of each two basic periods. Lopez and Kingsman (1991) summarized the theory and practice for ELSP.

$$\sum_{n_i \text{ odd}} (s_i + \rho_i n_i W) + \sum_{n_i \text{ even, } i \in B_1} (s_i + \rho_i n_i W) \leq W \tag{2.4}$$

$$\sum_{n_i \text{ odd}} (s_i + \rho_i n_i W) + \sum_{n_i \text{ even, } i \in B_2} (s_i + \rho_i n_i W) \leq W \tag{2.5}$$

Haessler (1979) extended the EBP policy further and presented the following necessary conditions for the solution to be feasible. Currently, the EBP policy refers to this more general policy extended by Haessler (1979), but not the original version of the EBP policy presented by Elmaghraby (1978). The EBP policy is a policy that assumes the cycle time is a multiple of the basic period without the capacity constraints used by Bomberger (1966). The difficulty of solving the ELSP under the EBP policy is how to ensure the feasibility of a production schedule.

$$\sum_{j=1}^{K} x_{ij} = k_i \text{ for all } i, \tag{2.6}$$

$$x_{ij} = x_{ij+n_i} \text{ for all } i \text{ provided } n_i \neq K, \text{ for } j = 1 \ \ldots \ K - n_i, \tag{2.7}$$

$$\sum_{i} x_{ij}(s_i + n_i \rho_i W) \leq W \text{ for all } j. \tag{2.8}$$

In the above expressions, x_{ij} is 1 if product i is produced in the jth basic period and 0 otherwise. K stands for the least common multiple of all n_i's and $k_i = K/n_i$, which is the number of times product i is produced in a complete cycle of K basic periods. Conditions (2.6), (2.7) and (2.8) are necessary conditions for a feasible solution if $n_i = 1$ for at least one product. Haessler (1979) also assumed that all the multipliers are powers of two. The error of rounding off the multipliers to powers of two is small. Roundy (1989) showed that the cost increase cannot exceed 6% by rounding off the intervals to powers of two for ELSP with necessary capacity constraints.

The EBP policy does not require the basic period to be long enough to accommodate the production of all products. Instead, it pools the machine time of consecutive periods to stagger out the production. It is very difficult to solve the problem optimally under the EBP policy. Several heuristics have been developed under this policy (Park and Yun, 1984; Boctor, 1987; Geng and Vickson, 1988; Larraneta and Onieva, 1988). Recently, a GA is presented by Chatfield (2007) and it is shown that GA performs particularly well for high utilization problems. Also, Yao and Huang (2005) solved the problem with deteriorating items using GA.

All the policies mentioned so far assume ELS. On the other hand, Dobson (1987) presented a formulation allowing the lot sizes for a given product to vary over the cycle. Together with the procedure designed by Zipkin (1991), the two algorithms comprise a simple, plausible heuristic for the ELSP as a whole. Dobson's approach is similar to that of Roger (1958), Maxwell (1964) and Delporte and Thomas (1977). This problem is also tackled by several

meta-heuristics (Moon et al., 2002; Raza and Akgunduz, 2005; Raza et al., 2006). The time-varying lot sizes policy is less restrictive than the policies assuming ELS. However, it often creates solutions with very long cycles in order to make room for setups (Chatfield, 2007). In this dissertation, we focus on discussing policies with ELS assumption.

So far, no one has characterized an optimal strategy for solving the general ELSP without imposing any requirements. However, there are papers discussing how to solve the general ELSP with two products. Vemuganti (1978) had shown that, given the number of setups for the two products over some time interval, a feasible schedule exists if a certain mixed integer linear program has a feasible solution. Boctor (1982) presented necessary and sufficient conditions for feasibility for the general ELSP when there are only two products and showed that the cycle lengths have to be integer multiples of some basic cycle time when there are more than two products. Lee and Danusaputro (1989) proposed an algorithm for the two-product problem.

Vast literatures have been devoted to the ELSP with deterministic demands. There are also a number of works done on the stochastic ELSP (SELSP), which considers the production of multiple products on a single machine under random demands (Karmarkar, 1987; Leachman and Gascon, 1988; Leachman et al., 1991; Bourland and Yano, 1994; Markowitz et al., 1995; Federgruen and Katalan, 1996; Pena and Zipkin, 1997; Federgruen, 1998; Wagner and Smits, 2004).

Another important extension of the ELSP is the lot scheduling problem with sequence-dependent setups, where the explicit costs associated with the setup and the lost productive time of the setup depend on the production sequence. Maxwell (1964) was the first to formulate and discuss the problem, followed by Geoffrion and Graves (1976), Singh and Foster (1987) and Driscoll and Emmons (1977). Subsequently, Dobson (1992) formulated the problem and provided a heuristic solution procedure. Recently, a search heuristic is presented by Wagner and Davis (2002).

2.2 MELSP

This problem can be described as a problem to determine the product assignment, economic lot sizes and production sequences on multiple machines. Similar to the single-machine problem, the objective of the MELSP is to minimize the total average setup and inventory holding costs for all the machines.

Maybe due to the problem complexity, there is very little literature on the MELSP, although it is an important problem as most real problems have more than one machine. Maxwell and Singh (1986) were the first to address the MELSP by proposing some conditions for developing effective heuristic methods. Carreno (1990) introduced a local search heuristic under the CC policy with the assumption that the production of a product cannot be split among the machines. Bollapragada and Rao (1999) investigated the nonidentical multiple-machine problem under the CC policy where the production of a product is allowed to be split among the machines.

Chapter 3

ELSP

3.1 The EBP and PoT Policy

The CC policy and the BP policy were introduced before the EBP policy in the literature. It is obvious that the CC policy is too restrictive as it assumes that all the products have the same cycle time. The BP policy allows the products to have different cycle times, but it cannot provide good solutions if the capacity is tight. A worst case analysis is performed for the CC policy and the BP policy in Appendix E.1, which shows that the solutions under these two policies can be arbitrarily bad compared to the solution under the EBP and PoT policy. However, to our knowledge, so far no one has presented an optimal solution for the problem under the EBP and PoT policy. A search algorithm that finds the optimal solution and a heuristic search algorithm for ELSP under the EBP and PoT policy are presented in this section. The policy is defined as:

- EBP: $T_i = n_i W$, n_i and W must satisfy

$$\sum_{i \in S_k} (s_i + \rho_i n_i W) \leq W$$

where S_k represents the set of products produced in the basic period k.

- PoT: $n_i = 2^{t_i}$, for some non-negative integer t_i.

3.1.1 The Formulation

The problem under the EBP and PoT policy can be formulated as follows:

PoT$(\cdot)$:

$$\min \quad \sum_{i=1}^{I} \sum_{t_i=\mu_i}^{\nu_i} \sum_{j_i=1}^{2^{t_i}} \left(\frac{A_i}{2^{t_i} W} + H_i 2^{t_i} W \right) x_{i,t_i,j_i}$$

subject to

$$\sum_{i=1}^{I} \sum_{t_i=\mu_i}^{\nu_i} \left(s_i + \rho_i 2^{t_i} W \right) x_{i,t_i,\alpha(t_i,k)} \leq W, \ k = 1, 2, \ldots, K \tag{3.1}$$

$$\sum_{t_i=\mu_i}^{\nu_i} \sum_{j_i=1}^{2^{t_i}} x_{i,t_i,j_i} = 1, \ i = 1, 2, \ldots, I$$

$$x_{i,t_i,j_i} \in \{0, 1\}, \ W > 0$$

where

$$\alpha(t_i, k) = \left((k-1) \mod 2^{t_i} \right) + 1$$

Here, the binary variables in x's capture the production sequence where $x_{i,t_i,j_i} = 1$ only when $n_i = 2^{t_i}$ and the production position of product i is j_i. Let

$\boldsymbol{j} = (j_1, \ldots, j_I)$ be the production positions. In other words, the cycle time of product i is $2^{t_i}W$ and it is produced at the j_ith period of every 2^{t_i} basic periods. W stands for the duration of the basic period. $\alpha(t_i, k)$ is used to ensure that period k captures the production time of product i. K constraints are imposed in (3.1) to ensure that the production time in every basic period is within the capacity of W. If W is given for **PoT**$(\cdot)$, the model is denoted by $PoT(W)$. Note that $PoT(W)$ is a pure integer linear programming problem.

μ_i and ν_i are the lower bound and upper bound of t_i. Given $L_i \leq n_i \leq U_i$ (See Appendix A), μ_i and ν_i are calculated as follows.

$$\mu_i = \lceil \log_2 L_i \rceil$$

$$\nu_i = \lfloor \log_2 U_i \rfloor$$

where $\lceil z \rceil$ is the smallest integer that is bigger than or equal to z and $\lfloor z \rfloor$ is the biggest integer that is smaller than or equal to z.

K is the number of basic periods in one common cycle. Under the PoT policy, the number of basic periods in a common cycle is equal to the maximum of multipliers. For example, if there are three products and the multipliers of those products are 1, 2 and 4, then the number of basic periods in a common cycle is 4. Therefore, $K = 2^{\max_i\{t_i\}}$. In the model, $K = 2^{\max_i\{\nu_i\}}$, which is the maximum of all the possible multipliers.

To ensure the feasibility of a production schedule, a capacity constraint is

imposed in each basic period in a common cycle. S_k, the set of products produced in the kth basic period, can be expressed by the binary decision variables x_{i,t_i,j_i}'s. Given a solution of x_{i,t_i,j_i}'s, the multiplier $\boldsymbol{n}$ and the production position $\boldsymbol{j}$ can be determined. Once the multipliers and production positions are known, the production schedule can be constructed. For example, assume that the 3-product problem has the multipliers 1, 2 and 4. If the production positions are 1, 2 and 3, then a possible production schedule is shown in Figure 3.1.

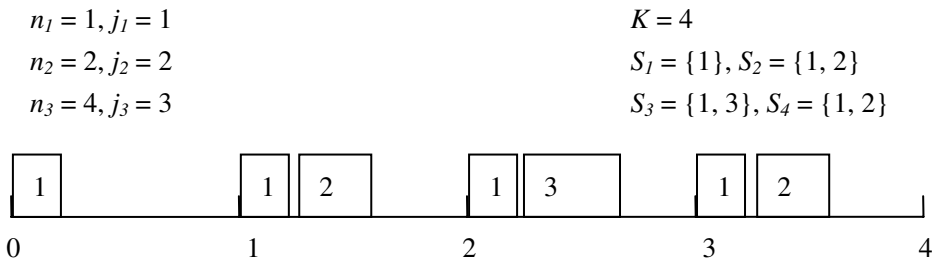

Figure 3.1 An explanation of K, j_i and S_k

The feasibility constraints under the EBP policy are $\sum_{i \in S_k}(s_i + \rho_i 2^{t_i} W) \leq W, \forall\ k$. The set of products S_k is modeled by the decision variables $x_{i,t_i,\alpha(t_i,k)}$, where $\alpha(t_i, k) = ((k-1) \mod 2^{t_i}) + 1$. $\alpha(t_i, k)$ is the production position for product i if its multiplier is 2^{t_i}. For example, in Figure 3.1 in the first basic period, $\alpha(t_i, 1) = 1, \forall\ t_i$. So the constraint in period 1 is $\sum_{i=1}^{I} \sum_{t_i=\mu_i}^{\nu_i} (s_i + \rho_i 2^{t_i} W) x_{i,t_i,1} \leq W$. It is not difficult to see that a product is produced in the first basic period only if the production position is 1. The constraint ensures that there is enough capacity for all the products produced in the first basic period. The constraints to ensure feasibility for the schedule in Figure 3.1 are:

$$(s_1 + \rho_1 W) \leq W, k = 1,$$

$$(s_1 + \rho_1 W + s_2 + 2\rho_2 W) \leq W, k = 2,$$

$$(s_1 + \rho_1 W + s_3 + 4\rho_3 W) \leq W, k = 3,$$

$$(s_1 + \rho_1 W + s_2 + 2\rho_2 W) \leq W, k = 4.$$

3.1.2 Discontinuity of the Problem

For brevity, we denote the optimal value or the minimum cost of $PoT(W)$ by $f(W)$. For a given value of W, $f(W)$ can be found by the linear integer programming techniques. Nevertheless, it is not easy to determine the optimal value of W because $f(W)$ is not continuous in general. This can be seen from the following 2-product example.

Table 3.1 A 2-product example

Product	$A(\$)$	$h(\$/unit/day)$	$r(/day)$	$p(/day)$	$s(day)$	$H(\$/day^2)$
1	18	1/30	400	4000	0.4	6
2	100	2/105	1500	5000	0.2	10

Table 3.1 gives the data of a 2-product case. The problem can be easily solved. The graph of the optimal cost function $f(W)$ is plotted in solid curves in Figure 3.2, where a discontinuity point occurs at $W = 2$. In the example, the optimal multiplier vector $\boldsymbol{n}$ and function value are

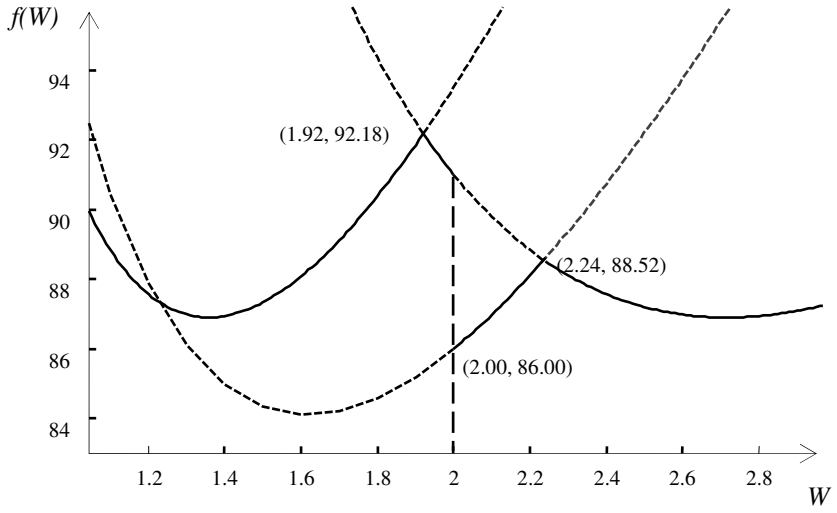

Figure 3.2 Graph of function $f(W)$

$$\boldsymbol{n} = \begin{cases} \text{infeasible}, & \text{when } W < 0.5; \\ (2,2), & \text{when } 0.50 \leq W < 1.92; \\ (1,1), & \text{when } 1.92 \leq W < 2.00; \\ (1,2), & \text{when } 2.00 \leq W < 2.24; \\ (1,1), & \text{when } W \geq 2.24. \end{cases}$$

$$f(W) = \begin{cases} (A_1 + A_2)/2W + 2(H_1 + H_2)W, & \text{when } 0.50 \leq W < 1.92; \\ (A_1 + A_2)/W + (H_1 + H_2)W, & \text{when } 1.92 \leq W < 2.00; \\ (A_1 + A_2/2)/W + (H_1 + 2H_2)W, & \text{when } 2.00 \leq W < 2.24; \\ (A_1 + A_2)/W + (H_1 + H_2)W, & \text{when } W \geq 2.24. \end{cases}$$

In general, it is possible to find the minimum W^* of $f(W)$ using a brute-

force method such as a small-step search, which finds $f(W)$ by decreasing W by a small value in each step. It has been observed that if a multiplier vector $\boldsymbol{n}$ is feasible for a particular W', it is also feasible for all $W > W'$ (Elmaghraby, 1978). This implies that if $PoT(W')$ is infeasible, $PoT(W)$ is infeasible for all $W < W'$. Hence, we can make the search from high to low values of W and stop the algorithm once $PoT(W)$ becomes infeasible.

3.1.3 A Lower Bound

Clearly, applying the small-step search over all possible values of W is inefficient. Moreover, the accuracy of the solution depends on the step size. To cut down the search range, a common technique is to employ the bounds. A natural lower bound of $f(W)$ is provided by removing constraints (3.1) from $PoT(W)$. This uncapacitated ELSP under the PoT policy was solved by Yao and Elmaghraby (2001). Their model is equivalent to ours by removing constraints (3.1) and setting $y_{i,t_i} = \sum_{j_i=1}^{2^{t_i}} x_{i,t_i,j_i}$ as shown by the objective function (3.2) and constraints (3.3). Let $g(W)$ be the optimal value of this uncapacitated model for a given W. It is clear that $f(W)$ is bounded below by $g(W)$ for all W.

$$\min \quad \sum_{i=1}^{I} \sum_{t_i=\mu_i}^{\nu_i} \left(\frac{A_i}{2^{t_i} W} + H_i 2^{t_i} W \right) y_{i,t_i} \tag{3.2}$$

subject to

$$\sum_{t_i=\mu_i}^{\nu_i} y_{i,t_i} = 1, \ \ i = 1, 2, \ldots, I \tag{3.3}$$

$$y_{i,t_i} \in \{0,1\},\ W > 0$$

We briefly introduce their methods as follows. First, the lower bound and the upper bound of W are calculated. Let W_{LB} be the lower bound and W_{UB} be the upper bound.

$$W_{LB} = \max_{i=1,\ldots,m} \{s_i(1+\rho_i)\}.$$

$$W_{UB} = T^* = \max\left\{\sqrt{\frac{\sum_{i=1}^{I} A_i}{\sum_{i=1}^{I} H_i}}, \frac{\sum_{i=1}^{I} s_i}{1-\sum_{i=1}^{I}\rho_i}\right\}.$$

where T^* is the optimal cycle time under the CC policy.

The curve associated with $g(W)$ is a piece-wise convex curve, which is the lower envelop of a finite number of convex curves. The junction points are defined as the W values where $g(W)$ is achieved by two convex curves. Five junction points are shown in Figure 3.3. In general, we assume that the junction points in $[W_{LB}, W_{UB}]$ are $w_1, w_2, \ldots, w_{n-1}$ for some positive integer n, where $w_j < w_{j+1},\ j = 1, \ldots, n-2$. $[W_{LB}, W_{UB}]$ is divided into n intervals $I_p = [\underline{w}_p, \overline{w}_p],\ p = 1, \ldots, n$, where $\underline{w}_1 = W_{LB}$, $\overline{w}_n = W_{UB}$ and $\overline{w}_p = \underline{w}_{p+1} = w_p,\ p = 1, \ldots, n-1$. By the convexity of $g(W)$ function in I_p, the minimum in I_p can be located easily. Let $\hat{w}_p$ be the value of W corresponding to the minimum point in I_p. With these minimums, the global minimum of $g(W)$ can be determined. Yao and Elmaghraby (2001) presented

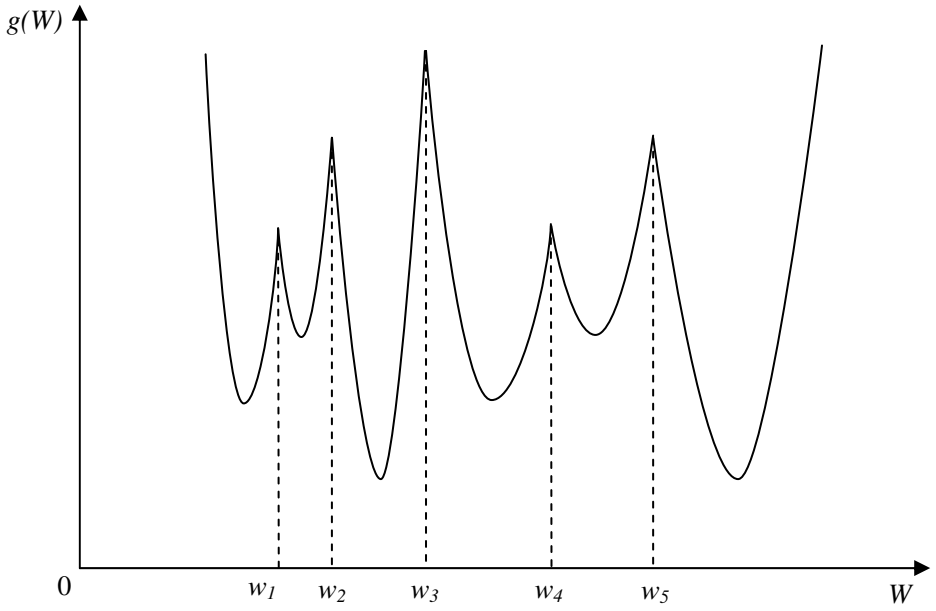

Figure 3.3 Junction points on a curve

a search algorithm to find all the junction points of $g(W)$.

3.1.4 The Parametric Search Algorithm

In this section, we will present a divide-and-conquer procedure to search for the minimum of $f(W)$ in $[W_{LB}, W_{UB}]$. We use a better lower bound than Yao and Elmaghraby (2001) for W as follows:

$$W_{LB} = \max_{i=1,\ ...,\ m} \left\{ \frac{s_i}{1-\rho_i} \right\} \geq \max_{i=1,\ ...,\ m} \left\{ s_i(1+\rho_i) \right\}.$$

To make the search more efficient, we make use of the lower bound for $f(W)$ provided by $g(W)$ and the upper bound provided by feasible solutions whenever they are found. We divide the search range $[W_{LB}, W_{UB}]$ into intervals

bounded by the junction points. The upper bound information is used to trim away points whose lower bounds are worse than the upper bound value. The lower bound information is used to identify good intervals to perform the small step search intensively. In the algorithm we always engage the interval with the smallest lower bound first. Within the engaged interval, a feasible solution will be sought on the point with the smallest lower bound. This step may lead to infeasibility, which means all values of W to the left of this point can be trimmed. On the other hand, when a feasible solution can be found, its objective value can be used to update the upper bound to trim away values of W whose lower bounds are worse than this value. After trimming, the small-step-search algorithm is used to find the best solution within the interval. These steps are repeated until all worthy intervals are explored. Upon completion, the solution associated with the latest upper bound is then the optimal solution to our problem. This divide-and-conquer procedure is presented in the next subsection.

The divide-and-conquer procedure

Let $C(W, \boldsymbol{n})$ be the cost function over W and $\boldsymbol{n}$. That is,

$$C(W, \boldsymbol{n}) = \sum_{i=1}^{I} \left(\frac{A_i}{n_i W} + H_i n_i W \right)$$

In the procedure, we let $\underline{C_p} = g(\hat{w}_p)$ be the minimum value of $g(W)$ in I_p and $\overline{C}$ be the global upper bound of $f(W)$. As $g(W)$ is convex over an interval which is bounded by the junction points, it is easy to trim away from it the values of W whose $g(W)$ are worse than $\overline{C}$. To elaborate, let the interval be

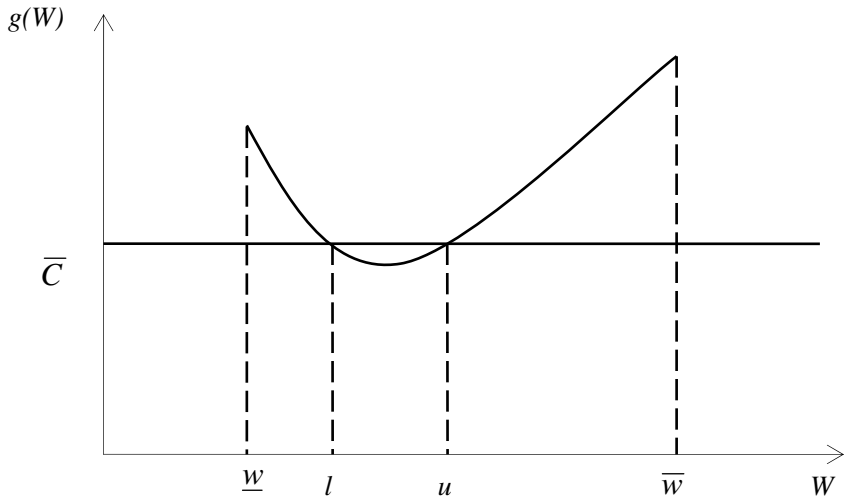

Figure 3.4 Trimmed interval

$[\underline{w}, \overline{w}]$ and $\boldsymbol{n}$ be the optimal multiplier vector for $g(W)$ in $[\underline{w}, \overline{w}]$. Let l and u $(l \leq u)$ be the solutions of W that solve

$$C(W, \boldsymbol{n}) = \overline{C}.$$

Then the trimmed interval is given by $[\max\{\underline{w}, l\}, \min\{\overline{w}, u\}]$. Figure 3.4 provides such an example graphically. The divide-and-conquer procedure is described next, where $\boldsymbol{n}^p$ denotes the optimal multiplier vector of $g(W)$ in I_p and each I_p is corresponding to one optimal multiplier vector only.

The divide-and-conquer procedure:

1. Determine W_{LB}, W_{UB}, and all junction points on the curve associated with $g(W)$. Suppose $n-1$ is the number of junction points in $[W_{LB},\ W_{UB}]$. Let $S = \{1,\ \ldots,\ n\}$. For $p \in S$, construct intervals I_p, determine $\boldsymbol{n}^p$, $\hat{w}_p$ and $\underline{C_p}$ on $g(W)$. Let $k = \arg\min_{p \in S}\ \underline{C_p}$ and set

$\overline{C} = \infty$.

2. If $PoT(\hat{w}_k)$ is feasible, let $w = \hat{w}_k$ and go to Step 3. If $PoT(\hat{w}_k)$ is infeasible and $PoT(\overline{w}_k)$ is feasible, let $w = \overline{w}_k$ and go to Step 3. Otherwise let $S = S - \{p : p \leq k\}$ and go to step 6.

3. Trim the interval I_k using $\overline{C} = \min\{f(w), \overline{C}\}$.

4. Determine the minimum of $f(W)$ in I_k and let w_k^* be its minimum. Let $S = S - \{k\}$. If $PoT(\underline{w}_k)$ is infeasible, let $S = S - \{p : p < k\}$.

5. If $f(w_k^*) < \overline{C}$, let $\overline{C} = f(w_k^*), W^* = w_k^*$, and $S = S - \{p : \underline{C_p} \geq \overline{C}\}$.

6. If S is empty, stop; otherwise, let $k = \arg\min_{p \in S} \underline{C_p}$. Go to step 2.

In step 4, we use the small-step search to determine the minimum of $f(W)$ in the interval I_k, and we call the whole procedure as the SS algorithm. After the procedure terminates, the optimal solution is W^* and its corresponding optimal cost is $\overline{C}$. Under this divide-and-conquer procedure, we are able to apply the time-consuming small-step search only to those intervals that may contain an optimal solution. Based on our experiments, this procedure greatly reduces the computational time compared with not using the divide-and-conquer. However, the efficiency of the search algorithm depends heavily on the computational time for the small-step search in each interval. Apparently, the small-step search is not efficient. In the next section, we present a much faster heuristic algorithm to be used in step 4, and we call the whole algorithm as the Efficient Heuristic (EH) algorithm.

The EH algorithm

The difference of the EH algorithm from the SS algorithm is that we use a different search algorithm to find the minimum of $f(W)$ in the interval I_k. Consider $f(W)$ in a small interval $[\underline{w}, \overline{w}]$, in which $g(W)$ is convex.

If $\boldsymbol{n}$ is known to be the optimal multiplier vector for all $W \in [\underline{w}, \overline{w}]$, then $f(W) = C(W, \boldsymbol{n})$, and $f(W)$ is also convex over $[\underline{w}, \overline{w}]$. Therefore, the minimum of $f(W)$ over $[\underline{w}, \overline{w}]$ is given by

$$w' = \min\{\max\{\underline{w}, \hat{w}\}, \overline{w}\} \tag{3.4}$$

where

$$\hat{w} = \sqrt{\frac{\sum_{i=1}^{I} A_i/n_i}{\sum_{i=1}^{I} H_i n_i}}$$

$\hat{w}$ is determined by solving

$$\frac{dC(W, \boldsymbol{n})}{dW} = -\frac{\sum_{i=1}^{I} A_i/n_i}{W^2} + \sum_{i=1}^{I} H_i n_i = 0$$

Equation (3.4) captures the three cases for the calculation of the minimum point $w^{'}$.

- If $\underline{w} \leq \hat{w} \leq \overline{w}$, $w^{'} = \hat{w}$.
- If $\hat{w} < \underline{w}$, $w^{'} = \underline{w}$.

- If $\overline{w} < \hat{w}$, $w' = \overline{w}$.

If there are more than one optimal multiplier vector in $[\underline{w}, \overline{w}]$, the minimum of $f(W)$ can still be found easily given that there is an easy way to divide $[\underline{w}, \overline{w}]$ into subintervals such that within a subinterval the optimal multiplier vector is the same. Unfortunately, there does not seem to exist an easy way to divide the interval into the desired subintervals. Its difficulty lies in the fact that a multiplier vector can be optimal at the two end points of an interval and yet not optimal over the entire interval. This can be seen from the previous 2-product example showing discontinuity, in which $\boldsymbol{n} = (1, 1)$ is optimal at $W = 1.95$ and 3.00, but not optimal at $W = 2.00$. However, it is perceived that these oddities may not be common and even if they occur, failing to uncover their corresponding values may not lead to a very bad solution. We therefore design a heuristic search algorithm based on assumption 1 which ignores these oddities. In the assumption, $\boldsymbol{n}(\underline{w})$ denotes the optimal multiplier vector of $f(W)$ at $\underline{w}$ and $\boldsymbol{n}(\overline{w})$ denotes the optimal multiplier vector of $f(W)$ at $\overline{w}$.

Assumption 1. If $\boldsymbol{n}(\underline{w})$ is optimal at $\overline{w}$ (i.e., $C(\overline{w}, \boldsymbol{n}(\underline{w})) = C(\overline{w}, \boldsymbol{n}(\overline{w}))$), then $\boldsymbol{n}(\underline{w})$ is optimal for all $W \in [\underline{w}, \overline{w}]$.

Following assumption 1, the minimum of $f(W)$ over an interval $[\underline{w}, \overline{w}]$ is identified by equation (3.4) if the end points are found to have a common optimal multiplier vector. On the other hand, if $\boldsymbol{n}(\underline{w}) \neq \boldsymbol{n}(\overline{w})$, we need to find a way to divide the interval into subintervals such that the two end points of each subinterval have a common optimal multiplier vector. To do this, we first make the following observations which are not difficult to show.

Observation 1: Let $\boldsymbol{n}^1$-curve and $\boldsymbol{n}^2$-curve be the curves corresponding to $C(W, \boldsymbol{n}^1)$ and $C(W, \boldsymbol{n}^2)$, respectively, where $\boldsymbol{n}^1 \neq \boldsymbol{n}^2$. Then the two curves intersect exactly once (Grznar and Riggle, 1997) at $w_a(\boldsymbol{n}^1, \boldsymbol{n}^2)$ where

$$w_a(\boldsymbol{n}^1, \boldsymbol{n}^2) = \sqrt{\frac{\sum_{i=1}^{I} A_i/n_i^1 - \sum_{i=1}^{I} A_i/n_i^2}{\sum_{i=1}^{I} H_i n_i^2 - \sum_{i=1}^{I} H_i n_i^1}}. \tag{3.5}$$

We can easily get $w_a(\boldsymbol{n}^1, \boldsymbol{n}^2)$ shown as follows:

$$C(W, \boldsymbol{n}^1) = \sum_{i=1}^{I} \left(\frac{A_i}{n_i^1 W} + H_i n_i^1 W \right)$$

$$C(W, \boldsymbol{n}^2) = \sum_{i=1}^{I} \left(\frac{A_i}{n_i^2 W} + H_i n_i^2 W \right)$$

From the following equation, we can find the intersection point of the two curves:

$$C(W, \boldsymbol{n}^1) = C(W, \boldsymbol{n}^2)$$

Observation 2: Let $w_b(\boldsymbol{x}(w))$ be the smallest value of W such that $\boldsymbol{x}(w)$ is feasible to $PoT(W)$, where $\boldsymbol{x}(w)$ is the optimal solution of $PoT(w)$. Then

$$w_b(\boldsymbol{x}(w)) = \max_{k:k=1,\ldots,K} \left\{ \frac{\sum_{i=1}^{I} \sum_{t_i=\mu_i}^{\nu_i} s_i x_{i,t_i,\alpha(t_i,k)}}{1 - \sum_{i=1}^{I} \sum_{t_i=\mu_i}^{\nu_i} \rho_i 2^{t_i} x_{i,t_i,\alpha(t_i,k)}} \right\} \tag{3.6}$$

This result can be deduced from the capacity constraints (3.1) easily, which requires:

$$W \geq \frac{\sum_{i=1}^{I} \sum_{t_i=\mu_i}^{\nu_i} s_i x_{i,t_i,\alpha(t_i,k)}}{1 - \sum_{i=1}^{I} \sum_{t_i=\mu_i}^{\nu_i} \rho_i 2^{t_i} x_{i,t_i,\alpha(t_i,k)}}, k = 1, \ \ldots, \ K$$

Observation 3: Let $\boldsymbol{n}^1 = \boldsymbol{n}(\underline{w}) \neq \boldsymbol{n}(\overline{w}) = \boldsymbol{n}^2$, and $f(\overline{w})$ is the optimal objective value at $\overline{w}$ achieved by the optimal solution $\boldsymbol{x}(\overline{w})$. $\boldsymbol{n}^1$ and $\boldsymbol{n}^2$ are associated with two different curves. We will have the following three cases.

1. $w_a(\boldsymbol{n}^1, \boldsymbol{n}^2) > \overline{w}$. In this case, $\underline{w} < w_b(\boldsymbol{x}(\overline{w})) \leq \overline{w}$ (Figure 3.5). If curve $\boldsymbol{n}^1$ and curve $\mathbf{n}^2$ intersect on the right of $\overline{w}$, the discontinuous point of curve $\boldsymbol{n}^2$ must be $\overline{w}$ or on the left of $\overline{w}$ and on the right of $\underline{w}$. Otherwise $\boldsymbol{n}^1$ cannot be the optimal multiplier vector at $\underline{w}$.

2. $w_a(\boldsymbol{n}^1, \boldsymbol{n}^2) < \underline{w}$. In this case, $\underline{w} < w_b(\boldsymbol{x}(\overline{w})) \leq \overline{w}$ (Figure 3.6). Similarly, if curve $\boldsymbol{n}^1$ and curve $\mathbf{n}^2$ intersect on the left of $\underline{w}$, the discontinuous point of curve $\boldsymbol{n}^2$ must be on the right of $\underline{w}$. Otherwise $\boldsymbol{n}^1$ cannot be the optimal multiplier vector at $\underline{w}$.

3. $\underline{w} \leq w_a(\boldsymbol{n}^1, \boldsymbol{n}^2) \leq \overline{w}$. This case is shown in Figure 3.7.

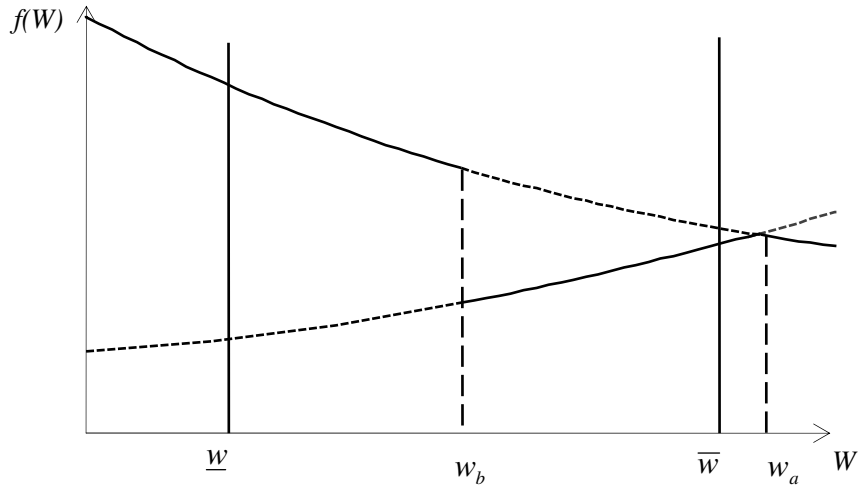

Figure 3.5 Case 1

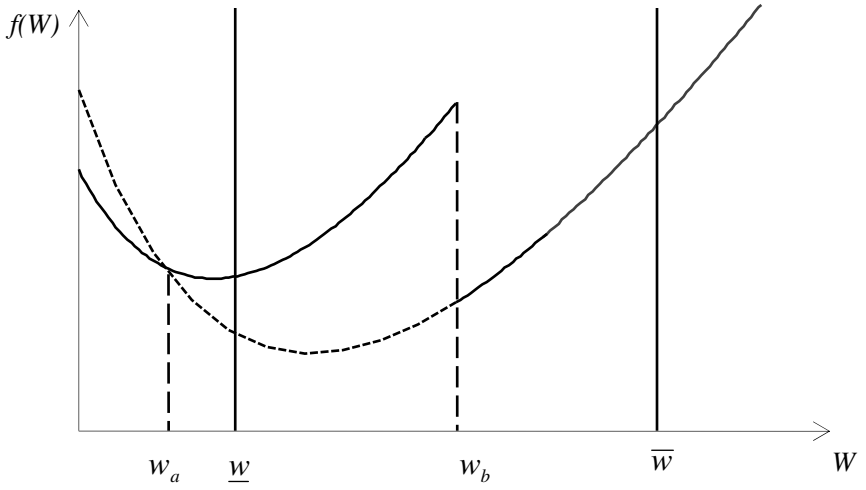

Figure 3.6 Case 2

For an interval $[\underline{w}, \overline{w}]$, when $\boldsymbol{n}^1 = \boldsymbol{n}(\underline{w}) \neq \boldsymbol{n}(\overline{w}) = \boldsymbol{n}^2$, we will divide the interval into two subintervals $[\underline{w}, w_c]$ and $[w_c, \overline{w}]$ based on the three observations. Formally,

$$w_c = \begin{cases} \max\{w_a(\boldsymbol{n}^1, \boldsymbol{n}^2), w_b(\boldsymbol{x}(\overline{w}))\}, & \text{if } \underline{w} \leq w_a(\boldsymbol{n}^1, \boldsymbol{n}^2) \leq \overline{w}; \\ w_b(\boldsymbol{x}(\overline{w})), & \text{otherwise.} \end{cases} \tag{3.7}$$

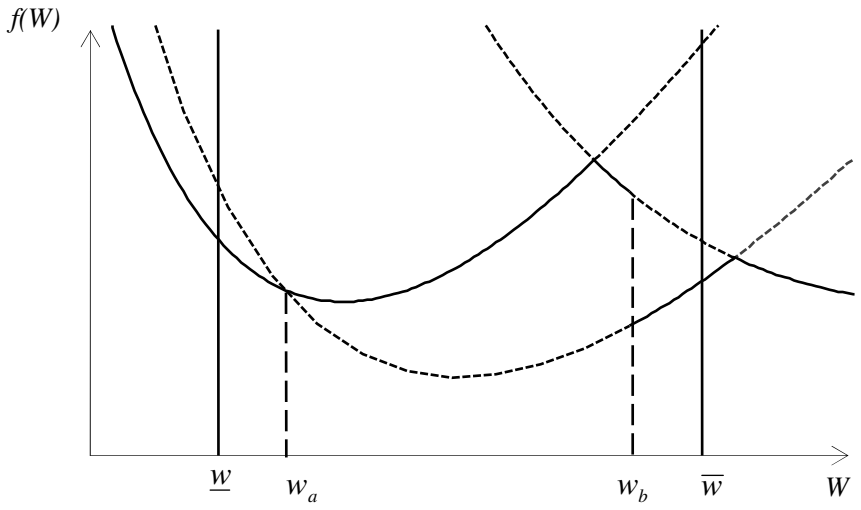

Figure 3.7 Case 3

It is clear that after the division each subinterval will be a proper subset of $[\underline{w}, \overline{w}]$ except for the special cases when $w_c = \overline{w}$ or $w_c = \underline{w}$. When $w_c = \overline{w}$, we avoid it by shortening the interval to be $[\underline{w}, \overline{w} - \epsilon]$ where ϵ is a small positive value. When $w_c = \underline{w}$, $\boldsymbol{n}(\overline{w})$ is also optimal at $\underline{w}$ and the optimal solution can be determined by equation (3.4).

Our heuristic procedure to find the minimum point of $f(W)$ over $[\underline{w}, \overline{w}]$ is presented next where **IntSearch** is the main procedure while **FeasProc** and **InfeasProc** are the two subprocedures.

IntSearch$(\underline{w}, \overline{w})$:

1. Solve $PoT(\underline{w})$ and $PoT(\overline{w})$. Let $c = f(\overline{w})$ and $w^* = \overline{w}$.

2. If $PoT(\underline{w})$ is feasible, call **FeasProc**$(\underline{w}, \overline{w})$;
   otherwise, call **InfeasProc**$(\underline{w}, \overline{w})$.

FeasProc$(\underline{w}, \overline{w})$:

1. If $C(\overline{w}, \boldsymbol{n}(\underline{w})) = C(\overline{w}, \boldsymbol{n}(\overline{w}))$, determine the minimum w' using equation (3.4) and $\boldsymbol{n} = \boldsymbol{n}(\underline{w})$; if $C(w', \boldsymbol{n}(\underline{w})) < c$, let $w^* = w'$ and $c = C(w', \boldsymbol{n}(\underline{w}))$. Exit the subprocedure.

2. Compute w_c using equations (3.5), (3.6), and (3.7).

3. Consider the following cases:

   (a) $\underline{w} < w_c < \overline{w}$: solve $PoT(w_c)$;
   call **FeasProc**$(\underline{w}, w_c)$;
   call **FeasProc**$(w_c, \overline{w})$.

   (b) $w_c = \overline{w}$: solve $PoT(\overline{w} - \epsilon)$;
   call **FeasProc**$(\underline{w}, \overline{w} - \epsilon)$.

   (c) $w_c = \underline{w}$: determine the minimum w' using equation (3.4) and $\boldsymbol{n} = \boldsymbol{n}(\overline{w})$; if $C(w', \boldsymbol{n}(\overline{w})) < c$, let $w^* = w'$ and $c = C(w', \boldsymbol{n}(\overline{w}))$.

InfeasProc$(\underline{w}, \overline{w})$:

1. Compute w_b using equation (3.6) and $\boldsymbol{x}(\overline{w})$.

2. Consider the following cases:

   (a) $w_b < \overline{w}$: solve $PoT(w_b)$;
   call **InfeasProc**$(\underline{w}, w_b)$;
   call **FeasProc**$(w_b, \overline{w})$.

(b) $w_b = \overline{w}$: solve $PoT(\overline{w} - \epsilon)$;

if $PoT(\overline{w} - \epsilon)$ is feasible, call **InfeasProc**$(\underline{w}, \overline{w} - \epsilon)$.

In the procedure, c and w^* represent the optimal cost in $[\underline{w}, \overline{w}]$ and its corresponding basic period value respectively. To start the procedure, it is assumed that a feasible solution exists at $\overline{w}$, i.e., $PoT(\overline{w})$ is feasible. **FeasProc** is used to search the interval $[\underline{w}, \overline{w}]$ in which a feasible solution exists at $\underline{w}$. **InfeasProc** is used to search the interval $[\underline{w}, \overline{w}]$ in which no feasible solution exists at $\underline{w}$.

The **FeasProc** first checks whether the optimal multiplier vector at $\underline{w}$ is also optimal at $\overline{w}$. If it is true, it determines the best solution on the curve corresponding to $\boldsymbol{n}(\underline{w})$. Otherwise, we must have $\boldsymbol{n}(\underline{w}) \neq \boldsymbol{n}(\overline{w})$. The interval is then divided by w_c and three cases are discussed. In case (i) and case (ii), $w_c = w_b(\boldsymbol{x}(\overline{w}))$. In case (iii), w_c is the maximum of $w_a(\boldsymbol{n}(\underline{w}), \boldsymbol{n}(\overline{w}))$ and $w_b(\boldsymbol{x}(\overline{w}))$. The **InfeasProc** first calculates the point $w_b(\boldsymbol{x}(\overline{w}))$. As the function is not feasible at $\underline{w}$, $w_b(\boldsymbol{x}(\overline{w}))$ must be greater than $\underline{w}$. If $w_b(\boldsymbol{x}(\overline{w})) < \overline{w}$, we divide the interval by $w_b(\boldsymbol{x}(\overline{w}))$. If $w_b(\boldsymbol{x}(\overline{w})) = \overline{w}$, we decrease $\overline{w}$ by ϵ, solve $PoT(\overline{w} - \epsilon)$ and repeat the **InfeasProc** in $[\underline{w}, \overline{w} - \epsilon]$. It is not difficult to see that both procedures will stop after a finite number of iterations.

3.2 Computational Results

To test the efficiency and quality of our SS and EH algorithms, we compare them against Haessler's heuristic (HH) algorithm (Haessler, 1979), which uses

the same policy as our algorithms. Haessler's heuristic provides a good reference point. The procedure is not enumerative in nature, does not involve ad hoc user intervention, and includes feasibility testing as part of the procedure. Recently, Chatfield (2007) presented a genetic algorithm (GLS) under the EBP policy without PoT restriction. As far as we know, the GLS is the most effective method for the ELSP, especially for high utilization problems. Therefore, a comparison is also made against the GLS on the high utilization problems reported by Chatfield (2007). Our algorithms are coded in C++ and the linear integer programming models are solved by ILOG CPLEX 10.0. All the experiments are conducted on a Pentium 4-2.6 GHz personal computer with a memory of 512 MB and run under the Windows XP operating system.

3.2.1 A Comparison under the EBP and PoT Policy

The comparison against the HH is made on five sets of randomly generated examples, where ten examples are generated in each set. The holding cost is based on an interest rate of 10% per year, and the number of working days per year is 240. The parameters of each product are generated randomly from a Uniform distribution with the ranges in Table 3.2. The load of the machine is measured with the utilization factor defined by $\rho = \sum_{i=1}^{I} \rho_i$. The ρ's of the examples are 0.7, 0.8, 0.9, 0.95 and 0.98 respectively. The demand rate of each example is scaled up to reach the five respective utilization levels. Ten products are assumed in the tested problems as the maximum of the product numbers in all the benchmark problems is ten.

We use the SS algorithm, in which the step size is set at 0.01 day to deter-

Table 3.2 Ranges of parameters for ELSP

Parameters	Dimension	Low	High
Production rate	Units per day	1000	30000
Demand rate	Units per day	10	400
Setup time	Day	0.125	1
Setup cost	Dollars	10	300
Unit cost	Dollars per unit	0.005	1

Table 3.3 Comparison for the algorithms under the EBP and PoT policy

		Average Ratio			Average Time (s)		
Set	ρ	OPT/LB	EH/OPT	HH/OPT	OPT	EH	HH
1	0.70	1.023	1.000	1.002	22.8	0.9	0.2
2	0.80	1.027	1.000	1.022	137.9	3.2	0.2
3	0.90	1.047	1.000	1.044	442.5	33.8	0.3
4	0.95	1.089	1.000	1.026	3420.6	168.5	0.2
5	0.98	1.127	1.000	1.062	22484.0	234.0	0.3

mine the optimal solution (OPT). The results are presented in Table 3.3. In the table, the second column shows the ρ in each problem set. The next three columns show the various ratios and the last three columns give the computer times in seconds. The lower bound (LB) is determined by the method described in Bomberger (1966). The average gap of the optimal solution to the lower bound is the smallest for set 1 (2.3%) and the biggest for set 5 (12.7%). From the table it is clear that the EH algorithm performs a lot better than the HH algorithm and it is much faster than the exact algorithm. In fact, the EH

algorithm finds the optimal solutions for 98% of all the randomly generated problems; the errors are negligible when it fails to find an optimal solution.

3.2.2 A Comparison with Other Policies for High Utilization Problems

When the utilization factor ρ increases, the ELSP becomes more difficult to solve. Chatfield (2007) tested the GLS on four high utilization problems and it is the best method for high utilization problems as far as we know. The GLS is under the EBP policy without PoT restriction, which is a less restrictive policy than ours. In Table 3.4, CC represents the optimal solution under the common cycle policy, BP represents the optimal solution under the basic period policy and EH is our heuristic under the EBP and PoT policy. It turns out that EH finds the optimal solutions under the EBP and PoT policy for these four problems. It is interesting to note that EH performs better than GLS even though it operates under a more restrictive policy.

Table 3.4 Computational results for high utilization problems

Problem	ρ	LB	CC	BP	EH	SS	GLS
1	0.88	7589	9880	8782	7697	7697	7697
2	0.92	7715	10086	9746	7947	7947	7947
3	0.95	8419	11950	12018	9098	9098	9140
4	0.98	15681	24458	24534	19004	19004	20500

3.3 Conclusions

Despite the fact that the ELSP has been studied extensively, only a few exact procedures are proposed for some simple policies. To the best of our knowledge, no one has presented an exact procedure for the ELSP under the EBP and PoT policy. In this chapter, we present an exact parametric search algorithm to solve the model under a divide-and-conquer framework. Further, a much faster heuristic is proposed which finds the optimal solutions for almost all the randomly generated examples and when the solutions found are not optimal, the errors are negligible.

Chapter 4

Genetic Algorithm for ELSP

4.1 Introduction to Genetic Algorithm

GA was invented by Holland (1975). Originally, Holland's goal was to study the phenomenon of adaptation as it occurs in nature to develop ways to import the mechanisms of natural adaptation into computer systems. Holland presented GA as an abstraction of biological evolution and gave a theoretical framework for adaptation under GA.

GA has been used to solve combinatorial optimization problems in the last thirty years. It searches in or out of the solution region to find the true or approximate optimal solution for the problem. With an encoding scheme, a solution is represented by a chromosome containing several genes. Normally, it is not straightforward to find a good encoding scheme for a combinatorial optimization problem. After the chromosome is defined, a population of chromosomes is kept to improve the solution from one generation to another.

To start a GA, an initial population is constructed. There are two ways to construct the initial population. The first way is to randomly generate the solutions in or out of the solution region. Another way is to generate good initial solutions with some fast heuristic. These two methods can be combined to construct the initial population. With the initial population, the mechanisms of natural adaptation is followed to improve from one generation to another. The defined crossover is used to generate new chromosomes from old chromosomes.

Similar to nature, random variations of the genetic material happen in each generation, denoted by mutation. Mutation enables the creation of genes which are lost in the current population and which cannot be gained if only the existent material is combined. Each new offspring is assigned a small probability of mutation.

4.1.1 Encoding Scheme

For any search, the way in which the solutions are encoded is an important factor in the success of a GA. The most popular encodings are binary encodings, integer encodings and real-valued encodings. Binary encodings use bit strings to encode a solution. They can also be extended to gray encoding and Hillis's diploid binary encoding scheme (Holland, 1975; Goldberg, 1989). The binary encodings have advantages, for example, simple, but they are unnatural and unwieldy for many problems.

Integer encodings are mainly used for combinatorial optimization problems. It is a natural way to represent the integer solutions. Binary encodings

are not good for many combinatorial optimization problems, where an array of integers is used to represent the solution.

The third popular encoding scheme is to use real encodings. It is natural to use real numbers to form chromosomes for many applications. Holland's schema-counting argument showed that GA should perform better with binary encodings. However, the performance depends very much on the problem and the details of the GA being used. Currently there are no rigorous guidelines for predicting which encoding will work best.

4.1.2 Selection

To choose the chromosomes to create offsprings for the next generation, a selection scheme needs to be defined. A selection emphasizes the fitter chromosomes in the hope that the offspring with higher fitness is generated. There are many selection methods and no rigorous guidelines are present for which method should be used for which problem. There are more technical comparisons of different selection methods (Goldberg and Deb, 1991; Hancock, 1994). The fitness-proportional selection, rank selection, tournament selection and elitism are the most popular selection methods.

Fitness-proportional selection relates the probability of selecting a chromosome to its fitness value. Specifically, two parents are selected from the current generation with probabilities inversely proportional to their fitness values.

Rank selection can prevent too-quick convergence, which ranks the chromo-

somes in the population according to fitness. The absolute differences in fitness are obscured as the parent chromosomes are selected only based on their ranks.

Tournament selection runs a tournament among a few chromosomes chosen at random from the population and selects the one with the best fitness for crossover. If the tournament size is larger, weak individuals have a smaller chance to be selected.

Most of the time, elitism is an addition to the selection method that forces GA to retain some best solutions in the population. Many researchers have found that elitism significantly improves GA's performance.

4.1.3 Genetic Operators

Genetic operators include crossover and mutation. The crossover operator is used to vary chromosomes from one generation to the next. After parent chromosomes are selected, the child chromosomes are generated by crossover.

Crossover

Many crossover techniques exist for the chromosomes with fixed length. The one-position crossover, the two-position crossover and the uniform crossover are the three main crossover operators. The one-position crossover selects one crossover point on the parent strings. All data beyond that point in the chromosomes is swapped between the two parent chromosomes. Two-position crossover selects two points in the chromosome strings. Everything between

the two points are swapped between the two parent chromosomes, rendering two child chromosomes. The uniform crossover swaps the bits in the two parent strings with a fixed probability, typically 0.5, to generate child chromosomes.

Mutation

Mutation is used to maintain genetic diversity from one generation of a population of chromosomes to the next. The classic mutation operator randomly selects a bit in the chromosome and changes it to an arbitrary value. The mutation operators should be designed according to the encoding scheme used. With the mutation, the population may generate lost genes in previous generations or even new genes.

4.2 The Formulation

The formulation PoT(.) in the previous chapter can be written as follows.

(P)

$$\min \sum_{i=1}^{I} \left(\frac{A_i}{n_i W} + H_i n_i W \right)$$

subject to

$$\sum_{i \in S_k} (s_i + n_i \rho_i W) \leq W, \quad k = 1, \ \ldots, \ K \tag{4.1}$$

$$S_k = \{ i : j_i \equiv (k-1) \pmod{n_i} + 1 \}, \quad k = 1, \ \ldots, \ K \tag{4.2}$$

$$W > 0, \quad n_i \in \{1, 2, 4, \ \ldots, \ \overline{n}_i\}, \quad j_i \in \{1, \ \ldots, \ n_i\}, \ i = 1, \ \ldots, \ I$$

where K is the number of basic periods in a complete cycle, which is the duration that the production schedule repeats. It has been stated in the previous chapter that K is equal to $\max_i\{n_i\}$. S_k is the set of products produced in the k^{th} basic period in a complete cycle. $\overline{n}_i$ is the upper bound value of the multiplier n_i. Constraints (4.1) are sufficient and necessary conditions for the ELSP under the EBP and PoT policy. The production positions j_i's and the multipliers n_i's are used to determine the set of products S_k produced in each basic period by equation (4.2).

4.3 Genetic Algorithm for ELSP

The idea of applying GA to the ELSP is not new. As mentioned earlier, Khouja et al. (1998) proposed a GA under the BP policy. Khouja showed that the GA is efficient for solving high utilization problems. Chatfield (2007) also used GA to solve the ELSP, albeit under a more general policy, the EBP policy (with no power-of-two restrictions). Chatfield utilizes a chromosome which represents the solution as a string, consisting of a basic period, W, a set of multipliers $\boldsymbol{n}$, and a set of production positions $\boldsymbol{j}$. Chatfield showed that the GA performs very well for the benchmark problems and it can find better solutions for the high utilization problems compared with the GA proposed by Khouja et al. (1998).

We focus on the EBP and PoT policy and present a GA that performs better than Haessler's heuristic. It will be shown that the performance of

Haessler's heuristic is poor for high utilization problems. It will also be shown that our results outperform those of Khouja et al. (1998) and Chatfield (2007).

4.3.1 Integer encoding scheme

A standard chromosome is an array of bits. For the ELSP, we use two arrays of integer numbers to represent a solution, consisting of a set of power-of-two multipliers $\boldsymbol{n}$ and a set of positions $\boldsymbol{j}$ as illustrated in Figure 4.1. A gene of the chromosome represents both n_i and j_i for a product. This integer chromosome does not include W explicitly. Instead, we will analytically determine its best value for given $\boldsymbol{n}$ and $\boldsymbol{j}$. To relate the chromosome to a feasible production schedule, we limit the ranges of its integers to be $1 \leq n_i < 1/\rho_i$ and $1 \leq j_i \leq n_i$.

n_1	n_2	...	n_I
j_1	j_2	...	j_I

Figure 4.1 Chromosome for ELSP under the EBP and PoT policy

4.3.2 Feasibility

The chromosome can represent a feasible solution if the following inequalities are satisfied:

$$\sum_{i \in S_k} n_i \rho_i < 1, \quad k = 1, \ \ldots, \ K \tag{4.3}$$

Inequalities (4.3) are deduced from (4.1) in (P). Given $\sum_{i \in S_k} n_i \rho_i < 1$ for all k, the production can be made feasible no matter what the values of the setup times are as the basic period can be increased sufficiently large so that there is enough time for the setups.

If inequalities (4.3) hold, the optimal basic period of (P) is given by:

$$W^* = \max \left\{ \sqrt{\frac{\sum_{i=1}^{I} A_i / n_i}{\sum_{i=1}^{I} H_i n_i}}, \ \max_{k=1,\ldots,K} \left\{ \frac{\sum_{i \in S_k} s_i}{1 - \sum_{i \in S_k} n_i \rho_i} \right\} \right\} \tag{4.4}$$

On the other hand, if $\sum_{i \in S_k} n_i \rho_i \geq 1$ for some k, the chromosome cannot represent a feasible solution. In this case, we attempt to repair the position vector $\boldsymbol{j}$ to reduce the degree of infeasibility.

In the repair procedure, we fix the values of the multipliers and attempt to find the production positions that may result in a feasible solution. Given the length of the basic period, the problem can be viewed as a variation of a bin-packing problem. Note that when we pack (produce) a product i in period $j \leq n_i$, all bins (basic periods) $k = j, n_i + j, 2n_i + j, \ \ldots, \ K + j - n_i$ will be utilized. We use a lower bound W_R of W calculated from a relaxed version of (P) as the length of basic period for the repair procedure. In the relaxed problem (R) displayed below, only one constraint is imposed, which is the sum

of constraints (4.1) over k.

$$\sum_{k=1}^{K}\sum_{i\in S_k}(\rho_i n_i W + s_i) \le KW \Longrightarrow \sum_{i=1}^{I}\left(K\rho_i W + \frac{s_i K}{n_i}\right) \le KW$$

$$\Longrightarrow \sum_{i=1}^{I}\left(\rho_i W + \frac{s_i}{n_i}\right) \le W$$

(R)

$$\min \sum_{i=1}^{I}\left(\frac{A_i}{n_i W} + H_i n_i W\right)$$

subject to

$$\sum_{i=1}^{I}\left(\rho_i W + \frac{s_i}{n_i}\right) \le W$$

$$W > 0, \quad n_i \in \{1, 2, 4, \ \ldots, \ \overline{n}_i\}$$

Given $\boldsymbol{n}$, the optimal W of (R) is

$$W_R = \max\left\{\sqrt{\frac{\sum_{i=1}^{I} A_i/n_i}{\sum_{i=1}^{I} H_i n_i}}, \ \frac{\sum_{i=1}^{I} s_i/n_i}{1-\sum_{i=1}^{I}\rho_i}\right\}. \tag{4.5}$$

To produce or pack the products in K different periods of length W_R, we first compute the sum of setup time and production time of each product which is $\sigma_i = s_i + \rho_i n_i W_R$ for $i = 1, \ \ldots, \ I$. Then, the products are ordered increasingly by n_i. For products with the same multiplier value, they are ordered decreasingly by σ_i. Based on the list, we apply the first-fit heuristic to pack the products one by one to the first basic period that can accommodate it. We keep the multiplier vector $\boldsymbol{n}$ fixed. Hence each product is packed in K/n_i periods. In the packing, it may happen that none of the basic periods have enough time to accommodate the production of some of the products. In this case, these products are packed in the period that has the smallest $\sum_i n_i\rho_i$.

As a result the length of the periods receiving the products will exceed W_R.

The usage of the first-fit heuristic in the repair procedure, based on a lower bound of the basic period W_R, generates a good solution if it is able to pack in all products. However, when the utilization is high, it is likely that W_R is not big enough to accommodate every product. In this case, it is more critical to take care of the feasibility. Hence the list scheduling heuristic is employed which uses different $\sum_i n_i \rho_i$ in different basic periods as the criteria to pack in the products. In summary, the repair procedure attempts to increase the chance of finding a feasible solution by assigning new values of $\boldsymbol{j}$ while fixing $\boldsymbol{n}$. It may also reduce the degree of infeasibility when a chromosome is not made feasible. In the evolutionary process, both feasible and infeasible chromosomes are kept in the population.

4.3.3 Fitness value

GA works with a finite population, which evolves from one generation to the next, governed by the fitness of the chromosome. We relate the fitness f of a chromosome to the objective value of the ELSP when its represented solution is feasible or can be made feasible by a simple repair, whereas a penalty will be imposed on the fitness value if the chromosome cannot be made feasible. The detailed calculation of f is described as follows.

First if inequalities (4.3) are satisfied, the chromosome can represent a feasible solution with the optimal basic period W^* given by (4.4). With W^* and $\boldsymbol{n}$, f is calculated by (4.6), which is the objective function of the ELSP.

$$f = \sum_{i=1}^{I} (A_i/n_i W^* + H_i n_i W^*) \tag{4.6}$$

On the other hand, if inequalities (4.3) are not satisfied, the chromosome will go through a fast repair procedure described earlier which assigns new values to $\boldsymbol{j}$ to induce the feasibility. However, if the repaired chromosome is still infeasible, the fitness value is given by:

$$f = \sum_{i=1}^{I} (A_i/n_i W_R + H_i n_i W_R) + \text{penalty} \tag{4.7}$$

We suggest the value of the penalty to be related to:

- The degree of infeasibility of the chromosome.
- The number of generations, denoted by g.
- The sum of densities or the utilization of the problem, denoted by $\rho = \sum_{i=1}^{I} \rho_i$.

We let penalty $= t(\boldsymbol{n}, \boldsymbol{j}) P(g) Q(\rho)$, where

$$\begin{aligned} t(\boldsymbol{n}, \boldsymbol{j}) &= \sum_{k=1}^{K} \left(\sum_{i \in S_k} n_i \rho_i - 1 \right)^+ \\ P(g) &= g \\ Q(\rho) &= 1/\rho \end{aligned}$$

Thus our penalty is proportional to the degree of infeasibility, which is measured by:

$$\sum_{k=1}^{K} \left(\sum_{i \in S_k} n_i \rho_i - 1 \right)^+$$

On the other hand it is inversely proportional to the problem utilization ρ. In other words, the scheme imposes higher penalty for lower utilization problems while it is less severe for high utilization problems where infeasible solutions may be required in the evolution to generate feasible offsprings. We also make the penalty proportional to the number of generations in the GA so that infeasible solutions will be discarded towards the later part of the evolution.

4.3.4 Population initialization

Initially N solutions are randomly generated to form a population. For each solution or chromosome, n_i is randomly generated from $\{1, 2, \ldots, \overline{n}_i\}$ and j_i is randomly generated from $\{1, 2, \ldots, n_i\}$, both following the uniform distribution. Here $\overline{n}_i$ is chosen to be the maximal value that n_i can assume, i.e., $\overline{n}_i = 2^{\lfloor \log_2(1/\rho_i) \rfloor}$.

4.3.5 Selection and reproduction

During each successive generation, a portion of the best found solutions is kept to the next generation so that the new offsprings have a better chance to be generated from good solutions. In this GA, 10% of the best solutions are kept and 90% are generated from the crossover and mutation. To perform the crossover, we select two different parents with probabilities inversely

proportional to their fitness values. The selection probability is:

$$Prob(\text{select the } n^{th} \text{ chromosome}) = \frac{1/f_n}{\sum_{i=1}^{N}(1/f_i)}$$

where f_n is the fitness value of chromosome n.

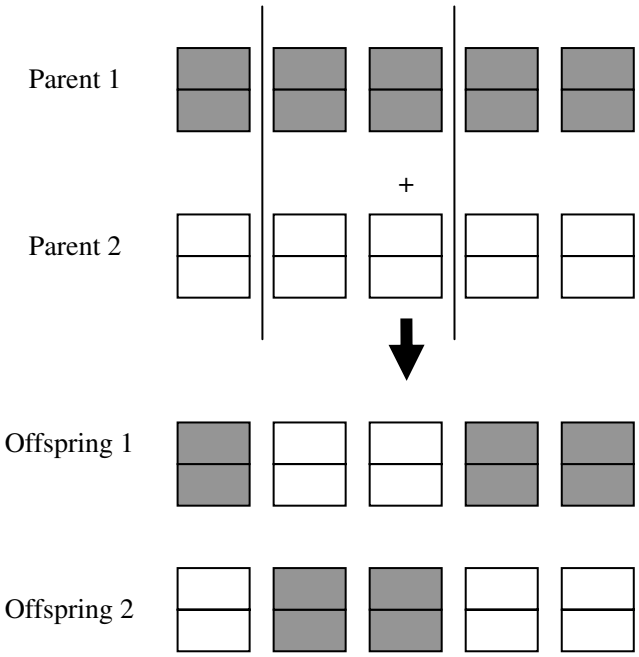

Figure 4.2 Two-position crossover for ELSP under the EBP and PoT policy

The reproduction of two offsprings is obtained by a two-position crossover on two parents as illustrated in Figure 4.2, where the two positions are generated randomly. The new offsprings share many characteristics of the parents. In addition, each offspring is assigned a small probability of mutation so that the solutions are more diversified. If a mutation takes place, the multiplier of each product is either changed or not with equal probability. If the multiplier

is changed, it is either doubled or halved with equal probability. In the case that the multiplier cannot be halved as it has a value of 1, it will be increased to 2.

When the mutation takes place and n_i is decreased, it is possible that $j_i > n_i$. If this happens, a new j_i is randomly generated in $\{1, 2, \ldots, n_i\}$ so that the chromosome can be a meaningful representation. If $j_i \leq n_i$, j_i will not be changed.

4.3.6 Values of parameters and the termination condition

The following values are used for the parameters:

- Population size: $N = 100$.
- Percentage of best solutions to be retained in the new generation: 10%.
- The termination condition is that either the best solution does not improve for 1,000 generations or 10,000 generations have been generated.
- Mutation rate: 0.1.

4.4 Computational Results

Our GA is coded in C++ and run on the same computer as mentioned in the previous chapter. For the GA, the computation time is roughly 2 seconds per

1000 generations.

4.4.1 Benchmark Problems

Elmaghraby (1978) developed six benchmark problems (see Appendix C), the first of which is the Bomberger problem. These problems assume 240 working days in a year and the interest rate is 10% per year. We apply our GA to these benchmark problems and compare it against other methods (see Table 4.1). Our GA finds good solutions whose costs are less than 1.5% away from the LB. Note that these solutions are, in fact, optimal EBP-PoT solutions and they can be solved by the SS algorithm very fast. The solutions found by Chatfield (2007) and Park and Yun (1984) are slightly better than those of our GA. This is expected as they adopt a more general policy.

No.	ρ	LB	GLS	GA	HH	P&Y	Fujita	Elmaphraby
1	0.88	7589	7697	7697	7697	7697	7823	8383
2	0.66	4663	4727	4731	4731	4728	4862	4944
3	0.71	8742	8801	8801	8801	8801	9347	9526
4	0.58	21418	21566	21717[a]	21716	21486	21799	21903
5	0.41	4169	4174	4194	4194	4191	4191	4216
6	0.59	21218	21399	21519	21519	21327	21612	21622

[a]Note that a smaller value of 21716 for problem number 4 was reported in Haessler (1979) which is due to difference in rounding. An optimal solution for this problem has 17.43 days as its basic period and (8,2,8,2,4,2,16,2,1,4) as its multiplier vector.

Table 4.1 Computational results for six benchmark ELSP problems

It is interesting to note that several algorithms, that perform well for the low utilization problems, may perform very poorly for the high utilization problems. In some cases, finding a good feasible solution can be difficult when the utilization is very high.

4.4.2 High Utilization Problems

Our GA is tested on high utilization problems, developed by Khouja et al. (1998). Table 4.2 shows that our GA outperforms other heuristics for high utilization problems. In Table 4.3, the basic periods, multipliers and production positions of our GA are reported.

Table 4.2 Computational results for high utilization problems with GA

No.	ρ	LB	Khouja	GLS	HH	Our GA
1	0.88	7589	8782	7697	7697	7697
2	0.92	7714	9746	7947	7972	7947
3	0.95	8420	12018	9140	11962	9097
4	0.98	15683	24534	20500	22526	19004

Note that our GA again finds optimal EBP-PoT solutions for all the tested problems. GLS is the algorithm proposed by Chatfield (2007) and HH is the heuristic proposed by Haessler (1979). Although Chatfield (2007) adopts a more general policy, the GA does not find a better solution than that of our GA. That is, it is more difficult to search for the global optima within a larger

Table 4.3 Multipliers and production positions for high utilization problems with GA

No.	ρ	W	$\boldsymbol{n}$	$\boldsymbol{j}$
1	0.88	23.42	(8,2,2,1,2,4,8,1,2,2)	(8,2,2,1,1,2,4,1,1,2)
2	0.92	23.18	(2,1,2,1,2,8,8,1,2,4)	(1,1,2,1,2,8,4,1,1,2)
3	0.95	40.44	(2,1,2,1,2,4,4,1,2,2)	(1,1,2,1,2,4,2,1,1,2)
4	0.98	101.32	(1,2,2,2,2,4,4,1,2,4)	(1,1,1,2,2,4,4,1,1,2)

feasible region. In addition, our GA keeps both feasible and infeasible chromosomes in the population to make the search more diversified. To quickly find a feasible solution, a repair procedure is used for the infeasible chromosomes, which is important when most randomly generated solutions are infeasible for the high utilization problems. It can be seen that HH does not perform well for high utilization problems. HH first determines the basic period and multipliers that can give lower cost without considering the feasibility and then uses a heuristic to determine production positions for the given basic period and multipliers. The heuristic fails in finding feasible production positions for high utilization problems in most cases so the resulting solution is not as good as for low utilization problems.

4.4.3 Randomly Generated Problems

Under the EBP and PoT policy, we make a comparison between our GA and HH for the 50 randomly generated problems used in the previous chapter. To generate these examples, we first randomly generate 10 examples following the uniform distribution based on the ranges given in Table 3.2 in the previous

chapter. The demand rate of each example is then scaled up to reach the five respective utilization levels. The inventory holding costs can be calculated from the unit costs.

Table 4.4 Computational results for randomly generated problems

	Average Values ($/day)				Number of Optimal EBP-PoT	
ρ	LB	Optimal EBP-PoT	GA	HH	GA	HH
0.70	33.58	34.35	34.35	34.42	10	6
0.80	35.63	36.59	36.59	37.40	10	4
0.90	37.57	39.30	39.31	41.10	9	2
0.95	43.04	46.87	46.92	48.07	9	3
0.98	80.22	90.55	90.89	95.66	9	1

As shown in Table 4.4, our GA finds optimal EBP-PoT solutions for most of the tested problems and the solutions are significantly better than those produced by HH. The convergence of the GA is shown in Appendix F.

4.5 Conclusions

In this chapter a GA is designed to solve the ELSP under the EBP and PoT policy. By recognizing that W can be best determined from $\boldsymbol{n}$ and $\boldsymbol{j}$, we use a lean representation which confines the search to the best solution among the solutions having the same $\boldsymbol{n}$ and $\boldsymbol{j}$. This not only makes the search efficient

by avoiding the inferior solutions sharing the same $\boldsymbol{n}$ and $\boldsymbol{j}$ but also it has cut down the search space by one dimension which speeds up the search tremendously. In the evolution, we allow infeasible solutions with proper penalties to be included. This feature is found to be useful in diversifying the search as well as finding good feasible solutions, especially for high utilization problems. For infeasible chromosomes, we also make them go through a repair which engages the help of a guided basic period plus the first-fit and the list-scheduling heuristics to induce good feasible solutions or to reduce the degrees of infeasibility. As a result, our computational experiment shows that our GA performs well for high utilization problems. For low utilization problems, optimal EBP-PoT solutions are normally close to optimal. Although heuristics with a more general policy may find a slightly better solution, its implementation could be significantly more difficult.

Chapter 5

MELSP

5.1 Problem Description

As mentioned in chapter 1, the problem assumes that all the machines are identical and the products are not allowed to be split on different machines. The CC policy for MELSP assumes that the products allocated to the same machine have the same cycle time, but different machines can have different cycle times. The EBP and PoT policy for MELSP is similar as in ELSP. Different machines are allowed to have different basic periods. The product cycle time is equal to a multiple of the basic period of the machine that produces the product.

Most notations in this section are the same as for ELSP, the followings are the extra notations for MELSP.

M = Number of machines;

$\bar{\rho}$ = $\sum_{i=1}^{I} \rho_i / M$, utilization of the problem;

T_m = Cycle time for products on machine m;

W_m = Basic period on machine m;

a_i = Machine number on which product i is produced;

K_m = Number of basic periods in a cycle for machine m;

S_{mk} = Set of products in the kth period in a cycle on machine m.

Under the CC policy, The cycle time of product i is equal to T_m if it is produced on machine m. Under the EBP and PoT policy, the cycle time of product i is equal to $n_i W_m$ if it is produced on machine m.

5.2 Genetic Algorithm for MELSP under the CC Policy

In this section, a GA is presented for the MELSP under the CC policy. It is assumed that the products produced on the same machine have a common cycle time. Under this assumption, the problem is to determine in the optimal way to allocate products on different machines so that the total cost is minimized. Once the allocation is determined, it is not difficult to find the optimal common cycle time for the products on each machine. With this observation, we present an encoding scheme to represent the product allocation and use it to search for the optimal solution.

5.2.1 Encoding scheme

An array of integers $\boldsymbol{a} = (a_1, a_2, \ldots, a_I)$ in Figure 5.1 is used to represent the allocation, where a_i represents the machine number on which product i is produced.

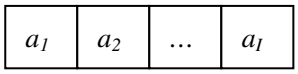

Figure 5.1 Chromosome for MELSP under the CC policy

We call $\boldsymbol{a}$ the chromosome in GA. Given $\boldsymbol{a}$, the set of products on machine m, denoted by P_m is determined. Formally, $P_m = \{i : a_i = m\}, m = 1, \ldots, M$. The cycle times T_m's are not encoded explicitly in the chromosome.

5.2.2 Feasibility

A chromosome represents a feasible solution if and only if

$$\sum_{i \in P_m} \rho_i < 1, \ m = 1, \ \ldots, \ M. \tag{5.1}$$

In the case that inequalities (5.1) are not satisfied for any of the machines, the chromosome does not represent a feasible solution. In GA, a number of chromosomes are kept in the solution pool (population) to evolve until the stopping criterion is met. Some of the genetic algorithms only keep the feasible chromosomes in the population. In our GA, we keep both feasible and infeasible chromosomes in the population as a big proportion of the possible

encoded chromosomes are infeasible especially for high utilization problems. With infeasible chromosomes, the search will be more diversified and is observed to converge faster, especially for high utilization problems.

To make full use of the infeasible chromosomes, a repair procedure is applied on them first so that the GA can move faster to feasible solutions. If a solution is not feasible, $\sum_{i \in P_m} \rho_i \geq 1$ for at least one machine. An intuitive way to repair the infeasibility is to repack the products so that $\sum_{i \in P_m} \rho_i < 1$ for all m. However, it is possible that the common cycle times of some of the machines may be unreasonably large due to the setup times of their assigned products. With this observation, we also consider setup times in repacking. As cycle times are not explicitly encoded in the chromosome, in order to include the setup times in repacking, we need to assume some common cycle time. It is easy to see that, for any feasible solution, we must have

$$\rho < \sum_{m=1}^{M} \sum_{i \in P_m} \left(\frac{s_i}{T_m} + \rho_i \right) \leq M.$$

We select only one cycle time value T_c for all the machines such that

$$\sum_{i=1}^{I} \left(\frac{s_i}{T_c} + \rho_i \right) = \frac{\rho + M}{2}.$$

where $(\rho + M)/2$ is the midpoint between ρ and M. That is,

$$T_c = \frac{2 \sum_{i=1}^{I} s_i}{M - \rho}.$$

Instead of using ρ_i to repack the products, we use $\rho_i^{'} = \rho_i + s_i/T_c$ in repacking. In the repair procedure, we will repair only those machines that are infeasible. First, products on them are removed. These products are then repacked one by one back into all the M machines. In the packing, the product with the largest value of $\rho_i^{'}$ is selected and assigned to the machine with the smallest sum of $\rho_i^{'}$.

The repair procedure attempts to find a feasible product allocation considering both setup times and production densities. Note that it is still possible that some machines remain infeasible, especially when the utilization is high. Despite this, the repair procedure will decrease their degrees of infeasibility.

5.2.3 Fitness value

Let $f = \sum_{m=1}^{M} f_m$ be the fitness value of a chromosome, where f_m is the fitness value of machine m. Given a feasible chromosome ($\sum_{i \in P_m} \rho_i < 1, \forall\ m$), the common cycle time on machine m is chosen to be T_m^* which minimizes $\sum_{i \in P_m}(A_i/T_m + H_i T_m)$ subject to $\sum_{i \in P_m}(s_i + \rho_i T_m) \leq T_m$. That is,

$$T_m^* = \max\left\{\sqrt{\frac{\sum_{i \in P_m} A_i}{\sum_{i \in P_m} H_i}},\ \frac{\sum_{i \in P_m} s_i}{1 - \sum_{i \in P_m} \rho_i}\right\}.$$

The average inventory and setup costs of machine m is

$$f_m = \sum_{i \in P_m}\left(\frac{A_i}{T_m^*} + H_i T_m^*\right).$$

We extend the definition to the infeasible chromosomes where penalty values will be imposed if they remain infeasible after the repair procedure. To be more specific, if machine m does not have enough capacity to produce its assigned products, the fitness value is penalized and it is given by

$$f_m = 2\sqrt{\sum_{i \in P_m} A_i \sum_{i \in P_m} H_i} + penalty_m.$$

Note that $2\sqrt{\sum_{i \in P_m} A_i \sum_{i \in P_m} H_i}$ is the minimum cost on machine m ignoring the capacity constraints. We suggest the value of $penalty_m$ to be related to:

- The degree of infeasibility.
- The average utilization of the problem, denoted by $\bar{\rho} = \rho/M$.
- The number of generations, denoted by g.

We let $penalty_m = t_m P(\bar{\rho}) Q(g)$, where

$$t_m = \sum_{i \in P_m} \rho_i - 1, \;\; P(\bar{\rho}) = 1/\bar{\rho}, \;\; Q(g) = g.$$

t_m is related to the degree of infeasibility on machine m. The bigger t_m is, the more the fitness value is penalized. The average utilization over all machines is an important factor that affects the difficulty of solving the problem. The bigger $\bar{\rho}$ is, the less the fitness value is penalized so that more infeasible solutions are kept in the population for the high utilization problem. The third factor of the product g makes the infeasible solutions less likely to stay in the

population after more generations have evolved.

5.2.4 Initialization

The initial population of N chromosomes are randomly generated from the product set of I sets, $\{1, 2, \ldots, M\} \times \{1, 2, \ldots, M\} \times \cdots \times \{1, 2, \ldots, M\}$. If a generated chromosome happens to be infeasible, it will go through a repair procedure first.

5.2.5 Selection

Selection plays an important role in GA. We use a binary tournament selection to form a parent pool of a target size for reproduction. To form a pool, the tournament selects two chromosomes randomly from the population and adds the one with the smaller fitness value to the pool if it has not been added earlier. Once a parent pool with a target size of N' is created, the reproduction will start which is done by repeatedly selecting two parents from the pool randomly to form a new offspring. Also to maintain good quality solutions in the subsequent generations, a percentage of the best solutions of the current generation are kept to the next generation.

5.2.6 Crossover and mutation

A uniform crossover is used to produce an offspring from two parents in which the genes of the new offspring are copied from the corresponding genes of the

two parents with equal probability. An illustration of the crossover is given in Figure 5.2.

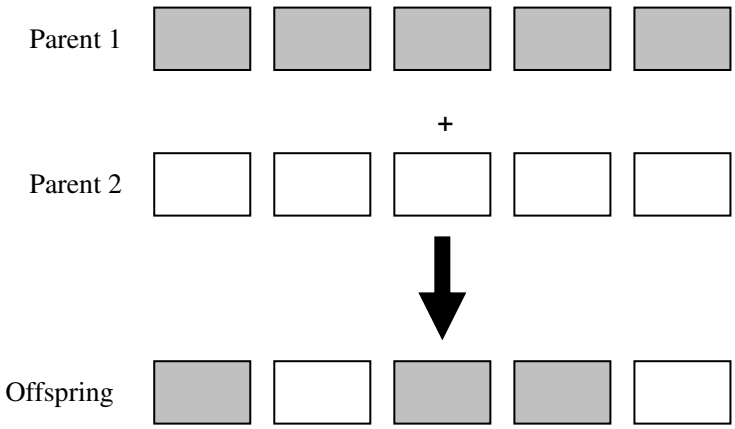

Figure 5.2 Uniform crossover for MELSP under the CC policy

Each new offspring is assigned a small probability for the possible mutation. If a mutation takes place, two of its genes will be randomly selected and the values of the two genes are interchanged.

5.2.7 Values of parameters and the termination condition

The values used in the GA are: $N = 50$, $N' = 30$. The mutation rate is 0.1. The GA terminates if the solution does not improve over 1,000 generations or the total number of generations reaches 10,000. The percentage of the best solutions to be kept to the next generation is 20%.

5.3 Genetic Algorithm for MELSP under the EBP and PoT Policy

The EBP and PoT policy assumes that each machine is associated with a basic period W_m. The cycle times of the products on the machine are power-of-two multiples of the basic period. Denote the multiplier of product i by n_i. Under this policy, a solution for the MELSP is specified by three decisions. The first involves the allocation of the products to the M machines. The second involves the determination of the basic periods for the M machines. The third involves the choice of the product cycle times expressed as power-of-two multiples of the basic periods as well as the staggering or production positions of the products.

5.3.1 Encoding scheme

We use three arrays of integers $(\boldsymbol{a}, \boldsymbol{n}, \boldsymbol{j})$ as a chromosome to represent a solution (see Figure 5.3). As before, $\boldsymbol{a} = (a_1, a_2, \ldots, a_I)$ defines the machine numbers on which the products are produced. As before, the multiplier vector $\boldsymbol{n} = (n_1, n_2, \ldots, n_I)$ and the production position vector $\boldsymbol{j} = (j_1, j_2, \ldots, j_I)$ respectively define the product cycle times expressed as power-of-two multiples of the basic periods and the production positions of the products.

For example, if we have two machines and four products and a chromosome has $a_1 = 1, a_2 = 2, a_3 = 1, a_4 = 2, n_1 = 1, n_2 = 1, n_3 = 2, n_4 = 2, j_1 = 1, j_2 = 1, j_3 = 1, j_4 = 2$, then the production schedule can be constructed as in Figure 5.4. Similar to GACC, we do not encode the values of the basic periods

a_1	a_2	...	a_I
n_1	n_2	...	n_I
j_1	j_2	...	j_I

Figure 5.3 Chromosome for MELSP under the EBP and PoT policy

in the chromosome explicitly.

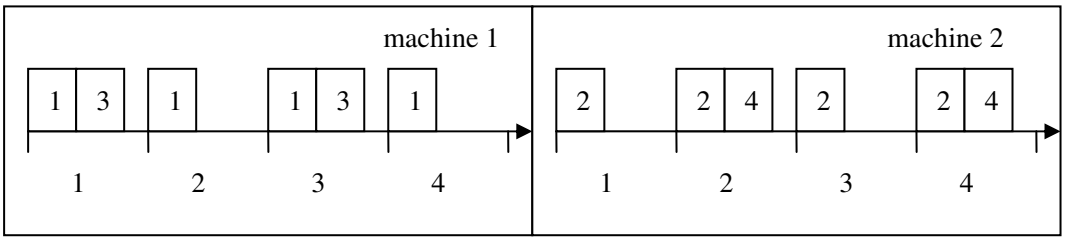

Figure 5.4 A simple example for MELSP

As before, we let P_m denote the set of products allocated to machine m. Let K_m be the number of basic periods until the product schedule repeats on machine m. Given that all the multipliers are powers of two, K_m is equal to the maximum of $n_i, i \in P_m$. For $m = 1, \ldots, M$, let S_{mk} be the set of products produced on the kth basic period of machine m. Formally, $P_m = \{i : a_i = m\}$, $m = 1, \ldots, M$; $K_m = \max_{i \in P_m}\{n_i\}$, $m = 1, \ldots, M$; and $S_{mk} = \{i \in P_m : j_i = (k-1)(\mod n_i)+1\}$, $m = 1, \ldots, M$ and $k = 1, \ldots, K_m$.

5.3.2 Feasibility

A chromosome is feasible only when each machine can produce its assigned products according to the given multipliers and production positions. Therefore, the feasibility is determined by the respective machine resource utilizations as well as the production frequencies and production positions. Ignoring the constraints imposed by the production frequencies and positions, the necessary conditions for a solution to be feasible are

$$\sum_{i \in P_m} \rho_i < 1, \ m = 1, \ \ldots, \ M. \tag{5.2}$$

These conditions state that the allocation of products to different machines should ensure that no machine is over utilized. If $\sum_{i \in P_m} \rho_i \geq 1$ for some m, the chromosome does not represent a feasible solution and it will be repaired by a repair procedure which is similar to that described in GACC. To use it, we need to select a common value for basic periods, denoted by W_c. In our work, we select the value of W_c such that

$$\sum_{i=1}^{I} \left(\frac{s_i}{n_i W_c} + \rho_i \right) = \frac{\rho + M}{2}.$$

That is,

$$W_c = \frac{2 \sum_{i=1}^{I} s_i / n_i}{M - \rho}.$$

The same repair procedure as in GACC is then applied except that $\rho_i^{'} = \rho_i + s_i/(n_i W_c)$. If $\sum_{i \in P_m} \rho_i$ remains greater than or equal to 1 for some of the

machines after the repair, the chromosome will be declared infeasible and a penalty described later will be imposed on the fitness value.

For machine m, necessary and sufficient conditions for feasibility under EBP and PoT are

$$\sum_{i \in S_{mk}} n_i \rho_i < 1, \quad k = 1, \ \ldots \ , K_m. \tag{5.3}$$

If conditions (5.3) hold for all $k = 1, \ldots, K_m$, then we can always find a basic period that is long enough to produce all the products in S_{mk} regardless of the values of setup times. For example, in Figure 5.4, $K_1 = 2, S_{11} = \{1, 3\}, S_{12} = \{1\}; K_2 = 2, S_{21} = \{2\}, S_{22} = \{2, 4\}$. If $\rho_1 + 2\rho_3 < 1$, then a feasible production schedule can be constructed for machine 1. Similarly if $\rho_2 + 2\rho_4 < 1$, then a feasible production schedule can be constructed for machine 2.

For the machines that satisfy (5.2), we check the necessary and sufficient conditions (5.3). If conditions (5.3) fail for machine m, a repair procedure described in the previous chapter will be applied. This procedure reassigns the production positions of products on machine m so as to induce the feasibility. To do this, we need to select a value of the basic period in the repair procedure. Following the method in the previous chapter, the selected value of the basic period

$$\underline{W_m} = \max \left\{ \sqrt{\frac{\sum_{i \in P_m} A_i / n_i}{\sum_{i \in P_m} H_i n_i}}, \ \frac{\sum_{i \in P_m} s_i / n_i}{1 - \sum_{i \in P_m} \rho_i} \right\} \tag{5.4}$$

is used to estimate the sum of setup time and processing time $\sigma_i = s_i +$

$\rho_i n_i \underline{W_m}$, $\forall\ i \in P_m$. To reassign the production positions, the products are ordered increasingly by n_i. For products with the same multiplier, they are ordered decreasingly by σ_i. Based on the list, the first-fit heuristic is applied to pack the products one by one to the first basic period that can accommodate the product. The production schedule for product i is repeated each n_i periods. Hence, each product is packed in K_m/n_i basic periods. In the packing, it may happen that none of the basic periods have enough time to accommodate the production of some of the products. In this case, these products are packed in the period that has the smallest sum of $n_i\rho_i$. As a result the length of the periods receiving the products will exceed $\underline{W_m}$. In the end, this repair procedure may give a feasible schedule on machine m or reduce the degree of infeasibility of the schedule.

5.3.3 Fitness value

Similar to GACC, we define $f = \sum_{m=1}^{M} f_m$ as the fitness value of a chromosome, where f_m is the fitness value of machine m. There are three cases for the calculation of f_m.

1. Conditions (5.3) hold for all k. This is a feasible solution. The optimal basic period that minimizes $\sum_{i\in P_m}(A_i/(n_i W_m) + H_i n_i W_m)$ subject to (5.3) is given by

$$W_m^* = \max\left\{\sqrt{\frac{\sum_{i\in P_m} A_i/n_i}{\sum_{i\in P_m} H_i n_i}},\ \max_{k=1,\ldots,K_m}\left\{\frac{\sum_{i\in S_{mk}} s_i}{1-\sum_{i\in S_{mk}} n_i\rho_i}\right\}\right\}.$$

Thus, the fitness value of machine m is given by

$$f_m = \sum_{i \in P_m} \left(\frac{A_i}{n_i W_m^*} + H_i n_i W_m^* \right).$$

2. Conditions (5.3) do not hold for some k and $\sum_{i \in P_m} \rho_i < 1$. In this case, a penalty will be imposed on the fitness value of machine m, and f_m is given by

$$f_m = \sum_{i \in P_m} \left(\frac{A_i}{n_i \underline{W_m}} + H_i n_i \underline{W_m} \right) + Penalty_{m,1},$$

where $\underline{W_m}$ is given by (5.4). In this algorithm, we let

$$Penalty_{m,1} = t_{m,1} P(\bar{\rho}) Q(g),$$

where

$$t_{m,1} = \sum_{k=1}^{K_m} \left(\sum_{i \in S_{mk}} n_i \rho_i - 1 \right)^+, \; P(\bar{\rho}) = 1/\bar{\rho}, \; Q(g) = g.$$

3. $\sum_{i \in P_m} \rho_i \geq 1$. In this case, machine m does not have enough capacity to produce the assigned products. Similar to GACC, the fitness value of machine m is given by

$$f_m = \sum_{i \in P_m} \left(\frac{A_i}{n_i \hat{W}_m} + H_i n_i \hat{W}_m \right) + Penalty_{m,2}, \tag{5.5}$$

where

$$\hat{W}_m = \sqrt{\frac{\sum_{i \in P_m} A_i / n_i}{\sum_{i \in P_m} H_i n_i}}$$

is the value that minimizes the total cost ignoring the capacity constraint. In this algorithm, we let

$$Penalty_{m,2} = t_{m,2} P(\bar{\rho}) Q(g),$$

where

$$t_{m,2} = \left(\sum_{i \in P_m} \rho_i - 1 \right) \max_{i \in P_m} \{n_i\}, \; P(\bar{\rho}) = 1/\bar{\rho}, \; Q(g) = g.$$

5.3.4 Initialization

N chromosomes are generated in the initial population. 10% of the chromosomes are generated using the following procedure.

To generate a chromosome representation, the products are added one by one to the machines. In each iteration, a product is selected randomly from the remaining products and added to the first machine that can accommodate

it. To check whether a product can be added to a machine, the sum of ρ_i for all the products already assigned to the machine plus the current product is calculated. If it is less than 1, the product is added to the machine. Otherwise, the product is assigned to the next machine that can fit it. If the machine happens to be the last machine, all the products will be assigned to the last machine without checking the sum of ρ_i. After all the products have been packed, $\boldsymbol{a}$ is known. Let $\boldsymbol{n} = \boldsymbol{j} = (1, 1, \ldots, 1)$. There is a great chance that the generated solution is feasible and with the injection of highly likely initial feasible solutions the algorithm often improves faster.

The other 90% of the initial chromosomes are randomly generated from the appropriate ranges. That is, for each chromosome, $\boldsymbol{a}$ is randomly generated from the product set of I sets, $\{1, 2, \ldots, M\} \times \{1, 2, \ldots, M\} \times \cdots \times \{1, 2, \ldots, M\}$, $\boldsymbol{n}$ is randomly generated from $\{1, 2, \ldots, \overline{n_1}\} \times \{1, 2, \ldots, \overline{n_2}\} \times \cdots \times \{1, 2, \ldots, \overline{n_I}\}$, and $\boldsymbol{j}$ is randomly generated from $\{1, 2, \ldots, n_1\} \times \{1, 2, \ldots, n_2\} \times \cdots \times \{1, 2, \ldots, n_I\}$, where $\overline{n_i}$ is an upper bound of n_i, which is chosen to be the biggest integer that is less than $1/\rho_i$.

5.3.5 Selection and crossover

The selection and crossover are similar to the ones used in GACC. In other words, a target size of a parent pool is selected by a binary tournament for reproduction. Also a uniform crossover (see Figure 5.5) is used to produce new offsprings from randomly selected parents. And a percentage of the best chromosomes is kept to the next generation.

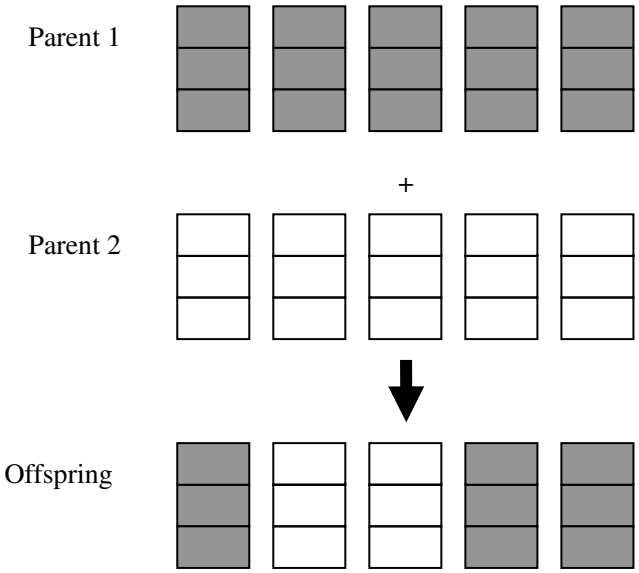

Figure 5.5 Uniform crossover under the EBP and PoT policy

5.3.6 Mutation

Each new offspring has a small probability of being mutated. When a mutation takes place, the following two operators will be used randomly with equal probability.

- MUT1: Each multiplier has a small probability of being changed. If a change occurs, a multiplier will be either doubled or halved randomly with equal probability. In the case that the multiplier is equal to 1, it will be simply increased to 2. Once a multiplier is changed, it may happen that $j_i > n_i$. In this case j_i is replaced by a random number in $\{1, 2, \ldots, n_i\}$.

- MUT2: Select a random number r from $\{2, 3, \ldots, M\}$. Select any r ma-

chines and for each of the selected machines, randomly choose a product from the machine. These products are then rotated from one machine to another among all the machines. For example, if three machines are selected, the product on the first machine will be moved to the second machine, the product on the second machine will be moved to the third machine and the product on the third machine will be moved to the first machine.

5.3.7 Values of the parameters and termination condition

The parameters used in GAEBP are summarized here. The population size is $N = 100$. The percentage of best chromosomes kept to the next generation is 20%. The size of the targeted parent pool is $N' = 60$. The mutation rate is 0.1. In MUT1, each multiplier has a probability of 0.1 to be changed. The termination condition is that either the best solution does not improve after 1,000 generations or 10,000 generations have been generated.

5.4 Computational Results

Three algorithms, Carreno's heuristic, GACC and GAEBP, are compared on randomly generated problems. The algorithms are coded in C++ and run on an Intel Pentium CPU 3GHz personal computer with a memory of 0.99 GB. The problems are generated for five different numbers of machines-2, 4, 6, 8 and 10. The number of products is five times of the number of machines. For

each problem setting, ten tested problems are generated where their parameters are randomly generated from the ranges given in Table 5.1. The settings in the table are the same as those used by Carreno (1990).

Table 5.1 Ranges of parameters for MELSP

Parameter	Mean	Range
Production Rate (unit/day)	14000.00	5000.00
Setup Cost ($)	200.00	400.00
Setup Time (day)	0.28	0.44
Holding Cost ($/unit/year)	0.35	0.70
Demand Rate (unit/day)	2500.00	4800.00

We use GAP = (solution's value - LB)/LB to measure the performance of the heuristics, where LB is the lower bound of the problem used in Carreno (1990). For easy of reference, a description of LB is included in Appendix D. For each number of machines and utilization, the average GAP of 10 problems is reported in Figures 5.6 - 5.11. The running time of these algorithms is reported in Figures 5.12 - 5.17. The cap of running time for Carreno's heuristic is set at 50 seconds. The x-axis represents the number of machines and the y-axis represents the running time in seconds. Convergence of GACC and GAEBP is reported in Appendix F.

From the computational results, we find that solutions of GACC dominate solutions of Carreno's heuristic for all the different settings. For the high utilization problems, GACC is a lot better than Carreno's heuristic. This is

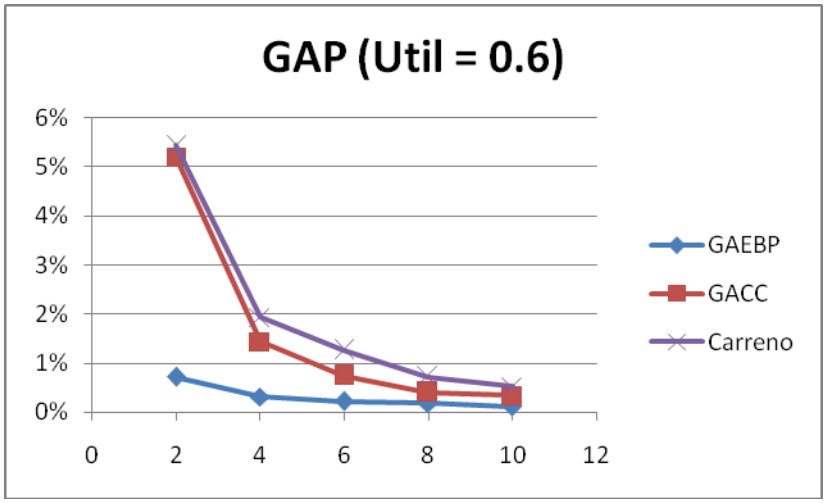

Figure 5.6 Computational results for utilization 0.6

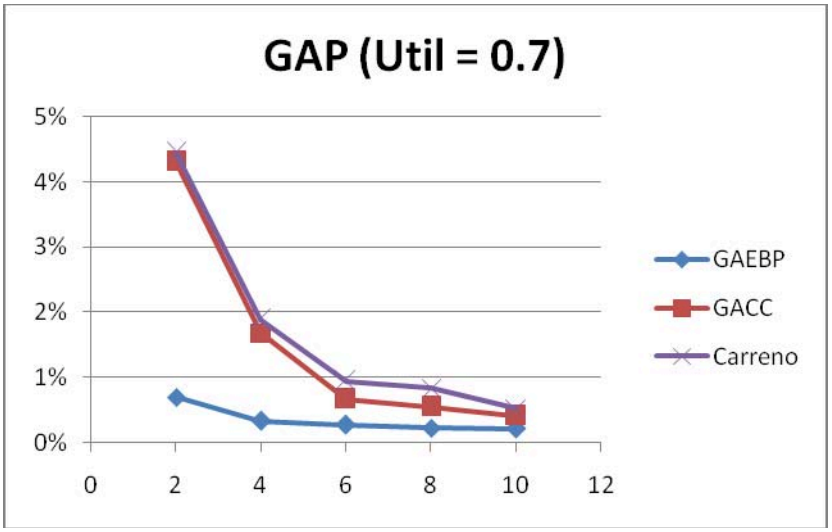

Figure 5.7 Computational results for utilization 0.7

mainly due to the difficulty of using Carreno's heuristic to solve the high utilization problems. The running time of GACC does not increase much when the number of machines increases, but the running time of Carreno's heuristic increases dramatically when the utilization and machine number increase and the cap of running time is reached for a number of them.

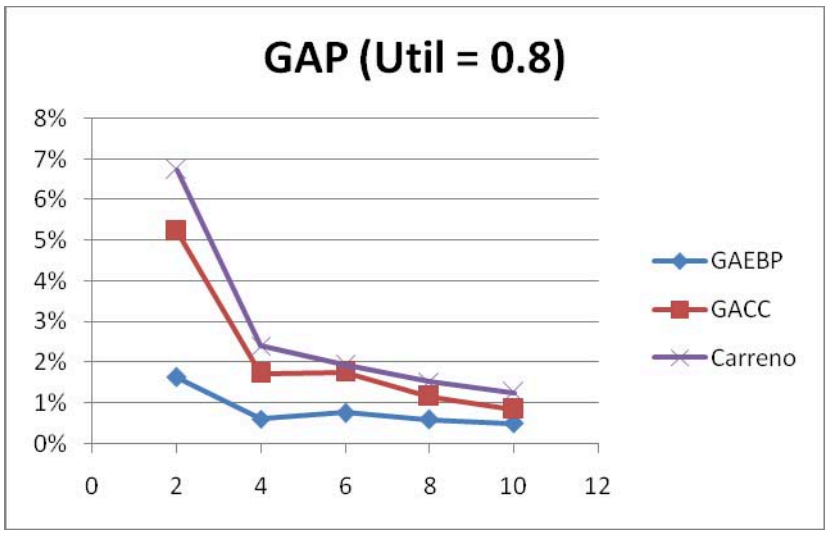

Figure 5.8 Computational results for utilization 0.8

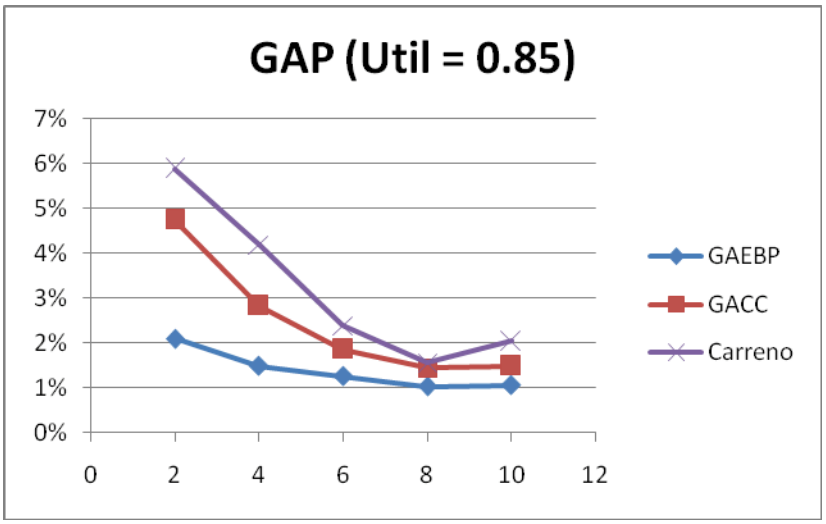

Figure 5.9 Computational results for utilization 0.85

GAEBP is slower than GACC, but the solution of GAEBP dominates GACC for all the settings. For the problems with two machines, GAEBP is much better than GACC for low utilization problems, but it is not difficult

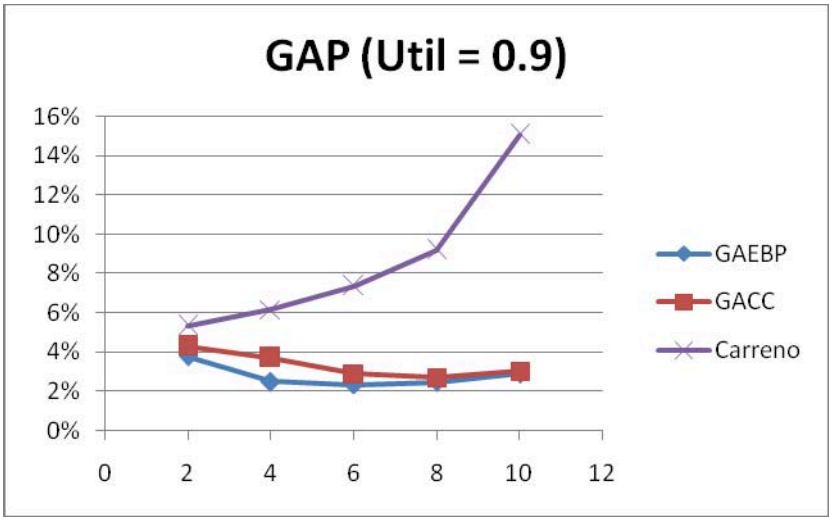

Figure 5.10 Computational results for utilization 0.9

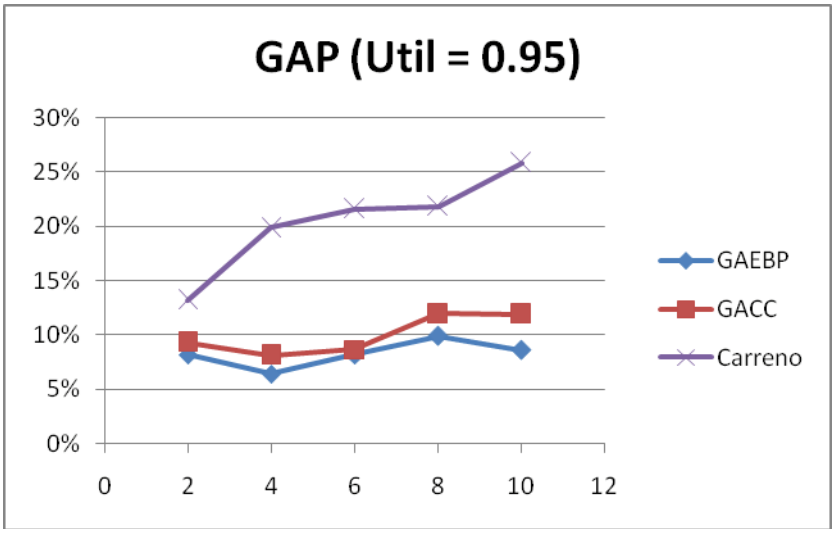

Figure 5.11 Computational results for utilization 0.95

to see that the improvement of GAEBP over GACC decreases as the number of machines increases. In general, the more machines we have, the more flexibility we have for cycle times under the CC policy. For the 10-machine problems, the algorithms under the CC policy perform well. Their average GAPs are around 1% or less for low utilization problems. This is because the

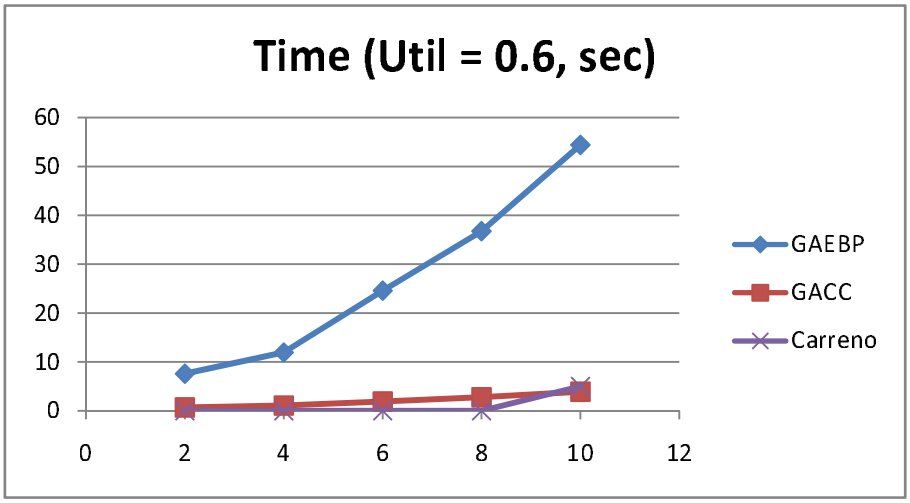

Figure 5.12 Running time for utilization 0.6

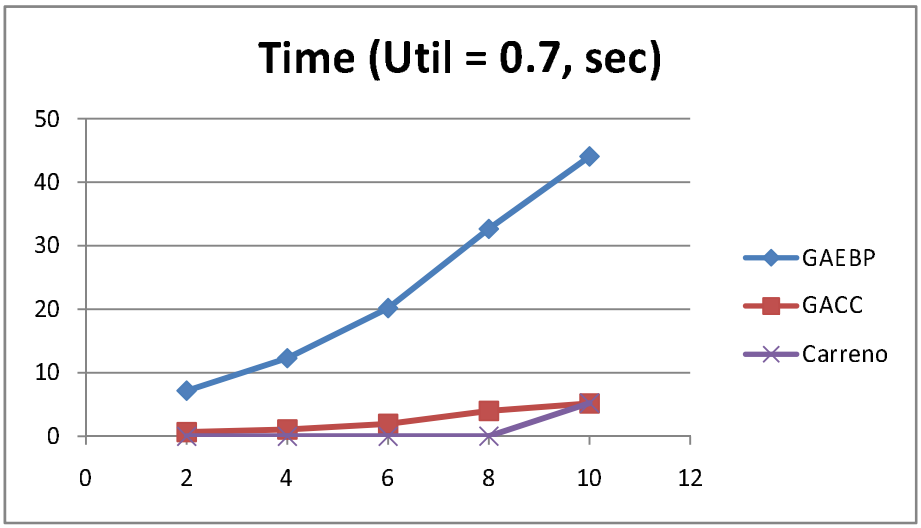

Figure 5.13 Running time for utilization 0.7

restriction of the CC policy is alleviated by the flexibility of using different cycle times on different machines. When there are ten machines, it is not difficult to find a good grouping so that the common cycle time of the machine is close to the independent cycle times of all products produced in that machine. However, when there are a few machines, products with much different

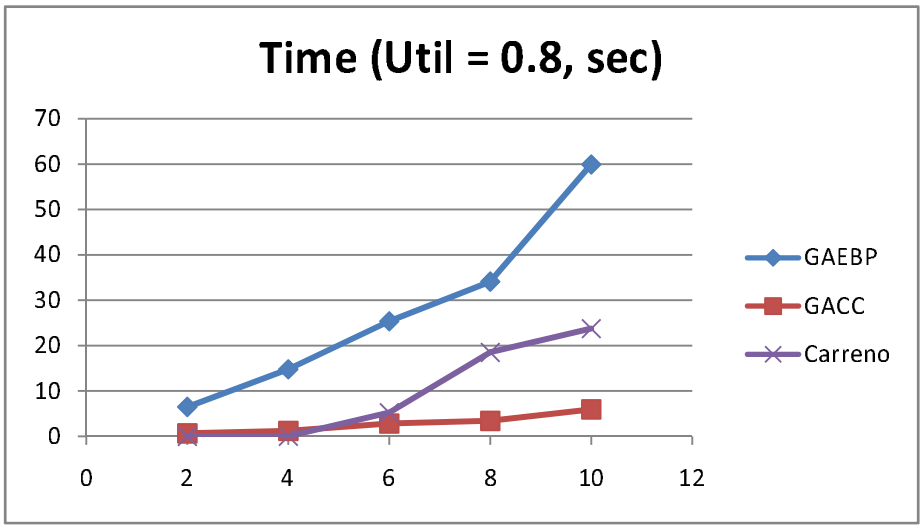

Figure 5.14 Running time for utilization 0.8

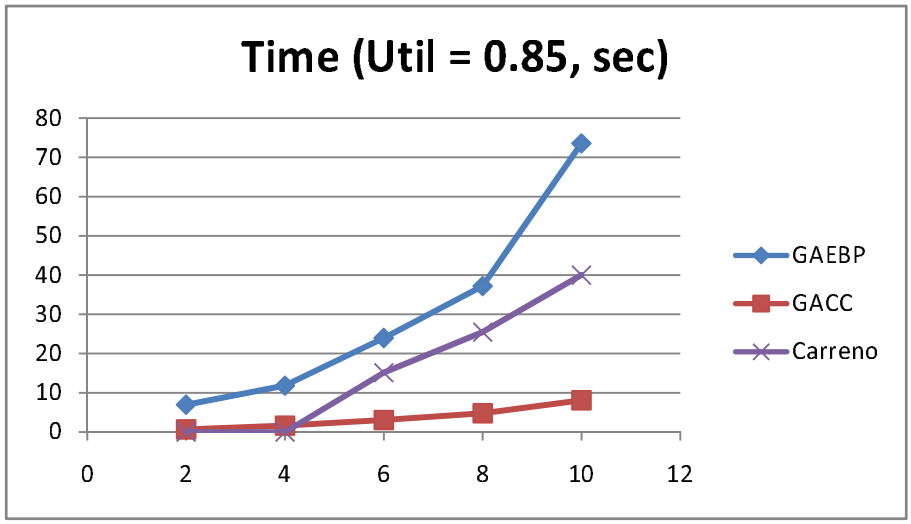

Figure 5.15 Running time for utilization 0.85

independent cycle times may have to be produced in the same machine. As a result, the performance of the heuristics under the CC policy deteriorates when the number of machines decreases. It is clearly seen that GAEBP performs much better than the other two heuristics, especially when the number of machines is small and the machine utilization is low.

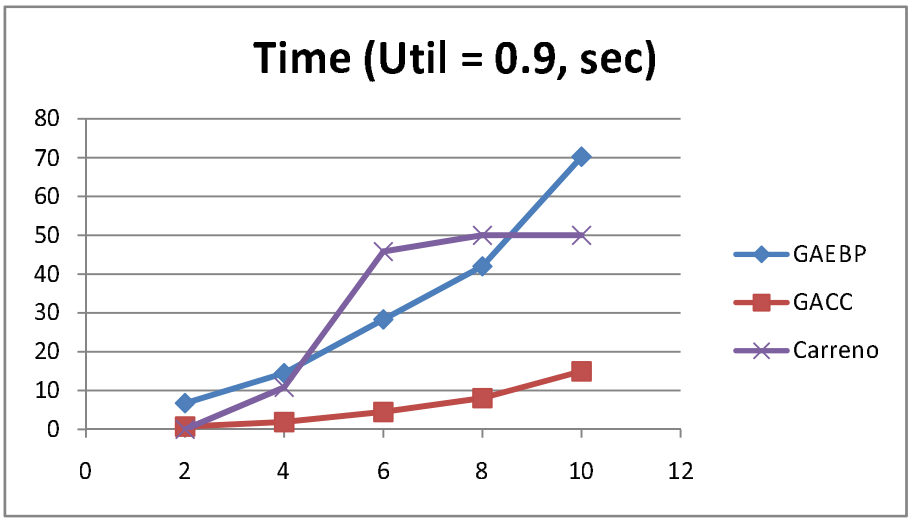

Figure 5.16 Running time for utilization 0.9

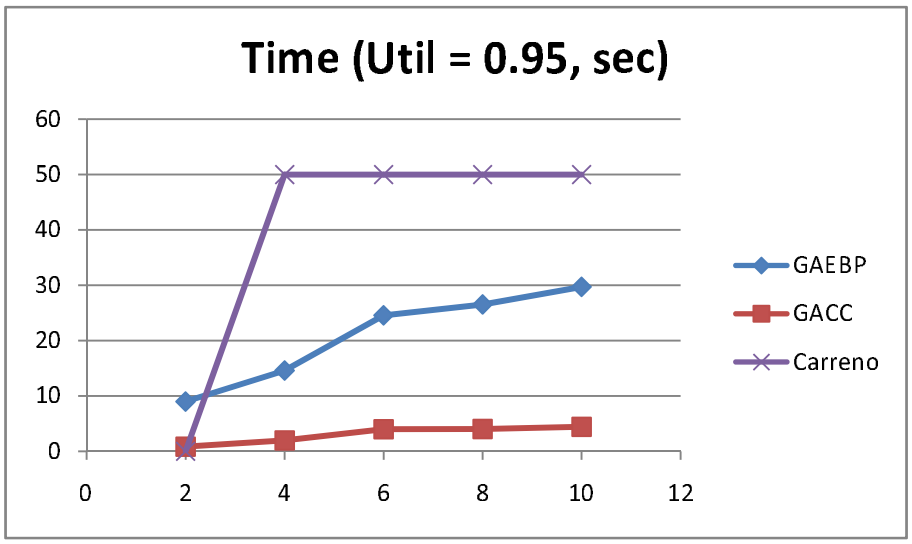

Figure 5.17 Running time for utilization 0.95

5.5 Conclusions

In this chapter, we discuss two different policies for the MELSP and propose two genetic algorithms under the two policies. The first policy we use is the CC policy, which assumes that the products produced on the same machine have a common cycle time. It is shown empirically that GACC can find better solutions than Carreno's heuristic for all the different settings and compared with the latter it requires less running time for the high utilization problems. Among all the three algorithms, GAEBP performs best though it requires a longer running time.

It is observed that the GAP of Carreno's heuristic is bigger when the number of machines is small. For the 2-machine problem, the average GAP can be improved dramatically by using a GA under the EBP and PoT policy. A genetic algorithm under this policy can find very good solutions for the MELSP. From the computational experiment, we see that the solution quality of GAEBP dominates GACC and Carreno's heuristic for all different settings. Particularly, it finds better solutions when the number of machines is small.

Chapter 6

Conclusions

6.1 Conclusions

In this dissertation, the ELSP under the EBP and PoT policy is discussed. A search algorithm that finds the optimal solution, an efficient search algorithm and a GA are presented for this problem. The MELSP is also discussed under the CC policy and under the EBP and PoT policy. Two genetic algorithms are presented for this problem under the two policies.

The ELSP under the EBP and PoT policy is formulated as a nonlinear integer programming problem. We formulate the problem in such a way that once one of the decision variables is treated as a parameter, the problem becomes an integer linear program. For each value of the parameter, an integer linear programming problem is solved optimally. After all the possible values of the parameter have been searched, the global optimal solution is determined. This algorithm is the first to find the optimal solution for the problem under this policy.

Based on the insights drawn from the algorithm, an efficient search heuristic is proposed. The heuristic focuses the search on the values of the parameter that are likely to give the global optimal solution. Though we show that the heuristic does not guarantee for a global optimal solution, it finds the optimal solutions for 98% of all the problems tested and it is much faster than the algorithm which is designed to find the optimal solutions.

We observe that the structure of the problem under the EBP and PoT policy is suitable for GA. A GA is proposed for solving the problem, and it is found that the GA can find optimal EBP-PoT solutions for most of the problems tested. The running time of GA does not increase much when the number of products increases or the utilization is high, which is not guaranteed by the previous two algorithms. In the literature, another GA (Chatfield, 2007) was presented under the EBP policy without the PoT restriction. It is shown that our GA under the more restricted policy performs better for several high utilization problems. This result shows that it is important to find a policy that can effectively reduce the complexity of the problem so that an efficient algorithm can be developed.

The MELSP schedules several products on multiple identical machines. We investigate two policies for the MELSP, the CC policy and the EBP and PoT policy. The CC policy reduces the complexity of the problem greatly. However, no one has found the optimal solution even under the CC policy. Carreno (1990) proposed a local search heuristic to solve the MELSP under the CC policy and it can find very good solutions for most of the problems tested.

However, the local search heuristic requires a generalized assignment problem to be solved in each iteration. When the problem becomes more complicated, the heuristic does not guarantee to finish the iterations in a short time due to the complexity of solving the assignment problem.

To overcome the disadvantage of the local search heuristic, a GA is presented for the MELSP under the CC policy. The problem is encoded with integer encodings and the GA is tested for randomly generated problems. The GA outperforms the local search heuristic for all the different settings tested. In addition, the GA ensures that the running time is reasonable for all the problems, which is not true for Carreno's heuristic.

When we test the GA and the local search heuristic from Carreno (1990) for the problems with only two machines, it is found that the solution errors are quite high compared with when there are five or ten machines. It is not difficult to see that this is due to the restriction of the CC policy. The restriction can only be alleviated when there are more machines. Therefore, we develop a GA under the EBP and PoT policy. This policy is very flexible and it is allowed that all the products to have different cycle times. Though it increases the problem complexity, the solution quality is improved a lot. It is found that the algorithm dominates the other two algorithms in terms of solution quality. And the less machines we have, the more improvement can be gained by using the EBP and PoT policy compared with using the CC policy.

6.2 Future Research

The search algorithm to find the optimal solution presented for the ELSP in this dissertation requires to search all the possible values of the parameter. The search algorithm will be more effective if more properties can be found for this problem under the EBP and PoT policy. One future direction is to find useful properties for the problem to speed up the search algorithm. Another direction is to extend the nonlinear integer programming model to solve the other extensions of the ELSP. As long as the extra constraints are linear, the model is tractable and can be very useful for analyzing the structure of the ELSP with other extensions.

The MELSP is very difficult as the allocation problem is combined with the scheduling problem. The CC policy reduces the complexity and the necessary and sufficient conditions were presented by Carreno (1990) by a nonlinear model under this policy. However, no algorithm can guarantee to find the optimal solution for the nonlinear programming problem. Future research can focus on formulating the problem in a different way so that the mathematical model is tractable and the optimal solution can be found by the linear programming or integer linear programming techniques under certain policies.

In this dissertation, the genetic algorithms are used for both ELSP and MELSP. It is found that the problem structures are suitable for GA. And there are several applications for the ELSP with other meta-heuristics. The future direction can focus on exploring the possibility of applying meta-heuristics to solve the MELSP with more realistic considerations.

Bibliography

Boctor, F.F. The two-product, single-machine, static demand, infinite horizon lot scheduling problem, Management Science, *28*, pp. 798-807. 1982.

Boctor, F.F. The g-group heuristic for single machine lot scheduling, International Journal of Production Research, *25*, pp. 363-379. 1987.

Bollapragada, R. and Rao, U. Single-stage resource allocation and economic lot scheduling on multiple, nonidentical production lines, Management Science, *45*, pp. 889-904. 1999.

Bomberger, E. A dynamic programming approach to a lot size scheduling problem, Management Science, *12*, pp. 778-784. 1966.

Bourland, K.E. and Yano, C.A. The strategic use of capacity slack in the economic lot scheduling problem with random demand, Management Science, *40*, pp. 1690-1704, 1994.

Carreno, J. J. Economic lot scheduling for multiple products on parallel

identical processors, Management Science, *36*, pp. 348-358. 1990.

Chatfield, D. The economic lot scheduling problem: A pure genetic search approach, Computers & Operations Research, *34*, pp. 2865-2881. 2007.

Clausen, J. and Ju, S. A hybrid algorithm for solving the economic lot and delivery scheduling problem in the common cycle case, European Journal of Operational Research, *175*, pp. 1141-1150. 2006.

Delporte, C.M. and Thomas, L.J. Lot sizing and sequencing for N products on one facility, Management Science, *23*, pp. 1070-1079. 1977.

Dobson, G. The economic lot scheduling problem: Achieving feasibility using time-varying lot sizes, Operations Research, *35*, pp. 764-771. 1987.

Dobson, G. The cyclic lot scheduling problem with sequence-dependent setups, Operations Research, *40*, pp. 736-749. 1992.

Doll, C.L. and Whybark, D.C. An interactive procedure for the single-machine multi-product lot scheduling problem, Management Science, *20*, pp. 50-55. 1973.

Driscoll, W.C. and Emmons, H. Economic lot scheduling problem: a resolution of feasibility, using time-varying lot sizes, AIIE Transactions, *35*, pp.

388-395. 1977.

Elmaghraby, S.E. The economic lot scheduling problem (ELSP): Review and extensions, Management Science, *24*, pp. 587-598. 1978.

Federgruen, A. and Katalan, Z. The stochastic economic lot scheduling problem: Cyclical base-stock policies with idle times, Management Science, *42*, pp. 783-796. 1996.

Federgruen, A. and Katalan, Z. Determining production schedules under base-stock policies in single facility multi-item production systems, Operations Research, *46*, pp. 883-898. 1998.

Fujita, S. The application of marginal analysis to the economic lot scheduling problem, AIIE Transactions, *10*, pp. 354-361. 1978.

Gallego, G. and Dong, X.S. Complexity of the ELSP with general cyclic schedules, IIE Transactions, *29*, pp. 109-113. 1997.

Galvin, T. Economic lot scheduling problem with sequence dependent setup costs, Production and Inventory Management, *1*, pp. 96-105. 1987.

Geng, P.C. and Vickson, R.G. Two heuristics for the economic lot scheduling problem: An experimental study, Naval Research Logistics, *35*, pp. 605-617.

1988.

Geoffrion, A.M. and Graves, G.W. Scheduling parallel production lines with changeover costs: practical application of a quadratic assignment/LP approach, Operations Research, *24*, pp. 595-610. 1976.

Goldberg, D.E. Genetic algorithms in search, optimization and machine learning. Addison-Wesley. 1989.

Goldberg, D.E. and Deb, K. A comparison analysis of selection schemes used in genetic algorithms, In G. Rawlins, ed., Foundation of Genetic Algorithms. Morgan Kaufman. 1991.

Goyal, S.K. Scheduling a multi-product single machine system, International Journal of Production Research, *24*, 261-266. 1973.

Grznar, J. and Riggle, C. An optimal algorithm for the basic period approach to the economic lot scheduling problem, Omega, *25*, pp. 355-364. 1997.

Gupta, D. On the economic lot scheduling problem with backlogging: the common cycle approach, Operations Research Letter, *12*, pp. 101-109. 1992.

Haessler, R. An improved extended basic period procedure for solving the

economic lot scheduling problem, AIIE Transactions, *11*, pp. 336-340. 1979.

Hahm, J. and Yano C.A. The economic lot delivery scheduling problem: The common cycle case, IIE Transactions, *27*, pp. 113-125. 1995.

Hancock, P.J.B. An empirical comparison of selection methods in evolutionary algorithms, In T. C. Fogarty, ed., Evolutionary Computing: AISB Workshop, Leeds, U.K., April 1994, Selected Papers. Springer-Verlag.

Hanssmann, F. Operations research in production and inventory control. John Wiley & Sons. 1962.

Holland, J.H. Adaption in natural and artificial systems. University of Michigan Press, Ann Arbor, MI. 1975.

Hsu, W. On the general feasibility test of scheduling lot sizes for several products on one machine, Management Science, *29*, pp. 93-105. 1983.

Jensen, M.T., Khouja, M. An optimal polynomial time algorithm for the common cycle economic lot and delivery scheduling problem, European Journal of Operational Research, *156*, pp. 305-311. 2004.

Jones, P.C. and Inman, R.R. When is the economic lot scheduling problem

easy? IIE Transactions, *21*, pp. 11-20. 1989.

Karmarkar, U.S., Lot sizes, lead times and in-process inventories, Management Science, *33*, pp. 409-418. 1987.

Khouja, M., Michalewicz, Z. and Wilmot, M. The use of genetic algorithms to solve the economic lot scheduling problem, European Journal of Operational Research, *110*, pp. 509-524. 1998.

Khoury, B.N., Abboud, N.E. and Tannous, M.M. The common cycle approach to the ELSP problem with insufficient capacity, International Journal of Production Economics, *73*, pp. 189-199. 2001.

Larraneta, J. and Onieva, L. The economic lot-scheduling problem: a simple approach, Journal of Operational Reseach Society, *39*, pp. 373-379. 1988.

Leachman, R.C. and Gascon, A. A heuristic scheduling policy for multi-item, single-machine production systems with time-varying stochastic demands, Management Science, *37*, pp. 377-390. 1988.

Leachman, R.C., Xiong, Z.K., Gascon, A. and Park, K. Note: An improvement to the dynamic cycle lengths heuristic for scheduling the multi-item, single-machine, Management Science, *37*, pp. 1201-1205. 1991.

Lee, C.Y. and Danusaputro, S. Economic Lot Scheduling for Two-Product Problem, IIE Transactions, *21*, pp. 162-169. 1989.

Lin, G.C., Kroll, D.E. and Lin, C.J. Determining a common production cycle time for an economic lot scheduling problem with deteriorating items, European Journal of Operational Research, *173*, pp. 669-682. 2006.

Lopez, M.A. and Kingsman, B.G. The economic lot scheduling problem: Theory and practice, International Journal of Production Economics, *23*, pp. 147-164. 1991.

Madigan, J.C. Scheduling a multi-product single machine system for an infinite planning period, Management Science, *14*, pp. 713-719. 1968.

Markowitz, D.M., Reiman M.I. and Wein, L.M. The stochastic economic lot scheduling problem: Heavy traffic analysis of dynamic cyclic policies, Management Science, *14*, pp. 713-719. 1968.

Maxwell, W. The scheduling of economic lot sizes, Naval Research Logistics Quarterly, *11*, pp. 89-124. 1964.

Maxwell, W. and Singh, H. The effect of restricting cycle times in the economic lot scheduling problem, IIE Transactions, *15*, pp. 235-241. 1983.

Maxwell, W. and Singh, H. Scheduling cyclic production on several identical machines, Operations Research, *34*, pp. 460-463. 1986.

Moon, I., Silver, E.A. and Choi, S. Hybrid genetic algorithm for the economic lot scheduling problem, International Journal of Production Research, *40*, pp. 809-824. 2002.

Park, K.S. and Yun, D.K. A stepwise partial enumeration algorithm for the economic lot scheduling problem, IIE Transactions, *16*, pp. 363-370. 1984.

Pena, A. and Zipkin, P. Dynamic Scheduling Rules for a Multiproduct Make-To-Stock Queue, Operations Research, *45*, pp. 919-930. 1997.

Raza, S.A. and Akgunduz A. The use of meta-heuristics to solve economic lot scheduling problem, Evolutionary Computation in Combinatorial Optimization, Springer Berlin, 2005.

Raza, S.A., Akgunduz A. and Chen M.Y. A tabu search algorithm for solving economic lot scheduling problem, Journal of Heuristics, *12*, pp. 413-426. 2006.

Roger, J. A Computational approach to the lot scheduling problem, Management Science, *3*, pp. 264-291. 1958.

Roundy, R. Rounding off to powers of two in continuous relaxations of capacitated lot sizing problems, Management Science, *35*, pp. 1433-1442. 1989.

Singh H. and Foster J.B. Production scheduling with sequence dependent setup costs, IIE Transactions, *19*, pp. 43-49. 1987.

Schweitzer, P.J. and Silver, E.A. Mathematical pitfalls in the one machine multiproduct economic lot scheduling problem, Operations Research, *31*, pp. 401-405. 1983.

Soman, C.A., Van Donk, D.P. and Gaalman, G.J.C. A basic period approach to the economic lot scheduling problem with self life considerations, International Journal of Production Research, *15*, pp. 1677-1689. 2004.

Stankard, M.F. and Gupta, S.K. A note on Bomberger's approach to lot size scheduling: heuristic proposed, Management Science, *15*, pp. 449-452. 1969.

Vemuganti, R.R. On the feasibility of scheduling lot sizes for two products on one machine, Management Science, *24*, pp. 1668-1673. 1978.

Wagner, B.J. and Davis, D.J. A search heuristic for the sequence-dependent economic lot scheduling problem, European Journal of Operational Research, *141*, pp. 133-146. 2002.

Wagner, M. and Smits, S.R. A local search algorithm for the optimization of the stochastic economic lot scheduling problem, International Journal of Production Economics, *90*, pp. 391-402. 2004.

Yao, M. and Elmaghraby, S. The economic lot scheduling problem under power-of-two policy, Computers & Mathematics with Applications, *41*, pp. 1379-1393. 2001.

Yao, M. and Huang, J.X. Solving the economic lot scheduling problem with deteriorating items using genetic algorithms, Journal of Food Engineering, *70*, pp. 309-322. 2005.

Zipkin, P. Computing optimal lot sizes in the economic lot scheduling problem, Operations Research, *39*, pp. 56-63. 1991.

Appendix A

Determination of L_i and U_i for ELSP

1. A choice of U_i.

   For product i, $s_i + \rho_i n_i W \leq W$. Hence, $n_i \leq (W - s_i)/\rho_i W$ and we can choose U_i to be $U_i = \lfloor (W - s_i)/\rho_i W \rfloor$.

2. A choice of L_i.

   For any feasible production schedule, the average machine utilization rate cannot exceed one. Mathematically this means

$$\sum_{i=1}^{I} \left(\frac{s_i}{n_i W} + \rho_i \right) \leq 1$$

   Therefore,

$$\frac{s_{i_0}}{n_{i_0} W} + \rho_{i_0} + \sum_{i \neq i_0} \left(\frac{s_i}{U_i W} + \rho_i \right) \leq 1$$

This implies

$$n_{i_0} \geq \frac{s_{i_0}}{W - \sum_{i \neq i_0}(s_i/U_i + \rho_i W) - \rho_{i_0} W}$$

Hence we can choose L_i to be

$$\max\left\{1, \left\lceil \frac{s_{i_0}}{W - \sum_{i \neq i_0}(s_i/U_i + \rho_i W) - \rho_{i_0} W} \right\rceil \right\}.$$

Appendix B

Bomberger's Stamping Problem

Bomberger's stamping problem data are given in Table B.1. Costs are based on 240 working days per year. Production is based on eight hours per day. The interest rate is 10% per year.

Table B.1 Bomberger's problem

Index	A_i(\$)	c_i (\$/unit)	p_i (units/day)	r_i (units/day)	s_i (hours)
1	15	0.0065	30000	400	1
2	20	0.1775	8000	400	1
3	30	0.1275	9500	800	2
4	10	0.1	7500	1600	1
5	110	2.785	2000	80	4
6	50	0.2675	6000	80	2
7	310	1.5	2400	24	8
8	130	5.9	1300	340	4
9	200	0.9	2000	340	6
10	5	0.04	15000	400	1

Appendix C

Other Benchmark Problems

Bomberger's problem is the first benchmark problems used in Chapter 4. The other five benchmark problems are listed as follows.

Table C.1 Benchmark problem 2

Index	A_i(\$)	c_i (\$/unit)	p_i (units/day)	r_i (units/day)	s_i (hours)
1	50	0.0146	11000	750	2
2	50	0.2644	2000	40	3
3	10	0.2869	1400	500	8
4	260	0.225	7000	160	4
5	70	6.2663	700	50	1
6	160	0.6187	2500	100	2
7	30	0.375	5500	150	1
8	40	0.2333	3000	45	1
9	30	2.025	6000	210	6
10	20	0.09	540216	4500	2

Table C.2 Benchmark problem 3

Index	A_i(\$)	c_i (\$/unit)	p_i (units/day)	r_i (units/day)	s_i (hours)
1	70	0.016	25189	1500	6
2	15	0.522	3770	200	4
3	30	0.0855	3900	130	2
4	30	0.9	1950	240	3
5	50	3.697	5000	600	6
6	10	0.027	15000	3000	8
7	100	1.628	20000	750	2
8	200	6.1	2000	95	1
9	20	0.2	6100	100	4
10	150	0.075	15000	300	1

Table C.3 Benchmark problem 4

Index	A_i(\$)	c_i (\$/unit)	p_i (units/day)	r_i (units/day)	s_i (hours)
1	185	0.2723	20000	200	6
2	300	0.269	37333	5600	8
3	85	0.183	4333	130	7
4	150	2.526	7496	425	1
5	140	0.5262	5498	320	3
6	360	3.414	4245	270	2
7	170	0.1941	2961	90	4
8	50	0.6186	4752	335	5
9	200	1.603	35503	2400	1
10	300	0.199	20000	950	2

Table C.4 Benchmark problem 5

Index	A_i(\$)	c_i (\$/unit)	p_i (units/day)	r_i (units/day)	s_i (hours)
1	50	0.1936	4500	90	5
2	20	0.1232	1539	50	1
3	60	0.2068	2401	40	6
4	45	0.2224	1200	30	8
5	5	0.748	2100	70	7
6	110	0.1056	18000	900	4
7	60	0.417	13714	2400	3
8	70	0.261	5600	70	2
9	90	0.167	6500	65	1
10	250	0.2956	5200	195	1

Table C.5 Benchmark problem 6

Index	A_i(\$)	c_i (\$/unit)	p_i (units/day)	r_i (units/day)	s_i (hours)
1	140	0.95	25000	900	3
2	70	0.235	6000	720	3
3	20	0.065	24000	420	5
4	30	0.22	600	30	8
5	60	0.23	7000	210	6
6	100	0.75	3000	210	7
7	300	1.055	90000	4500	1
8	60	0.14	21000	2100	1
9	55	0.625	9000	900	2
10	350	2.955	40000	900	4

Appendix D

Lower Bound for MELSP

A lower bound can be obtained from the optimal value of the following convex program (Carreno, 1990).

$$\min \sum_{i=1}^{I} \left(\frac{A_i}{T_i} + H_i T_i \right)$$

subject to

$$\sum_{i=1}^{I} \left(\rho_i + \frac{s_i}{T_i} \right) \leq M,$$

$$T_i > 0, \ \forall \ i.$$

where T_i is the cycle time of product i. The optimal solution of the convex program is given by:

$$t_i^* = \sqrt{\frac{A_i + \lambda s_i}{H_i}}, \ \forall \ i.$$

The value of λ is 0 when $\sum_{i=1}^{I} \left(\rho_i + \frac{s_i}{\sqrt{A_i/H_i}} \right) \leq M$. When $\sum_{i=1}^{I} \left(\rho_i + \frac{s_i}{\sqrt{A_i/H_i}} \right) > M$, λ and t_i^*'s satisfy the following system of equations:

$$t_i^* = \sqrt{\frac{A_i + \lambda s_i}{H_i}},\ \forall\ i \qquad \text{and}$$

$$\sum_{i=1}^{I} \left(\rho_i + \frac{s_i}{t_i^*} \right) = M.$$

After λ and t_i^*'s are determined, the lower bound for MELSP is

$$\min \sum_{i=1}^{I} \left(\frac{A_i}{t_i^*} + H_i t_i^* \right).$$

Appendix E

Worst Case Analysis

E.1 ELSP

Common cycle policy

We construct a series of examples to show that the solution under the CC policy can be arbitrarily bad compared to the optimal solution under the EBP and PoT policy. For a given n from the set $\{1, 2, 4, \ldots\}$, the example parameters are as follows:

Number of products:	$n+1$;
Setup cost:	$A_i = 1,\ i = 1,\ \ldots,\ n+1$;
Factor of holding cost:	$H_1 = 1$;
	$H_i = 1/n^2,\ i = 2,\ \ldots,\ n+1$;
Setup time:	$s_1 = 1/n$;
	$s_i = 1/2,\ \ i = 2,\ \ldots,\ n+1$;
Production density:	$\rho_1 = 1/n$;
	$\rho_i = 1/3n,\ \ i = 2,\ \ldots,\ n+1$.

An optimal solution of the problem under the EBP and PoT policy is to produce product 1 with cycle time 1 and produce product i $(2 \leq i \leq n+1)$ with cycle time n. The cost of the optimal solution is:

$$C_{EBP} = A_1/T_1 + H_1T_1 + \sum_{i=2}^{n+1}(A_i/T_i + H_iT_i) = 4$$

Under the CC policy, it is required that

$$T > \sum_{i=1}^{n+1} s_i > n/2$$

Therefore, the cost of a solution under the CC policy is

$$C_{CC} > A_1/T + H_1T > H_1T > n/2$$

and

$$\lim_{n\to\infty} \frac{C_{CC}}{C_{EBP}} > \lim_{n\to\infty} \frac{n/2}{4} = \infty$$

Basic period policy

The BP policy allows different products to have different cycle times. It is less restrictive than the CC policy, but it is still not flexible enough as it requires W to be big enough to accommodate the production of all the products. For the constructed example in the previous section, it is required that

$$W > \sum_{i=1}^{n+1} s_i > n/2$$

So the cost of a solution under the BP policy is

$$C_{BP} = \sum_{i=1}^{n+1}(A_i/n_iW + H_in_iW) > H_1n_1W \geq H_1W > n/2$$

and

$$\lim_{n\to\infty} \frac{C_{BP}}{C_{EBP}} > \lim_{n\to\infty} \frac{n/2}{4} = \infty$$

In the worst case, the solution under the BP policy is arbitrarily bad compared to the solution under the EBP and PoT policy.

E.2 MELSP

Again, we construct a series of examples to show that the solution under the CC policy can be arbitrarily bad compared to the optimal solution under the EBP and PoT policy. For a given n from the set $\{1, 2, 4, \ldots\}$, the example parameters are as follows:

Number of machines:	M;
Number of products:	$nM + 1$;
Setup cost:	$A_i = 1,\ i = 1,\ \ldots,\ nM + 1$;
Factor of holding cost:	$H_1 = 1$;
	$H_i = 1/n^2,\ i = 2,\ \ldots,\ nM + 1$;
Setup time:	$s_1 = 1/(M + 1)n$;
	$s_i = 1/(M + 1)^2,\ \ i = 2,\ \ldots,\ nM + 1$;

Production density: $\rho_i = M/(M+1)n,\ i = 1,\ \ldots,\ nM+1;$

An optimal solution under the EBP and PoT policy is to produce products $1,\ \ldots,\ n+1$ on machine 1 and produce products $(m-1)n+2,\ \ldots,\ nm+1$ on machine m $(2 \leq m \leq M)$. On the first machine, the cycle time of product 1 is 1 and the cycle time of product i $(2 \leq i \leq n+1)$ is n. On machine m $(2 \leq m \leq M)$, the cycle time of product i $((m-1)n+2 \leq i \leq mn+1)$ is n. The cost of this solution is:

$$C_{EBP} = \sum_{i=1}^{nM+1} (A_i/T_i + H_i T_i) = 2 + nM(1/n + 1/n) = 2M + 2$$

The cost of the solution under the CC policy is analyzed as follows. Machine m $(2 \leq m \leq M)$ can at most produce $(M+1)n/M - 1$ products as $\rho_i = M/(M+1)n$ $(i \geq 2)$, so at least $n/M + M$ products should be produced on machine 1, which is calculated by

$$\begin{aligned} 1 + nM - \left(\frac{(M+1)n}{M} - 1\right)(M-1) &= nM - (nM - \frac{n}{M} - M + 1) + 1 \\ &= \frac{n}{M} + M \end{aligned}$$

So the cycle time of product 1 is required to accommodate $n/M + M - 1$ products with setup time $1/(1+M)^2$, which is

$$T_1 > \left(\frac{n}{M} + M - 1\right)/(M+1)^2$$

so

$$C_{CC} > H_1 T_1 = \left(\frac{n}{M} + M - 1\right) / (M+1)^2$$

and

$$\lim_{n\to\infty} \frac{C_{CC}}{C_{EBP}} > \lim_{n\to\infty} \frac{\left(\frac{n}{M} + M - 1\right) / (M+1)^2}{2M+2} = \infty$$

Appendix F

Convergence for Genetic Algorithms for MELSP

The x-axis represents the number of generations and the y-axis represents the fitness value of the best feasible solution found.

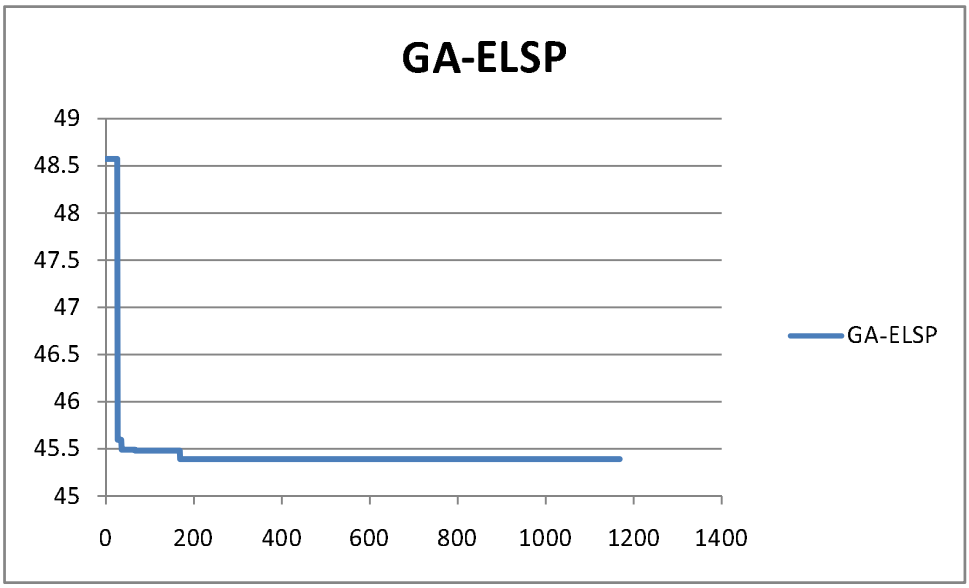

Figure F.1 Convergence of GA for ELSP

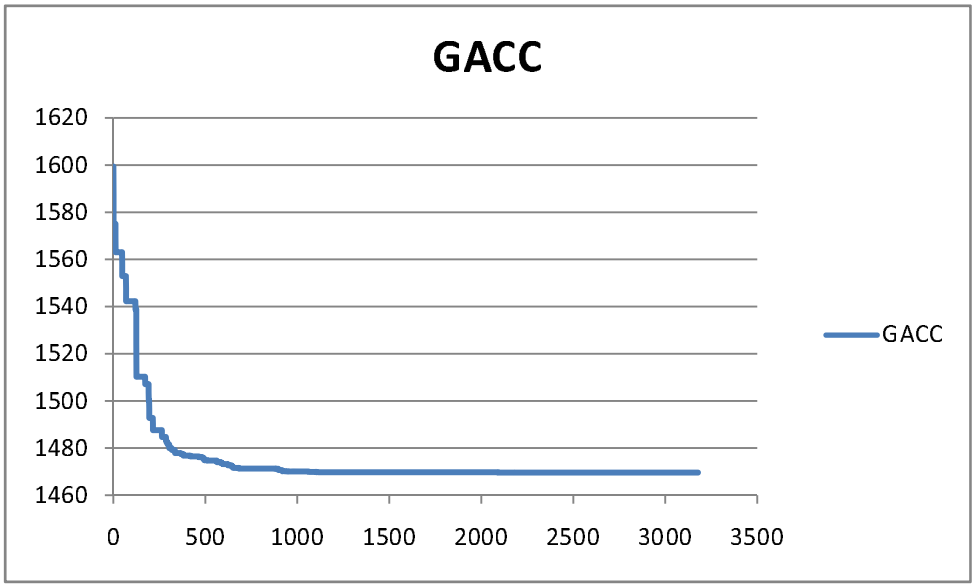

Figure F.2 Convergence of GACC

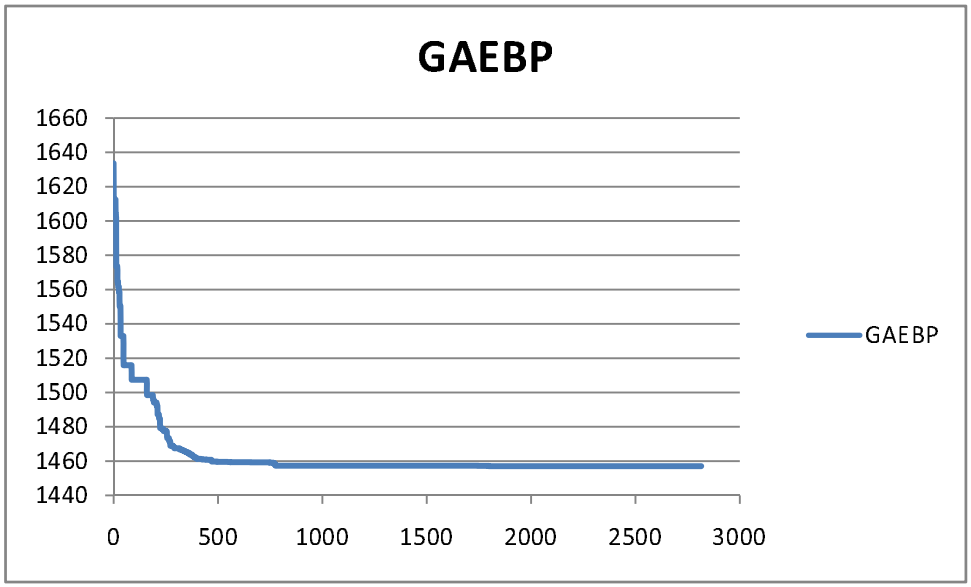

Figure F.3 Convergence of GAEBP